1. What is JavaScript?

Answer: JavaScript is a high-level, interpreted programming language that conforms to the ECMAScript specification. It is a multi-paradigm language that supports event-driven, functional, and imperative programming styles. JavaScript is primarily used for client-side scripting in web browsers but can also be used on the server-side (Node.js).

2. What are the data types in JavaScript?

Answer: JavaScript has 8 basic data types:

- Primitive types:
    - String - textual data
    - Number - integer and floating-point numbers
    - BigInt - arbitrary-length integers
    - Boolean - true/false
    - Undefined - uninitialized variable
    - Null - intentional absence of any object value
    - Symbol - unique identifier
- Non-primitive type:
    - Object - collections of key-value pairs

3. What is the difference between let, const, and var?

Answer:

- var: Function-scoped, can be redeclared and updated, hoisted to the top of its scope
- let: Block-scoped, can be updated but not redeclared, hoisted but not initialized
- const: Block-scoped, cannot be updated or redeclared, must be initialized during declaration

4. What is hoisting in JavaScript?

Answer: Hoisting is JavaScript's default behavior of moving declarations to the top of their containing scope. Variable and function declarations are hoisted, but not initializations. Only the declaration is hoisted, not the assignment.

5. What is closure in JavaScript?

Answer: A closure is a function that has access to its own scope, the outer function's variables, and global variables - even after the outer function has returned. Closures are created every time a function is created.

Example:

```
function outer() {
  let count = 0;
```

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

Functions and Scope

6. What is the difference between == and ===?

Answer:

- == (loose equality) compares values after performing type coercion if necessary
- === (strict equality) compares both value and type without type coercion

7. What are arrow functions?

Answer: Arrow functions are a concise syntax for writing function expressions in ES6. They don't have their own this, arguments, super, or new.target bindings, making them unsuitable for methods or constructors.

Example:

javascript

Copy

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

8. What is the this keyword in JavaScript?

Answer: this refers to the object that the function is a property of or the object that the function is called on. The value of this depends on how the function is called:

- In a method: this refers to the owner object
- In a function: this refers to the global object (window in browsers)
- In strict mode function: this is undefined
- In an event: this refers to the element that received the event
- With call(), apply(), bind(): this refers to the object passed as argument

9. What is the difference between call(), apply(), and bind()?

Answer:

- call(): Invokes the function with a given this value and arguments provided individually
- apply(): Similar to call() but accepts arguments as an array
- bind(): Returns a new function with a bound this value and optional preset arguments

10. What are higher-order functions?

Answer: Higher-order functions are functions that operate on other functions by taking them as arguments or returning them. Examples include map(), filter(), and reduce().

Asynchronous JavaScript

11. What is the event loop?

Answer: The event loop is what allows JavaScript to be non-blocking and asynchronous. It continuously checks the call stack and if it's empty, it takes the first task from the callback queue and pushes it to the call stack. It consists of:

- Call Stack
- Web APIs
- Callback Queue (Task Queue)
- Microtask Queue (for Promises)

12. What are promises?

Answer: A Promise is an object representing the eventual completion or failure of an asynchronous operation. It has three states:

- Pending: Initial state
- Fulfilled: Operation completed successfully
- Rejected: Operation failed

Example:

```
const promise = new Promise((resolve, reject) => {
  if (success) {
    resolve(value);
  } else {
    reject(error);
  }
});
```

13. What is async/await?

Answer: Async/await is syntactic sugar built on top of promises that allows writing asynchronous code that looks synchronous. An async function returns a promise, and await pauses the execution until the promise is resolved.

Example:

```
async function fetchData() {
  try {
    const response = await fetch('url');
    const data = await response.json();
    return data;
  } catch (error) {
    console.error(error);
  }
}
```

14. What is the difference between setTimeout and setInterval?

Answer:

- setTimeout() executes a function once after a specified delay
- setInterval() executes a function repeatedly with a fixed time delay between each call

Objects and Prototypes

15. What is prototypal inheritance?

Answer: JavaScript uses prototypal inheritance where objects can inherit properties from other objects. Each object has a prototype object from which it inherits properties. This forms a prototype chain until null is reached.

16. What is the difference between Object.create() and new?

Answer:

- Object.create() creates a new object with the specified prototype object and properties
- The new operator creates an instance of a user-defined object type or of one of the built-in object types that has a constructor function

17. What are getters and setters?

Answer: Getters and setters are functions that get or set the value of an object's property. They are defined using get and set keywords.

