PROMISE, ASYNC/AWAIT

I4GIC
By Thavorac

INTRODUCTION TO CALLBACK

Promise, Async/Await are all for the purpose of managing "**CallBacks**".

A callback is a function which is to be executed after another function has finished execution. A more formal definition would be - Any function that is passed as an argument to another function so that it can be executed in that other function is called as a callback function

CallBack can be triggered by an event and the event might be user-initiated such as mouse click or delay time or automated event.

PROMISE

Promises are used to handle asynchronous operations in JavaScript.

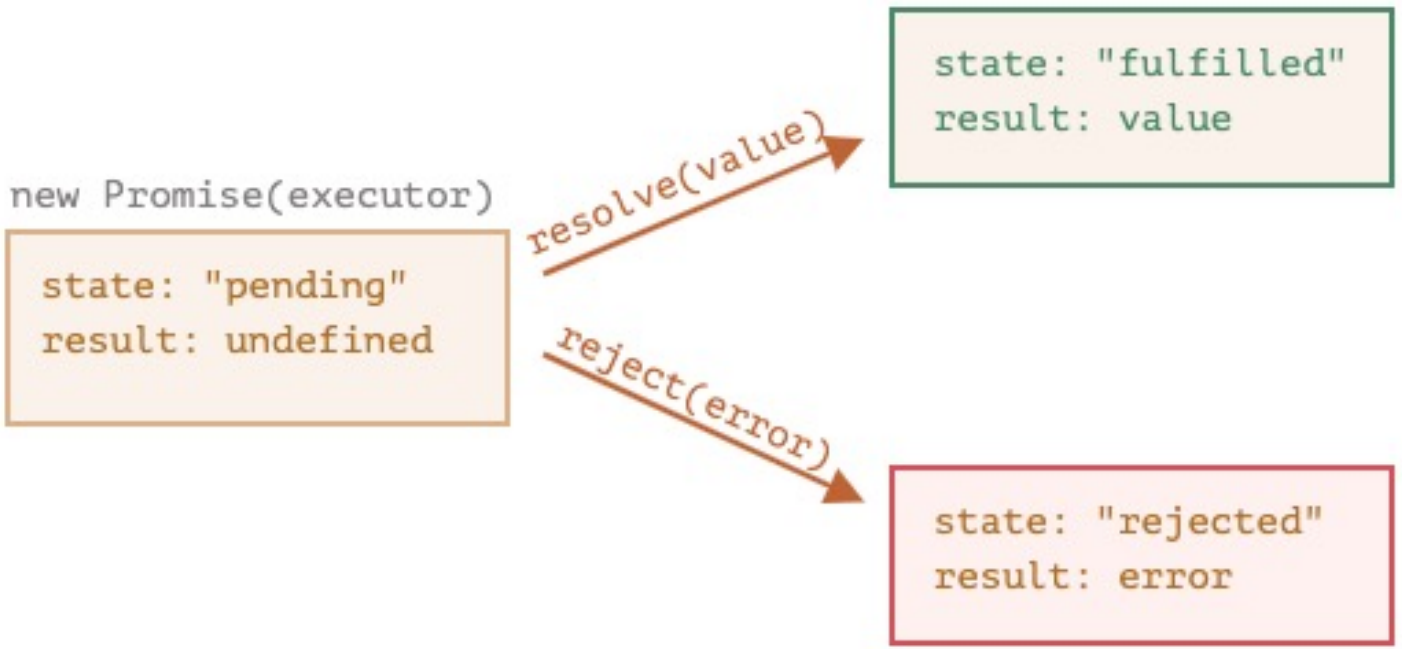
Benefits of Promises

- Improves Code Readability
- Better handling of asynchronous operations
- Better flow of control definition in asynchronous logic
- Better Error Handling

A Promise has four states:

- **fulfilled**: Action related to the promise succeeded
- **rejected**: Action related to the promise failed
- **pending**: Promise is still pending i.e. not fulfilled or rejected yet
- **settled**: Promise has fulfilled or rejected

```
let promise = new Promise(function(resolve, reject) {
  // executor (the producing code, "singer")
});
```



