Asynchronous JavaScript

Promises

What is a Promise?

| Simple Definition

A Promise is an object that represents a value that will be:

- available now
- available later
- or never

"I promise I'll give you the result... eventually."

| Why Promises Exist

- Handle async code cleanly
- Avoid nested callbacks
- Better error handling

Promise States

A promise is always in one of these states:

🟡 Pending

- Initial state
- Promise is still working

🟢 Fulfilled

- Operation completed successfully
- Result is available

🔴 Rejected

- Operation failed
- Error occurred

A promise can change state only once.

Creating Promises

Syntax

```
const promise = new Promise((resolve, reject) => {
  // async operation
});
```

Example Code

```
const fetchData = new Promise((resolve, reject) => {
  let success = true;

  if (success) {
    resolve("Data received");
  } else {
    reject("Something went wrong");
  }
});
```

- resolve() → fulfilled
- reject() → rejected

Consuming Promises (then, catch, finally)

.then()

Runs when promise is fulfilled

.catch()

Runs when promise is rejected

.finally()

Runs no matter what

Example Code

```
fetchData
  .then(result => {
    console.log(result);
  })
  .catch(error => {
    console.log(error);
  })
  .finally(() => {
    console.log("Done");
  });
```

Promise Chaining

What Is Chaining?

Using multiple .then() calls in sequence.

Example Code

```
fetchData
  .then(result => result + " 🔥")
  .then(updated => console.log(updated))
  .catch(err => console.log(err));
```

Each .then() receives the previous result.

Returning Promises from .then()

Important Rule

If you return a promise from .then(), the next .then() waits for it.

Example Code

```
fetchUser()
  .then(user => {
    return fetchOrders(user.id);
  })
  .then(orders => {
    console.log(orders);
  });
```

This keeps async code flat & readable.

Error Propagation

How Errors Work

- Errors automatically travel down the chain
- One .catch() can handle all errors

Example Code

```
fetchData
  .then(data => {
    throw new Error("Oops");
  })
  .then(result => console.log(result))
  .catch(error => console.log(error.message));
```

Once an error happens → next .then() is skipped.

Promise Immutability

What It Means

Once a promise is:

- fulfilled or
- rejected

its state can never change again

Example Code

```
const p = new Promise((resolve, reject) => {
  resolve("Success");
  reject("Fail"); // ignored
});
```

Only the first call matters.