{Variables and Primitives.}

Objectives

By the end of this chapter, you should be able to:

- Initialize and assign variables in JavaScript
- Store variables using the `prompt` function
- Write comments in your JavaScript code
- List all of the data types in JavaScript
- Compare and contrast primitive data types with objects

variables

What's a variable, and why would you ever want to use one? The word "variable" may be most familiar to you from math class, when you often use letters like *x* or *y* to represent numbers.

This idea also exists in programming languages. Using variables lets us write code that's easier to read and also easier to change. It's unlikely you'll ever write a significant amount of JavaScript code without using variables, so it's good to get used to them early.

To see why variables are useful, suppose you want to log a set of greetings to the console:

```
console.log ("Hi, Matt!");
console.log ("How are you doing, Matt?");
console.log ("See you later, Matt!");
```

This works fine, but what if we want to change the person's name from "Matt" to something else? We'll have to change three different spots in our text file, and there's a risk that we'll make a typo when fixing any one of these changes. Wouldn't it be better if we could just store a single copy of the name, and use it wherever we want?

Variables give us this ability. So let's write our first variable. In JavaScript, you initialize variables using the `let` keyword. Strictly speaking, using this keyword is not necessary, but in practice it should almost always be there (we'll discuss why a bit later).

Let's rewrite the example above to use a variable:

```
let firstName = "Matt";
console.log("Hi, " + firstName + "!");
console.log("How are you doing, " + firstName + "?");
console.log("See you later, " + firstName + "!");
```

If you enter this code correctly, you should see the same result as before - JavaScript knows that `firstName` corresponds to the name Matt!

There are a few different things going on here, so let's unpack the code a bit. On the first line, we're declaring a variable using the `let` keyword. A variable is just a way for us to save some data in JavaScript. When we typed in `let firstName = "Matt"`, JavaScript stored the word `Matt` in the variable `firstName`. From then on, any time you use `firstName` in the console while this window is still open, you'll see the value that `firstName` has stored.

On the subsequent lines, you'll see there's something else going on too: we're using the `+` operator to combine words made up of characters, or *strings*, together. In JavaScript, when you combine two strings with the `+` operator, you get a new string which is a combination of the two. You can think of this as adding the strings together; a more formal name for it is *concatenation*. For example, if you write the expression `"Hello" + " World"` in the console, you should see that the result is the single string `"Hello World"`.

Let's declare some more variables.

```
let firstName = "Matt";
let lastName = "Lane";
let fullName = firstName + " " + lastName;
```

In all of these examples, we have the keyword `let` in front, followed by the variable name, an assignment operator (the `=` sign), and then the value we want to assign to our variable.

These examples also illustrate a common convention when writing JavaScript: when declaring variables using multiple words, the standard is to capitalize each word after the first word, and otherwise use lower-case letters (e.g. `firstName`, not `firstname`, `first_name`, `FirstName`, or some other variation). This casing convention is called camel case, and while your JavaScript code will work just fine if you don't abide by this convention, it's good to get in the habit of camel casing your variables.

The `prompt` function

Let's revisit our earlier example:

```
let firstName = "Matt";
console.log("Hi, " + firstName + "!");
console.log("How are you doing, " + firstName + "?");
console.log("See you later, " + firstName + "!");
```

Since we've used a variable, if we want to change the name, now we only have to do it in one place. That's great! Try changing the value stored in `firstName` from "Matt" to your name.

Now suppose we wanted to ask the user for their name. In JavaScript, you can ask the user to provide some information using the `prompt` function. You won't use this function very often (there are better ways to get information from a user), but when you're first learning it's a helpful tool.

