Object Basics

Foundations Course

Introduction

Congratulations on making it to one of the last lessons in Foundations! By this point, you have learned many of the fundamentals of JavaScript. In this lesson, you will learn about Objects - a collection of key-value pairs - as well as some more powerful and commonly used *array* methods.

Before you know it, you'll have a better understanding of how powerful objects and arrays are and how both can be an indispensable part of your JavaScript tool kit!

Lesson overview

This section contains a general overview of topics that you will learn in this lesson.

- Creating objects.
- Accessing object properties.
- Using multiple object operators.
- Understanding the differences between object and primitive data types.

Objects

Objects are a *very* important part of the JavaScript language, and while for the most part you can accomplish simple and even intermediate tasks without worrying about them, any real project that you're going to attempt is going to feature Objects. The uses of Objects in JavaScript can get deep relatively quickly, so for the moment we're only going to cover the basics. There'll be an in-depth dive later.

1. This JavaScript.info article on objects is the best place to get started. You do not need to do the exercises at the end of the page.
2. Follow along to MDN's "JavaScript object basics". Skip the "Introducing constructors" section, as that's a more advanced topic we'll come to later in the curriculum.

Differences between objects and primitives

Earlier in the curriculum you've learned about primitive data types. Now that you've seen the object data type, which includes but is not limited to, objects ({key: value}), arrays, and functions. The main difference between the two is that primitives can contain only a single thing (string, number, etc). Objects data types are used to store a collection of data and more complex entities.

Besides the formal differences, there are also some technical differences which affect how we use each data type. When you define a primitive variable, it will contain a copy of the information provided to it:

```
let data = 42;
// dataCopy will store a copy of what data contains, so a copy of 42
let dataCopy = data;

// which means that making changes to dataCopy won't affect data
dataCopy = 43;

console.log(data); // 42
console.log(dataCopy); // 43
```

On the other hand, when you define an object variable, it will contain a *reference* to the object provided to it:

```
// obj contains a reference to the object we defined on the right si
const obj = { data: 42 };
// objCopy will contain a reference to the object referenced by obj
const objCopy = obj;

// making changes to objCopy will make changes to the object that it
objCopy.data = 43;

console.log(obj); // { data: 43 }
console.log(objCopy); // { data: 43 }
```

This behavior isn't new to you. In your last project, you made changes to the cells in the Etch-A-Sketch grid by using references. Let's take this code snippet as an example:

```
const element = document.querySelector("#container");
element.style.backgroundColor = "red";
```

We're mutating the variable we declared (`element`), yet the changes affect the corresponding node in the DOM. Why does it happen? The node we have in our code is a **reference** to the same node that our DOM uses. If `element` wasn't a reference, but instead was a copy of the node itself (like how primitive data types behave), mutating it would have **no** effect because changes would only affect the copy, not the original.

This behavior is also something to consider when we pass arguments to a function. Let's take the following functions for example:

```
function increaseCounterObject(objectCounter) {
  objectCounter.counter += 1;
}

function increaseCounterPrimitive(primitiveCounter) {
  primitiveCounter += 1;
}

const object = { counter: 0 };
let primitive = 0;

increaseCounterObject(object);
increaseCounterPrimitive(primitive);
```

Take a moment and guess what will happen to `object` and what will happen to `primitive` after we make the function calls.

If you answered that the object counter would increase by 1, and the primitive counter wouldn't change, you're correct. Remember that the parameter `objectCounter` contains a *reference* to the same object as the `object` variable we gave it, while `primitiveCounter` only contains a copy of the primitive value.

📝 Reassigning object data type variables

When we mutate an object that is referenced by multiple variables, the change will be visible through all those variables, since they all reference the same object. However, reassigning one of those variables to a new object will not also reassign the other variables. For example:

```
let animal = { species: "dog" };
let dog = animal;

// reassigning animal variable with a completely new object
animal = { species: "cat" };

console.log(animal); // { species: "cat" }
console.log(dog); // { species: "dog" }
```

Assignment

1. Now that you've been exposed to objects, go back to the array methods exercises at the end of the JavaScript.info array methods article from a few lessons ago. Do only the following exercises:
   - `Map to names`
   - `Map to objects`
   - `Sort users by age`
   - `Get average age`
   - `Create keyed object from array`
2. More practice with objects (Inside arrays!). Fork and clone Wes Bos' JavaScript30 repository. To follow along with these two video exercises, you'll want to open the `index-START.html` file.
   - 04 - Array Cardio Day 1
   - 07 - Array Cardio Day 2
3. Go back to the JavaScript exercises repository's `foundations` directory. Review each README file prior to completing the following exercises in order:
   - `12_calculator`
   - `13_palindromes`
   - `14_fibonacci`
   - `15_getTheTitles`
   - `16_findTheOldest`

   Note: Solutions for these exercises can be found in the `solution` folder of each exercise.

If you feel yourself getting overwhelmed or stuck, don't be afraid to go back and review or ask for help in the TOP Discord server!

Knowledge check

The following questions are an opportunity to reflect on key topics in this lesson. If you can't answer a question, click on it to review the material, but keep in mind you are not expected to memorize or master this knowledge.

- What is the difference between objects and arrays?
- How do you access object properties?
- How do primitives and object types differ when you assign them to other variables, or pass them into functions?

Improve on GitHub Report an issue See lesson changelog

View Course Mark Complete Next Lesson