JavaScript™

Third Edition

ABSOLUTE BEGINNER'S GUIDE

Kirupa Chinnathambi

Pearson

Contents at a Glance

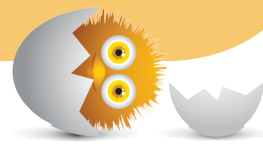
Reader Services

Register your copy of ***JavaScript™ Absolute Beginner's Guide***, **Third Edition** at informit.com for convenient access to downloads, updates, and corrections as they become available. To start the registration process, go to informit.com/register and log in or create an account*. Enter the product ISBN, **9780137959167**, and click Submit. Once the process is complete, you will find any available bonus content under Registered Products.

*Be sure to check the box that you would like to hear from us in order to receive exclusive discounts on future editions of this product.

IN THIS CHAPTER

- Learn how to use values to store data
- Organize your code with variables
- Get a brief look at variable naming conventions



2

VALUES AND VARIABLES

In JavaScript, every piece of data we provide or use is considered to contain a value. In our example from the previous chapter, we might think of **hello, world!** as just some words we pass in to the `alert` function:

```
alert("hello, world!");
```

To JavaScript, however, these words have a specific representation under the covers. They are considered **values**. We may not have thought much about that when we were typing those words, but when we are in JavaScript Country, every piece of data we touch is considered a value.

Now, why is knowing this important? It is important because we will be working with values a whole lot. Working with them in a way that doesn't drive you insane is a good thing. There are just two things we need to simplify our life working with values:

- We need to identify them easily.
- We need to reuse them throughout our application without unnecessarily duplicating them.

Those two things are provided by what we are going to be spending the rest of our time on: **variables**. Let's learn all about them here.

Using Variables

A variable is an identifier for a value. Instead of typing **hello, world!**, every time we want to use that phrase in our application, we can assign that phrase to a variable and use that variable whenever we need to use **hello, world!** again. This will make more sense in a few moments—I promise!

There are several ways to use variables. For most cases, the best way is by relying on the `let` keyword followed by the name you want to give your variable, like so:

```
let myText
```

In this line of code, we declare a variable called `myText`. Right now, our variable has simply been **declared**. It doesn't contain anything of value. It is merely an empty shell.

Let's fix that by **initializing** our variable to a value like, say, **hello, world!**, as shown here:

```
let myText = "hello, world!";
```

At this point, when this code runs, our `myText` variable will have the value **hello, world!** associated with it. Let's put all of this together as part of a full example. If you still have **hello_world.htm** open from earlier, replace the contents of your

<script> tag with the following, or you can create a new HTML file and add the following contents into it:

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8"">
  <title>An Interesting Title Goes Here</title>

  <style>

  </style>
</head>

<body>
  <script>
    let myText = "hello, world!";
    alert(myText);
  </script>
</body>

</html>
```


Notice that we are no longer passing in the **hello, world!** text to the `alert` function directly. Instead, we are now passing in the variable name `myText` instead. The end result is the same. When this script runs, an alert with **hello, world!** will be shown. What this change allows us to do is have one place in our code where **hello, world!** is being specified. If we wanted to change **hello, world!** to **The dog ate my homework!**, all we would have to do is just make one change to the phrase specified by the `myText` variable:

```
let myText = "The dog ate my homework!";
alert(myText);
```

Throughout our code, wherever we reference the `myText` variable, we will now see the new text appear. Although this is hard to imagine as being useful for something as simple as what we have right now, for larger applications, the convenience of having just one location where we can make a change that gets reflected everywhere is a major time-saver. You'll see more less-trivial cases of the value variables provide in subsequent examples.

More Variable Stuff

What we learned in the previous section will take us far in life. At least, it will in the parts of our life that involve getting familiar with JavaScript. We won't dive too much further into variables here—we'll do all of that as part of future chapters where the code is more complex and the importance of variables is more obvious. With that said, there are a few odds and ends we should cover before calling it a day.

Naming Variables

We have a lot of freedom in naming our variables however we see fit. Ignoring what names we should give things based on philosophical/cultural/stylistic preferences, from a technical point of view, JavaScript is very lenient on what characters can go into a variable name.

This leniency isn't infinite, so we should keep the following points in mind when naming our variables:

- Variables can be as short as one character, or they can be as long as you want—think thousands and thousands of characters.
- Variables can start with a letter, underscore, or dollar sign ($). They can't start with a number.
- Outside of the first character, our variables can be made up of any combination of letters, underscores, numbers, and $ characters. We can also mix and match lowercase and uppercase letters to our heart's content.
- Spaces are not allowed.

Here are some examples of valid variable names:

