M1W2 - Fun With Callbacks

LIGHTHOUSE LABS

“Don't trust javascript programmers
All they do is promises but they
never callback.”

AGENDA

Functions as First-class Objects

Higher-Order Functions

Callbacks

Single Responsibility Principle

Anonymous & Arrow Functions

Scope Chain

Breakout Exercise

LIGHTHOUSE LABS

Functions as First-class Objects

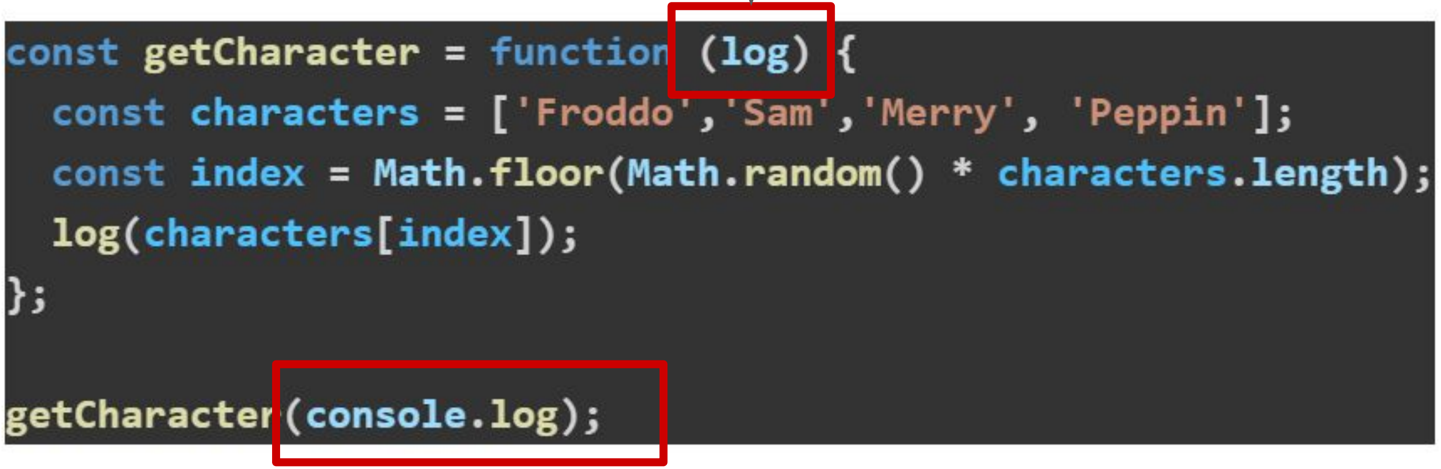
- A function can be treated like **any other value** in JS
- It can be assigned to a variable
- It can be passed as an argument
- It can returned by another function

Higher Order Function

A function that accepts another function as an **input parameter** or **return another function**.

