1.9 Promises & async-await

Table of Content

1. Promises in Javascript

In JavaScript, a promise is a good way to handle **asynchronous** operations. It is used to find out if the asynchronous operation is successfully completed or not.

A promise may have one of three states.

- Pending
- Fulfilled
- Rejected

A promise starts in a pending state. That means the process is not complete. If the operation is successful, the process ends in a fulfilled state. And, if an error occurs, the process ends in a rejected state.

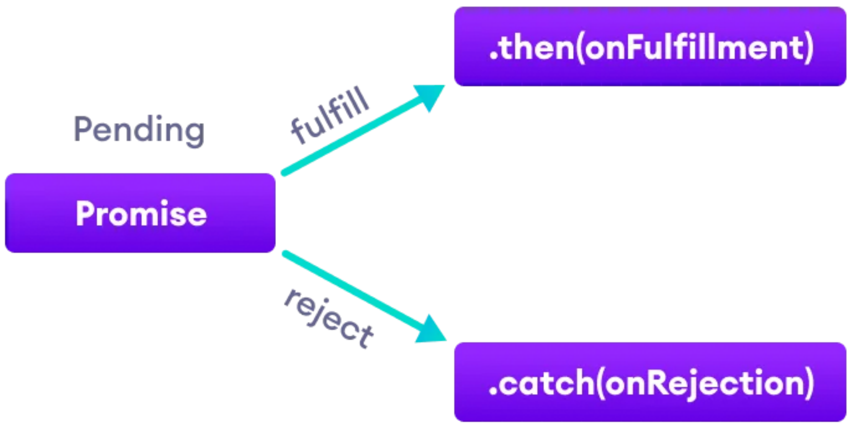
▼ **Create a Promise**

To create a promise object, we use the `Promise()` constructor.

```
let promise = new Promise(function(resolve, reject){
     //do something
});
```

The `Promise()` constructor takes a function as an argument. The function also accepts two functions `resolve()` and `reject()`.

If the promise returns successfully, the `resolve()` function is called. And, if an error occurs, the `reject()` function is called.



▼ Promises Chaining

Promises are useful when you have to handle more than one asynchronous task, one after another. For that, we use promise chaining.

You can perform an operation after a promise is resolved using methods `then()`, `catch()` and `finally()`.

▼ JavaScript then() method

The `then()` method is used with the callback when the promise is successfully fulfilled or resolved.

The syntax of `then()` method is:

```
promiseObject.then(onFulfilled, onRejected);
```

▼ Javascript catch() method

The `catch` () method is used with the callback when the promise is rejected or if an error occurs.

```
api().then(function(result) {
    return api2() ;
}).then(function(result2) {
    return api3();
}).then(function(result3) {
    // do work
}).catch(function(error) {
    //handle any error that may occur before this poi
nt
});
```

2. async-await in Javascript

We use the `async` keyword with a function to represent that the function is an asynchronous function. The async function returns a promise.

The syntax of `async` function is:

```
async function name(parameter1, parameter2, ...paramaterN)
{
    // statements
}
```

Here,

- **name** - name of the function
- **parameters** - parameters that are passed to the function

The syntax to use await is:

```
let result = await promise;
```

The use of `await` pauses the async function until the promise returns a result (resolve or reject) value. For example,

▼ **Benefits of async-await**

- The code is more readable than using a callback or a promise.
- Error handling is simpler.
- Debugging is easier.

Note: These two keywords `async/await` were introduced in the newer version of JavaScript (ES8). Some older browsers may not support the use of async/await.

Assignments

1. Create a process for cart checkout Page using callback & Promises with async-await with the following steps. Here each step can contain a setTimeOut with 2 seconds to mimic the asynchronous delay.
    a. getOrderInfo
    b. checkIfAvailable
    c. placeOrder
    d. returnSuccess

2. Create a process for user signup using callback & Promises with async-await with the following steps. Here each step can contain a setTimeOut with 2 seconds to mimic the asynchronous delay.
    a. getUserInfo
    b. checkIfAlreadyPresent
    c. createAccount
    d. sendSignUpEmail
    e. and returnSuccess