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 us 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.

2 Variables

2.1 Variables

In programming, a *variable* is a container for a value. You can think of variables as little containers for information that live in a computer's memory. Information stored in variables, such as a username, account number, or even personalized greeting can then be found in memory.

Variables also provide a way of labeling data with a descriptive name, so our programs can be understood more clearly by the reader and ourselves.

In short, variables label and store data in memory. There are only a few things you can do with variables:

1. Create a variable with a descriptive name.
2. Store or update information stored in a variable.
3. Reference or "get" information stored in a variable.

It is important to distinguish that variables are not values; they contain values and represent them with a name. Later, we will cover how to use the `var`, `let`, and `const` keywords to create variables.

2.2 Create a Variable: var

There were a lot of changes introduced in the ES6 version of JavaScript in 2015. One of the biggest changes was two new keywords, `let` and `const`, to create, or declare, variables. Prior to the ES6, programmers could only use the `var` keyword to declare variables.

```
var myName = "Arya";
console.log(myName); // Output: Arya
```

Let's consider the example above:

1. `var`, short for variable, is a JavaScript *keyword* that creates, or *declares*, a new variable.
2. `myName` is the variable's name. Capitalizing in this way is a standard convention in JavaScript called *camel casing*. In camel casing you group words into one, the first word is lowercase, then every word that follows will have its first letter uppercased. (e.g. camelCaseEverything).
3. `=` is the assignment operator. It assigns the value (`'Arya'`) to the variable (`myName`).
4. `'Arya'` is the value assigned (`=`) to the variable `myName`. You can also say that the `myName` variable is initialized with a value of `'Arya'`.
5. After the variable is declared, the string value `'Arya'` is printed to the console by referencing the variable name: `console.log(myName)`.

There are a few general rules for naming variables:

- Variable names cannot start with numbers.
- Variable names are case sensitive, so `myName` and `myname` would be different variables. It is bad practice to create two variables that have the same name using different cases.

- Variable names cannot be the same as *keywords*. For a comprehensive list of keywords check out MDN's keyword documentation.
  https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

Later, we will learn why ES6's `let` and `const` are the preferred variable keywords by many programmers. Because there is still a ton of code written prior to ES6, it is helpful to be familiar with the pre-ES6 `var` keyword. To learn more about `var` and the quirks associated with it, check out the MDN var documentation.
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/var

2.3 Create a Variable: let

As mentioned before, the `let` keyword was introduced in ES6. The `let` keyword signals that the variable can be reassigned a different value. Take a look at the example:

```
let meal = "Enchiladas";
console.log(meal); // Output: Enchiladas
meal = "Burrito";
console.log(meal); // Output: Burrito
```

Another concept that we should be aware of when using `let` (and even `var`) is that we can declare a variable without assigning the variable a value. In such a case, the variable will be automatically initialized with a value of `undefined`:

```
let price;
console.log(price); // Output: undefined
price = 350;
console.log(price); // Output: 350
```

Notice in the example above:

- If we don't assign a value to a variable declared using the `let` keyword, it automatically has a value of `undefined`.
- We can reassign the value of the variable.

2.4 Create a Variable: const

The `const` keyword was also introduced in ES6, and is short for the word constant. Just like with `var` and `let` you can store any value in a `const` variable. The way you declare a `const` variable and assign a value to it follows the same structure as `let` and `var`. Take a look at the following example:

```
const myName = "Gilberto";
console.log(myName); // Output: Gilberto
```

However, a `const` variable cannot be reassigned because it is constant. If you try to reassign a `const` variable, you'll get a `TypeError`.

Constant variables must be assigned a value when declared. If you try to declare a `const` variable without a value, you'll get a `SyntaxError`.

If you're trying to decide between which keyword to use, `let` or `const`, think about whether you'll need to reassign the variable later on. If you do need to reassign the variable use `let`, otherwise, use `const`.

2.5 Mathematical Assignment Operators

Let us consider how we can use variables and math operators to calculate new values and assign them to a variable. Check out the example below:

```
let w = 4;
w = w + 1;
console.log(w); // Output: 5
```

In the example above, we created the variable `w` with the number `4` assigned to it. The following line, `w = w + 1`, increases the value of `w` from `4` to `5`.