Higher Order Function

getCharacter accepts a function as an **input parameter**

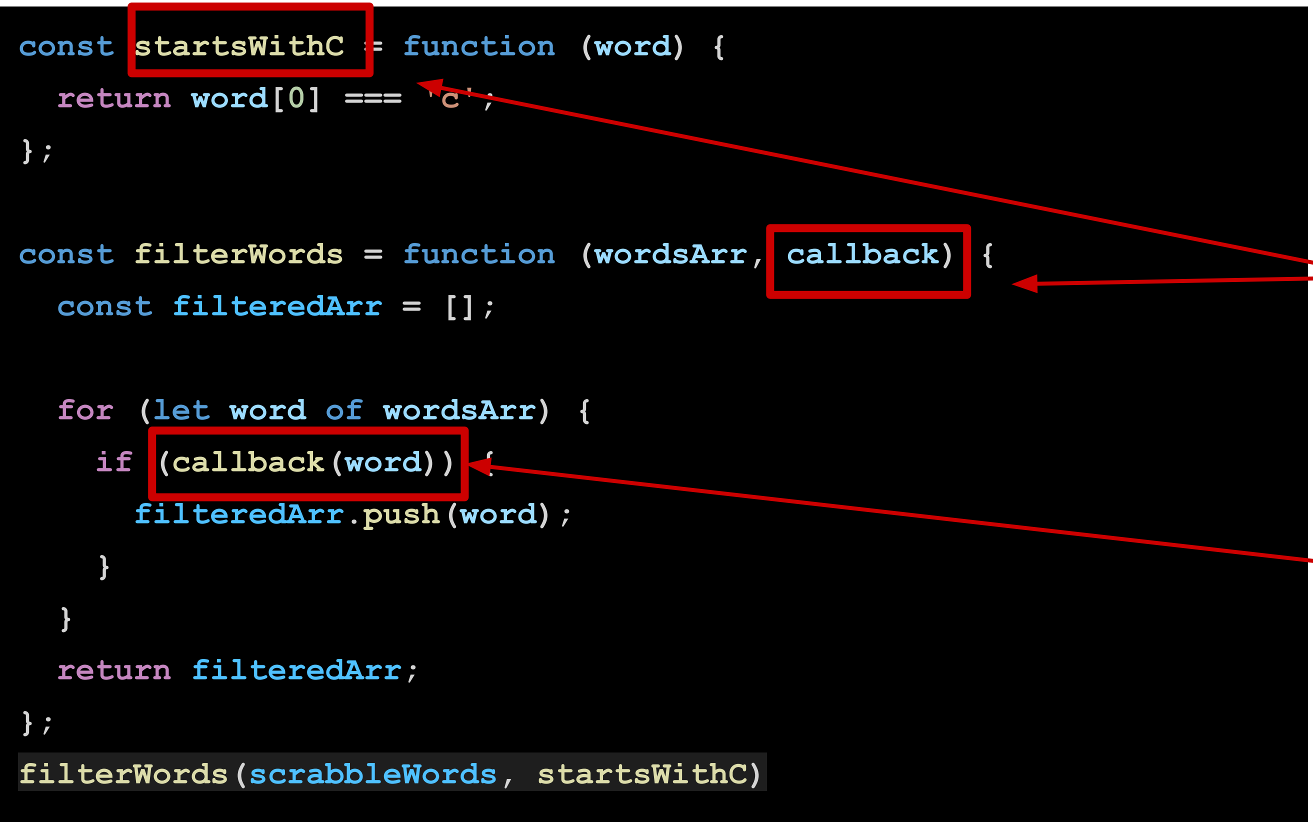


```
const getCharacter = function (log) {
  const characters = ['Froddo','Sam','Merry', 'Peppin'];
  const index = Math.floor(Math.random() * characters.length);
  log(characters[index]);
};

getCharacter(console.log);
```

