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

JavaScript is a single-threaded language . Single threaded processes contain the execution of instructions in a single sequence. In other words, one command is processes at a time, that means it has only one call stack that is used to execute the program. The opposite of Single threaded processes is multithreaded processes . These processes allow the execution of multiple parts of a program at the same time. These are lightweight processes available within the process.

Multithreaded processes can be implemented as user level threads or kernal level threads.

1. **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.

A promise will be in one of 3 possible states:

1. **Fulfilled:** onFulfilled() will be called (e.g., resolve() was called))
2. **Rejected:** onRejected() will be called (e.g., reject() was called))
3. **Pending:** not yet fulfilled or rejected.

**The benefits of Promises**

* + Promises are used to handle asynchronous operations in JavaScript.
  + They make asynchronous code synchronous
  + They reduce the number of embedded nested callbacks in NodeKS code
  + Easy usage .
  + Promise error handling in Fantastic.

1. **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

An async function is a function that is declared with the async keyword and allows the await keyword inside it. The async and await keywords allow asynchronous, promise-based behavior to be written more easily and avoid configured promise chains. The async keyword may be used with any of the methods for creating a function.

## JavaScript Await

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.

1. **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.

1. **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.

1. **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.
2. **Write the explicit equivalent of Promise.resolve(100);**

Answer written in visual studio code .

1. **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.

1. **What happens if neither resolve nor reject is called within a promise?**

Then failing to resolve or reject a promise just fails to ever change the state from "pending" to anything else.

1. **What happens if multiple resolve or 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).

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

The finally() method returns a Promise. When the promise is settled, i.e 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. This helps to avoid duplicating code in both the promise's then() and catch() handlers.

1. **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

1. **Write a simple print() function that accepts a callback, which gets called after 3 seconds. The callback function could be anything that prints out on the screen.**

Answer written in visual studio code .

1. **Delay with a promise We know that setTimeout() uses a callback. Create a promise based alternative. The function named delay(ms) should return a promise. We should be able to achieve something like this: delay(3000).then(() => alert('Should alert after 3 seconds'))**

Answer written in visual studio code .

1. **Consume the following promise using `await`. let promise = new Promise((res,rej) => { setTimeout(() => res("Resolved!"),3000) } ); promise.then((res) => alert("Response value is: " + res)); // this line needs to get replaced.**

Answer written in visual studio code .

1. **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.**

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);

…But not here:

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.