JavaScript Syntax & Structure

- JavaScript
- Monday October 19th, 2020

As with any language, programming languages are defined by sets of rules. The rules (or syntax) are what we follow when we write our code, which form the logical structure of our programs.

This article marks the beginning of a series I'll be writing on learning the JavaScript fundamentals. Be sure to check back often!

Let's dive right in with the building blocks of JavaScript. We'll be looking at values (literals & variables), camel casing, unicode, semi colons, indentation, white spacing, commenting, case sensitivity, keywords, operators and expressions!

By taking the time to learn the fundamentals, we'll be well on our way toward building more functional and readable code!

JavaScript Values

In JavaScript there are two types of values: Fixed values (or **literals**) and Variable values (**variables**).

Literals

Literals are defined as values that are written in our code, such as numbers, strings, booleans (true or false), as well as object and array literals (don't worry too much about these just yet). Some examples include:

```
10            // A number (can be a decimal eg. 10.5)
'Boat'        // A string (can be in double "" or single ''
quotes)
true          // A boolean (true or false)
['a', 'b']                          // An array
{color: 'blue', shape: 'Circle'}    // An object
```

Note: I'll be looking at data types in detail in the next article in this series, stay tuned!

Variables

Variables are named values which store data. In JavaScript we declare our variables with the `var`, `let` or `const` keywords, and we assign values with the equal sign `=`.

For example, `key` is defined as a variable. Which is assigned the value `abc123`:

```
let key;
key = abc123;
```

When to use `var`? Don't. It really should only be used when working with legacy code. Its the old pre-ES6 syntax.

When to use `let`? Use it if your variable needs to be updated within the program (it can be reassigned).

When to use `const`? Use it if your variable holds a constant value. It must be assigned at the time of declaration and it cannot be reassigned.

Camel Case

What if our variable name consists of more than one word? For example, how would we declare a variable we wish to name "first name"?

Could we use **hyphens?** e.g. `first-name`Nope, `-`'s are reserved for subtractions in JavaScript.

What about **underscores?** e.g. `first_name` We could, but it has a tendency to make our code look messy and confusing.

The solution? **camel case**! e.g. `firstName`. The first word is lower-case, the first letter of any subsequent words are upper-case. This is the convention within the community.

Note: It's quite acceptable however, to name your `const` variables in upper-case with underscores e.g. `const DEFAULT_PLAYBACK_SPEED = 1;` This would make it clear to others that the value is fixed. Otherwise just stick with camelCase!

Unicode

JavaScript uses the unicode character set. Unicode covers just about all of the characters, punctuations, and symbols that there are! Check out the complete reference. This is great as we can write our names in any language, and we could even use emojis as variable names (because why not? □□♂).

Semicolons

JavaScript programs are made up of a number of instructions known as statements. Such as:

```
// These are all examples of JavaScript statements:

let a = 1000;

a = b + c;

const time = Date.now();
```

JavaScript statements often end in a semicolon `;`.

However, **semicolons aren't always mandatory!** JavaScript does not have any issues if you don't use them.

```
// Still perfectly valid!

let a = 1000

a = b + c

const time = Date.now()
```

There are however, some situations where they are mandatory. For instance when we use a `for` loop, like so:

```
for (i = 0; i < .length; i++) {
  // code to execute
}
```

When using a block statement however, semicolons are not to be included after the curly braces, for example:

```
if (name == "Samantha") {
  // code
}                            // <- no ';'
//or,
function people(name) {
  // code
}                            // <- no ';'
```

If we're using an object however, such as:

```
const person = {
  firstName: "Samantha",
  lastName: "Doe",
  age: 30,
  eyeColor: "blue"
};                           // the ';' is mandatory
```

Then our `;`'s are required!

Over time you'll start to memorize where semicolons can and can't be used. Until then it's wise to use them at the end of all statements (with the exception of block statements!)

Plus it really is a common convention is to use them regardless, it's considered good practice as its reduces the probability of errors.

Note: Once you really get going with JavaScript, start using a linter such as ESLint. It'll automatically find syntax errors in your code and make life much easier!

