🔹 Core JavaScript Concepts

1. **What is the difference between `**, **`, and ``?**
2. `var` is function-scoped and allows hoisting (initialized as `undefined`).
3. `let` is block-scoped and also hoisted but not initialized (temporal dead zone).
4. `const` is block-scoped and must be initialized at the time of declaration. It cannot be reassigned.
5. **What are data types in JavaScript?**
6. **Primitive Types**: String, Number, Boolean, Null, Undefined, BigInt, Symbol.
7. **Non-Primitive Types**: Object (includes arrays, functions, dates, etc.).
8. **Explain the difference between `** and **`.**
9. `==` checks for value equality with type coercion.
10. `===` checks for both value and type equality (strict equality).
11. **What is hoisting in JavaScript?**
12. Hoisting is JavaScript's default behavior of moving declarations (not initializations) to the top of the scope.
13. Applies to variables (`var`) and function declarations.
14. **What is the difference between `** and **`?**
15. `undefined` means a variable has been declared but not assigned a value.
16. `null` is an intentional assignment of no value.
17. **What is scope in JavaScript?**
18. Scope determines the accessibility of variables.
19. Types: Global Scope, Function Scope, Block Scope (with `let` and `const`).
20. **What is closure?**
21. A closure is a function that retains access to its lexical scope even when executed outside its scope.

22. Enables private variables and state preservation.
23. **What is the difference between function declaration and function expression?**
24. Function Declaration: `function foo() {}` - hoisted.
25. Function Expression: `const foo = function() {}` - not hoisted.
26. **Explain the concept of callback functions.**
27. A callback is a function passed as an argument to another function and is invoked after some operation.
28. **What are higher-order functions?**
29. Functions that take other functions as arguments or return them.
30. Example: `map()`, `filter()`, `reduce()`.

🔹 ES6+ Concepts

1. **What are arrow functions and how are they different from regular functions?**
2. Shorter syntax: `const add = (a, b) => a + b`.
3. Do not bind their own `this`, `arguments`, or `super`.
4. **What are template literals in ES6?**
5. String literals allowing embedded expressions using backticks.
6. Example: ``` `Hello, ${name}` ```.
7. **What is destructuring in JavaScript?**
8. Allows unpacking values from arrays or properties from objects.
9. Example: `const {name, age} = person;`
10. **What are default parameters?**
11. Allow function parameters to have default values.
12. Example: `function greet(name = 'Guest') {}`

13. **What is the spread operator and rest operator?**

14. **Spread**: `...` to expand iterable into individual elements.

15. `const arr2 = [...arr1]`
16. **Rest**: `...` to collect all remaining arguments.

17. `function sum(...args) {}`

18. **What are Promises and how do you use `** and **`?**

19. Promises represent eventual completion or failure of async operations.

20. `.then()` handles success, `.catch()` handles errors.

21. **What is async/await? How is it better than Promises?**

22. Syntactic sugar over Promises, making async code look synchronous.

23. Uses `await` for promise resolution, `try...catch` for errors.

24. **What are classes in JavaScript?**

25. ES6 syntax for creating objects and inheritance using `class`, `constructor`, and `extends`.

26. **What is the difference between shallow copy and deep copy?**

27. **Shallow Copy**: Copies object references (e.g., `Object.assign`).

28. **Deep Copy**: Copies nested objects as well (e.g., `structuredClone`, JSON methods).

29. **What is the difference between `**, **`, `**, and **`?**

30. `map()`: Transforms each element and returns new array.

31. `forEach()`: Executes callback for each element (no return).
32. `filter()`: Returns array with elements that pass condition.
33. `reduce()`: Accumulates values into a single output.

🔹 Asynchronous JavaScript

1. **How does the event loop work in JavaScript?**

2. Continuously checks the call stack and task queues to execute functions in a non-blocking way.

3. **What is the call stack, task queue, and microtask queue?**

4. **Call Stack**: Tracks function calls.

5. **Task Queue**: Queues callbacks from `setTimeout`, DOM events.
6. **Microtask Queue**: Queues from promises (`then`, `catch`). Runs before task queue.

7. **What is the difference between synchronous and asynchronous code?**

8. **Synchronous**: Executes line by line, blocking.

9. **Asynchronous**: Executes non-blocking, via event loop.

10. **What is the use of `** and **`?**

11. `setTimeout(fn, delay)`: Executes `fn` once after delay.

12. `setInterval(fn, interval)`: Repeats `fn` at every interval.

13. **What is debouncing and throttling? When do you use them?**

14. **Debouncing**: Executes function after a delay of no calls.

15. **Throttling**: Ensures function runs at most once in a given time.
16. Used in scroll, resize, search input handlers.

🔹 DOM & Browser APIs

1. **How do you manipulate the DOM using JavaScript?**

2. Using methods like `getElementById`, `querySelector`, `innerHTML`, `createElement`, `appendChild`.

3. **What is event delegation?**

4. Attaching a single event listener to a parent element to handle events from children using `event.target`.

5. **What is bubbling and capturing in event propagation?**

6. **Bubbling**: Event travels from target element up to the root.

7. **Capturing**: Event travels from root down to the target.

8. **How do you prevent default browser behavior in events?**
9. Using `event.preventDefault()` in event handler.
10. **What is localStorage, sessionStorage, and cookies?**
11. **localStorage**: Persistent storage, survives reloads.
12. **sessionStorage**: Data cleared after tab closes.
13. **Cookies**: Sent to server with each request, has size limitations.
14. **What is the difference between `** and **`?**
15. `Object.is()` is like `===` but also correctly handles `NaN`, `-0`, and `+0`.
16. **How does `` work?**
17. Returns a Promise for making network requests.
18. Use `fetch(url).then(res => res.json())`.
19. **How do you handle errors in async functions?**
20. Use `try...catch` block with `async/await`.

🔹 Miscellaneous & Advanced

1. **What is the `` keyword and how does it behave in different contexts?**
2. Refers to the object executing the function.
3. In strict mode or arrow functions, behavior differs.
4. **What are modules in JavaScript? (CommonJS vs ES6 Modules)**
5. **CommonJS**: `require()` and `module.exports` (Node.js).
6. **ES6 Modules**: `import` and `export` (browser, modern JS).
7. **What is memory leak in JavaScript?**
8. When memory is no longer needed but not released.

9. Causes: global variables, event listeners, closures.

10. **What is currying in JavaScript?**

11. Transforming a function with multiple arguments into a sequence of functions with one argument.

12. `f(a, b) -> f(a)(b)`

13. **What is prototype and prototypal inheritance?**

14. Objects inherit from other objects via the prototype chain.

15. `obj.__proto__` or `Object.create(proto)`.

16. **What is the difference between `**, **`, and ``?**

17. All change the context (`this`) of a function.

18. `call(thisArg, ...args)`
19. `apply(thisArg, [args])`

20. `bind(thisArg)` returns a new function.

21. **Explain the concept of `` with an example.**

22. The event loop checks the call stack and task queues.

23. Example:

```
console.log("Start");
setTimeout(() => console.log("Timeout"), 0);
Promise.resolve().then(() => console.log("Promise"));
console.log("End");
```

Output: Start -> End -> Promise -> Timeout