Asynchronous Programming in JavaScript

- ➢ JavaScript is single-threaded, meaning it can only process one task at a time.
- ➢ Asynchronous programming allows JavaScript to execute tasks in the background without blocking the main thread, allowing other code to run while it's waiting for an I/O operation to complete.
- ➢ JavaScript provides several methods for implementing asynchronous programming, including:
  - i. Callbacks: Functions passed as arguments to other functions, which are executed when the first function is done.
  - ii. Promises: Objects representing the eventual completion or failure of an asynchronous operation.
  - iii. Async/await: Syntax built on top of Promises, making it easier to write asynchronous code that is similar in structure to synchronous code.
- ➢ Asynchronous code is typically used when working with APIs, performing network requests, reading/writing files, or other I/O operations that could take a long time to complete.
- ➢ In order to avoid callback hell (a situation where callback functions are nested multiple levels deep, making the code difficult to read and maintain), it's important to structure asynchronous code in a way that makes it easy to understand and debug. This can be achieved by breaking down complex tasks into smaller, composable functions, and using tools like Promises and async/await to manage the flow of execution.

Difference Between Callbacks, Promises and Async/Await

- ✗ Callbacks: Callbacks are functions that are passed as arguments to other functions and are executed when the first function is done. They are the oldest and most basic way to handle asynchronous operations in JavaScript. Callbacks are often used for simple, one-off tasks, but can quickly become unreadable and difficult to maintain when dealing with more complex flow control.

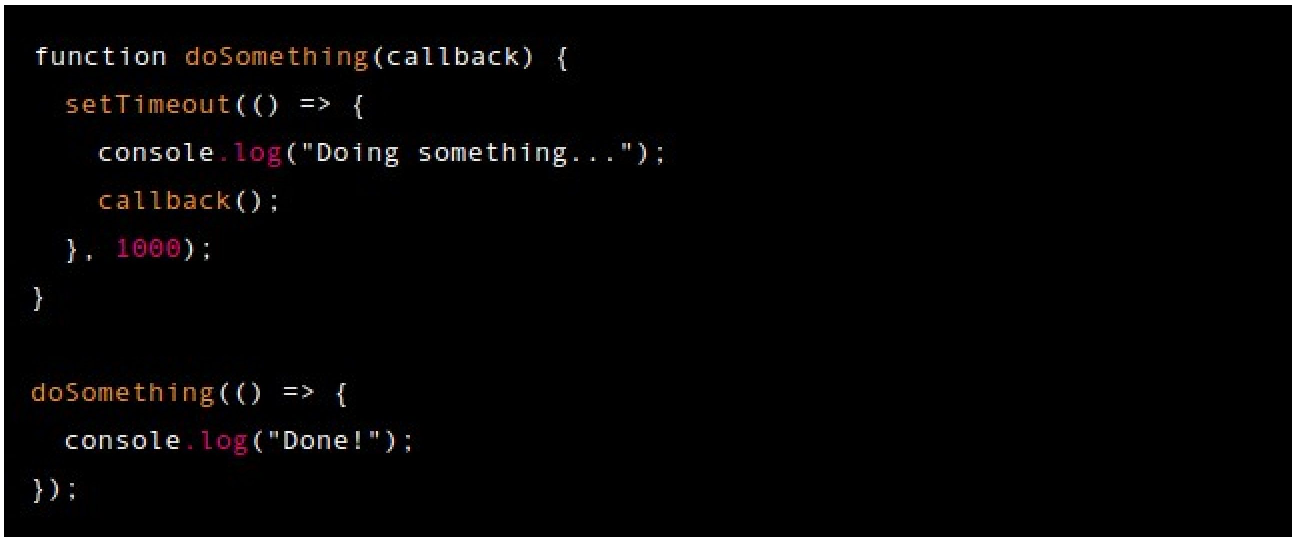
- ✗ Promises: Promises are objects that represent the eventual completion or failure of an asynchronous operation. They provide a way to handle asynchronous operations in a more structured and composable manner, compared to callbacks. Promises can be chained together to handle complex flow control, and can also be combined using Promise.all() or Promise.race().
- ✗ Async/await: Async/await is a syntax built on top of Promises that makes it easier to write asynchronous code that is similar in structure to synchronous code. With async/await, you can use the `await` keyword to pause execution until a Promise is resolved, making it possible to write asynchronous code that looks and behaves like synchronous code. This makes it easier to read and understand asynchronous code, compared to callbacks or Promises.

