

**Hafiz Hamza Ahmed Siddiqui**

When you want to visit JavaScrip book official website to download in pdf way please [**click here**](http://wccftech.com/wp-content/uploads/2014/10/JavaScript.pdf) !!

**copyright © 2020 by Hafiz Hamza Ahmed Siddiqui**

Chapters

* How I propose to cut your effort in half

by using technology

* ALERTS
* Variables for Strings
* Variables for Numbers
* Variable Names Legal and Illegal
* Math Expressions: familiar operators

**How I propose to**

**cut your effort in half**

**by using technology.**

When you set out to learn anything as complicated as JavaScript, you sign up for some heavy cognitive lifting. If I had to guess, I'd say the whole project of teaching yourself a language burns at least a large garden-cart load of brain glucose. But here's what you may not realize: When you teach yourself, your cognitive load doubles. Yes, all the information is right there in the book if the author has done a good job. But learning a language entails far more than reading some information. You need to commit the information to memory, which requires some kind of plan. You need to practice. How are you going to structure that? And you need some way to correct yourself when you go off-course. Since a book isn't the best way to help you with these tasks, most authors don't even try. Which means all the work of designing a learning path for yourself is left to you. And this do-it yourself meta-learning, this struggle with the question of how to master what the book is telling you, takes more effort than the learning itself.

Traditionally, a live instructor bridges the gap between reading and learning. Taking a comprehensive course or working one-on-one with a mentor is still the best way to learn JavaScript if you have the time and can afford it. But, as long as many people prefer to learn on their own, why not use the latest technology as a substitute teacher? Let the book lay out the principles. Then use an interactive program for memorization, practice, and correction. When the computer gets into the act, you'll learn twice as fast, with half the effort. It's a smarter way to learn JavaScript. It's a smarter way to learn anything.

And as long as we're embracing new technology, why not use all the tech we can get our hands on to optimize the book? Old technology—i.e. the paper book—has severe limitations from an instructional point of view. New technology—i.e. the **ebook**—is the way to go, for many reasons. Here are a few:

Color is a marvelous information tool. That's why they use it for traffic lights. But printing color on paper multiplies the cost. Thanks to killer setup charges, printing this single word —color—in a print-on-demand book adds thirty dollars to the retail price. So color is usually out, or else the book is priced as a luxury item. With an **ebook**, color is free. Paper itself is expensive, so there usually isn't room to do everything the author would like to do. A full discussion of fine points? Forget it. Extra help for the rough spots? Can't afford it. Hundreds of examples? Better delete some. But no such limitation applies to an **ebook**. What do an extra hundred digital pages cost? Usually nothing

Here, then, is how I propose to use current technology to help you learn JavaScript in half the time, with half the effort.

* **Cognitive portion control.** Testing showed me that when they're doing hard-core learning, even strong-minded people get tired faster than I would have expected. You may be able to read a novel for two hours at a stretch, but when you're studying something new and complicated, it's a whole different ballgame. My testing revealed that studying new material for about ten minutes is the limit, before most learners start to fade. But here's the good news: Even when you've entered the fatigue zone after ten minutes of studying, you've still got the mental wherewithal to practice for up to thirty minutes. Practice that's designed correctly takes less effort than studying, yet teaches you more. Reading a little and practicing a lot is the fastest way to learn.
* **500 coding examples that cover every aspect of what you're learning**. Examples make concepts easy to grasp and focus your attention on the key material covered in each chapter. Color cues embedded in the code help you commit rules to memory. Did I go overboard and put in more examples that you need? Well, if things get too easy for you, just skip some them.
* **Tested on naive users.** The book includes many rounds of revisions based on feedback from programming beginners. It includes extra-help discussions to clarify concepts that proved to be stumbling blocks during testing. Among the testers: my technophobe wife, who discovered that, with good instruction, she could code—and was surprised to find that she enjoyed it. For that matter, I got a few surprises myself. Some things that are simple to me turned out not to be not so simple to some readers.
* **Free interactive coding exercises paired with each chapter—1,750 of them in all.** They're the feature that testers say helps them the most. No surprise there. According to the New York Times, psychologists "have shown that taking a test—say, writing down all you can remember from a studied prose passage—can deepen the memory of that passage better than further study." I would venture that this goes double when you're learning to code. After reading each chapter, go online and practice everything you learned. Each chapter ends with a link to its accompanying online exercises. Find an index of all the exercises at [**http://www.ASmarterWayToLearn.com/js/.**](http://www.ASmarterWayToLearn.com/js/)

