Riverside JS Workshop
JavaScript Skills Ladder, Lesson 1.2

Working with JavaScript Variables

Lesson Objectives:

When you complete this skills ladder lesson, you will be able to:

- *declare variables.*
- *enter values into variables.*
- *change variable types.*
- *use standard naming conventions for your variables.*

What is a Variable?

A variable is a container for a value. You've actually experimented with variables a bit already in this course. It looked something like this:

```
var age = 10;
```

In this example `var` is a *keyword*, a special word that tells JavaScript you're declaring a variable. The variable is age, and the value you're putting into the age container is the number 10. When you write `var age = 10`, you're actually doing *two* things: you're *declaring* the variable (with `var`) and you're *initializing* it, by setting its value to 10. You can also just declare a variable without initializing it, like this:

```
var age;
```

In this example, the variable's value is undefined until you give it a value, so it's more likely that you'll declare and initialize a variable at the same time.

You only need the `var` keyword when you *declare* a variable. *Declaring a variable* means you're telling JavaScript about it for the first time, and usually giving that variable a value.

Note You may have seen scripts that don't use the **var** keyword to declare variables (the **var** keyword is omitted). JavaScript is pretty forgiving and will let you do this, but it's not a good idea because of certain assumptions that are made about the variable. We'll come back to this later; for now, just get in the habit of using the **var** keyword whenever you declare a variable!

Values

So, what kinds of values can you put into variables?

Numbers

You can put *numbers* into variables. Try this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title>Storing Numbers</title>
  <meta charset="utf-8">
  <script>
      var x = 3;
      alert("x = " + x);

      var y = 3.0;
      alert("y = " + y);

      var z = 3.1;
      alert("z = " + z);

      var big = 3.0E12;
      alert("big = " + big);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *var_number.html***,** and load the page in your browser. Make note of the values you see. Are they what you expected?

In JavaScript, there are two kinds of numbers: integers and floating point. You probably recall from your school days that integers are positive and negative whole numbers (...-2, -1, 0, 1, 2, 3...). To declare a variable x with the value 3, you would type:

OBSERVE:

```
var x = 3;
```

To declare a variable y with the value -3, you would type:

OBSERVE:

```
var y = -3;
```

Floating point numbers are real numbers (that is, a number that can contain a fractional part), like 1.1, -3.145, and 36034.55. To declare these numbers, you would type, for example:

OBSERVE:

```
var x = 1.1;
var y = -3.145;
var z = 36034.55;
```

Floating point numbers don't always have to contain a decimal point. In fact, in JavaScript, most of the time you don't have to don't need to be concerned with whether a number has a decimal point; you can just use it as a number. There are some exceptions to that rule though; we'll go over them later.

So, suppose you want to represent a number like 3,000,000,000,000 (that's 3 trillion). You could write out the whole long thing like this:

OBSERVE:

```
var big = 3000000000000;
```

Or, you could use *scientific notation*, and write it like this:

OBSERVE:

```
var big = 3.0E12;
```

The E12 tells JavaScript to multiply 3.0 by 1 followed by 12 zeros (or 1 trillion). You probably won't use this notation too often, but once in a while it can come in handy!

Note Make sure you leave out the commas when you're writing a number value in JavaScript!

Strings

Another type of value you can put into JavaScript variables is a *string*. A string is a sequence of characters with quotation marks around it, like this: **"Fido"**, or this:

OBSERVE:

```
var dogLikes = "Fido likes to go for long walks";
```

A string may contain many characters, including spaces, numbers, and even other quotes.

Suppose you want to write a string containing another string in JavaScript. For instance, you want to include the string: "And the President said, "I cannot tell a lie." "; Including quotation marks in your string presents a challenge. (We are not challenged by punctuation, by the way. We're just using two sets of quotation marks for the sake of our example, sacrificing sound grammar for technology. Sometimes it has to be this way.) How could you do it? Try experimenting with strings and quotation marks. Try typing a string that contains double quotation marks:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> A famous quote </title>
  <meta charset="utf-8">
  <script>
      var quote = "And the President said, "I cannot tell a lie." ";
      alert(quote);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *var_string.html*, and load the page in your browser. Do you see the alert? Probably not, because the code causes an error.