Example:

```
let promise = new Promise(function(resolve, reject) {
  // the function is executed automatically when the promise is constructed

  // after 1 second signal that the job is done with the result "done"
  setTimeout(() => resolve("done"), 1000);
});
```


```
let promise = new Promise(function(resolve, reject) {
  // after 1 second signal that the job is finished with an error
  setTimeout(() => reject(new Error("Whoops!")), 1000);
});
```

```
var promise = new Promise(function(resolve, reject) {
  const x = "geeksforgeeks";
  const y = "geeksforgeeks"
  if(x === y) {
    resolve();
  } else {
    reject();
  }
});

promise.
    then(function () {
        console.log('Success, You are a GEEK');
    }).
    catch(function () {
        console.log('Some error has occurred');
    });
```

PRACTICE

Write 3 functions which print “Hello from 1” , “Hello from 2” and “Hello from 3” respectively. Each message need to be displayed in your console 1 after another with 2s delay.

Create this scenario by using Javascript’s promise.

PROMISE API

Promise.all() : Let's say we want many promises to execute in parallel and wait until all of them are ready.

```
1  let promise = Promise.all([...promises...]);
```

Example:

```
Promise.all([
  new Promise(resolve => setTimeout(() => resolve(1), 3000)), // 1
  new Promise(resolve => setTimeout(() => resolve(2), 2000)), // 2
  new Promise(resolve => setTimeout(() => resolve(3), 1000))  // 3
]).then(alert); // 1,2,3 when promises are ready: each promise
  contributes an array member
```

Promise. allSettled() : just waits for all promises to settle, regardless of the result..

Example:

```
let urls = [
  'https://api.github.com/users/iliakan',
  'https://api.github.com/users/remy',
  'https://no-such-url'
];

Promise.allSettled(urls.map(url => fetch(url)))
  .then(results => { // (*)
    results.forEach((result, num) => {
      if (result.status == "fulfilled") {
        alert(`${urls[num]}: ${result.value.status}`);
      }
      if (result.status == "rejected") {
        alert(`${urls[num]}: ${result.reason}`);
      }
    });
  });
```

Promise. race() : waits only for the first settled promise and gets its result (or error).

Example:

```
Promise.race([
  new Promise((resolve, reject) => setTimeout(() => resolve(1), 1000)),
  new Promise((resolve, reject) => setTimeout(() => reject(new Error
    ("Whoops!")), 2000)),
  new Promise((resolve, reject) => setTimeout(() => resolve(3), 3000))
]).then(alert); // 1
```

Promise. any() : Similar to Promise.race, but waits only for the first fulfilled promise and gets its result. If all of the given promises are rejected, then the returned promise is rejected with AggregateError.

Example:

```
Promise.any([
  new Promise((resolve, reject) => setTimeout(() => reject(new Error
    ("Whoops!")), 1000)),
  new Promise((resolve, reject) => setTimeout(() => resolve(1), 2000)),
  new Promise((resolve, reject) => setTimeout(() => resolve(3), 3000))
]).then(alert); // 1
```

ASYNC/AWAIT

There's a special syntax to work with promises in a more comfortable fashion, called "async/await". It's surprisingly easy to understand and use.

```
1 async function f() {
2   return 1;
3 }
```

The word "async" before a function means one simple thing: a function always returns a promise. Other values are wrapped in a resolved promise automatically.

```
async function f() {
  return 1;
}

f().then(alert); // 1
```

Is the same as :

```
async function f() {
  return Promise.resolve(1);
}

f().then(alert); // 1
```

async ensures that the function returns a promise, and wraps non-promises in it.

Await

Await only works inside Async function

```
// works only inside async functions
let value = await promise;
```

Example:

```
async function f() {

  let promise = new Promise((resolve, reject) => {
    setTimeout(() => resolve("done!"), 1000)
  });

  let result = await promise; // wait until the promise resolves (*)

  alert(result); // "done!"
}

f();
```