**ALERTS**

An alert is a box that pops up to give the user a message. Here's code for an alert that displays the message "Thanks for your input!"

**alert("Thanks for your input!");**

alert is a keyword—that is, a word that has special meaning for JavaScript. It means, "Display, in an alert box, the message that follows." Note that alert isn't capitalized. If you capitalize it, the script will stop. The parentheses are a special requirement of JavaScript, one that you'll soon get used to. You'll be typing parens over and over again, in all kinds of JavaScript statements. In coding, the quoted text "Thanks for your input!" is called a text string or simply a string. The name makes sense: it's a string of characters enclosed in quotes. Outside the coding world, we'd just call it a quotation. Note that the opening parenthesis is jammed up against the keyword, and the opening quotation mark is hugging the opening parenthesis. Since JavaScript ignores spaces (except in text strings), you could write...

**alert("Thanks for your input!");**

But I want you to know the style conventions of JavaScript, so I'll ask you to always omit spaces before and after parentheses.

In English, a careful writer ends every declarative sentence with a period. In scripting, a careful coder ends every statement with a semicolon. (Sometimes complex, paragraph-like statements end with a curly bracket instead of a semicolon. That's something I'll cover in a later chapter.) A semicolon isn't always necessary, but it's easier to end every statement with a semicolon, rather than stop to figure out whether you need one. In this training, I'll ask you to end every statement (that doesn't end with a curly bracket) with a semicolon.

|  |
| --- |
| **CODING ALTERNATIVES TO BE AWARE OF** |

* Some coders write window.alert instead of, simply, alert. This is a highly formal but perfectly correct way to write it. Most coders prefer the short form. We'll stick to the short form in this training.

Find the interactive coding exercises for this chapter at: [**http://www.ASmarterWayToLearn.com/js/1.htm**](http://www.ASmarterWayToLearn.com/js/1.htm)

**Variables for Strings**

Please memorize the following facts.

* My name is Mark.
* My nationality is U.S.

Now that you've memorized my name and nationality, I won't have to repeat them, literally, again. If I say to you, "You probably know other people who have my name," you'll know I'm referring to "Mark."

f I ask you whether my nationality is the same as yours, I won't have to ask, "Is your nationality the same as U.S.?" I'll ask, "Is your nationality the same as my nationality?" You'll remember that when I say "my nationality," I'm referring to "U.S.", and you'll compare your nationality to "U.S.", even though I haven't said "U.S." explicitly

A variable is created when you write var (for variable) followed by the name that you choose to give it. It takes on a particular value when you assign the value to it. This is a JavaScript statement that creates the variable name and assigns the value "Mark" to it.

**var** name = "Mark";

Now the variable name refers to the text string "Mark". Note that it was my choice to call it name. I could have called it myName, xyz, lol, or something else. It's up to me how to name my variables.

With the string "Mark" assigned to the variable name, my JavaScript code doesn't have to specify "Mark" again. Whenever JavaScript encounters name, JavaScript knows that it's a variable that refers to "Mark". For example, if you ask JavaScript to print name, it remembers the value that name refers to, and prints...

