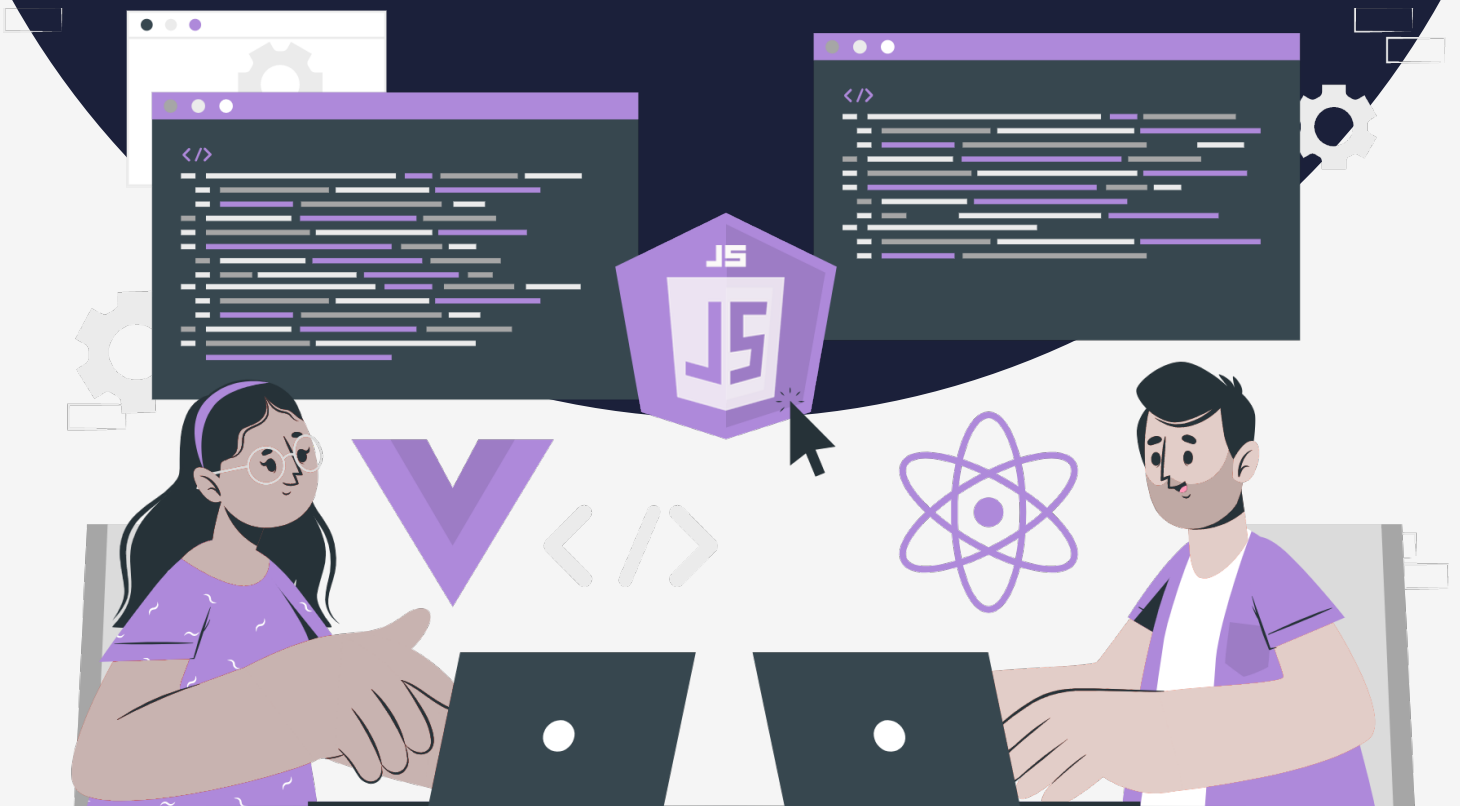


Lesson:

Callback, Returning function



List of content :

1. What is callback
2. Example
3. What is callback hell
4. Returning a function
5. Examples

What is a callback?

A callback is a function that is to be executed after another function has finished execution. Alternatively, we can also define it as any function that is passed as an argument to another function so that it may be executed in that other function is referred to as a callback function.

Example:

```
function hiFunction(name, callback) {
  console.log('Hi' + ' ' + name + ' I am the actual function');
  callback();
}
// callback function
function exampleFunction() {
  console.log('I come from the callback function');
}
// passing function as an argument
hiFunction('Folks', exampleFunction);
```

In this example, we have created two functions viz `hiFunction()` and `exampleFunction()`. While calling the `hiFunction()` function, two arguments (a string value `name` and a function) are passed. This is the function from where the execution starts.

The `exampleFunction()` function is a callback function and starts its execution on encountering the `callback()`.

Output:

```
[Running] node "c:\Users\ACER\Desktop\Co
Hi Folks I am the actual function
I am the callback function
[Done] exited with code=0 in 0.093 secon
```

The benefit of using a callback function is that you can wait for the result of a previous function call and before executing another function call.

Callback Hell :

It is nested callbacks stacked below one another forming a pyramid kind of structure. Every callback here, waits for the previous callback, thereby affecting the readability and maintainability of the code.

Example:

```
getProduct(18, (user) => {
  console.log("Get Products", user);
  getUserInfo(user.username, (name) => {
    console.log(name);
    getAddress(name, (item) => {
      console.log(item);
      // this goes on and on...    })
  })
})
```

In the example, `getUserInfo` takes in an argument which is dependent or needs to be extracted from the result produced by `getProducts` which is inside `user`. The same dependency can be observed with `getAddress` also. This is callback hell.

There are a few ways to avoid these callback hells which we will discuss in the lectures ahead.

Returning function

As we already know, when a return statement is encountered in a function body, the execution of the function stops. If specified, a given value is returned to the function caller.

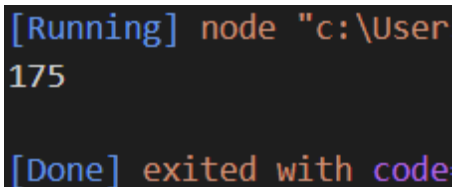
Till now we have seen examples of functions returning/not returning a value. However, in higher-order function concepts, we saw that functions can also return a function.

Let us look at the example below for a better understanding:

Example 1: Demonstrates a simple function that returns a function.

```
multiplyBy7() {
  return function(x) { return x * 7; };
}
const changedValue = multiplyBy7();
console.log(changedValue(25));
```

Output:



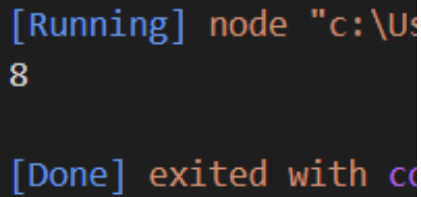
```
[Running] node "c:\User
175
[Done] exited with code
```

Here, as you can observe, the function `changeNumber()` returns a function `change()` which in itself does some operation before returning the result.

Example 2: The example below shows an implementation where both the parent function and returned functions are taking parameters.

Example:

```
function makeAdder(n1) {  
  return function(n2) { return n1 + n2;};  
}  
console.log( makeAdder(5)(3) );
```

Output:

```
[Running] node "c:\Us  
8  
[Done] exited with co
```

Here, you can see that the function *additionFunction()* takes a parameter *n1* and returns the addition of *n1* and *n2* (*n2* being a parameter passed to the function *returnFunc()* returned by *additionFunc()*).