10 advanced JavaScript questions

1. Question: What is the Event Loop in JavaScript, and how does it work?

Answer: The Event Loop is a fundamental part of JavaScript's concurrency model. It's responsible for managing the execution of asynchronous code. JavaScript is single-threaded, meaning it can only execute one task at a time, but it can perform asynchronous operations like I/O in a non-blocking way.

Explanation: The Event Loop constantly checks the call stack (the place where synchronous code is executed) and the message queue (where asynchronous tasks are queued). If the call stack is empty, the Event Loop takes the first task from the message queue and pushes it onto the call stack for execution. This process continues, ensuring that asynchronous tasks are processed without blocking the main thread.

2. Question: Explain closures in JavaScript. Provide an example.

Answer: A closure is a function that "closes over" variables from its outer (enclosing) function scope. It retains access to those variables even after the outer function has completed.

Explanation: Here's an example:

```
function outer() {
  const outerVar = 'I am from the outer function';
  return function inner() {
    console.log(outerVar);
  };
}

const innerFunction = outer();
innerFunction(); // Outputs: "I am from the outer function"
```

In this example, the inner function closes over the outerVar variable, allowing it to access outerVar even after the outer function has finished executing.

3. Question: What is the "this" keyword in JavaScript, and how is its value determined?

Answer: The this keyword refers to the context in which a function is executed. Its value is determined dynamically at runtime based on how a function is called.

Explanation: The value of this can be:

In a function called as a method of an object, this refers to the object itself.

In a simple function call, this typically refers to the global object (e.g., window in a browser).
In a constructor function, this refers to the newly created object.
When using .call() or .apply(), you can explicitly set the value of this.

4. Question: What is a promise in JavaScript, and how does it differ from callbacks?

Answer: A promise is an object that represents the eventual completion (or failure) of an asynchronous operation. It provides a cleaner and more structured way to handle asynchronous tasks compared to traditional callback functions.

Explanation: Promises have two states: "fulfilled" (resolved) and "rejected" (failed). You can attach .then() and .catch() handlers to a promise to handle success and error cases. Promises help avoid callback hell and make asynchronous code more readable and maintainable.

5. Question: What is "hoisting" in JavaScript, and how does it work?

Answer: Hoisting is a JavaScript behavior where variable and function declarations are moved to the top of their containing scope during compilation. However, only declarations are hoisted, not initializations.

Explanation: For example:

```
console.log(myVar); // Output: undefined
var myVar = 10;
```

In this code, myVar is hoisted to the top of the scope, but it's not yet assigned a value. That's why console.log(myVar) doesn't throw an error but logs undefined.

6. Question: Explain the concepts of "currying" and "partial application" in JavaScript.

Answer: Currying is the process of converting a function that takes multiple arguments into a series of functions that each take a single argument. Partial application is a similar concept where a function is partially applied to some of its arguments, producing a new function.

Explanation: Here's an example of currying:

```
const add = a => b => a + b;
const add5 = add(5);
const result = add5(10); // Result is 15
```

In this example, add is a curried function that takes two arguments. We partially apply it with 5, resulting in a new function add5 that takes a single argument, which adds 5 to it.

7. Question: What is the purpose of the "use strict" directive in JavaScript?

Answer: "use strict" is a pragma that enables strict mode, which helps catch common coding mistakes and "unsafe" actions in JavaScript. It enforces better coding practices and makes the code less error-prone.

Explanation: In strict mode, certain actions that were previously silent errors become explicit errors, like assigning values to undeclared variables or using reserved keywords. It encourages better coding practices and enhances performance in some cases.

8. Question: Explain the concept of "prototypal inheritance" in JavaScript.

Answer: In JavaScript, objects can inherit properties and methods from other objects through their prototype chain. Prototypal inheritance is a mechanism

where an object can inherit from another object, serving as the basis for creating more complex objects.

Explanation: When a property or method is accessed on an object, JavaScript will first look on the object itself. If it's not found, it will traverse up the prototype chain to find it in the object's prototype, and so on. This allows for a flexible and dynamic way to create and share functionality among objects.

9. Question: What are generators in JavaScript, and how do they work?

Answer: Generators are a type of function that can be paused and resumed. They allow you to control the flow of execution manually, which is useful for asynchronous operations and creating iterators.

Explanation: Here's a simple example:

```
function* countUpTo(n) {
  let count = 1;
  while (count <= n) {
    yield count;
    count++;
  }
}
```

```
const counter = countUpTo(5);
console.log(counter.next().value); // Output: 1
console.log(counter.next().value); // Output: 2
// ...
```

The yield keyword allows the function to pause and resume, making it handy for asynchronous tasks and lazy evaluation.

