## 1. Explain the difference between `var`, `let`, and `const`.

- `var` : Function-scoped, can be redeclared and hoisted.
- `let` : Block-scoped, cannot be redeclared, but can be reassigned.
- `const` : Block-scoped, cannot be redeclared or reassigned (though objects can be mutated).

## 2. What is the purpose of closures in JavaScript, and how do they work?

- A closure is created when an inner function accesses variables from its outer function even after the outer function has returned. This is possible because of lexical scoping.

## 3. What is the difference between shallow copy and deep copy?

- A shallow copy only copies the first level of an object, whereas a deep copy recursively copies all nested objects. Shallow copies share references to nested objects, while deep copies create independent objects.

## 4. Explain how prototypal inheritance works in JavaScript.

- In JavaScript, objects inherit properties and methods from their prototype. Every object has a `[[Prototype]]` reference to another object, and when a property is not found on the object itself, JavaScript looks it up on the prototype chain.

## 5. What is the event loop, and how does it work in JavaScript?

- The event loop allows JavaScript to perform non-blocking I/O operations by pushing asynchronous callbacks (e.g., from `setTimeout` ) into a queue, which are executed after the current call stack is empty.

## 6. How do `async` and `await` work in JavaScript?

- `async` functions return promises. Inside an `async` function, `await` pauses execution until the promise is resolved, allowing asynchronous code to be written in a synchronous style.

## 7. What are JavaScript Promises, and how do they work?

- A Promise represents an asynchronous operation that can either resolve or reject. It has three states: pending, fulfilled, or rejected. Handlers ( `.then` , `.catch` , `.finally` ) can be attached for subsequent actions.

## 8. What is the concept of 'hoisting' in JavaScript?

- Hoisting is JavaScript's behavior of moving variable and function declarations to the top of their scope before execution. Only declarations are hoisted, not initializations.

## 9. What is currying, and how does it benefit JavaScript code?

- Currying is the process of transforming a function that takes multiple arguments into a sequence of functions that each take a single argument. It improves modularity and reusability.

## 10. What is a JavaScript generator, and how does it differ from regular functions?

- A generator function (defined with `function*` ) allows you to yield values one by one using `yield` . It can pause execution and later resume from where it left off.

## 11. Explain the concept of memoization and how it can be implemented in JavaScript.

- Memoization is an optimization technique where you cache the result of expensive function calls and return the cached result when the same inputs occur again. This can be implemented using closures or objects.

## 12. What are higher-order functions in JavaScript?

- Higher-order functions are functions that can take other functions as arguments or return functions. Examples include `map` , `filter` , and `reduce` .

## 13. Explain the difference between `call` , `apply` , and `bind` .

- `call` : Invokes a function with a specified `this` context and arguments passed individually.
- `apply` : Similar to `call` , but arguments are passed as an array.
- `bind` : Returns a new function with the specified `this` context and pre-set arguments.

## 14. What is the significance of the `this` keyword in JavaScript?

- `this` refers to the context in which a function is executed. Its value depends on how a function is called: in a method, it refers to the object; in a function, it can be `undefined` in strict mode or the global object.

## 15. Explain the module pattern in JavaScript.

- The module pattern encapsulates private and public variables and methods using closures. It helps in maintaining a clean global namespace by exposing only the necessary public API.

## 16. What is function composition, and how is it achieved in JavaScript?

- Function composition is the process of combining two or more functions such that the output of one function becomes the input of the next. It can be implemented like `const composed = f(g(h(x)));` .

## 17. What are IIFEs, and why are they useful?

- Immediately Invoked Function Expressions (IIFEs) are functions that are executed right after they are defined. They are useful for creating local scopes and avoiding polluting the global scope.

## 18. What is the difference between synchronous and asynchronous code?

- Synchronous code is executed line by line, blocking further execution until each line is completed. Asynchronous code allows multiple tasks to be executed concurrently by running some tasks in the background.

## 19. How do you prevent object mutation in JavaScript?

- Object mutation can be prevented using `Object.freeze` (makes the object immutable), `Object.seal` (prevents adding/removing properties but allows modification of existing ones), or by using deep copies.

## 20. What are ES6 symbols, and how can they be used?

- Symbols are unique and immutable values used as property keys that prevent name collisions. They can be used to define non-enumerable properties or implement private-like properties in objects.

## 21. Explain the purpose of `Object.assign` and how it differs from the spread operator.

- `Object.assign` copies all enumerable properties from one or more source objects to a target object. The spread operator `...` also copies properties, but can be used for more operations, like spreading arrays or other iterables.

## 22. What is tail call optimization, and when is it applied in JavaScript?

- Tail call optimization allows recursive functions to execute without increasing the call stack size when the recursive call is the last operation in a function. This reduces memory usage and prevents stack overflows.

## 23. What is a proxy in JavaScript, and how can it be used?

- A proxy allows you to intercept and redefine fundamental operations on objects, such as property access and assignment. It is useful for creating custom behaviors for objects.

## 24. Explain the concept of “call stack” and “task queue” in JavaScript.

- The call stack holds function calls to be executed in a last-in-first-out (LIFO) manner. The task queue contains asynchronous callbacks that are pushed to the call stack when the stack is empty (via the

event loop).

## 25. What is the difference between `undefined` and `null` in JavaScript?

- `undefined` represents the absence of a value where no value was assigned, while `null` is an assignment value that explicitly indicates no value or an empty object reference.

