Learn web development

Storing the information you need — Variables

Previous | Overview: First steps | Next

After reading the last couple of articles you should now know what JavaScript is, what it can do for you, how you use it alongside other web technologies, and what its main features look like from a high level. In this article we will get down to the real basics, looking at how to work with most basic building blocks of JavaScript — Variables.

Prerequisites:	Basic computer literacy, a basic understanding of HTML and CSS, an understanding of what JavaScript is.
Objective:	To gain familiarity with the basics of JavaScript variables.

Tools you need

Throughout this article, you'll be asked to type in lines of code to test your understanding of the content. If you are using a desktop browser, the best place to type your sample code is your browser's JavaScript console (see What are browser developer tools for more information on how to access this tool).

However, we have also provided a simple JavaScript console embedded in the page below for you to enter this code into, in case you are not using a browser with a JavaScript console easily available, or find an in-page console more comfortable.

What is a variable?

A variable is a container for a value, like a number we might use in a sum, or a string that we might use

as part of a sentence. But one special thing about variables is that their contained values can change. Let's look at a simple example:

```
<button>Press me</button>
```

```
var button = document.querySelector('button');

button.onclick = function() {
  var name = prompt('What is your name?');
  alert('Hello ' + name + ', nice to see you!');
}
```

Press me

In this example pressing the button runs a couple of lines of code. The first line pops a box up on the screen that asks the reader to enter their name, and then stores the value in a variable. The second line displays a welcome message that includes their name, taken from the variable value.

To understand why this is so useful, let's think about how we'd write this example without using a variable. It would end up looking something like this:

```
var name = prompt('What is your name?');

if (name === 'Adam') {
  alert('Hello Adam, nice to see you!');
} else if (name === 'Alan') {
  alert('Hello Alan, nice to see you!');
} else if (name === 'Bella') {
  alert('Hello Bella, nice to see you!');
} else if (name === 'Bianca') {
  alert('Hello Bianca, nice to see you!');
} else if (name === 'Chris') {
  alert('Hello Chris, nice to see you!');
}

// ... and so on ...
```

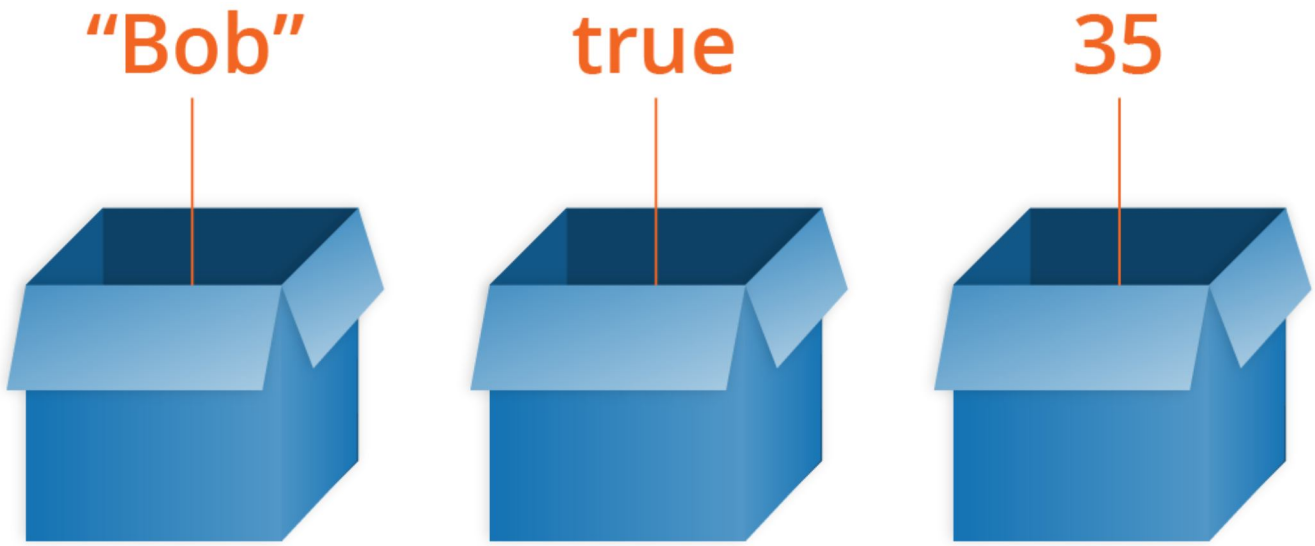
You may not fully understand the syntax we are using (yet!), but you should be able to get the idea — if we didn't have variables available, we'd have to implement a giant code block that checked what the