getCharacter is a **Higher Order Function**

Passing a **function as an argument** when calling getCharacter

Callbacks

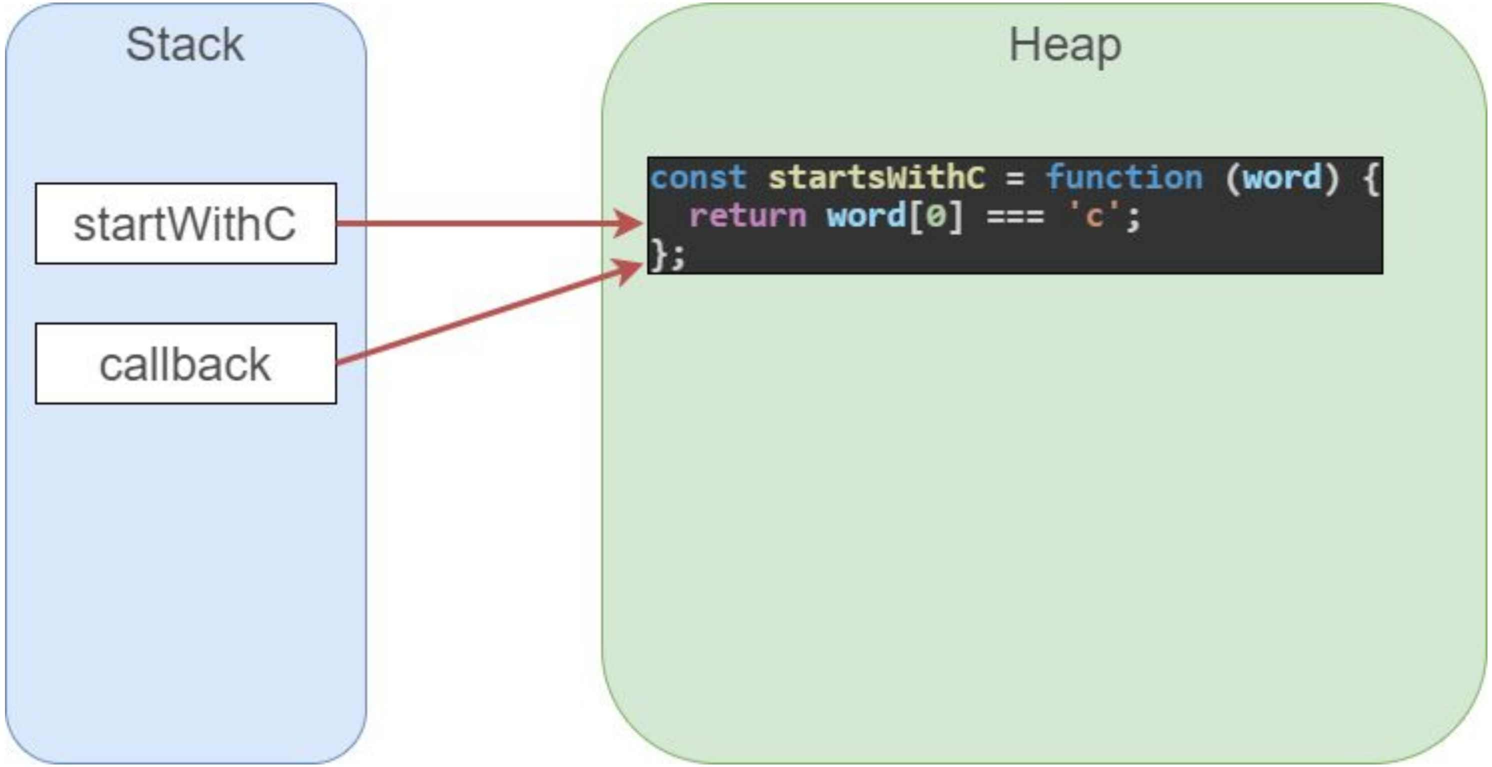


```
const startsWithC = function (word) {
  return word[0] === 'c';
};

const filterWords = function (wordsArr, callback) {
  const filteredArr = [];

  for (let word of wordsArr) {
    if (callback(word)) {
      filteredArr.push(word);
    }
  }
  return filteredArr;
};
filterWords(scrabbleWords, startsWithC)
```

startsWithC and **callback** are *2 variables* that point to the **same anonymous function.**

Executing *callback(word)* is **executing the same anonymous function** as *startWithC*.

Callbacks



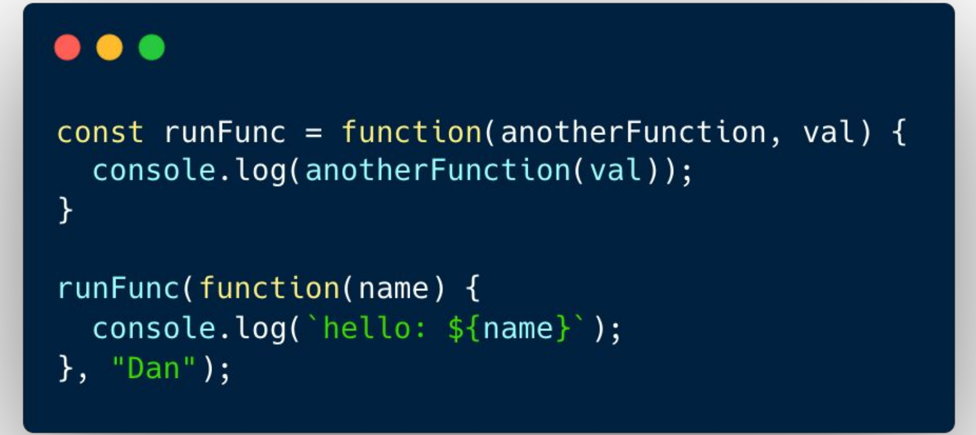
```
const startsWithC = function (word) {
  return word[0] === 'c';
};
```

