Introduction to Programming with Javascript

Qitian Liao

Aug 3, 2020

Contents

1 Introduction to Javascript

1.1 What is Javascript?

Last year, millions of learners from our community started with JavaScript. Why? JavaScript is primarily known as the language of most modern web browsers, and its early quirks gave it a bit of a bad reputation. However, the language has continued to evolve and improve. JavaScript is a powerful, flexible, and fast programming language now being used for increasingly complex web development and beyond!

Since JavaScript remains at the core of web development, it's often the first language learned by self-taught coders eager to learn and build. We're excited for what you'll be able to create with the JavaScript foundation you gain here. JavaScript powers the dynamic behavior on most websites, including this one.

In this lesson, you will learn introductory coding concepts including data types and built-in objects—essential knowledge for all aspiring developers. Make sure to take notes and pace yourself. This foundation will set you up for understanding the more complex concepts you'll encounter later.

1.2 Console

The console is a panel that displays important messages, like errors, for developers. Much of the work the computer does with our code is invisible to us by default. If we want to see things appear on our screen, we can print, or log, to our console directly.

In JavaScript, the `console` keyword refers to an object, a collection of data and actions, that we can use in our code. Keywords are words that are built into the JavaScript language, so the computer will recognize them and treats them specially.

One action, or method, that is built into the `console` object is the `.log()` method. When we write `console.log()` what we put inside the parentheses will get printed, or logged, to the console. It is going to be very useful for us to print values to the console, so we can see the work that we are doing.

```
console.log(5);
```

This example logs 5 to the console. The semicolon denotes the end of the line, or statement. Although in JavaScript your code will usually run as intended without a semicolon, we recommend learning the habit of ending each statement with a semicolon so you never leave one out in the few instances when they are required.

You'll see later on that we can use `console.log()` to print different kinds of data.

1.3 Comments

Programming is often highly collaborative. In addition, our own code can quickly become difficult to understand when we return to it. For these reasons, it's often useful to leave notes in our code for other developers or ourselves.

As we write JavaScript, we can write comments in our code that the computer will ignore as our program runs. These comments exist just for human readers. Comments can explain what the code is doing, leave

instructions for developers using the code, or add any other useful annotations. There are two types of code comments in JavaScript:

1. A *single line comment* will comment out a single line and is denoted with two forward slashes `//` preceding it.

   ```
   // Prints 5 to the console
   console.log(5);
   ```

   You can also use a single line comment to comment after a line of code:

   ```
   console.log(5);  // Prints 5
   ```

2. A *multi-line comment* will comment out multiple lines and is denoted with `/*` to begin the comment, and `*/` to end the comment.

   ```
   /*
   This is all commented
   console.log(10);
   None of this is going to run!
   console.log(99);
   */
   ```

   You can also use this syntax to comment something out in the middle of a line of code:

   ```
   console.log(/*IGNORED!*/ 5);  // Still just prints 5
   ```

1.4 Data Types

Data types are the classifications we give to the different kinds of data that we use in programming. In JavaScript, there are seven fundamental data types:

1. *Number*: Any number, including numbers with decimals: `4`, `8`, `1516`, `23.42`.

2. *String*: Any grouping of characters on your keyboard (letters, numbers, spaces, symbols, etc.) surrounded by single quotes: `' ... '` or double quotes `" ... "`. Though we prefer single quotes. Some people like to think of string as a fancy word for text.

3. *Boolean*: This data type only has two possible values— either `true` or `false` (without quotes). It's helpful to think of booleans as on and off switches or as the answers to a "yes" or "no" question.

4. *Null*: This data type represents the intentional absence of a value, and is represented by the keyword `null` (without quotes).

5. *Undefined*: This data type is denoted by the keyword `undefined` (without quotes). It also represents the absence of a value though it has a different use than `null`.

6. *Symbol*: A newer feature to the language, symbols are unique identifiers, useful in more complex coding. No need to worry about these for now.

7. *Object*: Collections of related data.

The first 6 of those types are considered *primitive data types*. They are the most basic data types in the language. *Objects* are more complex, and you will learn much more about them as you progress through JavaScript. At first, seven types may not seem like that many, but soon you will observe the world opens with possibilities once you start leveraging each one. As you learn more about objects, you'll be able to create complex collections of data. But before we do that, let us first get comfortable with strings and numbers!

```
console.log("Location of The Metropolitan: 2301 Durant Ave, Berkeley");
console.log(40);
```

In the example above, we first printed a string. Our string is not just a single word; it includes both capital and lowercase letters, spaces, and punctuation. Next, we printed the number 40, notice we did not use quotes.

1.5 Arithmetic Operators

Basic arithmetic often comes in handy when programming. An *operator* is a character that performs a task in our code. JavaScript has several built-in in *arithmetic operators*, that allow us to perform mathematical calculations on numbers. These include the following operators and their corresponding symbols:

1. Add: `+`
2. Subtract: `-`
3. Multiply: `*`
4. Divide: `/`
5. Remainder: `%`

The first four work how you might guess:

```
console.log(3 + 4); // Prints 7
console.log(5 - 1); // Prints 4
console.log(4 * 2); // Prints 8
console.log(9 / 3); // Prints 3
```

Note that when we `console.log()` the computer will evaluate the expression inside the parentheses and print that result to the console. If we wanted to print the characters `3 + 4`, we would wrap them in quotes and print them as a string.

```
console.log(11 % 3); // Prints 2
console.log(12 % 3); // Prints 0
```

The remainder operator, sometimes called *modulo*, returns the number that remains after the right-hand number divides into the left-hand number as many times as it evenly can: `11 % 3` equals 2 because 3 fits into 11 three times, leaving 2 as the remainder.