entered name was, and then display the appropriate message for that name. This is obviously really inefficient (the code is a lot bigger, even for only four choices), and it just wouldn't work — you couldn't possibly store all possible choices.

Variables just make sense, and as you learn more about JavaScript they will start to become second nature.

Another special thing about variables is that they can contain just about anything — not just strings and numbers. Variables can also contain complex data and even entire functions to do amazing things. You'll learn more about this as you go along.

Note that we say variables contain values. This is an important distinction to make. Variables aren't the values themselves; they are containers for values. You can think of them being like little cardboard boxes that you can store things in.



Declaring a variable

To use a variable you've first got to create it — more accurately, we call this declaring the variable. To do this, we type the keyword var followed by the name you want to call your variable:

```
var myName;
var myAge;
```

Here we're creating two variables called `myName` and `myAge`. Try typing these lines in now in your web browser's console, or in the below console (You can open this console in a separate tab or window if you'd prefer that). After that, try creating a variable (or two) with your own name choices.

```
>
```

> **Note**: In JavaScript, all code instructions should end with a semi-colon (`;`) — your code may work correctly for single lines, but probably won't when you are writing multiple lines of code together. Try to get into the habit of including it.

You can test whether these values now exist in the execution environment by typing just the variable's name, e.g.

```
myName;
myAge;
```

They currently have no value; they are empty containers. When you enter the variable names, you should either get a value of `undefined` returned. If they don't exist, you'll get an `error` message — try typing in

```
scoobyDoo;
```

> **Note**: Don't confuse a variable that exists but has no value defined with a variable that doesn't exist at all — they are very different things.

Initializing a variable

Once you've defined a variable, you can initialize it with a value. You do this by typing the variable name, followed by an equals sign (=), followed by the value you want to give it. For example:

```
myName = 'Chris';
myAge = 37;
```

Try going back to the console now and typing in these lines. You should see the value you've assigned to the variable returned in the console to confirm it, in each case. Again, you can return your variable values by simply typing their name into the console — try these again:

```
myName;
myAge;
```

You can declare and initialize a variable at the same time, like this:

```
var myName = 'Chris';
```

This is probably what you'll do most of the time, as it is quicker than doing the two actions on two separate lines.

Note: If you write a multiline JavaScript program that declares and initializes a variable, you can actually declare it after you initialize it and it will still work. This is because variable declarations are generally done first before the rest of the code is executed. This is called **hoisting** — read var hoisting for more detail on the subject.

Updating a variable

Once a variable has had a value assigned, you can update that value by simply giving it a different value. Try entering the following lines into your console:

```
myName = 'Bob';
myAge = 40;
```

An aside on variable naming rules

You can call a variable pretty much anything you like, but there are limitations. Generally you should stick to just using Latin characters (0-9, a-z, A-Z) and the underscore character.

- You shouldn't use other characters because they may cause errors or be hard to understand by an international audience.
- Don't use underscores at the start of variable names — this is used in certain JavaScript constructs to mean specific things, so may get confusing.

- Don't use numbers at the start of variables. This isn't allowed and will cause an error.
- A safe convention to stick to is so-called "lower camel case", where you stick together multiple words, using lower case for the whole first word and then capitalize subsequent words. We've been using this for our variable names in the article so far.
- Make variable names intuitive, so they describe the data they contain. Don't just use single letters/numbers, or big long phrases.
- Variables are case sensitive — so `myage` is a different variable to `myAge`.
- One last point — you also need to avoid using JavaScript reserved words as your variable names — by this, we mean the words that make up the actual syntax of JavaScript! So you can't use words like `var`, `function`, `let`, and `for` as variable names. Browsers will recognize them as different code items, and so you'll get errors.