In summary, callbacks are the most basic method for handling asynchronous operations in JavaScript, but can quickly become difficult to maintain as the complexity of the code increases. Promises provide a more structured and composable way to handle asynchronous operations, and async/await makes it possible to write asynchronous code that looks and behaves like synchronous code.

Examples:

Here's a simple example demonstrating each of the three methods of asynchronous programming in JavaScript:

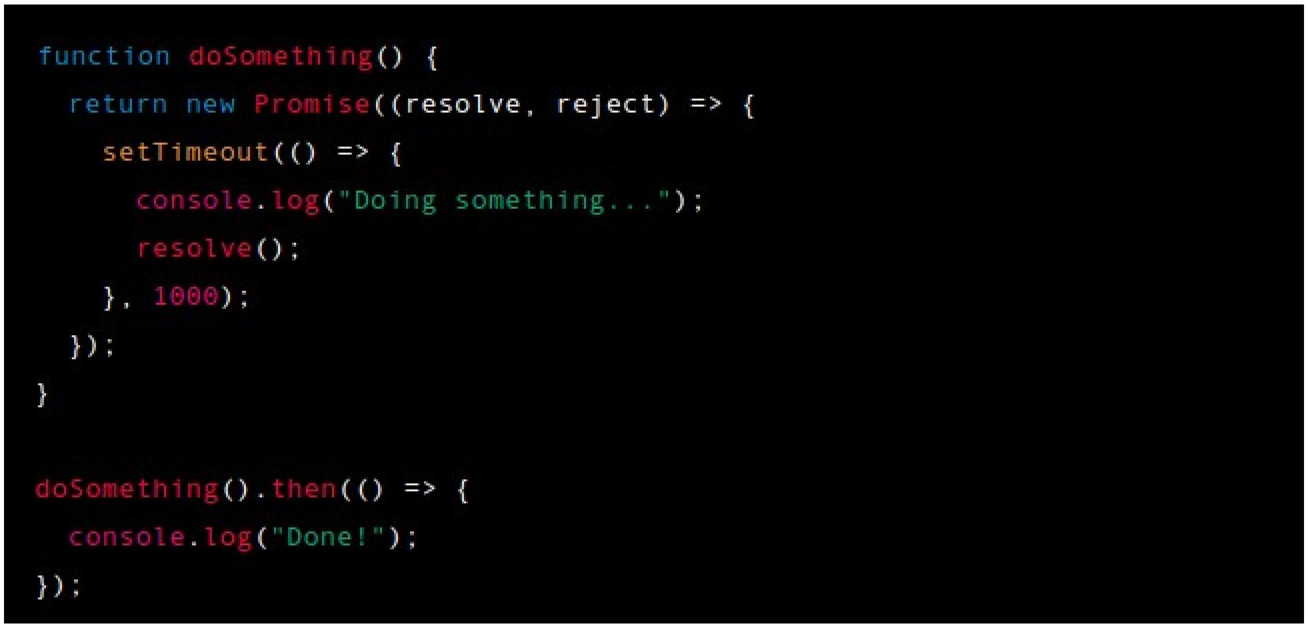
➔ **Callbacks**



```javascript
function doSomething(callback) {
  setTimeout(() => {
    console.log("Doing something...");
    callback();
  }, 1000);
}

doSomething(() => {
  console.log("Done!");
});
```