1.6 String Concatenation

Operators are not just for numbers! When a `+` operator is used on two strings, it appends the right string to the left string:

```
console.log("hi" + "ya"); // Prints "hiya"
console.log("wo" + "ah"); // Prints "woah"
console.log("I love to " + "code."); // Prints "I love to code."
```

This process of appending one string to another is called *concatenation*. Notice in the third example we had to make sure to include a space at the end of the first string. The computer will join the strings exactly, so we needed to make sure to include the space we wanted between the two strings.

```
console.log("front " + "space"); // Prints "front space"
console.log("back" + " space"); // Prints "back space"
console.log("no" + "space"); // Prints "nospace"
console.log("middle" + " " + "space"); // Prints "middle space"
```

Just like with regular math, we can combine, or chain, our operations to get a final result:

```
console.log("One" + ", " + "two" + ", " + "three!"); // Prints "One, two, three!"
```

1.7 Properties

When you introduce a new piece of data into a JavaScript program, the browser saves it as an instance of the data type. Every string instance has a property called `length` that stores the number of characters in that string. You can retrieve property information by appending the string with a period and the property name:

```
console.log("Hello".length); // Prints 5
```

The `.` is another operator! We call it the dot operator. In the example above, the value saved to the `length` property is retrieved from the instance of the string, `'Hello'`. The program prints `5` to the console, because `Hello` has five characters in it.

1.8 Methods

Remember that methods are actions we can perform. JavaScript provides a number of string methods. We *call*, or use, these methods by appending an instance with:

- a period (the dot operator)
- the name of the method
- opening and closing parentheses

E.g. `'example string'.methodName()`.
The syntax looks familiar. When we use `console.log()` we're calling the `.log()` method on the `console` object. Let us see `console.log()` and some real string methods in action!

```
console.log("hello".toUpperCase()); // Prints "HELLO"
console.log("Hey".startsWith("H")); // Prints true
```

Let's look at each of the lines above:

- On the first line, the `.toUpperCase()` method is called on the string instance `'hello'`. The result is logged to the console. This method returns a string in all capital letters: `'HELLO'`.

- On the second line, the `.startsWith()` method is called on the string instance `'Hey'`. This method also accepts the character `'H'` as an input, or argument, between the parentheses. Since the string `'Hey'` does start with the letter `'H'`, the method returns the boolean `true`.

You can find a list of built-in string methods in the JavaScript documentation.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/String
Developers use documentation as a reference tool. It describes JavaScript's keywords, methods, and syntax.

1.9 Built-in Objects

In addition to `console`, there are other objects built into JavaScript.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects
Down the line, you'll build your own objects, but for now these "built-in" objects are full of useful functionality. For example, if you wanted to perform more complex mathematical operations than arithmetic, JavaScript has the built-in `Math` object.

The great thing about objects is that they have methods! Let's call the `.random()` method from the built-in `Math` object:

```
console.log(Math.random()); // Prints a random number between 0 and 1
```

In the example above, we called the `.random()` method by appending the object name with the dot operator, the name of the method, and opening and closing parentheses. This method returns a random number between 0 and 1.

To generate a random number between 0 and 50, we could multiply this result by 50, like so:

```
Math.random() * 50;
```

The example above will likely evaluate to a decimal. To ensure the answer is a whole number, we can take advantage of another useful `Math` method called `Math.floor()`.

`Math.floor()` takes a decimal number, and rounds down to the nearest whole number. You can use `Math.floor()` to round down a random number like this:

```
Math.floor(Math.random() * 50);
```

In this case:

1. `Math.random` generates a random number between 0 and 1.
2. We then multiply that number by `50`, so now we have a number between 0 and 50.
3. Then, `Math.floor()` rounds the number down to the nearest whole number.

If you wanted to see the number printed to the terminal, you would still need to use a `console.log()` statement:

```
console.log(Math.floor(Math.random() * 50)); // Prints a random whole number between 0
    and 50
```

To see all of the properties and methods on the Math object, take a look at the documentation here.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Math
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Number

1.10 Review

Let's take one more glance at the concepts we just learned:

- Data is printed, or logged, to the console, a panel that displays messages, with `console.log()`.
- We can write single-line comments with `//` and multi-line comments between `/*` and `*/`.
- There are 7 fundamental data types in JavaScript: strings, numbers, booleans, null, undefined, symbol, and object.
- Numbers are any number without quotes: `23.8879`
- Strings are characters wrapped in single or double quotes: `'Sample String'`
- The built-in arithmetic operators include `+`, `-`, `*`, `/`, and `%`.
- Objects, including instances of data types, can have properties, stored information. The properties are denoted with a `.` after the name of the object, for example: `'Hello'.length`.
- Objects, including instances of data types, can have methods which perform actions. Methods are called by appending the object or instance with a period, the method name, and parentheses. For example: `'hello'.toUpperCase()`.
- We can access properties and methods by using the `.`, dot operator.
- Built-in objects, including `Math`, are collections of methods and properties that JavaScript provides.