Lesson | Weekend

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / JavaScript Data Types

Text | Cheat sheet

Now that you have the basics of HTML and CSS under your belt, we're going to start learning our first programming language: JavaScript.

In JavaScript, information is separated into different data types. **Data** is simply a piece of information, and its **type** is what the piece of information is categorized as. So, **data types** (also just known as "types") are the different names we give to distinguish and categorize data. Each data type has different **functionality**, meaning it does different things.

For example, some data represents numbers, like your age. While other data represents text, like the title of your favorite book or movie. In JavaScript, number data and text data are given the name "number" and "string" respectively. We'll learn more about numbers and strings in just a few paragraphs!

It's important to note that all programming languages have data types built-in to them. They are building blocks of the functionality in our code.

In JavaScript, data types are separated into two categories: **objects** and **primitives**.

Primitives

Primitives are the simplest elements of JavaScript. There are seven different types of primitives in JavaScript, but during this course, we'll see and use only five of the primitives regularly:

- **Number:** A number is just what it sounds like. It could be `221`, `-4`, or `1.97`. As you might guess, we'll use these a lot.
- **String:** A string is a set of characters enclosed in quotations. `"hello"` is a string. So is `"Good morning, everyone!!!"` And, just to make things a little tricky, `"21"` is a string, not a number. That's because it's enclosed in quotations. Any set of characters that's enclosed in quotations — whether that's letters, numbers, nineteen exclamation points in a row, or some combination thereof — is a string. We will be using strings all of the time.
- **Boolean:** A boolean can be one of two values: `true` or `false`. That's it. We will use these a lot, too.
- **Null:** Null just means nothing. We will sometimes want to assign a `null` value to a thing because we don't know what value it should have yet.
- **Undefined:** This is a fun one... or rather, most of the time it's not so fun when it shows up in our code. It just means something hasn't been defined yet. The reason it's often not so fun is because we'll see that something is `undefined` in our code when we don't want it to be. This means there's a bug we need to fix. Don't worry, we will get a lot of practice with `undefined`, whether we want to or not!

- **Symbol:** A newer primitive that was introduced in 2015. We won't be using this one during the program — and because we don't want to overwhelm you with too much information, we won't go into detail about it here. If you really must know more right this moment, see the Mozilla documentation on Symbols (https://developer.mozilla.org/en-US/docs/Glossary/Symbol).
- **BigInt:** An even newer primitive that was introduced in 2020. We also won't be using this during the program. In brief, a BigInt is a numeric data type like "number". BigInt represents whole numbers (meaning without decimals) that are larger than 2 to the power of 53, or 2^53. You can learn more about BigInt on the Mozilla documentation on BigInt (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Data_structures#bigint_type).

Objects

In the Intermediate JavaScript course, we dedicate an entire section to learning about JavaScript objects. However, JavaScript objects are so important that we'll begin to learn about them now in a conceptual way. Later in this course section, we'll learn about specific tools that are powered by objects. So, what is an object?

Let's first consider what an object is in English: an object is something that can be seen or touched. We could describe an object as a thing, often inanimate. We could also describe an object similar to how we describe a noun: a person, place, or thing. By these definitions, a computer is an object, so let's use a computer as an example.

How do we describe what a computer is? Well, we describe a computer by its features: its screen size, color, hardware, make and model, and its operating system. We also describe a computer by what it does: it turns on and off, it runs an operating system that in

turn runs programs, and it connects to the internet. What an object is in English, and how we describe it, is much the same as what an object is in JavaScript.

In JavaScript, we use objects to represent things and concepts — like animals, political parties, and computers. A JavaScript **object** is simply a container for related data. We can describe an object by listing its data and functionality, the same as describing a computer by listing its features and what it can do.

More generally speaking, this definition of an object is true across computer programming languages. As we'll learn later in this section, our web browsers use objects to represent things like the browser window and the webpages that we create with HTML and CSS.

JavaScript objects can hold multiple types of data and they can be assigned many different types of functionality. The data describes what an object is, and the functionality describes what an object can do. This is in contrast to JavaScript primitives which only represent one piece and type of data. Also, primitives don't *do* anything, meaning we can't give them actions to perform.

For example, an object can be made up of number and string primitives, while a string primitive can only be one thing, a string. Because objects can contain lots of data of any data type, objects are really powerful tools! And this is the main takeaway: objects enable us to structure complex data in an organized way, as well as to house related data in one container.

More Object Analogies

Think of a bag of groceries as an object. It does something (functionality): it holds food and other items so that people can easily transport the goods from the store to home. It also has something (data): perhaps 3 oranges, 2 packages of tofu, 1 pound of green beans, and a block of cheese. All of the separate pieces of data define the bag of groceries (our object), and are also now

connected to each other by being held together in that bag of groceries. This is an example of how an object is a container that holds data, data about what the object is and can do.

Now, what if we had a website that provided online grocery shopping. How would we represent the bag of groceries that the customer leaves the store with in this new online context? In this case, we would create an object to represent a shopping cart that contains all of the different grocery items in it. This shopping cart object might have data about the customer (name, home address, etc.) so that we can connect the customer to the shopping cart. This shopping cart object might also have checkout functionality, so that the customer can buy the groceries. This is another example of how we can use a JavaScript object to represent a concept (a shopping cart) and to package information and functionality related to that concept in one container.

Let's consider another example object: a cat. We can describe what a cat is by itemizing different aspects about it. A cat has:

- eyes and other organs
- colors in its coat
- a name
- a personality
- age
- favorite activities
- perhaps a family and friends?
- how about enemies or prey?

And what can a cat do? A cat can:

- purr
- scratch up the furniture
- eat
- meow
- hunt
- jump, and certainly more!

If we had a video game in which we lived and explored the world as a cat, we'd want to be able to define what the cat is (its data) and what the cat can do (its functionality), so that whoever is playing the video game can have the immersive cat experience! Objects let us do just that. With an object, we can define the complex data structure of a cat in a single container.

Custom Versus Built-In Objects

JavaScript (like most programming languages) provides built-in objects as well as the ability to make custom objects. What's the difference? Functionality that is **built-in** means that an object's category (name or type) and functionality (what it can do) is predefined in the JavaScript language. To use built-in objects, we simply need to learn about their syntax and then we can apply them in our code. Soon, we will learn more about built-in JavaScript objects.

Creating our own **custom** object means we are using JavaScript-provided tools to define our own objects, what it is (name or type) and functionality (what it can do), like the cat or the shopping cart objects we talked about. We will learn more about creating custom objects in Intermediate JavaScript.

Takeaways

If primitives and objects are still fuzzy, that's entirely expected — we'll be working through examples of these data types in upcoming lessons. Our next steps are to learn what JavaScript looks like and what we can do with JavaScript. That means writing our first JavaScript!

Previous (/introduction-to-programming/javascript-and-web-browsers/using-the-devtools-console-for-practice-and-pair-programming)
Next (/introduction-to-programming/javascript-and-web-browsers/arithmetic)

Lesson 6 of 75
Last updated March 24, 2023

disable dark mode

(http://www.epicodus.com)