10. Question: What is the difference between "call" and "apply" in JavaScript?

Answer: Both `call` and `apply` are methods used to invoke functions with a specific context (the value of `this`) and a set of arguments. The key difference is in how arguments are passed to the function.

Explanation:

- `call`: Arguments are passed as a comma-separated list. For example, `func.call(context, arg1, arg2, arg3)`.
- `apply`: Arguments are passed as an array or array-like object. For example, `func.apply(context, [arg1, arg2, arg3])`.

Which one to use depends on the function and how its arguments are structured.

These advanced JavaScript questions cover a range of important topics. Understanding these concepts can greatly improve your ability to work with JavaScript effectively.

11. Question: What is the "call stack" in JavaScript, and how does it work?

Answer: The call stack is a data structure in JavaScript that keeps track of function calls. When a function is invoked, a new frame is pushed onto the stack, and when the function completes, its frame is popped off.

Explanation: This ensures that JavaScript, being single-threaded, maintains a record of where it should return after executing a function. When the call stack is empty, the program has finished executing.

12. Question: Explain the difference between "debouncing" and "throttling" in JavaScript.

Answer: Debouncing and throttling are techniques used to control the rate of execution of certain functions. Debouncing ensures a function is only executed after a pause, while throttling limits the execution rate to a fixed interval.

Explanation: For example, debouncing can be used to delay a search function until the user has stopped typing, while throttling can be used to limit the frequency of API requests to avoid overloading a server.

13. Question: What is a closure leak in JavaScript, and how can you avoid it?

Answer: A closure leak occurs when a function unintentionally retains references to objects that should have been released. This can lead to memory leaks. To avoid it, ensure that you nullify references to objects when they are no longer needed.

Explanation: For example, in event handlers, remove event listeners to prevent closures from holding references to DOM elements after they are removed from the document.

14. Question: What are Web Workers in JavaScript, and how do they work?

Answer: Web Workers allow you to run JavaScript code in a separate thread, enabling concurrent processing without blocking the main thread. They communicate with the main thread through a message-passing system.

Explanation: Web Workers are ideal for CPU-intensive tasks, like data processing or rendering, while keeping the main thread responsive for user interactions.

15. Question: Explain the concept of "Promise.all" and "Promise.race" in JavaScript.

Answer: Promise.all and Promise.race are methods for handling multiple promises. Promise.all waits for all promises to resolve before resolving itself, while Promise.race resolves as soon as the first promise in the array resolves or rejects.

Explanation: Promise.all is useful for executing multiple asynchronous operations in parallel and waiting for all of them to complete. Promise.race can be used when you want the fastest response from multiple sources.

16. Question: What is the difference between "weak map" and "map" in JavaScript, and when would you use one over the other?

Answer: Both Map and WeakMap are key-value data structures. The key difference is that WeakMap keys are weakly held, allowing their associated values to be garbage collected when there are no other references to the key.

Explanation: Use a Map when you need to associate data with specific objects, and you want to ensure those objects won't be collected. Use a WeakMap when you want to avoid memory leaks and allow objects to be collected when they are no longer needed.

17. Question: What is the "prototype chain" in JavaScript, and how does it relate to inheritance?

Answer: The prototype chain is a mechanism in JavaScript that allows objects to inherit properties and methods from their prototype objects. It's fundamental to JavaScript's inheritance model.

Explanation: When you access a property or method on an object, JavaScript will look up the prototype chain to find it. This enables object-oriented programming and code reuse.

18. Question: Explain the concept of "async/await" in JavaScript and how it simplifies asynchronous code.

Answer: async/await is a syntax for handling asynchronous code that makes it appear more like synchronous code. It simplifies error handling and improves code readability.

Explanation: By using the async keyword in front of a function, you can use await within the function to pause execution until a promise is resolved, making the code easier to reason about and maintain.

19. Question: What is the "Temporal Dead Zone" in JavaScript, and how does it relate to the "let" and "const" declarations?

Answer: The Temporal Dead Zone (TDZ) is the period between entering a scope and the actual declaration of a variable. Variables declared with let and const are hoisted but not initialized in the TDZ.

Explanation: Accessing a variable within its TDZ results in a ReferenceError. The TDZ ensures that variables are accessed only after they've been declared.

20. Question: How does JavaScript handle memory management, and what are some best practices to avoid memory leaks?

Answer: JavaScript uses automatic memory management through garbage collection. To avoid memory leaks, developers should:

Nullify references to objects when they are no longer needed.
Remove event listeners when DOM elements are removed.
Avoid circular references that can't be collected.
Explanation: Memory leaks occur when objects are still referenced and not eligible for garbage collection, so being mindful of object lifecycles and reference management is crucial.

These additional advanced JavaScript questions cover a broader range of topics and challenges faced by experienced JavaScript developers. Understanding these concepts is essential for writing robust and efficient JavaScript code.