When you use the `prompt` function, a pop-up window will appear on the page and ask the user to fill in a text box. You can then store what the user types into a variable. Try it out with this modification to our example:

```
let firstName = prompt("What is your first name?");
// Now firstName should correspond to whatever the user typed!
console.log("Hi, " + firstName + "!");
console.log("How are you doing, " + firstName + "?");
```

```
console.log("See you later, " + firstName + "!");
```

One last thing. See that line in there that starts with two slashes? That indicates a *comment*. Javascript ignores comments; they are there purely to make notes about the code and generally help with its readability. You can create single-line comments with `//`; if you want a multiline comment, here's a haiku that shows how it's done:

```
/* this is the start of
a multiline comment, and
this is the ending. */
```

Primitive Data Types in JavaScript

So far, we've only used variables to store *strings*. But JavaScript can work with other types of data as well. Let's take a look at the primitive data types (you don't need to worry about what primitive means just yet).

JavaScript has 6 primitive data types, but we'll only talk about 5 of them. Here's what they look like:

- string - `let greeting = "hello";`
- number - `let favoriteNum = 33;`
- boolean - `let isAwesome = true;`
- undefined - `let foo;` or `let setToUndefined = undefined;`
- null - `let empty = null;`

JavaScript is known as a "weakly" typed language. What this means is that when you create variables and assign them to values, you do not have to specify the type of data you are working with. In statically (or strongly) typed languages, like Java and C++, you do need to specify the type.

Now let's look at data types a little more.

string

As we saw above, a string is a set of characters enclosed in quotes. A string can be defined using double quotes:

```
let greeting = "Hello Whiskey";
```

or using single quotes:

```
let greeeting = 'Hello World';
```

So what is the difference between the two ways of initializing a string? Well, first of all, if you want quotes in your string, it's nice to have another option to start and end the string:

```
let phrase = 'Matt said, "I haven\'t been to Chile", the other day.';
```

What would happen if you try to use double quotes to create the previous string instead of using single quotes? Try it in your console.

Also notice that there is a backslash before the single quote in haven't. The backslash is called an escape character and it tells JavaScript that the single quote in the string should not be used to end the string. Try removing the backslash from the string and seeing what happens in your JavaScript console.

number

JavaScript numbers can be positive:

```
let num = 5;
```

negative:

```
let num = -25;
```

decimal numbers:

```
let piApproximation = 3.14159265;
```

and we can also do all of the math expressions you'd expect:

```
let x = 1 + 3;
let a = 4.5;
let b = 5.9;
let c = Math.sqrt(a * a + b * b);
let studentTeacherRatio = 4 / 1;
```

If you need to do any kind of calculation in the application you're building, chances are you'll be relying heavily on the number type.

boolean

A boolean type can only be in one of two states, true or false. In other words:

```
let pizzaIsGood = true;
let pizzaIsBad = false;
```

Boolean types are a very useful tool for controlling our program. For example, if a user is signed in, you might want to show them a link to update their profile; but if a user is not logged in, you'd probably just want to show them a sign-in link. This sort of behavior, where the code that gets executed is conditioned on something else, happens all the time in programming. We'll learn more about how to deal with these situations in the next chapter.

undefined

Any variable that is created in JavaScript that is not assigned a value is undefined:

```
let noValue;  // The value here will be undefined
```

You can also explicitly set a variable to undefined:

```
let favoriteFood = "Candy";
// Changed your mind
let favoriteFood = undefined;
```

null

Null is not the same as undefined. It signifies an intentional absence of data.

```
let secondEmailAddress = null;
```

It is important to remember that `null` and `undefined` are different types in JavaScript! This can be a confusing feature of JavaScript, even for people who know other programming languages. The distinction can seem somewhat arbitrary when you're first learning the language, but as you get more comfortable the distinction will become clearer. For now, you don't need to worry about it too much; if you're interested, you can read some discussion about the differences here and here.

Figuring out a variable's type in JavaScript

In JavaScript, we have a keyword called `typeof` that returns the type of the variable. While this seems pretty fool-proof, there are some quirks that we should be aware of. In the Chrome console, let's type out each one of these:

- `typeof "";` - "string"
- `typeof 5;` - "number"
- `typeof false;` - "boolean"
- `typeof Symbol();` - "symbol"
- `typeof undefined;` - "undefined"
- `typeof null;` // hmmm, this is not what we expect, it returns "object"!

