Callbacks in JavaScript

A **callback** is a function passed as an argument to another function, which is then executed after some operation has been completed. It helps in handling asynchronous operations, like reading a file or making a web request.

Example of a Callback:

```
function fetchData(callback) {
  console.log("Fetching data...");
  setTimeout(() => {
    const data = { name: "John", age: 25 };
    callback(data); // Call the callback function after data is ready
  }, 2000); // Simulating a 2-second delay
}

function processData(data) {
  console.log("Data received:", data);
}

fetchData(processData); // Passing processData as a callback
```

Here:

- `fetchData` starts an asynchronous operation (like fetching data).
- When the operation finishes after 2 seconds, it calls `processData` with the fetched data.

Downside of Callbacks:
When callbacks are nested inside each other, especially for complex operations, they lead to **callback hell**, making code difficult to understand and maintain.

Promises in JavaScript

Promises are a cleaner way to handle asynchronous operations. They represent a value that will be available in the future (either successfully or with an error).

A Promise can have three states:

1. **Pending** – Initial state, the operation is not complete.
2. **Fulfilled** – The operation completed successfully, and a value is returned.
3. **Rejected** – The operation failed, and an error is returned.

Example of a Promise:

```
function fetchData() {
  return new Promise((resolve, reject) => {
    console.log("Fetching data...");
    setTimeout(() => {
      const success = true; // Simulate success or failure
      if (success) {
        resolve({ name: "John", age: 25 }); // If successful, resolve the promise
      } else {
        reject("Error: Data not found"); // If failed, reject the promise
      }
    }, 2000); // Simulating a 2-second delay
  });
}

// Handling the Promise
fetchData()
  .then((data) => {
    console.log("Data received:", data); // If promise is resolved
  })
  .catch((error) => {
    console.error(error); // If promise is rejected
  });
```

Here:

- `fetchData()` returns a Promise.
- `resolve` is called if the operation is successful.
- `reject` is called if there's an error.
- `.then()` is used to handle the success, while `.catch()` is used to handle errors.

Difference between Callbacks and Promises:

1. **Readability:** Promises make the code more readable and easier to handle than nested callbacks.
2. **Chaining:** Promises allow chaining (`.then()`) for multiple asynchronous tasks, making it easier to manage sequential operations.
3. **Error Handling:** Promises have better error handling with `.catch()`, making it easier to catch all errors in one place.

Callback Hell Example:

```
fetchData1(function(result1) {
  fetchData2(result1, function(result2) {
    fetchData3(result2, function(result3) {
      console.log("Final result:", result3);
    });
  });
});
```

Promise Chaining Example (same task):

```
fetchData1()
  .then(result1 => fetchData2(result1))
  .then(result2 => fetchData3(result2))
  .then(result3 => console.log("Final result:", result3))
  .catch(error => console.error(error));
```

Promises provide a clearer, more manageable way to handle asynchronous code compared to callbacks.

JavaScript Promise

A **JavaScript Promise** is a special object that helps you handle **asynchronous tasks** (like fetching data from a server) by making a "promise" that it will either:

1. **Resolve** (succeed) with a result, or
2. **Reject** (fail) with an error.

Why Use Promises?

When you perform tasks that take time (like loading data), Promises allow your code to **keep running** without waiting for the task to finish. Instead, the Promise "promises" to tell you the result **later**.

Simple Example:

```
let promise = new Promise((resolve, reject) => {
  let success = true; // Simulate a task
  if (success) {
    resolve("Task completed!"); // If successful, resolve the promise
  } else {
    reject("Task failed!"); // If failed, reject the promise
  }
});

promise
  .then((message) => {
    console.log(message); // Runs if the promise is resolved (success)
  })
  .catch((error) => {
    console.error(error); // Runs if the promise is rejected (failure)
  });
```

How It Works:

- **`new Promise`** : Creates a promise.
- **`resolve`** : This is called if the task is successful, sending back the result.
- **`reject`** : This is called if the task fails, sending back an error.
- **`.then()`** : Defines what to do when the promise is resolved (successful).
- **`.catch()`** : Defines what to do when the promise is rejected (failed).

In Simple Terms:

- A Promise is like ordering something online.
- You order it (start a task) and the website promises to deliver it **later**.
- If the delivery is successful, you receive your item (**resolve**).
- If there's an issue, you get an error (**reject**).
- **`.then()`** is what you do when you get the item, and **`.catch()`** is what happens if something goes wrong.