

# Callbacks and Timers

## Goals

- Understand the terms first class functions and higher order functions (HOF)
- Learn how to use and build your own callbacks
- Learn how to manipulate timers in JavaScript

## First Class Functions

- Functions in JavaScript are quite flexible because they are essentially treated just like other data types.
- And as you already know, you can assign functions to variables.
- For example, you can pass functions as arguments to other functions

```
function takeAFunction(anotherFunc) {  
  return anotherFunc(); // calling the parameter as a function  
}  
  
let sayHi = function() {  
  console.log('hello')  
};
```

This is what we mean by first class functions!

## Higher Order Functions

A function is a HOF if it does at least one of the following:

- Accepts another function as a parameter
- Returns another function
- HOFs are a general concept in mathematics, not just JavaScript. However, they are pretty straightforward!

```
function myHOF() {  
  return function() {  
    console.log('Returning this function also makes me a HOF!');  
  }  
}
```

## Callbacks

Now that we know about first class functions and HOFs, we finally know how to define callback functions!

```
function markWahlberg(animal, callback) {  
  console.log(`Hey ${animal}, how you doin'?`);  
  callback();  
}  
  
function marksCallback() {  
  console.log('Say hi to your mother for me, alright?');  
}
```

Simply put, a callback is a (first class) function that gets passed as a parameter to another function (a HOF).

The HOF will invoke the callback at some point.

```
markWahlberg('chicken', marksCallback);
```

## Why callbacks?

- They can reduce repetition and re-definition of functions
- They are commonplace with more advanced array methods!

## An example

Imagine you are building a simple calculator, let's start with some basic functions

```
function add(a, b){  
  return a + b;  
}  
  
function subtract(a, b){  
  return a - b;  
}  
  
function multiply(a, b){  
  return a * b;  
}  
  
function divide(a, b){  
  return a / b;  
}
```

This seems great, but:

- What happens when we want to do other operations like square roots
- What happens if we want to do multiple operations with **a** and **b** like  **$a * b + b * a$** ?
- We need to keep defining new functions each time!

## Using a callback

```
function doMath(a, b, callback){  
  return callback(a,b)  
}  
  
doMath(10, 20, function(first, second){  
  return first + second  
})  
  
doMath(5, 10, function(first, second){  
  return first * second / second + first  
})  
  
doMath(5, 5, add) // 10  
doMath(5, 5, subtract) // 0
```