Fotios Floros Follow
Full stack developer @Effect, Programmer, Photographer, Cinema enthusiast, coffee lover
Jan 1 · 3 min read

Explaining Javascript Callbacks

Callback functions are a concept derived from functional programming and specifies the use of functions as arguments.

In Javascript every function is a first class object, which means that every function is an Object and can be used like any other object(String, Number etc.). This allows the use of a function as a parameter in another function which is the fundamental idea of callback functions.

Note that when passing a callback function as a parameter to another function we are only passing the callback function definition.

Those are the main aspects that I am going to cover here

1. Use anonymous functions as callbacks
2. Use named functions as callbacks
3. Parameters to callback functions
4. Multiple callback functions
5. Is it optional? Is it a function?

Use anonymous functions as callbacks

Example: jQuery is widely using callback functions. In the following example there is a callback function that is executed every time a certain button is clicked.

```
$("#submitButton").click(function(){
   console.log("submit");
})
```

Example: Let’s see another example, parse an array and print it’s elements.

```
var fruits = ["apple","bannana","orange"];
   fruits.forEach(function(fruit, index){
   console.log(index+1+". "+fruit);
});
```

Use named functions as callbacks

Example: In it’s definition testFunction() has *callback*(which can be any function) as a parameter. Later when we call testFunction() we use the already defined function namedCallback()

```
function namedCallback(){
   alert("namedCallback()");
}
function testFunction(callback){
   callback();
}
testFunction(namedCallback);
```

Parameters to callback functions

We can also pass parameters to callback functions, just like we do in any other function.

Example: In the following example we pass as a parameter the name of the author of the function

```
var author = “FF”;

function namedCallback(param){

   alert(“namedCallback() called by “+param);

}

function testFunction(callback){

   callback();

}

testFunction(namedCallback(author));
```

Multiple callback functions

Multiple callback functions can be used as parameters of another function.

Example: jQuery’s AJAX function

```
Var someUlr = …;

function successCallback(){

   //success code

}

function completeCallback(){
   //complete code

}

function errorCallback(){

   //error code

}

$.ajax({

   url: someUrl,

   success: successCallback,

   complete: completeCallback,

   error: errorCallback

})
```

Is it optional? Is it a function?

It is a good idea to make your callback function optional, which means that your other function(the one that is using the callback) can run without the callback as a parameter so that your code is as general and maintainable as possible.

You should also check that that the callback is indeed a function!

Example

```
var message = “hello there!”;

function myCallbackFunction(){

   alert(“myCallbackFunction is called()”);

}

function testCallback(param, callback){

   alert(param);
```

```
    if(callback && typeof(callback) === “function”){

        callback();

    }

}

testCallback(message); // see what happens when you add
myCallbackFunction as a parameter here
```

Summarize

I have met Javascript developers who do not have realized the concept of callback functions and how powerful they are. Even if you do not use your own callback functions, it’s of utmost importance to understand that you can use callback functions:

- For asynchronous execution
- For event listeners/handlers
- For generalization

Moreover callback functions will allow you to:

- Write more clean and readable code
- DRY—Do not repeat yourself
- Have better maintainability
- Understand the use of maaany Javascript libraries that are using callbacks