Single Responsibility Principle

- A function should do only 1 thing
- filterWords and startWithC are now doing only 1 thing each

Anonymous Functions

We can declare our functions inline in our code and without name



```
const runFunc = function(anotherFunction, val) {
  console.log(anotherFunction(val));
}

runFunc(function(name) {
  console.log(`hello: ${name}`);
}, "Dan");
```

Arrow Functions

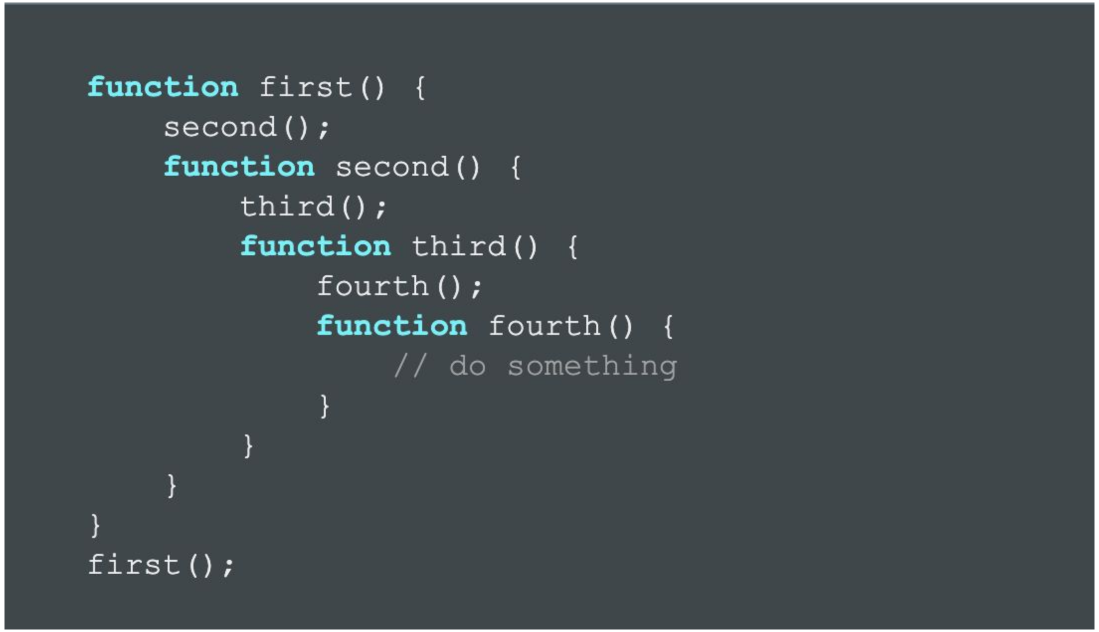
We can transform our function expressions into arrow functions.

```
const startsWithC = function (word) {
  return word[0] === 'c';
};
```

```
const startsWithC = (word) => word[0] === 'c';
```

- Uses them ALWAYS going forward UNLESS you need to use 'this' inside of your function

Scope Chain



```
function first() {
    second();
    function second() {
        third();
        function third() {
            fourth();
            function fourth() {
                // do something
            }
        }
    }
}
first();
```

Scope Chain

- There are two types of scope in JS:
  - Global Scope
  - Local Scope
- Variables defined inside a function are in the local scope
- Variables defined outside of a function are in the global scope.
- Each function when invoked creates a new scope.

Breakout Exercise

Let’s build our own higher order function forEachReverse

Questions?

LIGHTHOUSE LABS