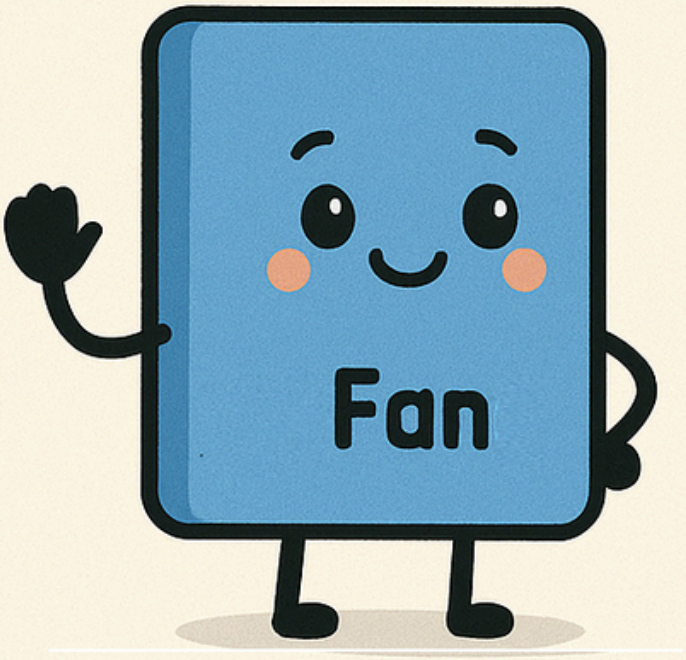
Mastering Functions, Callbacks & Promises in JavaScript

FUNCTION

A function is simply a set of statements that perform a specific task when called.




```
function
greet(name) {
  return `Hello,
${name}!`;
}
console.log(greet
("Rishabh"));
```

FIRST-CLASS FUNCTION

In JavaScript, functions are first-class citizens — meaning we can use them as values, pass them as arguments, or even return them.

🔷 Higher-Order Function

A higher-order function takes another function as an argument or returns one.

Examples: map(), filter(), reduce(), forEach(), find(), every(),

```
const numbers = [1, 2, 3];
const doubled = numbers.map
  num => num * 2);
console.log(doubled); // [2, 4, 6]
```

Callback Function

A callback function is passed into another function and executed later.

```
function fetchData(callback) {
 console.log("Fetching data...",
 callback("Data received")
}
fetchData(message => console.log(message)
// Output: Fetching data… Data received
```

Callbacks can be:

1. **Synchronous** — executed immediately after the outer function.

   👉 Example: map(), forEach()

2. **Asynchronous** — executed later, after an async task completes.

   👉 Example: fetch(), setTimeout(), Promise.then)

⚠️ Problems with Callbacks

Callback Hell – nested callbacks create unreadable "pyramid of doom" code.

Inversion of Control – we hand over too much control to external functions.

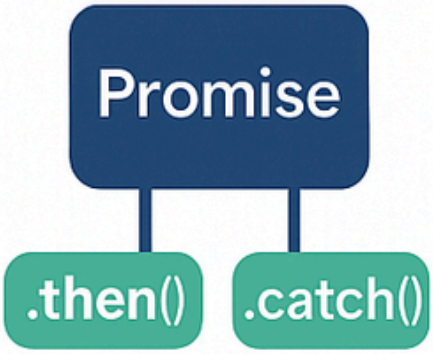
Why do we use Promises?

Avoid Callback Hell:

Promises help eliminate nested callbacks (also known as the "pyramid of doom") by allowing clean promise chaining.

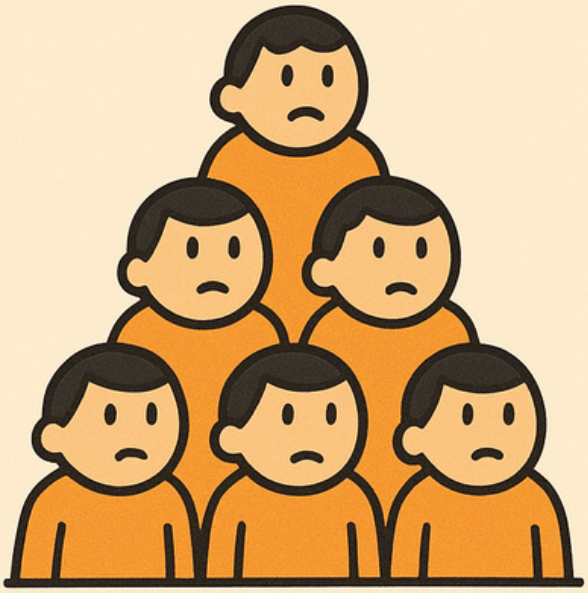
Inversion of Control (IoC):

Instead of passing callbacks into functions, Promises return an object. We can then attach `.then()` or `.catch()` handlers, giving developer greater control over program flow.



AVOID CALLBACK HELL

no nested callbacks



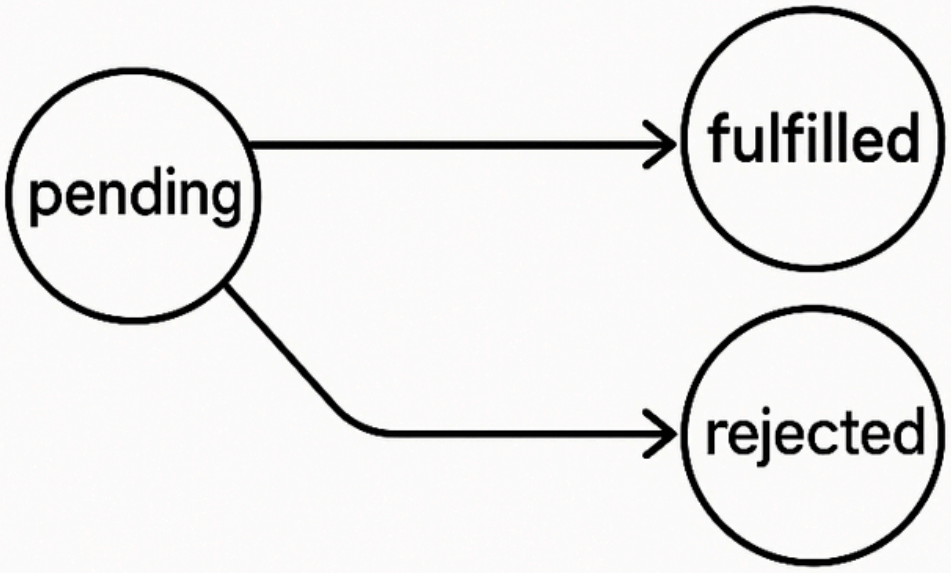
INVERSION OF CONTROL (IoC)

return object

What are Promises?

A Promise is a JavaScript object used to handle asynchronous operations.

It acts as a placeholder for a value that will be available in the future, once the async task completes.



A Promise can be in one of three states: pending, fulfilled, or rejected.

In simple terms, a Promise represents the eventual success (resolved) or failure (rejected) of an asynchronous operation.

PROMISES