Example:

```
const obj = {
  _name: '',
  get name() {
```

```
  return this._name;
 },
 set name(value) {
  this._name = value;
 }
};
```

ES6+ Features

18. What are template literals?

Answer: Template literals are string literals allowing embedded expressions, multi-line strings, and string interpolation using backticks (`) and ${expression}.

Example:

```
const name = 'John';
console.log(`Hello ${name}!`);
```

19. What are default parameters?

Answer: Default parameters allow named parameters to be initialized with default values if no value or undefined is passed.

Example:

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

20. What is destructuring assignment?

Answer: Destructuring assignment is a syntax that allows unpacking values from arrays or properties from objects into distinct variables.

Example:

```
// Array destructuring
const [a, b] = [1, 2];

// Object destructuring
const {name, age} = {name: 'John', age: 30};
```

Error Handling

21. How do you handle errors in JavaScript?

Answer: Errors can be handled using try...catch...finally blocks. Custom errors can be thrown using the throw statement.

Example:

```
try {
  // Code that may throw an error
  throw new Error('Something went wrong');
} catch (error) {
  console.error(error.message);
} finally {
  // Code that runs regardless of error
}
```

Browser and DOM

22. What is the DOM?

Answer: The Document Object Model (DOM) is a programming interface for HTML and XML documents. It represents the page so that programs can change the document structure, style, and content.

23. What is event delegation?

Answer: Event delegation is a technique where instead of adding event listeners to individual elements, you add a single event listener to a parent element that will fire for all descendants matching a selector, whether they exist now or are added in the future.

24. What is localStorage and sessionStorage?

Answer:

- localStorage: Stores data with no expiration date, cleared only through JavaScript or clearing browser cache
- sessionStorage: Stores data for one session (data is lost when the browser tab is closed)

Advanced Concepts

25. What is memoization?

Answer: Memoization is an optimization technique where the results of expensive function calls are cached so that when the same inputs occur again, the cached result can be returned.

Example:

javascript

Copy

```
function memoize(fn) {
```

```
  const cache = {};
  return function(...args) {
    const key = JSON.stringify(args);
    if (cache[key]) {
      return cache[key];
    }
    const result = fn.apply(this, args);
    cache[key] = result;
    return result;
  };
}
```

26. What is currying?

Answer: Currying is a technique of evaluating functions with multiple arguments into a sequence of functions with single arguments.

Example:

javascript

Copy


```
function curry(fn) {
  return function curried(...args) {
    if (args.length >= fn.length) {
      return fn.apply(this, args);
    } else {
      return function(...args2) {
        return curried.apply(this, args.concat(args2));
      };
    }
  };
}
```

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

Answer:

- Shallow copy: Copies only the top-level properties, nested objects are shared between original and copy

- Deep copy: Creates a new copy of all levels of the object hierarchy

28. What is the Temporal Dead Zone (TDZ)?

Answer: The TDZ is the period between entering scope and being declared where variables (let and const) cannot be accessed. Attempting to access them results in a ReferenceError.

Testing and Debugging

29. How do you debug JavaScript code?

Answer: Common debugging techniques include:

- Using console.log()
- Browser developer tools (breakpoints, step-through debugging)
- Debugger statement (debugger;)
- Linting tools
- Unit testing frameworks

Performance

30. How do you optimize JavaScript performance?

Answer: JavaScript performance can be optimized by:

- Minimizing DOM access and manipulation
- Using event delegation
- Debouncing/throttling frequent events
- Using Web Workers for CPU-intensive tasks
- Code splitting and lazy loading
- Minimizing repaints and reflows
- Using efficient algorithms and data structures

This list covers a wide range of JavaScript interview questions from basic to advanced concepts. Be prepared to explain these concepts clearly and provide practical examples during interviews.

continue

31. What are Immediately Invoked Function Expressions (IIFE)?

Answer: An IIFE is a function that runs as soon as it is defined. It helps create private scope and avoid polluting the global namespace.

javascript

Copy

(function() {

```
  console.log('This runs immediately');
})();
```

32. What is the difference between null and undefined?

Answer:

- undefined means a variable has been declared but not assigned a value
- null is an assignment value representing no value or no object

33. What are JavaScript design patterns?

Answer: Common JavaScript design patterns include:

- Module Pattern
- Singleton Pattern
- Factory Pattern
- Observer Pattern
- Prototype Pattern
- MVC Pattern

34. What are Web Workers?

Answer: Web Workers allow running scripts in background threads separate from the main execution thread, preventing UI freezing during heavy computations.

javascript

Copy

```
const worker = new Worker('worker.js');
worker.postMessage(data);
worker.onmessage = function(e) {
  console.log('Message from worker:', e.data);
};
```

35. What is CORS?

Answer: Cross-Origin Resource Sharing (CORS) is a mechanism that allows restricted resources on a web page to be requested from another domain outside the domain from which the first resource was served.