Indentation

In theory we could write an entire JavaScript program on one line. However this would be just about impossible to read and maintain. Which is why we use lines and indentation. Lets use a conditional statement as an example:

```
if (loginSuccessful === 1) {
  // code to run if true
} else {
  // code to run if false
}
```

Here we can see that any code inside a block is indented. In this case its our comment code `// code to run if true` and then `// code to run if false.` You can choose to indent your lines with either a few spaces (2 or 4 are common) or a tab. It's entirely your choice, the main thing is to be consistent!

If we are nesting our code, we'd indent further like so:

```
if (loginAttempts < 5){
  if (loginAttempts < 3){
    alert("< 3");
  } else {
    alert("between 3 and 5");
  }
} else {
  if (loginAttempts > 10){
    alert("> 10");
  } else {
    alert("between 5 and 10");
  }
}
```

By applying indentation you'll have much cleaner, more maintainable and easier to read code!

White Space

JavaScript only requires one space between keywords, names and identifiers, otherwise any white space is completely ignored. This means that as far as the language is concerned, there is no difference between the following statements:

```
const visitedCities="Melbourne, "+"Montreal, "+"Marrakech";
const visitedCities = "Melbourne, " + "Montreal, " +
"Marrakech";
```

I’m sure you’ll find the second line much more readable. And another example:

```
let x=1*y;
let x = 1 * y;
```

Again, the second line is much easier to read and debug! So feel free to space out your code in a way that makes sense! For that reason, this is also an acceptable use of white space:

```
const cityName          = "Melbourne";
const cityPopulation    = 5000001;
const cityAirport       = "MEL";
```

Commenting

A comment is un-executable code. They’re useful for providing an explanation of some code within a program. And also to ‘comment out’ a section of code to prevent execution — often used when testing an alternative piece of code.

There are two types of comments in JavaScript:

```
// Comment goes here

/* Comment goes here */
```

The first syntax is a single line comment. The comment goes to the right of the `//`

The second a multi-line comment. The comment goes in between the asterisks `/* here */`

A longer multi-line comment:

```
/* This is
a comment spanning
multiple lines */
```

Identifiers

In JavaScript, the name of a variable, function, or property is known as an identifier. Identifiers may consist of letters, numbers, `$` and `_`. No other symbols are permitted, and they cannot begin with a number.

```
// Valid □

Name
name
NAME
_name
Name1
```

```
$name

// Invalid □

1name
n@me
name!
```

Case Sensitivity

JavaScript is case sensitive! An identifier named `test` is different from `Test`.

The following will throw an error:

```
function test() {
  alert("This is a test!");
}
function showAlert() {
  Test();                    // error! test(); is correct
}
```

In order to ensure that our code is readable, it's best to try to vary our names, so no identifiers are found looking too similar.

Reserved Words

There are a number of words within JavaScript that may not be used as identifiers. Those words are reserved by the language, as they have built-in functionality. Such as:

```
break, do, instanceof, typeof, case, else, new, var, catch,
finally, return, void, continue, for, switch, while, debugger,
function, this, with, default, if, throw, delete, in, try,
class, enum, extends, super, const, export, import.
```

See the full list of reserved keywords.

You certainly don't need to commit these words to memory! If you get any strange syntax errors pointing to a variable, you can check the list and change the name.

JavaScript Operators

Arithmetical operators `+` `-` `*` and `/` are primarily used when performing calculations within JavaScript, such as:

```
(2 + 2) * 100
```

The assignment operator `=` is used to assign values to our variables:

```
let a, b, c;
a = 1;
b = 2;
c = 3;
```

JavaScript Expressions

An expression is when we combine values, variables and operators to compute a new value (known as a evaluation). Such as:

```
10 * 10    // Evaluates to 100

let x = 5
x * 10     // Evaluates to 50

const firstName = "Samantha";
const lastName = "Doe";
firstName + " " + lastName;    // Evaluates to: Samantha Doe
```

Wrapping up

And there we go! This article aimed to provide a general overview of the basic syntax and structure of JavaScript. We’ve looked at many of the common conventions, however, remember you can be somewhat flexible — especially when working in collaborative environments with their own particular standards.

Syntax and structuring both have an important role the play for both the functionality of our programs as well as for code readability and maintainability.