Promises

[Doaa Ashraf Taha]

- A Promise is a JavaScript object that links producing code and consuming code.
- "Producing code" is code that can take some time.
- "Consuming code" is code that must wait for the result.

```
let myPromise = new Promise(function(resolve, reject) {

        // "Producing Code" (May take some time)

       resolve(); // when successful

        reject();  // when error

});


// "Consuming Code" (Must wait for a fulfilled Promise)

myPromise.then(

  function(value) { /* code if successful */ },

  function(error) { /* code if some error */ }

);
```

- · A promise can only succeed or fail once. It cannot succeed or fail twice, neither can it switch from success to failure or vice versa.
- · The Promise object supports two properties:

state and **result**.

- **state** — initially "pending", then changes to either "fulfilled" or "rejected.
- **result** — initially undefined, then changes to value when resolve(value)is called or error when reject(error) is called.
- The properties state and result of the Promise object are internal. We can't directly access them. We can use the methods .then/.catch/.finally for that.

Using .then()

- .then() is used to chain promises together .
- The first argument of .then() is a function that runs when the promise is resolved, and receives the result.
- The second argument of .then() is a function that runs when the promise is rejected . It also return a promise that can be chained again with .then().

Using .catch()

- .catch() is used to handle error properly and is called only in case the promise has failed .

  So we can use .then for success only if we want using the first callback and let .catch deal with the error .

```
myPromise
  .then(value => { do something with the value})
  .then(value => { do another thing with the processed value  })
  .then(value => { console.log(value) })
  .catch(err => { console.log(err) });
```

· Using .finally()

- The finally() method can be useful if you want to do some processing or cleanup once the promise is settled, regardless of its outcome..
- · A finally handler has no arguments. In finally we don't know whether the promise is successful or not.

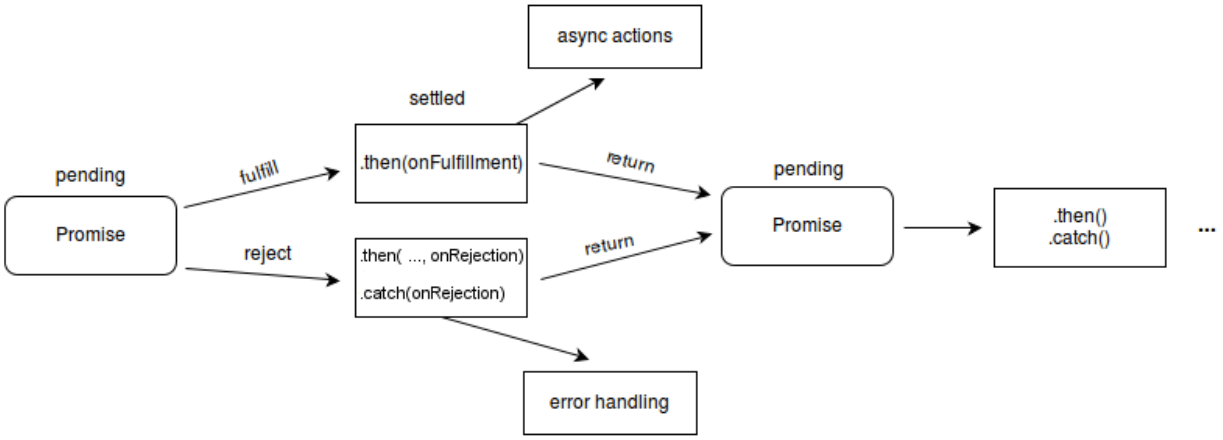
```
new Promise((resolve, reject) => {

 setTimeout(() => resolve("value"), 2000);

}).finally(() => alert("Promise ready")) // triggers first
   .then(result => alert(result));
```

[Mostafa Essa Ahmed]

Introduction:
A **Promise** is a proxy for a value not necessarily known when the promise is created. It allows you to associate handlers with an asynchronous action's eventual success value or failure reason. This lets asynchronous methods return values like synchronous methods: instead of immediately returning the final value, the asynchronous method returns a *promise* to supply the value at some point in the future.

A Promise is in one of these states:

- *pending*: initial state, neither fulfilled nor rejected.
- *fulfilled*: meaning that the operation was completed successfully.
- *rejected*: meaning that the operation failed.



Consumers: then, catch

A Promise object serves as a link between the executor (the “producing code” or “singer”) and the consuming functions (the “fans”), which will receive the result or error. Consuming functions can be registered (subscribed) using methods .then and .catch.

References
[**1**] https://javascript.info/promise-basics

[**2**]https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Promise#description

---

[Mohamed Raafat]

So I think we should begin with the first question that comes to mind when discussing a new topic like promises, what is promises and why does it exist in the first place?
promises in javascript is like promises in real life, when you make a promise in real life there are two outcomes either you fulfill that promise which is called resolve in javascript or you don't fulfill that promise which is called reject in javascript so for example if i promise you i will give you 1000$ and i actually gave it to you then i resolved my promise and if i didn't give that 1000$ (which i am sure i won't) then i rejected my promise.

So what does that promise look like?

```
//make a new promise
let p = new Promise((resolve, reject) => {
        let iGiveYou1000 = false
        // check if i fulfill my promise or not
        if(iGiveYou1000 == true)
        {
                resolve('fulfilled')
        }
        else
        {
                reject('not fulfilled')
        }
})
```

So how do we call or interact with the promise we just made?

we just use .then() and pass a function that does something when the promise is fulfilled or so called resolved and .catch() and pass a function that does something when the promise is not fulfilled or so called rejected

Why does such a thing exist?

well we can say that promises is neat or elegant way to replace callbacks and also the code with promises is a lot cleaner because when you start nesting callbacks things get tedious pretty quickly where your code keeps getting indented and pushed out of view where in promises all you have to do is just add another .then() instead of having a callback inside a callback inside a callback which is really bad