36. What are generators in JavaScript?

Answer: Generators are functions that can be exited and later re-entered, with their context saved across re-entrances. They are defined with function* and use yield to pause execution.

javascript

Copy

```
function* idGenerator() {
  let id = 1;
  while(true) {
    yield id++;
  }
}
const gen = idGenerator();
console.log(gen.next().value); // 1
console.log(gen.next().value); // 2
```

37. What are JavaScript modules?

Answer: JavaScript modules are reusable pieces of code that can be exported from one program and imported for use in another. ES6 introduced import/export syntax.

javascript

Copy

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



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

38. What is the difference between forEach and map?

Answer:

- forEach: Executes a provided function once for each array element (no return value)
- map: Creates a new array with the results of calling a provided function on every element

39. What is the purpose of the reduce method?

Answer: reduce executes a reducer function on each element of the array, resulting in a single output value.

javascript

Copy

```
const sum = [1, 2, 3].reduce((acc, val) => acc + val, 0);
// sum = 6
```

40. What is the difference between slice and splice?

Answer:

- slice: Returns a shallow copy of a portion of an array (doesn't modify original)
- splice: Changes the contents of an array by removing/replacing elements (modifies original)

41. What are JavaScript Proxies?

Answer: Proxies allow you to create a proxy for another object that can intercept and redefine fundamental operations for that object.

javascript

Copy

```
const handler = {
  get(target, prop) {
    return prop in target ? target[prop] : 37;
  }
};
const p = new Proxy({}, handler);
p.a = 1;
console.log(p.a, p.b); // 1, 37
```


42. What is the purpose of the void operator?

Answer: The void operator evaluates an expression and then returns undefined. It's often used to obtain the undefined primitive value.

javascript

Copy

```
void function test() {
  console.log('Test');
}();
// Returns undefined
```

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

Answer:

- Object.freeze(): Makes an object immutable (can't add, modify, or delete properties)
- Object.seal(): Prevents adding or removing properties, but allows modifying existing properties

44. What are tagged template literals?

Answer: Tagged templates are function calls that use template literals. The function can process the template literal and return a manipulated string.

javascript

Copy

```
function highlight(strings, ...values) {
  return strings.reduce((result, str, i) =>
    `${result}${str}<span class="hl">${values[i] || ''}</span>`, '');
}

const name = 'John';
const age = 30;
const sentence = highlight`Hello ${name}, you are ${age} years old.`;
```

45. What is the with statement and why is it discouraged?

Answer: The with statement extends the scope chain for a statement. It's discouraged because:

- Makes code harder to understand
- Slower performance
- Can lead to confusing bugs
- Strict mode doesn't allow it

46. What are WeakMap and WeakSet?

Answer:

- WeakMap: Collection of key/value pairs where keys are objects and weakly referenced
- WeakSet: Collection of objects where each object may occur only once (weakly referenced)

They don't prevent garbage collection of their elements.

47. What is the difference between Array.from() and the spread operator?

Answer:

- Array.from(): Creates a new array from array-like or iterable objects
- Spread operator (...): Expands iterable objects into individual elements

Array.from() can also take a mapping function as second argument.

48. What are the benefits of using strict mode?

Answer: Benefits include:

- Prevents accidental global variables
- Makes assignments that would silently fail throw errors

- Makes this undefined in functions called globally
- Disallows duplicate parameter names
- Makes eval safer by not leaking variables
- Throws error on attempts to delete undeletable properties

49. What is the difference between in and hasOwnProperty?

Answer:

- in: Returns true if property exists in object or its prototype chain
- hasOwnProperty: Returns true only if property exists directly on the object

50. How does JavaScript handle type coercion?

Answer: JavaScript performs implicit type conversion when operators are applied to values of different types. Rules include:

- + prefers string concatenation
- Comparison operators (==) perform type coercion
- Logical operators (&&, ||, !) coerce to boolean
- Explicit coercion can be done with Number(), String(), Boolean()

51. What is tail call optimization?

Answer: Tail call optimization is where a recursive function call is the last operation in the function, allowing the engine to reuse the current stack frame instead of creating a new one. ES6 specifies support for TCO but it's not widely implemented.

