Episode 1 : Callback

- There are 2 Parts of Callback:

    i. Good Part of callback - Callback are super important while writing asynchronous code in JS

    ii. Bad Part of Callback - Using callback we can face issue:
    - Callback Hell
    - Inversion of control

- Understanding of Bad part of callback is super important to learn Promise in next lecture.

> 💡 JavaScript is synchronous, single threaded language. It can Just do one thing at a time, it has just one call-stack and it can execute one thing at a time. Whatever code we give to Javascript will be quickly executed by Javascript engine, it does not wait.

```
console.log('Namaste');
console.log('JavaScript');
console.log('Season 2');
// Namaste
// JavaScript
// Season 2

// 💡 It is quickly printing because `Time, tide & Javascript waits for none.`
```

But what if we have to delay execution of any line, we could utilize callback, How?

```
console.log('Namaste');
setTimeout(function () {
  console.log('JavaScript');
}, 5000);
console.log('Season 2');
// Namaste
// Season 2
// JavaScript

// 💡 Here we are delaying the execution using callback approach of setTimeout.
```

🛒 e-Commerce web app situation

Assume a scenario of e-Commerce web, where one user is placing order, he has added items like, shoes, pants and kurta in cart and now he is placing order. So in backend the situation could look something like this.

```
const cart = ['shoes', 'pants', 'kurta'];
// Two steps to place a order
// 1. Create a Order
// 2. Proceed to Payment

// It could look something like this:
api.createOrder();
api.proceedToPayment();
```

Assumption, once order is created then only we can proceed to payment, so there is a dependency. So How to manage this dependency. Callback can come as rescue, How?

```
api.createOrder(cart, function () {
  api.proceedToPayment();
});
// 💡 Over here `createOrder` api is first creating a order then it is responsible to call `api.proceedToPayment()` as part of callback approach.
```

To make it a bit complicated, what if after payment is done, you have to show Order summary by calling `api.showOrderSummary()` and now it has dependency on `api.proceedToPayment()` Now my code should look something like this:

```
api.createOrder(cart, function () {
  api.proceedToPayment(function () {
    api.showOrderSummary();
  });
});
```

Now what if we have to update the wallet, now this will have a dependency over `showOrderSummary`

```
api.createOrder(cart, function () {
  api.proceedToPayment(function () {
    api.showOrderSummary(function () {
      api.updateWallet();
    });
  });
});
// 💡 Callback Hell
```

When we have a large codebase and multiple apis and have dependency on each other, then we fall into callback hell. These codes are tough to maintain. These callback hell structure is also known as **Pyramid of Doom**.

Till this point we are comfortable with concept of callback hell but now lets discuss about `Inversion of Control`. It is very important to understand in order to get comfortable around the concept of promise.

> 💡 Inversion of control is like that you lose the control of code when we are using callback.

Let's understand with the help of example code and comments:

```
api.createOrder(cart, function () {
  api.proceedToPayment();
});

// 💡 So over here, we are creating a order and then we are blindly trusting `createOrder` to call `proceedToPayment`.

// 💡 It is risky, as `proceedToPayment` is important part of code and we are blindly trusting `createOrder` to call it and handle it.

// 💡 When we pass a function as a callback, basically we are dependant on our parent function that it is his responsibility to run that function. This is called `inversion of control` because we are dependant on that function. What if parent function stopped working, what if it was developed by another programmer or callback runs two times or never run at all.

// 💡 In next session, we will see how we can fix such problems.
```

> 💡 Async programming in JavaScript exists because callback exits.

more at `http://callbackhell.com/`

Watch Live On Youtube below:

Edit this page