```
let myText;
let $;
let r8;
let _counter;
let $field;
```

```
let thisIsALongVariableName_butItCouldBeLonger;
let __$abc;
let OldSchoolNamingScheme;
```

To see if a variable name is valid, check out the really awesome and simple **JavaScript Variable Name Validator** at **https://bit.ly/namevalidator**.

Outside of valid names, there are other things to focus on as well, such as naming conventions, how many people commonly name variables, and other things you identify with a name. We will touch on these items in other chapters.

More on Declaring and Initializing Variables

One of the things you will learn about JavaScript is that it is a very forgiving and easy-to-work-with language.

Declaring a Variable Is Optional

For example, we don't have to use the `let` keyword to declare a variable. We could just do something like the following:

```
myText = "hello, world!";
alert(myText);
```

Notice the `myText` variable is being used without formally being declared with the `let` keyword. While not recommended, this is completely fine. The end result is that we have a variable called `myText`. The only thing is that by declaring a variable this way, we are declaring it globally. Don't worry if the last sentence makes no sense. We'll look at what *globally* means when talking about variable scope later.

Declaring and Initializing on Separate Lines Is Cool

There is one more thing to call out, and that is this: The declaration and initialization of a variable do not have to be part of the same statement. We can break them up across multiple statements:

```
let myText;
myText = "hello, world!";
alert(myText);
```

In practice, we will find ourselves breaking up our declaration and initialization of variables all the time.

Changing Variable Values and the const Keyword

Lastly, we can change the value of a variable declared via `let` to whatever we want, whenever we want:

```
let myText;
myText = "hello, world!";
myText = 99;
myText = 4 * 10;
myText = true;
myText = undefined;
alert(myText);
```

If you have experience working with languages that are more strict and don't allow variables to store a variety of data types, this leniency is one of the features people both love and hate about JavaScript. With that said, JavaScript does provide a way for you to restrict the value of a variable from being changed after you initialize it. That restriction comes in the form of the `const` keyword, which we can declare and initialize our variables with:

```
const siteURL = "https://www.google.com";
alert(siteURL);
```

By relying on `const`, we can't change the value of `siteURL` to something other than **https://www.google.com**. JavaScript will complain if we try to do that. There are some gotchas with using the `const` keyword, but it does a great job overall in preventing accidental modifications of a variable. We'll cover those pesky gotchas in bits and pieces when the time is right.

TIP Jump Ahead—Variable Scoping

Now that you know how to declare and initialize variables, a very important topic is that of **visibility**. You need to know when and where a variable you declared can actually be used in your code. The catch-all phrase for this is **variable scope**. If you are curious to know more about it, you can jump ahead and read Chapter 8, "Variable Scope."

THE ABSOLUTE MINIMUM

Values store data, and variables act as an easy way to refer to that data. There are a lot of interesting details about values, but those are details you do not need to learn right now. Just know that JavaScript enables you to represent a variety of values such as text and numbers without a lot of fuss.

To make your values more memorable and reusable, you declare variables. You declare variables using the `let` keyword and a **variable name**. If you want to initialize the variable to a default value, you follow all of that up with an equal sign (=) and the value you want to initialize your variable with.

? Ask a question: **https://forum.kirupa.com**

✔ Practice by building real apps: **https://bit.ly/coding_exercises**

Errors/known issues: **https://bit.ly/javascript_errata**