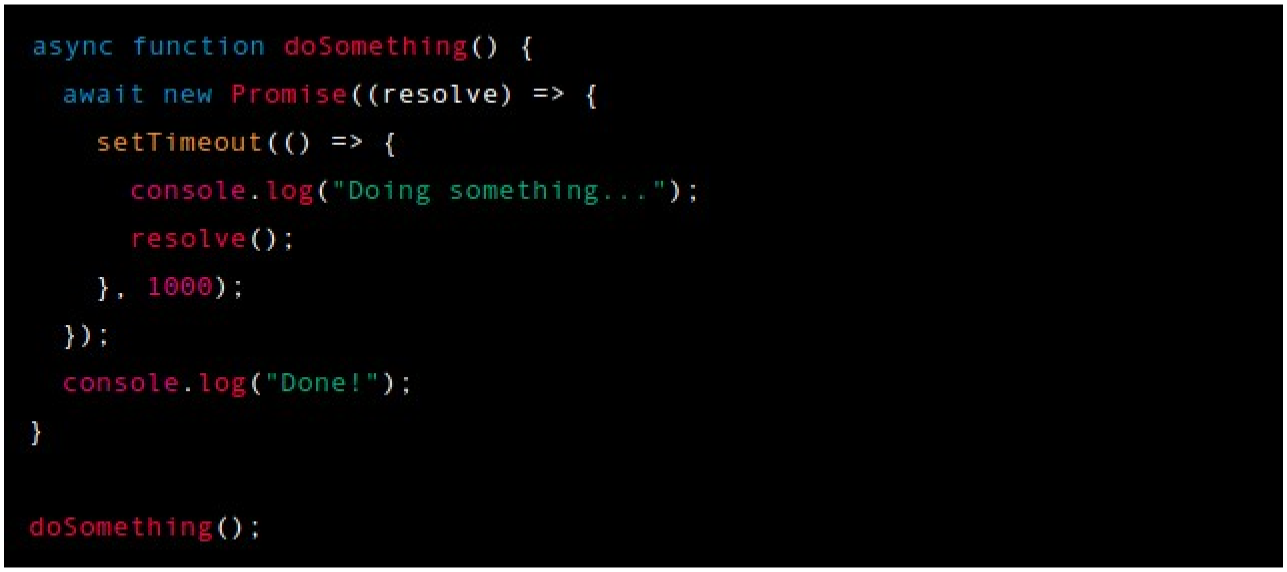
➔ Promises



```javascript
function doSomething() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      console.log("Doing something...");
      resolve();
    }, 1000);
  });
}

doSomething().then(() => {
  console.log("Done!");
});
```

➔ Async/Await

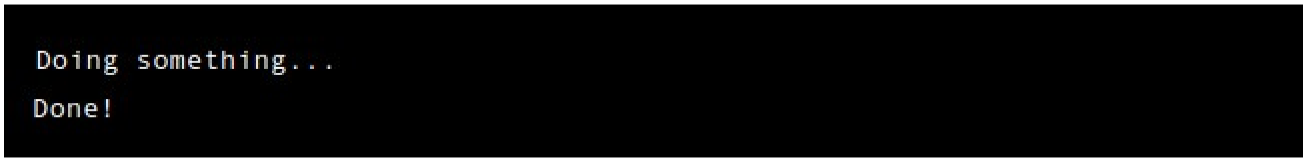


```
async function doSomething() {
  await new Promise((resolve) => {
    setTimeout(() => {
      console.log("Doing something...");
      resolve();
    }, 1000);
  });
  console.log("Done!");
}

doSomething();
```

In each of these examples, doSomething performs an asynchronous operation (a 1 second delay) and then calls a callback, resolves a Promise, or logs a message, depending on the method being used. The output of each of these examples is the same:

Sample Output:



```
Doing something...
Done!
```

Links:

https://www.w3schools.com/js/js_callback.asp

https://www.w3schools.com/js/js_asynchronous.asp

https://www.w3schools.com/js/js_promise.asp

https://www.w3schools.com/js/js_async.asp