52. What are the differences between async/await and Promises?

Answer:

- async/await is syntactic sugar over Promises
- async/await makes asynchronous code look synchronous
- Error handling is simpler with try/catch in async/await
- async/await may be easier to read and debug
- Both ultimately work with Promises under the hood

53. What is the arguments object?

Answer: The arguments object is an array-like object available inside functions that contains the values of the arguments passed to that function. It's not available in arrow functions.

javascript

Copy

function sum() {

```
  let total = 0;
  for(let i = 0; i < arguments.length; i++) {
    total += arguments[i];
  }
  return total;
}
```

54. What is the purpose of the Symbol type?

Answer: Symbols are unique and immutable primitive values that can be used as object property keys. They help:

- Create unique property keys that won't clash
- Add properties to objects you don't own without risk of name collisions
- Define well-known symbols to customize object behavior

55. What is the difference between Object.keys() and Object.getOwnPropertyNames()?

Answer:

- Object.keys(): Returns an array of enumerable own property names
- Object.getOwnPropertyNames(): Returns all own property names (enumerable and non-enumerable)

56. What are decorators in JavaScript?

Answer: Decorators are a proposal for extending JavaScript classes and properties at design time (currently stage 3 proposal). They use the @decorator syntax.

javascript

Copy

```
@log
class MyClass {
  @readonly
  method() {}
}
```

57. What is the difference between throw Error and throw new Error()?

Answer: There's no practical difference - both create a new Error object. throw Error() is shorthand for throw new Error().

58. What is the purpose of the finally block in try/catch?

Answer: The finally block executes after the try and catch blocks, regardless of whether an exception was thrown or caught. It's useful for cleanup code.

59. What are the different ways to create objects in JavaScript?

Answer:

1. Object literal: const obj = {};
2. Constructor function: new Object()
3. Object.create()
4. Class syntax: class MyClass {}; new MyClass()
5. Factory functions
6. Using new with built-in constructors (new Array(), etc.)

60. What is the difference between instanceof and typeof?

Answer:

- typeof: Returns a string indicating the type of the unevaluated operand
- instanceof: Tests whether an object has a constructor's prototype in its prototype chain

61. What is the purpose of the ArrayBuffer object?

Answer: ArrayBuffer represents a fixed-length raw binary data buffer. It's used for working with binary data directly, often in conjunction with typed arrays (Int8Array, Uint32Array, etc.) and DataView.

62. What is the Temporal API?

Answer: The Temporal API is a new date/time API proposal for JavaScript that aims to fix problems with the existing Date object by providing:

- Better immutability
- More comprehensive time zone support
- More intuitive API
- Better parsing and formatting
- More precise arithmetic operations

63. What is the difference between globalThis and window?

Answer:

- window: The global object in browser environments
- globalThis: Standard way to access the global object in any environment (browsers, Node.js, etc.)

64. What are the different ways to handle asynchronous code in JavaScript?

Answer:

1. Callbacks

2. Promises
3. Async/await
4. Generators with promises
5. Event emitters
6. Observables (RxJS)

65. What is the purpose of the Intl object?

Answer: The Intl object provides language-sensitive string comparison, number formatting, and date/time formatting through its constructors (Intl.Collator, Intl.NumberFormat, Intl.DateTimeFormat).

66. What is the difference between Array.of() and Array() constructor?

Answer:

- Array(3): Creates an array with length 3 and empty slots
- Array.of(3): Creates an array with one element: [3]

67. What is the purpose of the at() method for arrays?

Answer: The at() method takes an integer value and returns the item at that index, allowing for positive and negative integers (negative integers count back from the last item).

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

arr.at(-1); // 3
```


68. What is the difference between Object.assign() and the spread operator for objects?

Answer:

- Both can be used to shallow copy objects
- Spread syntax is more concise
- Object.assign() can modify an existing object (first parameter)
- Spread syntax can't add properties to an existing object directly

69. What is the purpose of the ?? (nullish coalescing) operator?

Answer: The ?? operator returns its right-hand side operand when its left-hand side operand is null or undefined, otherwise returns its left-hand side operand.

