**Qn-0. Is JavaScript single threaded or multithreaded? What does it mean to be any?**

JavaScript is a single-threaded language, it is synchronous in nature. JavaScript is a single-threaded language, which means it has only one call stack that is used to execute the program. The call stack is the same as the stack data structure that you might read in Data structures. As we know stacks are FILO that is First In Last Out. Similarly, within the call stack, whenever a line of code gets inside the call stack it gets executed and move out of the stack.

**Qn-01. What are promises? Why are they used?**

Promises are used to handle asynchronous operations in JavaScript. They are easy to manage when dealing with multiple asynchronous operations where callbacks can create callback hell leading to unmanageable code.

**Fulfilled:** onFulfilled() will be called (e.g., resolve() was called)

**Rejected:** onRejected() will be called (e.g., reject() was called)

**Pending:** not yet fulfilled or rejected

**>Following are the benefits of Promises:-**

* Improves Code Readability
* Better handling of asynchronous operations
* Better flow of control definition in asynchronous logic
* Better Error Handling

**Qn-02. What do async/await do? Explain it in your own words.**

"async and await make promises easier to write"

**async** makes a function return a Promise

**await** makes a function wait for a Promise

**Async/Await** is an extension of **promises** that we get as language support.

**JavaScript async** keyword turns a method into an async method, which allows you to use the await keyword in its body. When the await keyword is applied, it suspends the calling method and yields control back to its caller until the awaited task is complete. await can only be used inside an async method.

**JavaScript Await** function is used to wait for the promise. It could only be used inside the async block. It instructs the code to wait until the promise returns a response. It only delays the async block. Await is a simple command that instructs JavaScript to wait for an asynchronous action to complete before continuing with the feature. It's similar to a **"pause until done"** keyword. The await keyword is used to retrieve a value from a function where we will usually be used the **then()** function. Instead of calling after the asynchronous function, we'd use await to allocate a variable to the result and then use the result in the code as we will in the synchronous code.

**Qn-03. How do we catch errors in async functions?**

With async/await, a common way to handle errors when awaiting a promise is to wrap it with a try/catch block. This leads to a relatively straightforward failure case: if you do anything else inside your try block, any exceptions thrown will be caught.

**Qn-04. What do async functions return?**

Async functions always return a promise. If the return value of an async function is not explicitly a promise, it will be implicitly wrapped in a promise. Note: Even though the return value of an async function behaves as if it's wrapped in a Promise.resolve , they are not equivalent.

**Qn-05. What do then() consumers in promises return?**

The then method returns a Promise which allows for method chaining. If the function passed as handler to then returns a Promise , an equivalent Promise will be exposed to the subsequent then in the method chain.

**Qn-07. What are the states a promise can be in?**

A Promise is in one of these states:

* pending: initial state, neither fulfilled nor rejected.
* fulfilled: meaning that the operation was completed successfully.
* rejected: meaning that the operation failed.

**Qn-08. What happens if neither resolve nor reject is called within a promise?**

A promise can be only resolve or reject once, another tries will do nothing (no error, no warning, no then invocation).

**Qn-09. What happens if multiple resolve or reject is called within a promise?**

I faced the same thing a while ago, indeed a promise can be only resolved once, another tries will do nothing (no error, no warning, no then invocation). just pass your function as a callback and invoke it as many times you wish! Hope that makes sense.

**Qn-10. What does the finally() method on promise do? Provide your explanation?**

finally() The finally() method returns a Promise . When the promise is settled, either fulfilled or rejected, the specified callback function is executed. This provides a way for code to be run whether the promise was fulfilled successfully or rejected once the Promise has been dealt with.

**Qn-11. What are microtasks in JS?**

A microtask is a short function which is executed after the function or program which created it exits and only if the JavaScript execution stack is empty, but before returning control to the event loop being used by the user agent to drive the script's execution environment.

**Qn-15. What is the difference between the following two lines of code:**

**promise.then(f1).catch(f2);**

**and**

**promise.then(f1, f2);**

**Provide an explanation for your answer.**

The short answer is: no, they are not equal:

The difference is that if an error happens in f1, then it is handled by catch here:promise.then(f1).catch(f2);

promise.then(f1, f2);That’s because an error is passed down the chain, and in the second code piece there’s no chain below f1.

In other words, .then  passes results/errors to the next .then/catch. So in the first example, there’s a catch below, and in the second one there isn’t, so the error is unhandled.