Async and **Await**

ROHAN CHHETRY

Callbacks

Callbacks are functions passed as arguments to other functions and are executed after the completion of an asynchronous operation.

Problem 1

Create a function fetchData that simulates fetching data from an API using a callback.

Solution 1

```
function fetchData(callback) {
  setTimeout(() => {
    const data = { id: 1, name: 'Example Data' };
    callback(data);
  }, 1000);
}

function processResult(data) {
  console.log(`Processed data: ${JSON.stringify(data)}`);
}

fetchData(processResult);
// Output (after 1 second): Processed data: {"id":1,"name":"Example Data"}
```

Asynchtonous

Asynchronous operations allow the program to execute tasks without waiting for the completion of each operation before moving to the next one.

Problem 2

Create an asynchronous function simulateAsyncTask that logs a message after a delay.

Solution 2

```
function simulateAsyncTask() {
  setTimeout(() => {
    console.log('Async task completed.');
  }, 2000);
}

console.log('Start of program');
simulateAsyncTask();
console.log('End of program');
// Output:
// Start of program
// End of program
// Async task completed. (after 2 seconds)
```

Promises

Promises are objects representing the eventual completion or failure of an asynchronous operation, providing a cleaner alternative to callbacks.

Problem 3

Create a function fetchDataPromise that returns a promise to simulate fetching data.

03 Solution 3

```
function fetchDataPromise() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = { id: 1, name: 'Example Data' };
      resolve(data);
    }, 1000);
  });
}

fetchDataPromise()
  .then(data => console.log(`Fetched data: ${JSON.stringify(data)}`))
  .catch(error => console.error(`Error: ${error}`));
// Output (after 1 second): Fetched data: {"id":1,"name":"Example Data"}
```

Async/Await

Async/Await is a syntactic sugar built on top of promises, providing a more readable and concise way to work with asynchronous code.

Problem 4

Create an asynchronous function fetchAndProcessData using Async/Await.

Solution 4

```
function fetchDataPromise() {
  return new Promise((resolve, reject) => {
    setTimeout(() => {
      const data = { id: 1, name: 'Example Data' };
      resolve(data);
    }, 1000);
  });
}

async function fetchAndProcessData() {
  try {
    const data = await fetchDataPromise();
    console.log(`Fetched and processed data: ${JSON.stringify(data)}`);
  } catch (error) {
    console.error(`Error: ${error}`);
  }
}

fetchAndProcessData();
// Output (after 1 second): Fetched and processed data: {"id":1,"name":"Example Data"}
```