Promises (creation, chaining, error handling)

Here's a **detailed explanation of Promises** in JavaScript, covering creation, chaining, and error handling:

🔐 What is a Promise?

A **Promise** in JavaScript is an object that represents the **eventual completion (or failure)** of an asynchronous operation and its resulting value.

🧠 States of a Promise:

1. `pending` – Initial state
2. `fulfilled` – Operation completed successfully
3. `rejected` – Operation failed

🔨 Promise Creation

You create a Promise using the `Promise` constructor:

```
const myPromise = new Promise((resolve, reject) ⇒ {
  let success = true;

  if (success) {
    resolve("Promise resolved!");
  } else {
    reject("Promise rejected!");
  }
});
```

✅ Handling a Promise

You handle results using `.then()` and errors using `.catch()`:

```
myPromise
  .then((result) => {
    console.log(result); // "Promise resolved!"
  })
  .catch((error) => {
    console.error(error); // "Promise rejected!"
  });
```

🔗 Promise Chaining

You can **chain** `.then()` calls to perform multiple sequential asynchronous operations:

```
function first() {
  return new Promise((resolve) => {
    setTimeout(() => resolve("First done"), 1000);
  });
}

function second(message) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(`${message} → Second done`), 1000);
  });
}

function third(message) {
  return new Promise((resolve) => {
    setTimeout(() => resolve(`${message} → Third done`), 1000);
  });
}

first()
  .then(second)
  .then(third)
  .then(console.log); // "First done → Second done → Third done"
```

❌ Promise Error Handling

You can handle errors anywhere in the chain with `.catch()`:

```
fetch("https://invalid-api.com")
  .then((res) ⇒ res.json())
  .then((data) ⇒ console.log(data))
  .catch((error) ⇒ console.error("Something went wrong:", error));
```

You can also use `.finally()`:

Runs **regardless** of success or failure:

```
someAsyncTask()
  .then((res) ⇒ console.log(res))
  .catch((err) ⇒ console.error(err))
  .finally(() ⇒ console.log("Done!"));
```

🔁 Promise Example: Simulated API

```
function fakeAPI(success) {
  return new Promise((resolve, reject) ⇒ {
    setTimeout(() ⇒ {
      success ? resolve("Data loaded") : reject("API failed");
    }, 1000);
  });
}

fakeAPI(true)
```

```
    .then((data) => console.log(data))
    .catch((err) => console.error(err));
```

🧠 Key Concepts

Concept	Description
`Promise`	Handles async operations
`.then()`	Handles resolved value
`.catch()`	Handles rejected value
`.finally()`	Runs after `.then()` or `.catch()`
Chaining	Passing results to next `.then()`

🆚 Callbacks vs Promises

Feature	Callback	Promise
Readability	Hard (callback hell)	Easy with chaining
Error handling	Manual (nested)	Centralized `.catch()`
Composability	Difficult	Simple and clean

Let me know if you'd like to convert callback code to promises or explore `async/await` next.