When JavaScript sees the first quotation mark before the word *And*, it recognizes it as the beginning of a string. When JavaScript gets to the quotation mark in front of the word *I*, it thinks it's the *end* of the string. The word *I*, and all the words following it, aren't valid JavaScript, so those words cause an error.

Note When you display the string value in a variable, you won't see double quotation marks; they are only present to tell JavaScript where the string begins and ends.

To place quotations inside a string, you need to *escape* them, by preceding them with a special character, the backslash (\):

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> A famous quote </title>
  <meta charset="utf-8">
  <script>
    var quote = "And the President said \"I cannot tell a lie.\" ";
    alert(quote);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. Now your alert works fine.

The backslash tells JavaScript to *escape* from its normal interpretive process and treat the double quotation marks that follow as a punctuation mark rather than as a string delimiter.

You can also use an apostrophe within a string, like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> A famous quote </title>
  <meta charset="utf-8">
  <script>
    var quote = "It's no exaggeration to say that the undecideds could go one way or another.";
    alert(quote);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. You'll see the string in the alert.

We're using an apostrophe in the word *It's*. We don't have to use a backslash in front of it because it won't cause any problems. JavaScript knows that the string doesn't end until it sees another set of quotation marks.

One last bit of information to keep in mind regarding strings. If you need to type a really long string, you should write it all on one big long line, without any carriage returns. Give it a try. Type the code below as shown:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> A famous quote </title>
  <meta charset="utf-8">
  <script>
      var quote = "Thomas Jefferson once said, \"We should never judge
a
                   president by his age, only by his works.\"";
      alert(quote);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser.

JavaScript expects a string to start and end on the same line. If you split your string into two separate lines like this, JavaScript will get confused! You'll see shortly how you can split up strings over multiple lines. For now, just keep typing your strings in one continuous line.

Booleans

The other basic value you can put into a JavaScript variable is a *Boolean*. A Boolean value is true or false. You can write it like this:

OBSERVE:

```
var isBig = true;
var isOld = false;
```

We don't include quotation marks around the values *true* and *false*.

You'll learn about Boolean values in greater detail later, but for now, try this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Booleans </title>
  <meta charset="utf-8">
  <script>
    var age = 14;
    var isYoung = age < 21;
    alert("isYoung = " + isYoung);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *var_boolean.html*. Notice that instead of setting the variable `isYoung` directly to a Boolean true or false value, we set it to the result of an *expression*. Can you guess what this code does? Try it out.

What happens if you change the value of `age` to 22, or if you change the number you're comparing `age` to, from 21 to 13? Do some experimenting with Booleans!

Types

Even though we talked about the types of the values (number, string, or Boolean) we put into our variables, there is nothing in the variable declaration that indicates which type the variable should be. An unassigned variable like `var x` could contain any of those types.

In other languages you must indicate the type of the variable in order to tell the computer which kind of value should be in that variable. This is not required in JavaScript. This characteristic is known as *dynamic typing*; it means that JavaScript is pretty good at guessing the type from the value. It also means that if you really wanted to, you could *change* the type of a variable halfway through a program, like this:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Types </title>
  <meta charset="utf-8">
  <script>
```

```
      var x = 3;
      alert(x);
      x = "dog";
      alert(x);
  </script>
</head>
<body>
</body>
</html>
```

We initialize the variable x to the number 3, and then change its value to a string, "dog." Save it in your work folder as *var_boolean.html* and load the page in your browser. Does it work?

This is perfectly legal in JavaScript, but it's not a particularly good idea. Why? Well, suppose you have a big program that first initializes `x = 9;`. Then, you write code that expects x to be a number, like, `x = x + 10`. Then, you set `x = "W"` and *forget* your earlier `x = x + 10` code. If you happen to loop back to that code again and try to add 10 to "W," JavaScript will not be pleased!

Sticking with one type for a variable is a good idea and will help reduce the number of bugs you need to solve! It will also make it a lot easier to read your code later when you're trying to remember how or why you did something.

Which leads us to our next topic...

Naming Conventions for variables

We've talked a lot about variables and the kinds of values you can store in them, but we haven't talked much yet about the kinds of names you can use for your variables. We've used names like `x` and `y`, as well as `quote` and `isBig`. All of these are either letters or strings, which brings us to the first of three rules you need to know to create good variable names:

Rule 1: Begin your variable names with a letter.

All variable names must begin with a letter. You're also allowed to use "_" and "$" to begin a variable name, but because those characters are often used by variables in JavaScript libraries, it's best not to use them to begin a variable name unless you really know what you're doing. We'll just stick with letters. Try these:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Variable Names </title>
  <meta charset="utf-8">
  <script>
      var myName = "Scott";
      var isRaining = true;
      var _libVar = 0;
      var $importantVariable = 99;

      alert(myName);
      alert(isRaining);
      alert(_libVar);
      alert($importantVariable);
  </script>
</head>
<body>
</body>
</html>
```