Another way we could have reassigned `w` after performing some mathematical operation on it is to use built-in *mathematical assignment operators*. We could re-write the code above to be:

```
let w = 4;
w += 1;
console.log(w); // Output: 5
```

In the second example, we used the `+=` assignment operator to reassign `w`. We're performing the mathematical operation of the first operator `+` using the number to the right, then reassigning `w` to the computed value.

We also have access to other mathematical assignment operators: `-=`, `*=`, and `/=` which work in a similar fashion.

```
let x = 20;
x -= 5; // Can be written as x = x - 5
console.log(x); // Output: 15

let y = 50;
y *= 2; // Can be written as y = y * 2
console.log(y); // Output: 100

let z = 8;
z /= 2; // Can be written as z = z / 2
console.log(z); // Output: 4
```

2.6 The Increment and Decrement Operator

Other mathematical assignment operators include the increment operator (`++`) and decrement operator (`--`). The increment operator will increase the value of the variable by 1. The decrement operator will decrease the value of the variable by 1. For example:

```
let a = 10;
a++;
console.log(a); // Output: 11
```

```
let b = 20;
b--;
console.log(b); // Output: 19
```

Just like the previous mathematical assignment operators (`+=`, `-=`, `*=`, `/=`), the variable's value is updated and assigned as the new value of that variable.

2.7 String Concatenation with Variables

Before we assigned strings to variables. Now, let us go over how to connect, or concatenate, strings in variables. The `+` operator can be used to combine two string values even if those values are being stored in variables:

```
let myPet = "armadillo";
console.log("I own a pet " + myPet + "."); // Output: "I own a pet armadillo."
```

In the example above, we assigned the value 'armadillo' to the myPet variable. On the second line, the `+` operator is used to combine three strings: 'I own a pet', the value saved to myPet, and '.'. We log the result of this concatenation to the console as:

```
I own a pet armadillo.
```

2.8 String Interpolation

In the ES6 version of JavaScript, we can insert, or *interpolate*, variables into strings using *template literals*. Check out the following example where a template literal is used to log strings together: `

```
const myPet = 'armadillo';
console.log(`I own a pet ${myPet}.`); // Output: I own a pet armadillo.
```

Notice that:

- a template literal is wrapped by backticks `` ` `` (this key is usually located on the top of your keyboard, left of the 1 key).
- Inside the template literal, you'll see a placeholder, ${myPet}. The value of myPet is inserted into the template literal.
- When we interpolate `` `I own a pet ${myPet}.` ``, the output we print is the string: 'I own a pet armadillo.'

One of the biggest benefits to using template literals is the readability of the code. Using template literals, you can more easily tell what the new string will be. You also do not have to worry about escaping double quotes or single quotes.

2.9 typeof operator

While writing code, it can be useful to keep track of the data types of the variables in your program. If you need to check the data type of a variable's value, you can use the typeof operator. The typeof operator checks the value to its right and *returns*, or passes back, a string of the data type.

```
const unknown1 = 'foo';
console.log(typeof unknown1); // Output: string

const unknown2 = 10;
console.log(typeof unknown2); // Output: number

const unknown3 = true;
console.log(typeof unknown3); // Output: boolean
```

Let us break down the first example. Since the value unknown1 is 'foo', a string, typeof unknown1 will return 'string'.

2.10 Review Variables

Variables is a powerful concept you will use in all your future programming endeavors. Let us take one more glance at the concepts we just learned:

- Variables hold reusable data in a program and associate it with a name.
- Variables are stored in memory.
- The `var` keyword is used in pre-ES6 versions of JS.
- `let` is the preferred way to declare a variable when it can be reassigned, and `const` is the preferred way to declare a variable with a constant value.
- Variables that have not been initialized store the primitive data type `undefined`.
- Mathematical assignment operators make it easy to calculate a new value and assign it to the same variable.
- The `+` operator is used to concatenate strings including string values held in variables.
- In ES6, template literals use backticks `` ` `` and `${}` to interpolate values into a string.
- The `typeof` keyword returns the data type (as a string) of a value.