15. Callback, Promises & async-await

Table of Content

1. setTimeOut and setInterval methods

▼ setTimeOut() method

The setTimeout() method calls a function after a number of milliseconds. The `setTimeout()` method executes a block of code after the specified time. The method executes the code only once.

The commonly used syntax of JavaScript setTimeout is:

```
setTimeout(function, milliseconds);
```

Its parameters are:

- **function** - a function containing a block of code
- **milliseconds** - the time after which the function is executed

The `setTimeout()` method returns an **intervalID**, which is a positive integer.

▼ clearTimeOut() method

You generally use the `clearTimeout()` method when you need to cancel the `setTimeout()` method call before it happens.

```
// program to stop the setTimeout() method

let count = 0;

// function creation
function increaseCount(){

    // increasing the count by 1
    count += 1;
    console.log(count)
}

let id = setTimeout(increaseCount, 3000);

// clearTimeout
clearTimeout(id);
console.log('setTimeout is stopped.');
```

Output

```
setTimeout is stopped.
```

▼ setInterval() method

The `setInterval()` method is useful when you want to repeat a block of code multiple times. For example, showing a message at a fixed interval.

The commonly used syntax of JavaScript setInterval is:

```
setInterval(function, milliseconds);
```

Its parameters are:

- **function** - a function containing a block of code
- **milliseconds** - the time interval between the execution of the function

The `setInterval()` method returns an **intervalID** which is a positive integer.

▼ clearInterval() method

The syntax of `clearInterval()` method is:

```
clearInterval(intervalID);
```

Here, the `intervalID` is the return value of the `setInterval()` method.

---

2. Promises in Javascript

▼ Callback Functions

A callback function is passed as an argument to another function. It helps us to notify about certain events from the called function.

```
// function
function greet(name, callback) {
    console.log('Hi' + ' ' + name);
    callback();
}

// callback function
function callMe() {
    console.log('I am callback function');
}

// passing function as an argument
greet('Peter', callMe);
```

If we have a long chain of such callback functions, it can create a chain of nested function calls aka the callback hell.

▼ Promises

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.

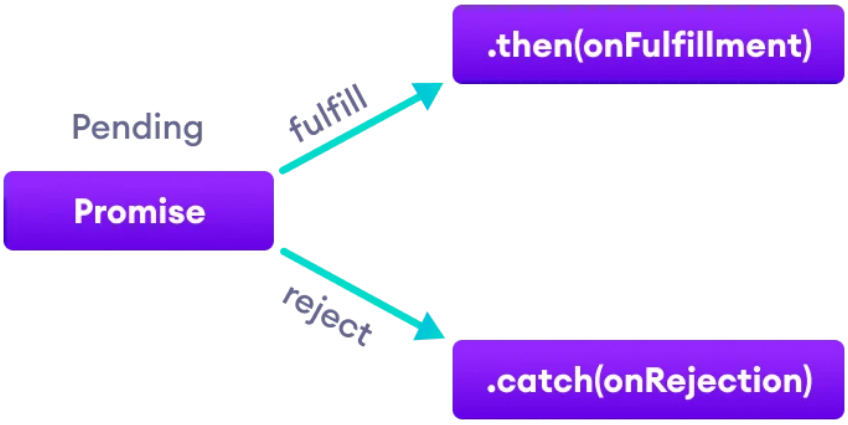
▼ Create a Promise

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.

▼ Promise vs Callback

▼ JavaScript Promise

1. The syntax is user-friendly and easy to read.
2. Error handling is easier to manage.
3. Example**:**

```
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 point
});
```

▼ **JavaScript Callback**

1. The syntax is difficult to understand.
2. Error handling may be hard to manage.
3. Example**:**

```
api(function(result){
    api2(function(result2){
        api3(function(result3){
             // do work
            if(error) {
                // do something
            }
            else {
                // do something
            }
        });
    });
});
```

---

3. 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