Lesson Weekend

Introduction to Programming (/introduction-to-programming) / JavaScript and Web Browsers (/introduction-to-programming/javascript-and-web-browsers) / Review of JavaScript Conventions

Text Cheat sheet

Over the course of learning our first JavaScript, we also covered a few conventions. A **convention** is the way something is usually done, and a convention in computer programming has to do with how we write our code. Conventions are important to computer programming because they make it so that everyone is writing code that can be easily read and understood by anyone. For example, with HTML and CSS, we learned that indentation and spacing is an important convention to follow.

While we could write a CSS rule like this:

```
h1{color:blue;background-color:orange;}
```

The convention is to write CSS with specific indentation and spacing so that it is easier to read:

```
h1 {
  color: blue;
  background-color: orange;
}
```

Following conventions makes collaboration much easier, because we can all share expectations around style and code structure. It's important to know that convention can be set by individual organizations, like a software company, as well as commonly held across user groups, like developers who program in JavaScript. For example, a company might decide to follow a specific naming convention for project folders and files so that their programmers can better collaborate. All this is to say that conventions can be broadly held or specific to an organization. If you are curious for another example, check out MDN's guidelines for contributors on writing JavaScript examples (https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript#use_expanded_syntax).

Let's review the JavaScript conventions that we've learned about thus far! There will be more that we learn about (like indentation and spacing) in upcoming lessons and weeks.

JavaScript Conventions

Use `let` or `const` to Declare Variables

Variable declaration is what we call the process of creating new variables, like `const myVariable;`. For `let` and `var` we can optionally initialize a variable with a value, like `var greeting = "Howdy!";` or `let myCat = "Waffles";`. For constant variables declared with `const`, we always need to initialize these with a value, because they can only be assigned a value once, and never be reassigned.

The convention with declaring variables is to use `let` and `const`, and never `var`. This has to do with scope, which we'll learn about more in coming lessons. For now, remember that `var` is outmoded, and that `let` and `const` was added to JavaScript in 2015 to improve the language. We should always use `let` and `const`.

Use Lower camelCase in Variable Names

Variable names should always be written in lower camelCase (https://developer.mozilla.org/en-US/docs/MDN/Guidelines/Code_guidelines/JavaScript#variable_naming), where there are no spaces in the name, the first letter is lowercase, and the first letter of every subsequent word in the variable name is uppercase. Here are some examples:

```
const myCodeSchool = "Epicodus";
let myFavoriteFloweringShrub = "Daphne";
```

Also note that variable names need to begin with a letter, and they are case sensitive. This means, `myNumber` is a *different* variable than `myNUMBER`.

Use Descriptive Names for Variables

Variable names should be descriptive of what they represent. For example, if you saw a variable that is named `num1`, what data type do you think the variable's value is? I would guess a number.

Remember that this is a subjective process and there's no one right choice. The main goal is that your code is easy to understand to anyone who comes across it. To that end, it's best to avoid the following:

- Variable names like `a`, `b`. Instead choose something like `inputA` or `inputB`.

- Variable names that are super long, like `myFavoriteFloweringShrub`. While this isn't an awful practice, because our variable name is very descriptive, it can be a drag to type this out over and over. Instead choose a name like `favShrub`.

Use Semicolons after Statements and Expressions

We add semicolons at the end of statements and expressions. A **statement** (https://developer.mozilla.org/en-US/docs/Glossary/Statement) is a piece of code that tells our computer to do something. A computer program is essentially made up of a list of statements. Statements can contain operators, keywords, and expressions. Semicolons are added at the end of a statement to indicate where it ends. For example, the semicolon in the following code separates two statements:

```
> const favTree = "Douglas Fir"; let favNumParity = "odd";
> favTree;
"Douglas Fir"
> favNumParity;
"odd"
```

However, the above isn't typical — usually we declare variables on separate lines, like this:

```
> const favTree = "Douglas Fir";
> let favNumParity = "odd";
> favTree;
"Douglas Fir"
> favNumParity;
"odd"
```

Statements can be made up of expressions. An **expression** (https://developer.mozilla.org/en-US/docs/Web/JavaScript/Guide/Expressions_and_Operators#expressions) is a piece of code that evaluates to a value. The difference between statements and expressions can get technical and detailed. However in simple terms we can distinguish the two like so: an expression is a piece of code that evaluates to a value, and a statement is used more generally to describe code that performs actions, whether or not they evaluate to a value.

Here are examples of expressions we've used so far:

```
> favNumParity.toUpperCase();
"ODD";
> parseInt("3");
3
> true;
true
> typeof true;
"boolean"
> "hello world";
"hellow world"
> 3;
3
> 1 + 3;
4
> 10 > 9;
true
> "boat" === "boat";
true
> "boat" !== "boat";
false
```

Here is an example of statements:

```
> let cat;
> const favTree = "Douglas Fir";
> let favNumParity = "odd";
> const firstNum = 1; const secondNum = 2;
```

A case when we don't add a semicolon is at the end of a function declaration:

```
function add(num1, num2) {
  return num1 + num2;
}  // <-- no semicolon!
```

It's worth reiterating that semicolons are a tricky subject in JavaScript. When we execute our code, JavaScript interprets it on-the-fly into code that our machines can read. In the process, it automatically adds semicolons between sections of our code. *However*, there are certain situations where JavaScript incorrectly adds a semicolon, which breaks our JavaScript code — and these situations are obscure for beginners.

To deal with this situation, some developers add semicolons themselves to be thorough, while others only add them when needed because JavaScript will do it automatically. In order to be in the latter camp, you need to know those gotcha situations. For that reason, **at Epicodus, we follow the convention of adding semicolons at the end of (most of) our statements and expressions**.

We will continue to revisit this topic in upcoming lessons, including the difference between statements and expressions. Because this is a tricky topic, we don't expect you to get semicolons right all of the time. Do your best, and this convention will make more sense in time.

Previous (/introduction-to-programming/javascript-and-web-browsers/another-look-at-javascript-objects)
Next (/introduction-to-programming/javascript-and-web-browsers/journal-2)

Lesson 22 of 75
Last updated more than 3 months ago.

disable dark mode

(http://www.epicodus.com)

© 2023 Epicodus (http://www.epicodus.com/), Inc.