The value that a variable refers to can change. Let's get back to the original examples, the facts I asked you to memorize. These facts can change, and if they do, the terms my name and my nationality will refer to new values. I could go to court and change my name to Ace. Then my name is no longer Mark. If I want you to address me correctly, I'll have to tell you that my name is now Ace. After I tell you that, you'll know that my name doesn't refer to the value it…

ew value (Ace). If I transfer my nationality to U.K., my nationality is no longer U.S. It's U.K. If I want you to know my nationality, I'll have to tell you that it is now U.K. After I tell you that, you'll know that my

**Variables for Numbers**

A string isn't the only thing you can assign to a variable. You can also assign a number

**var** weight = 150;

Having coded the statement above, whenever you write weight in your code, JavaScript knows you mean 150. You can use this variable in math calculations. If you ask JavaScript to add 25 to weight...

...JavaScript, remembering that weight refers to 150, will come up with the sum 175. Unlike a string, a number is not enclosed in quotes. That's how JavaScript knows it's a number that it can do math on and not a text string, like a ZIP code, that it handles as text.

But then, since it's not enclosed in quotes, how does JavaScript know it's not a variable? Well, because a number, or any combination of characters starting with a number, can't be used as a variable name. If it's a number, JavaScript rejects it as a variable. So it must be a number

If you enclose a number in quotation marks, it's a string. JavaScript can't do addition on it. It can do addition only on numbers not enclosed in quotes.

Now look at this code.

**var** originalNum = 23;

**var** newNum = originalNum + 7;

In the second statement in the code above, JavaScript substitutes the number 23 when it encounters the variable originalNum. It adds 7 to 23. And it assigns the result, 30, to the variable newNum.

JavaScript can also handle an expression made up of nothing but variables.

**For example...**

**var** originalNum = 23;

**var** numToBeAdded = 7;

**var** newNum = originalNum + numToBeAdded;

**Variable Names Legal and Illegal**

You've already learned three rules about naming a variable: You can't enclose it in quotation marks. The name can't be a number or start with a number. It can't be any of JavaScript's keywords—the special words that act as programming instructions, like alert and var.

**Here are the rest of the rules:**

* A variable name can't contain any spaces.
* A variable name can contain only letters, numbers, dollar signs, and underscores.
* Though a variable name can't be any of JavaScript's keywords, it can contain keywords. For example, userAlert and myVar are legal.
* Capital letters are fine, but be careful. Variable names are case sensitive. A rose is not a Rose. If you assign the string "Floribundas" to the variable rose, and then ask
* JavaScript for the value assigned to Rose, you'll come up empty. I teach the camelCase naming convention. Why "camelCase"? Because
* there is a hump or two (or three) in the middle if the name is formed by more than one word. A camelCase name begins in lower case.
* If there's more than one word in the name, each subsequent word gets an initial cap, creating a hump. If you form a variable name with only one word, like response, there's no hump. It's a camel that's out of food. Please adopt the camelCase convention. It'll make your variable names more readable, and you'll be less likely to get variable names mixed up**.**

**Examples:**

**userResponse**

**userResponseTime**

**userResponseTimeLimit**

**response**

**Math expressions:**

**Familiar operators**

Wherever you can use a number, you can use a math expression. For example, you're familiar with this kind of statement.

**var** popularNumber = 4;

**But you can also write this.**

**var** popularNumber = **2 + 2;**

**You can also write:**

**alert(2 + 2);**

is displays the messge "4" in an alert box. When it sees a math expression, JavaScript always does the math and delivers the result. Here's a statement that subtracts 24 from 12, assigning -12 to the variable.

**var** popularNumber = 12 - 24;

**This one assigns the product of 3 times 12, 36, to the variable.**

**var** popularNumber = 3 \* 12;

In this one, the number 10 is assigned to a variable. Then 1 is added to the variable, and the sum, 210, is assigned to a second variable. As usual, you can mix variables and numbers.

**var** num = 10;

**var** popularNumber = num + 200;

**You can also use nothing but variables**

**var** num = 10;

**var** anotherNum = 1;

**va**r popularNumber = num + anotherNum;

****

Figure 1

****

**Signature And Word Art:**

Signature And Word Art:



**E**

**quation:**

***Made By : Hafiz Hamza Ahmed Siddiqui***