Hyperiondev

TASK

# Higher-order Functions

Visit our website

# Introduction

## WELCOME TO THE HIGHER-ORDER FUNCTIONS TASK!

In this task, you will learn about one of the most important fundamental concepts in Javascript, Higher-Order Functions (HOF). This is a concept that is essential to understanding more advanced concepts in Javascript, which will be introduced later in this bootcamp.

Get in touch

**Connect for support**

Remember that with our courses, you're not alone! You can contact an expert code reviewer to get support on any aspect of your course.

The best way to get help is to login to Discord at **https://discord.com/invite/hyperdev** where our specialist team is ready to support you.

Our team is happy to offer you support that is tailored to your individual career or education needs. Do not hesitate to ask a question or for additional support!

## WHAT ARE HIGHER-ORDER FUNCTIONS?

Higher-order functions (HOF) are functions that take *a function* as *an argument* and/or *return* a function. We use them a lot in functional programming. They are a way to define reusable functionality and a less tedious alternative to class inheritance.

Higher-order functions allow JavaScript to be suitable for functional programming. You can find higher-order functions used extensively in Javascript. You've already encountered higher-order functions when you dealt with curried functions that returned other functions.

In order to understand this concept well, it is beneficial first to understand what functional programming is, and importantly the concept of first-class functions.

## WHAT IS FUNCTIONAL PROGRAMMING?

Functional programming is a programming paradigm in which you operate on functions as you would with any other kinds of values. This makes functions in functional programming first-class, meaning they are just like any other value you work with. Higher order functions and closures are fine examples of how you would treat functions as values.

### First-class functions

JavaScript treats functions as first-class objects. Functions in Javascript are values. That means that they can be assigned to variables and they can also, therefore, be passed as arguments because they are values (objects.) This is the most important thing to remember when working with higher-order functions.

In Javascript, functions are a special kind of object called Function objects. We can assign functions as values to variables because they are objects.

Let us look at the following example.

```
//this is a simple example of a higher-order function
function higherOrderExample() {
    //the function returns the greeting function
    return greeting = () => 'Hello, World!'
  }

//here we are calling the higher order function and by doing so,
//we are assigning the greeting function object to a variable
let useGreeting = higherOrderExample();

//calling the useGreeting function will return the "Hello world" string
console.log(useGreeting());
```

## BUILT-IN HIGHER ORDER FUNCTIONS

Javascript actually already has a bunch of built-in higher-order functions that make our lives easy. The Array class has built-in methods that take functions as arguments such as the map(), filter() and reduce methods. Let's look at an example of how the map() method can be used.

Let's say we have an array of numbers. We want to create a new array that will contain the doubled values of the first one. Let's see how we can solve this problem with and without a higher-order function.

Conventionally we could achieve it like below with an empty array and a loop that does the calculation on each element and pushes the values to the empty array:

```
function doubleAllValues() {
    const arr1 = [1, 2, 3, 4]
    //creating an empty array to push the values to
    const arr2 = []
    for (let i = 0; i < arr1.length; i++) {
      arr2.push(arr1[i] * 2) // the arr2 array grows in a loop
    }

    return arr2 // 2, 4, 6, 8 only without spaces
```

```
  }
```

Alternatively, we can use the map method to create a much cleaner solution by passing a function as an argument to the map method:

```
function doubleAllValues() {
    const arr1 = [1, 2, 3, 4, 5]
    // create an arrow function that multiplies a number by 2
    const multTwo = item => item * 2
    /* we are passing the function multTwo as an argument to the map method.
      It will multiply each element by 2 and create a new array with the
adjusted values*/
    const arr2 = arr1.map(multTwo)
    return arr2
  }
```

## CREATING A HIGHER-ORDER FUNCTION

Let's create our own higher-order function (HOF) that mimics the behaviour of the built-in map method so that we get a better understanding of how this works. Our higher-order function **`myMapper()`**, will take an array of strings and a function as an argument. This will then apply the function to every element in the array passed as an argument and return a new array. When we pass a function as an argument we call that a **callback function** - we will cover this in more detail as we progress further in the bootcamp. Also note: a function without a name is called an anonymous function. Read more on anonymous functions **here**.

```
const wordsArray = ['Express', 'JavaScript', 'React', 'Next']
// higher order function countLetters() takes a hypothetical function fn and
an array arr
let myMapper = arr => fn => {
    const  arrayAfterMapping = []
    for (let i = 0; i < arr.length; i++) {
          arrayAfterMapping.push(fn(arr[i])) // We apply the callback function
fn() to each element
    }
```

```
    return arrayAfterMapping // Returning a new array
}
// we can now use the countLetters HOF with an anonymous function as one of
the arguments (returning a length as a number followed by a space)
const outputArray = myMapper(wordsArray)(item => item.length + ' ')

//we can now output it to the console
console.log('Length of words: ' + outputArray) // Length of words: 7 ,10 ,5 ,4
```

For this task, you will be making your own higher-order function that mimics the behaviour of a built-in method that already exists. Before doing this, please look at how the filter() method in Javascript works, from this **resource**.

## Compulsory Task 1

Follow these steps:

- Create a higher-order function named **`myFilterFunction()`** in a file named **`higherOrder.js`**
- This function should take the following 2 arguments:
    - An array of strings with 10 words, where at least 3 of the words have 6 letters.
    - A callback function that returns a boolean based on whether or not a word has 6 letters,
- **`myFilterFunction()`** should return a new array that contains only the words that are 6 letters long and no other words.
- **You cannot use the built-in filter method to complete this task!**

Rate us

# Share your thoughts

HyperionDev strives to provide internationally-excellent course content that helps you achieve your learning outcomes.

Think that the content of this task, or this course as a whole, can be improved, or think we've done a good job?

**Click here** to share your thoughts anonymously.