Save it in your work folder as *var_names.html***,** and load the page in your browser. You'll see all the alerts with their correct values. What happens if you type these variable names instead? Try this code:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Variable Names </title>
  <meta charset="utf-8">
  <script>
      var 5year = 5;
      var %interest = 9.89;
      var ~test = false;

      alert(5year);
      alert(%interest);
      alert(~test);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. Do you see any alerts? You probably don't, because these are *not* valid variable names!

Rule 2: After the first character, you can use letters, numbers, underscores, and dollar signs.

Once you've got the first character of your variable set, then you can use letters, numbers, "_," and "$" as much as you like. Variable names can be as long as you like, and while you don't want to go overboard with super long names, it's a good idea to have *reasonably* long, descriptive names for variables. Descriptive names make it easier to understand your code. Try these valid names:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Variable Names </title>
  <meta charset="utf-8">
  <script>
      var amount$ = 9.99;
      var tax35 = 35;
      var keep_real = true;

      alert(amount$);
      alert(tax35);
      alert(keep_real);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. You'll see alerts. As long as you stick with letters, numbers, "_," and "$" for your variable names, you're good to go. Try using these variable names:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Variable Names </title>
  <meta charset="utf-8">
  <script>
      var twoand1/2men = "crazy";
      var last-name = "Gray";
      var two parts = true;
```

```
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. All of these variables have characters that aren't allowed in variable names because those characters have other meanings.

Note There's a common convention of using underscores (_) to represent spaces in variable names, so "two_parts" might be a better choice for that last one.

Also, variable names are *case sensitive*, which means that a variable named `tax35` is a different variable from one named TAX35.

Rule 3: Avoid using JavaScript's reserved keywords and built-in function names.

We've already encountered one keyword in this lesson: `var`. It would be mighty confusing to both you and JavaScript if you wrote `var var!` Try it:

CODE TO TYPE:

```
<!doctype html>
<html lang="en">
<head>
  <title> Variable Names </title>
  <meta charset="utf-8">
  <script>
      var var;
      alert(var);
  </script>
</head>
<body>
</body>
</html>
```

Save it, and load the page in your browser. You don't see anything, right? JavaScript will definitely complain if you do this, so stay away from JavaScript's keywords.

You'll get familiar with them fairly quickly and most of them make sense once you get to know the language. You can find a comprehensive list of all the reserved words online; here are a few:

- boolean
- break
- catch
- char
- class
- continue
- default
- delete
- do
- double
- else
- false
- final
- float
- for
- function
- goto
- if
- interface
- long
- namespace
- new
- null
- return
- super
- this
- throw
- true
- try
- var
- while
- with

We'll encounter several of these reserved words throughout the course.

Creating a Good Name

As we've already mentioned, it's a great idea to use *meaningful* names so when you go back to read your code later, you'll know what you meant!

Another good tip for making variable names, is to use *camel case*. Camel case means that the first letter of each word (except the first one) is capitalized within the variable name, so the capital letters look like the humps of a camel. Like camelCase (a one-hump camel) or myFavoriteBook (a two-hump camel). By convention, variable names usually begin with a lower case letter except in some special circumstances.

Finally, remember to start your variables with a letter, rather than "_" or "$" unless you have a very good reason (you'll know when you do). This helps you to avoid variable name clashes with JavaScript libraries.

You've got the low-down on variables now, so go make some! Try your hand at creating good names, and use **alert** to test them out. Have fun! In the next lesson you'll learn all about how to *do* more with variables, creating expressions and statements. See you there!

More about the material in this lesson

The Riverside JS Workshop would like to acknowledge the generosity of O'Reilly Media, Inc. for making this material available to us through the Creative Commons License. We would also like to acknowledge the great work of Elisabeth Robson who authored this content. She is one of our favorites. Elisabeth has written a number of *Head First* programming books for web developers. We recommend them highly. You can browse her work here.