## 26. Explain destructuring and its use cases.

- Destructuring allows you to unpack values from arrays or properties from objects into distinct variables. It simplifies the extraction of multiple values from arrays or objects.

## 27. What is the purpose of `Object.create`, and how is it different from a constructor function?

- `Object.create` creates a new object with a specified prototype object and properties. Unlike constructor functions, it allows more flexibility in prototype chain manipulation.

## 28. What are WeakMap and WeakSet, and how do they differ from Map and Set?

- WeakMap and WeakSet store only weak references to objects, meaning if no other references exist, they can be garbage collected. They are used for cases where memory efficiency is needed without holding strong references.

## 29. What is the Temporal Dead Zone (TDZ) in JavaScript?

- The TDZ is the time between when a variable is declared with `let` or `const` and when it is initialized. Accessing the variable in this period will throw a ReferenceError.

## 30. Explain how JavaScript's `new` keyword works under the hood.

- The `new` keyword creates a new object, sets the `this` context of the constructor function to the newly created object, and links the object to the constructor's prototype. It returns the object unless the constructor explicitly returns another object.

## 31. How can the `typeof` operator be misleading?

- `typeof null` returns `"object"`, which is a long-standing bug in JavaScript. Similarly, `typeof NaN` returns `"number"`, though NaN is not a valid number.

## 32. Explain how `Object.defineProperty` works and when you would use it.

- `Object.defineProperty` allows you to define a new property or modify an existing one on an object with precise control over property descriptors (e.g., writable, enumerable, configurable). It is useful for creating non-enumerable or read-only properties.

## 33. What is the difference between `Array.prototype.map` and `Array.prototype.forEach`?

- `map` returns a new array with the results of applying a function to every element, while `forEach` iterates over

the array but does not return a new array.

## 34. What are "rest" and "spread" operators in JavaScript?

- The rest operator ( `...args` ) collects all remaining arguments into an array, while the spread operator ( `...` ) expands an array or object into individual elements or properties.

## 35. Explain `Function.prototype.apply` and its use cases.

- `apply` allows you to call a function with a specified `this` value and arguments provided as an array. It's useful when you want to invoke a function with an array of arguments.

## 36. What are template literals, and how do they improve string handling?

- Template literals are enclosed by backticks and allow embedding expressions via `${}`. They simplify string concatenation and enable multi-line strings.

## 37. How does destructuring work with function arguments in JavaScript?

- Destructuring can be used in function parameters to directly extract properties from objects or elements from arrays, making code cleaner when handling complex input data.

## 38. What is the purpose of `Object.getOwnPropertyDescriptors`, and how does it differ from `Object.getOwnPropertyDescriptor`?

- `Object.getOwnPropertyDescriptors` returns all property descriptors of an object, while `Object.getOwnPropertyDescriptor` returns the descriptor for a single property.

## 39. What is the difference between the global execution context and the function execution context in JavaScript?

- The global execution context is the default environment when the script starts, while each function call creates a new function execution context with its own local variables and scope chain.

## 40. Explain the `instanceof` operator and how it works.

- `instanceof` checks whether an object's prototype chain includes the prototype of the constructor function. It returns `true` if the object was created using that constructor or its prototype.

## 41. What are arrow functions, and how do they handle the `this` keyword differently?

- Arrow functions inherit the `this` value from their surrounding context (lexical `this`). Unlike regular functions, they do not create their own `this` binding.

## 42. What are tagged template literals, and how do they work?

- Tagged template literals allow you to call a function with a template literal as the argument, where the function can process the literal parts and interpolated expressions separately.

## 43. What is the difference between `Object.keys` and `Object.entries`?

- `Object.keys` returns an array of an object's own enumerable property names, while `Object.entries` returns an array of key-value pairs.

## 44. What are mixins in JavaScript, and how can they be used?

- Mixins allow objects or classes to share functionality by copying properties or methods from one object to another. This promotes code reuse without traditional inheritance.

## 45. Explain the concept of "throttling" in JavaScript.

- Throttling ensures that a function is executed at most once in a specified time period, preventing it from being called repeatedly in quick succession (e.g., during scrolling).

## 46. What is "debouncing" in JavaScript, and how does it differ from throttling?

- Debouncing delays the execution of a function until after a certain period of inactivity. If the function is invoked again before the delay ends, the timer resets. It ensures the function is executed once after the events stop firing.

## 47. What is the purpose of `Reflect` in JavaScript?

- `Reflect` provides methods to manipulate and interact with objects. It helps standardize low-level object operations, such as creating, defining, or deleting properties, in a more consistent way than direct object manipulation.

## 48. How does garbage collection work in JavaScript?

- JavaScript uses automatic garbage collection. It tracks object references and automatically reclaims memory when an object is no longer reachable from the root (global scope or function stack).

## 49. What is the difference between a promise and an observable in JavaScript?

- A promise represents a single asynchronous value or completion, whereas an observable can handle a stream of values and allow multiple emissions over time, making it more flexible for reactive programming.

## 50. What is the significance of immutability in JavaScript, and how can it be enforced?

- Immutability ensures that objects cannot be modified after creation, which simplifies debugging and enhances predictability in code. It can be enforced using tools like `Object.freeze`, and by using immutable data structures.