JavaScript

An *asynchronous operation* is one that allows the computer to “move on” to other tasks while waiting for the asynchronous operation to complete.  Asynchronous programming means that time-consuming operations don’t have to bring everything else in our programs to a halt.

Promises are objects that represent the eventual outcome of an asynchronous operation.

 A Promise object can be in one of three states:

* **Pending**: The initial state— the operation has not completed yet.
* **Fulfilled**: The operation has completed successfully and the promise now has a *resolved value*. For example, a request’s promise might resolve with a JSON object as its value.
* **Rejected**: The operation has failed and the promise has a reason for the failure. This reason is usually an Error of some kind.

We refer to a promise as *settled* if it is no longer pending.

setTimeout() is a Node API (a comparable API is provided by web browsers) that uses callback functions to schedule tasks to be performed after a delay. setTimeout() has two parameters: a callback function and a delay in milliseconds.

Promise objects come with an aptly named .then() method. It allows us to say, “I have a promise, when it settles, **then** here’s what I want to happen.”

.then() is a higher-order function— it takes two callback functions as arguments. We refer to these callbacks as *handlers*. When the promise settles, the appropriate handler will be invoked with that settled value.

* The first handler, sometimes called onFulfilled, is a *success handler*, and it should contain the logic for the promise resolving.
* The second handler, sometimes called onRejected, is a *failure handler*, and it should contain the logic for the promise rejecting.

.then() is that it always returns a promise.

One way to write cleaner code is to follow a principle called *separation of concerns*. Separation of concerns means organizing code into distinct sections each handling a specific task. It enables us to quickly navigate our code and know where to look if something isn’t working.

This process of chaining promises together is called *composition.*

Promise.all() accepts an array of promises as its argument and returns a single promise. That single promise will settle in one of two ways:

* If every promise in the argument array resolves, the single promise returned from Promise.all() will resolve with an array containing the resolve value from each promise in the argument array.
* If any promise from the argument array rejects, the single promise returned from Promise.all() will immediately reject with the reason that promise rejected. This behavior is sometimes referred to as *failing fast*.

Review

* Promises are JavaScript objects that represent the eventual result of an asynchronous operation.
* Promises can be in one of three states: pending, resolved, or rejected.
* A promise is settled if it is either resolved or rejected.
* We construct a promise by using the new keyword and passing an executor function to the Promise constructor method.
* setTimeout() is a Node function which delays the execution of a callback function using the event-loop.
* We use .then() with a success handler callback containing the logic for what should happen if a promise resolves.
* We use .catch() with a failure handler callback containing the logic for what should happen if a promise rejects.
* Promise composition enables us to write complex, asynchronous code that’s still readable. We do this by chaining multiple .then()‘s and .catch()‘s.
* To use promise composition correctly, we have to remember to return promises constructed within a .then().
* We should chain multiple promises rather than nesting them.
* To take advantage of concurrency, we can use Promise.all().