First-Class Functions

- Due No Due Date
- Points 1
- Submitting a website url

(https://github.com/learn-co-curriculum/phase-1-first-class-functions) (https://github.com/learn-co-curriculum/phase-1-first-class-functions/issues/new)

Learning Goals

- Define "first-class function"
- Use inline functions
- Use functions as return values
- Define "higher-order function"

Introduction

Sometimes in life, we need to take a first step. Since life isn't scripted like a reality TV show, anything can happen after that initial step. We need to be able to adjust accordingly.

Imagine an exercise routine: every morning, we run 5 miles. But afterwards — depending on the day — we might lift weights, go for a swim, or run an extra 5 miles.

In programming-speak, we could write out a function for every day (follow along by writing out these examples in a REPL, or in the `index.js` file!):

```
function Monday() {
  console.log("Go for a five-mile run");
  console.log("Pump iron");
}

function Tuesday() {
  console.log("Go for a five-mile run");
```

```
  console.log("Swim 40 laps");
}

function Wednesday() {
  console.log("Go for a five-mile run");
  console.log("Go for a five-mile run");
}

function Thursday() {
  console.log("Go for a five-mile run");
  console.log("Pump iron");
}

function Friday() {
  console.log("Go for a five-mile run");
  console.log("Swim 40 laps");
}
```

But that's pretty tedious. And we already know that functions are supposed to help us *reduce* this kind of repetition.

What if we pull all of our five-mile runs into their own function?

```
function runFiveMiles() {
  console.log("Go for a five-mile run");
}
```

Okay, that cuts down *slightly* on how much code we need to write. Let's do the same thing for lifting weights and swimming:

```
function liftWeights() {
  console.log("Pump iron");
}

function swimFortyLaps() {
```

```
  console.log("Swim 40 laps");
}
```

Awesome! We've cut down a little bit more: `Monday()` could now look like:

```
function Monday() {
  runFiveMiles();
  liftWeights();
}
```

While it is a tiny bit shorter than before, there is definitely still room for improvement. We know that every day, our routine includes two activities. We also know that the first activity is always a run. That means that the second activity can be variable. What if we created a function that took the second activity as a parameter?

```
function exerciseRoutine(postRunActivity) {
  runFiveMiles();
  postRunActivity();
}
```

Notice that, in `exerciseRoutine()`, the `postRunActivity` parameter is a *callback function* — we call it after we call `runFiveMiles()`. Now let's try to use this new function we created in our `Monday()` function:

```
function Monday() {
  exerciseRoutine(liftWeights);
}

function exerciseRoutine(postRunActivity) {
  runFiveMiles();
  postRunActivity();
}
```

Note that we aren't *calling* `liftWeights`. When we want to pass a function as a value, we pass it by *reference* by omitting the parentheses at the end of the function. We're not running the function at this point. It's up to `exerciseRoutine()` to call the function when it is needed.

If we call `Monday()`, we'll see that we run five miles, and then we lift weights — awesome!

Define First-Class Functions

Functions in JavaScript are **first-class objects**, which means they can be treated like any other object: they can be assigned to a variable, passed as values to other functions, returned as the value from another function, etc. They're super useful, as you can see — they even help us exercise in the mornings!

> Note that we stated above that JavaScript functions can be treated like any *other* object. In JavaScript, functions are a special type of object!

Inline Functions

What if, though, we want to have a one-off day of Pilates in our exercise routine? Keep in mind that our `exerciseRoutine()` function requires a function as its first (and only) parameter. However, that function doesn't have to be defined beforehand! We can pass an *anonymous function* to `exerciseRoutine()`.

To start with, let's use the full function syntax we've come to know and love:

```
exerciseRoutine(function () {
  console.log("Stretch! Work that core!");
});

// "Go for a five-mile run"
// "Stretch! Work that core!"
```

We can rewrite this to be more concise by using an arrow function:

```
exerciseRoutine(() => {
  console.log("Stretch! Work that core!");
```

```
});

// Or even shorter:
exerciseRoutine(() => console.log("Stretch! Work that core!"));
```

Because we only need to use our function this one time, there's no need to give it a name or assign it to a variable. Instead, we define it inline as an anonymous function, passing it as the argument when we call `exerciseRoutine()`.

Returning Functions

Functions can also return other functions. This is useful when we want to package up a function and its environment, but don't want to call it *just yet*.

For example, let's say our morning routine involves drinking a cup of coffee, exercising immediately, and then at some point later (depending on how we feel), eating breakfast. What we'll have for breakfast depends on what kind of exercise we're doing.

Let's translate this to a function:

```
function morningRoutine(exercise) {
  let breakfast;

  if (exercise === liftWeights) {
    breakfast = "protein bar";
  } else if (exercise === swimFortyLaps) {
    breakfast = "kale smoothie";
  } else {
    breakfast = "granola";
  }

  exerciseRoutine(exercise);

  // we could give this function a name if we wanted to, but since
  // it's only available _inside_ morningRoutine(), we don't need to
  return function () {
```

```
    console.log(`Nom nom nom, this ${breakfast} is delicious!`);
  };
}
```

Now when we call `morningRoutine()`, our exercise routine will be logged as before, but we'll also get a function back:

```
const afterExercise = morningRoutine(liftWeights);
// LOG: Go for a five-mile run
// LOG: Pump iron

afterExercise;
//=> ƒ () { console.log(`Nom nom nom, this ${breakfast} is delicious!`); }
```

And we can call that function later:

```
afterExercise();
// LOG: Nom nom nom, this protein bar is delicious!
```

If you haven't been following along, it's vitally important that you go back and do so. First-class functions are one of JavaScript's most powerful features, but it takes some practice for them to sink in.

Higher-Order Functions

A higher-order function is a function that can accept functions as arguments and/or return a function. You can read more about them **here (https://www.freecodecamp.org/news/a-quick-intro-to-higher-order-functions-in-javascript-1a014f89c6b/)** and **here (https://dmitripavlutin.com/javascript-higher-order-functions)**.

Instructions

If you haven't already, **fork and clone** this lab into your local environment. Navigate into its directory in the terminal, then run `code .` to open the files in Visual Studio Code.

To get more practice with first-class functions, this lesson has three tests to pass that require you to write the following functions in the `index.js` file:

- The `receivesAFunction` function should:
  - take a *callback function* as an argument
  - call the callback function
  - it doesn't matter what this function returns, so long as it calls the callback function
- The `returnsANamedFunction` function should:
  - take no arguments
  - return a *named function*
- The `returnsAnAnonymousFunction` function should:
  - take no arguments
  - return an *anonymous function*

When you're done, remember to commit and push your changes up to GitHub, then submit your work to Canvas using CodeGrade.

Resources

- **Wikipedia: First-class function (https://en.wikipedia.org/wiki/First-class_function)**
- **FreeCodeCamp: A Quick Intro to Higher-Order Functions in JavaScript (https://www.freecodecamp.org/news/a-quick-intro-to-higher-order-functions-in-javascript-1a014f89c6b/)**
- **Dmitri Pavlutin: What are Higher-Order Functions in JavaScript? (https://dmitripavlutin.com/javascript-higher-order-functions)**
- **MDN Function Expression (named vs anonymous functions) (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/function)**