Converting between types

Very often you'll need to convert a value from one type to another. For example, maybe you want to do some math on a couple of numbers, but you get the numbers from a form and they have a value of string. In some cases JavaScript will change types

implicitly, in a process that's often referred to as (implicit) type coercion. We'll discuss the details of this later.

For now, though, let's take a look at some ways to explicitly change the type of a value. Here are a few examples:

Converting to a string: `toString`

The `toString` method will convert any value which is not undefined or null into a string. Here are a couple of examples:

```
let num = 5;
let bool = true;

num.toString(); // "5";
bool.toString(); // "true";
```

Converting to a number

There are several ways you can convert a value to a number. One way is to parse the number, using `parseInt` or `parseFloat`: each function will look at a string from left to right and try to make sense of the characters it sees as numbers. Here are some examples:

```
parseInt("2"); // 2
parseFloat("2"); // 2
parseInt("3.14"); // 3
parseFloat("3.14"); // 3.14
parseInt("2.3alkweflakwe"); // 2
parseFloat("2.3alkweflakwe"); // 2.3
parseInt("w2.3alkweflakwe"); // NaN (not a number)
parseFloat("w2.3alkweflakwe"); // NaN (not a number)
```

Both of these functions parse from left to right. If they see numbers, they'll continue parsing until they find a non-numerical character, and will either return an integer or a float, depending on which function was used. That's why `parseInt("2.3alkweflakwe")` returns a valid integer, but `parseInt("w2.3alkweflakwe")` does not.

Somewhat related is the `Number` function. This doesn't parse, it simply tries to convert the entire string directly to a number. Sometimes this can lead to slightly different behavior compared to the parsing functions:

```
Number("2"); // 2
Number("3.14"); // 3.14
Number("2.3alkweflakwe"); // NaN
Number("w2.3alkweflakwe"); // NaN
```

A nice shorthand for this conversion is the unary operator `+`:

```
+"2"; // 2
+"3.14"; // 3.14
+"2.3alkweflakwe"; // NaN
+"w2.3alkweflakwe"; // NaN
```

NaN

When you try to perform a numeric operation converting to a number with a value that can not be converted, you will get a special value called `NaN` (short for Not a Number).

You commonly get back `NaN` when JavaScript does not know how to convert something to a number - here are some examples:

```
    parseInt("taco") // NaN
    Number("no way!") // NaN
```

If you ever need to check if an expression evaluates to `NaN` you can use the handy `isNaN` function which returns `true` if the expression can not be converted to a number.

```
    let validConversion = Number("2")
    console.log("Is validConversion NaN?", isNaN(validConversion))

    // Is validConversion NaN? false

    let invalidConversion = Number("hello world")
    console.log("Is invalidConversion NaN?", isNaN(invalidConversion))
```

```
    // Is invalidConversion NaN? true
```

Converting to a boolean: `!!`

We'll talk about the `!` operator on booleans in a later chapter. For now, all you need to know is that `!!` will convert a value to its boolean equivalent. Here are a couple of examples:

```
let greeting = "hi";
let nothing = 0;

!!greeting; // true
!!nothing; // false
```

A very quick note on `var` and `const`

There are two other keywords that we can use to declare variables in JavaScript, `var` and `const`. You might see `var` in some older tutorials; it is much more rarely used in modern JavaScript. `const` is an alternative to `let` but it does not allow you to redeclare, which makes it useful for variables that you do not want changed. This is a very simple introduction, and you'll learn much more about these differences later, but for now, just use `let` to declare variables.

Exercises

1. Create the following variables
    - `name`, which is a string set to your current name
    - `dayOfBirth`, which is a number set to the day of your birth month
2. See what happens when you have multiple variables of the same name. Which one takes precedence?
3. Write some JavaScript that prompt's the user for their favorite color. Once the user has submitted a favorite color, log that color to the console along with a friendly message.
4. Create a string that contains both single quotes and double quotes.

5. What is the difference between null and undefined?
6. What is `NaN` in JavaScript? What is the `typeof NaN`?
7. You can declare a variable by typing `let thing;` What is the value of `thing`?