> **Note**: You can find a fairly complete list of reserved keywords to avoid at Lexical grammar — keywords.

Good name examples:

```
age
myAge
init
initialColor
finalOutputValue
audio1
audio2
```

Bad name examples:

```
1
a
_12
myage
MYAGE
var
Document
skjfndskjfnbdskjfb
thisisareallylongstupidvariablenameman
```

Try creating a few more variables now, with the above guidance in mind.

Variable types

There are a few different types of data we can store in variables (data types). In this section we'll describe these in brief, then in future articles you'll learn about them in more detail.

So far we've looked at the first two, but there are others.

Numbers

You can store numbers in variables, be those whole numbers like 30 (also called integers) or decimal numbers like 2.456 (also called floats or floating point numbers). JavaScript doesn't have different data types for different number types, like some programming languages do. When you give a variable a number value, you don't include quotes:

```
1 var myAge = 17;
```

Strings

Strings are pieces of text. When you give a variable a string value, you need to wrap it in single or double quote marks, otherwise JavaScript will try to intepret it as another variable name.

```
1 var dolphinGoodbye = 'So long and thanks for all the fish';
```

Booleans

Booleans are true/false values — they can have two values, `true` or `false`. These are generally used to test a condition, after which code is run as appropriate. So for example, a simple case would be:

```
1 var iAmAlive = true;
```

Whereas in reality it would be used more like this:

```
1 var test = 6 < 3;
```

This is using the "less than" operator (<) to test whether 6 is less than 3. As you might expect, it will return `false`, because 6 is not less than 3! You will learn a lot more about such operators later on in the course.

Arrays

An array is a single object that contains multiple values enclosed in square brackets and separated by commas. Try entering the following lines into your console:

```
var myNameArray = ['Chris', 'Bob', 'Jim'];
var myNumberArray = [10,15,40];
```

Once these arrays are defined, you can access their individual values using syntax like the following. Try these lines:

```
myNameArray[0]; // should return 'Chris'
myNumberArray[2]; // should return 40
```

The square brackets here contain an index value that specifies the position of the value you want returned. You might have noticed that computers count from 0, instead of 1 like us humans do.

You'll learn a lot more about arrays in a future article.

Objects

In programming, an object is a structure of code that models a real life object. You can have a simple object that represents a car park and contains information about its width and length, or you could have an object that represents a person, and contains data about their name, height, weight, what language they speak, how to say hello to them, and more.

Try entering the following line into you console:

```
var dog = { name : 'Spot', breed : 'Dalmatian' };
```

To retrieve the information stored in the object, you can use the following syntax:

```
dog.name
```

We won't be looking at objects any more for now — you can learn more about those in a future module.

Loose typing

JavaScript is a "loosely typed language", which means that, unlike some other languages, you don't need to specify what data type a variable will contain (e.g. number? string?).

For example if you declare a variable and give it a value with quotes round it, the browser will know it is a string:

```
var myString = 'Hello';
```

It will still be a string, even if it contains numbers, so be careful:

```
var myNumber = '500'; // oops, this is still a string
typeof(myNumber);
myNumber = 500; // much better — now this is a number
typeof(myNumber)
```

Try entering the four lines above into your console one by one, and see what the results are (don't type in the comments). You'll notice that we are using a special function called `typeof()` — this returns the data type of the variable you pass to it. The first time it is called in the above sequence, it should return `string`, as at that point the `myNumber` variable contains a string, `'500'`. Have a look and see what it returns the second time you call it.

Summary

By now you should know a reasonable amount about JavaScript variables and how to create them. In the next article we'll focus on numbers in more detail, looking at how to do basic math in JavaScript.

Previous | Overview: First steps | Next