ASYNCHRONOUS PROGRAMMING IN JAVASCRIPT

Asynchronous programming in JavaScript allows you to perform tasks without blocking the main execution thread, making your application more efficient and responsive.

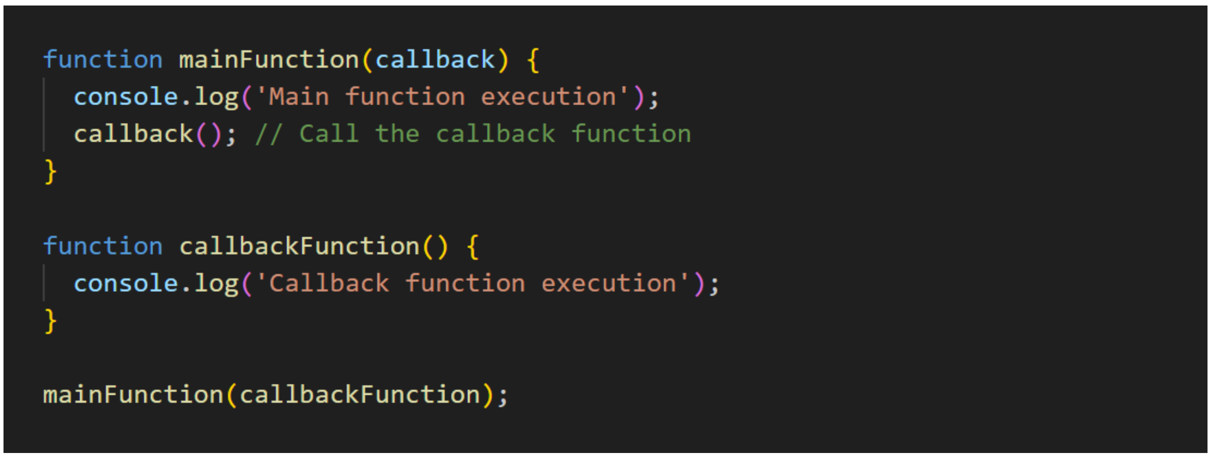
Since JS is single threaded, Asynchronous programming helps in handling operations like network requests, file reading, and timers without freezing the UI

Key Concepts

1. Callbacks: Functions passed as arguments to other functions, executed after the completion of an asynchronous operation

Usage: Commonly used in scenarios where an operation takes an unknown amount of time to complete.

Syntax:



```
function mainFunction(callback) {
  console.log('Main function execution');
  callback(); // Call the callback function
}

function callbackFunction() {
  console.log('Callback function execution');
}

mainFunction(callbackFunction);
```

Explanation:

- mainFunction takes a callback function as its parameter
- Inside mainFunction, the callback is called, executing the code defined in callbackFunction

Example:

```
function greet(name) {
  console.log('Hello, ' + name);
}

function processUserInput(callback) {
  let name = 'Alice';
  callback(name);
}

processUserInput(greet);
```

Explanation:

- greet is a callback function that takes a name and logs a greeting
- processUserInput calls the callback function with a name

2. Promises: Promise in JavaScript is an object that represents the completion or failure of an asynchronous operation and its resulting value.

It allows you to write cleaner asynchronous code without deeply nested callbacks.

Syntax:

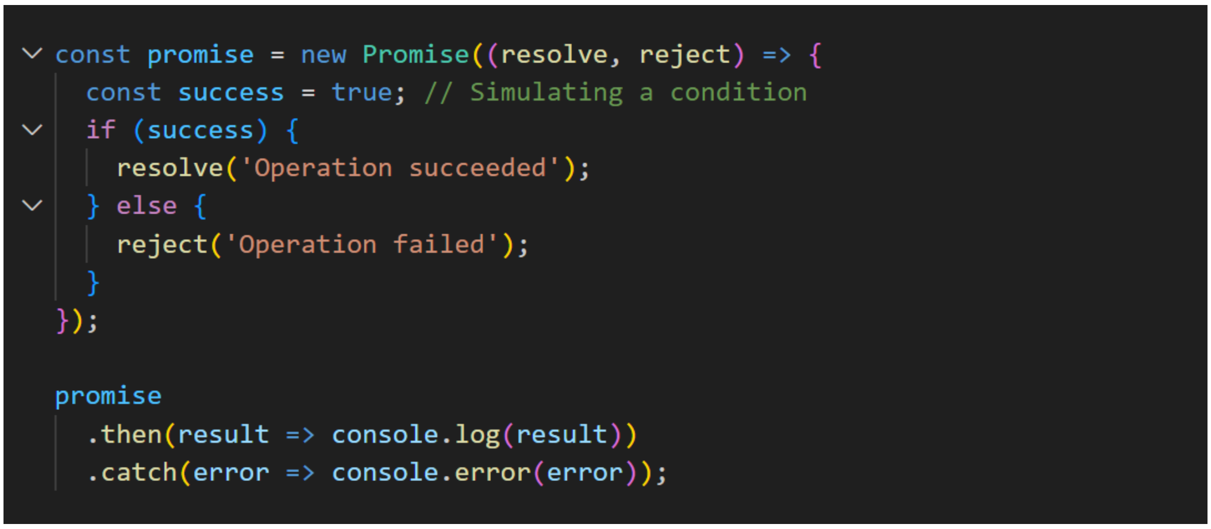
```
const promise = new Promise((resolve, reject) => {
  resolve('Success');
});

promise.then(result => console.log(result)).catch(error => console.error(error));
```

Explanation:

- A Promise is created with resolve for success and reject for failure
- The then method handles the resolved value, while the catch method handles any errors

Example:



```javascript
const promise = new Promise((resolve, reject) => {
  const success = true; // Simulating a condition
  if (success) {
    resolve('Operation succeeded');
  } else {
    reject('Operation failed');
  }
});

promise
  .then(result => console.log(result))
  .catch(error => console.error(error));
```

Explanation:

- The Promise is created with resolve for success and reject for failure
- The then method handles the resolved value ('Operation succeeded'), while the catch method handles any errors ('Operation failed')