```
const value = null ?? 'default'; // 'default'
```

70. What is the difference between ?. (optional chaining) and && for property access?

Answer:

- Optional chaining (?.): Returns undefined if reference is null or undefined
- && operator: Checks for all falsy values (not just null/undefined)

obj?.prop; // undefined if obj is null/undefined

obj && obj.prop; // undefined if obj is any falsy value

This comprehensive list covers fundamental to advanced JavaScript concepts that are commonly asked in technical interviews. Being able to explain and demonstrate these concepts will significantly improve your chances in JavaScript interviews.

DOM

1. What is the DOM?

Answer:
The Document Object Model (DOM) is a programming interface for HTML and XML documents that represents the page as a tree of objects (nodes). JavaScript can manipulate these objects to:

- Change HTML elements and attributes
- Modify CSS styles
- Add/remove elements
- Respond to user events

// Example: Accessing the DOM

document.getElementById("myElement");

2. What are the different ways to select DOM elements?

Answer:
Common selection methods:

// Single element selectors

document.getElementById("id"); // Returns single element

document.querySelector(".class"); // Returns first match

// Multiple element selectors

document.getElementsByClassName("class"); // HTMLCollection

document.getElementsByTagName("div"); // HTMLCollection

document.querySelectorAll("p.highlight"); // NodeList

3. What's the difference between getElementById and querySelector?

Answer:

- getElementById() only selects by ID and returns a single element
- querySelector() uses CSS selectors and returns the first match

- querySelector() is more versatile but slightly slower for ID selection

```
document.getElementById("header");  // Faster for IDs
document.querySelector("#header");  // More flexible
```

4. How do you create and add new elements to the DOM?

Answer:
Three-step process:

```
// 1. Create element
const newDiv = document.createElement("div");

// 2. Configure element
newDiv.textContent = "Hello World";
newDiv.classList.add("box");

// 3. Add to DOM
document.body.appendChild(newDiv);       // End of body
parentElement.insertBefore(newDiv, referenceElement); // Specific position
```

5. What's the difference between innerHTML and textContent?

Answer:

- innerHTML: Gets/sets HTML content (parses tags)
- textContent: Gets/sets text content (escapes tags)

```
element.innerHTML = "<strong>Bold</strong>";   // Renders bold text
element.textContent = "<strong>Bold</strong>"; // Shows tags as text
```

Security Note: innerHTML can expose XSS vulnerabilities if used with untrusted content.

6. How do you handle events in the DOM?

Answer:
Three approaches:

```
// 1. HTML attribute (avoid)
<button onclick="handleClick()">Click</button>

// 2. DOM property (good for single handler)
element.onclick = function() { console.log("Clicked"); };
```

```
// 3. addEventListener (best practice - multiple handlers)
element.addEventListener("click", function(e) {
  console.log("Clicked", e.target);
});
```

7. What is event delegation and why is it useful?

Answer:
Event delegation attaches a single event listener to a parent element to handle events from multiple child elements. Benefits:

- Better performance (fewer listeners)
- Works for dynamically added elements
- Less memory usage

```
document.getElementById("parent").addEventListener("click", function(e) {
  if(e.target.classList.contains("child")) {
    console.log("Child clicked:", e.target);
  }
});
```


8. What's the difference between window.load and DOMContentLoaded?

Answer:

- DOMContentLoaded: Fires when initial HTML is parsed (no styles/images needed)
- window.load: Fires after all resources (images, styles) finish loading

```
document.addEventListener("DOMContentLoaded", () => {
  console.log("DOM ready!");
});


window.addEventListener("load", () => {
  console.log("All resources loaded");
});
```

9. How do you modify element styles?

Answer:
Three approaches:

```
// 1. style property (inline styles)
```

```
element.style.color = "blue";
element.style.backgroundColor = "#eee";

// 2. classList API (recommended)
element.classList.add("active");
element.classList.remove("hidden");
element.classList.toggle("visible");

// 3. Direct className assignment (overwrites all classes)
element.className = "new-class";
```

10. What's the difference between parentNode and parentElement?

Answer:

- parentNode: Returns any type of parent node (including text nodes, document fragments)
- parentElement: Returns only element parent nodes (null if parent isn't an element)

```
// Usually identical for element nodes
element.parentNode;    // Works in all cases
element.parentElement; // Returns null for document nodes
```