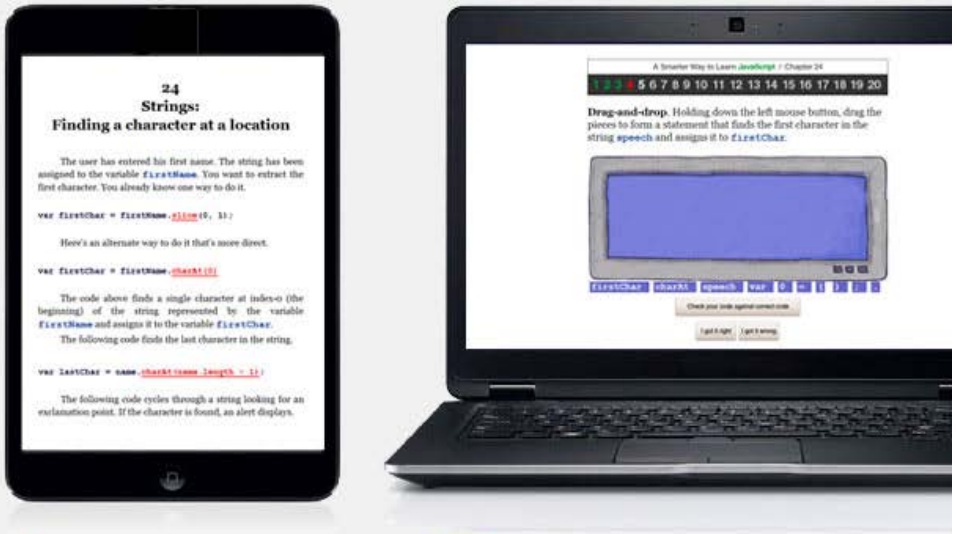
A Smarter Way to Learn JavaScript

The new approach that uses technology to cut your effort in half



1 Read a 10-minute chapter of this book to get each concept.

2 Code for 20 minutes at ASmarterWayToLearn.com to own the skill. (It's free.)

Mark Myers

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.

When a book is published traditionally, it may take up to a year for the manuscript to get into print. This means there isn't time for extensive testing on the target audience, or for the revisions that testing would inevitably suggest. And once the book is in print, it's a big, expensive deal to issue revised editions. Publishers put it off as long as possible. Reader feedback usually doesn't lead to improvements for years. An ebook can go from manuscript to book in a day, leaving lots of time for testing and revision. After it's published, new editions with improvements based on reader feedback can come out as often as the author likes, at no cost.

With all this going for them, is there any doubt that all the best instructional books are going to be ebooks? And would anyone deny that the most helpful thing an author can do for

you, in addition to publishing a good book electronically, is to take on the whole teaching job, not just part of it, by adding interactivity to help you with memorization, practice, and correction?

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. Rewriting ensued.

- **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/.
- **Live coding experience.** In scripting, the best reward is seeing your code run flawlessly. Most practice sessions include live coding exercises that let you see your scripts execute in the browser.

How to use this book

This isn't a book quite like any you've ever owned before, so a brief user manual might be helpful.

- **Study, practice, then rest.** If you're intent on mastering the fundamentals of JavaScript, as opposed to just getting a feel for the language, work with this book and the online exercises in a 15-to-30-minute session, then take a break. Study a chapter for 5 to 10 minutes. Immediately go online at http://www.ASmarterWayToLearn.com/js and code for 10 to 20 minutes, practicing the lesson until you've coded

everything correctly. Then go for a walk.

- **Use the largest, most colorful screen available.** This book can be read on small phone screens and monochrome readers, but you'll be happier if things appear in color on a larger screen. I use color as an important teaching tool, so if you're reading in black-and-white, you're sacrificing some of the extra teaching value of a full-color ebook. Colored elements do show up as a lighter shade on some black-and-white screens, and on those devices the effect isn't entirely lost, but full color is better. As for reading on a larger screen — the book includes more than 2,000 lines of example code. Small screens break long lines of code into awkward, arbitrary segments, jumbling the formatting. While still decipherable, the code becomes harder to read. If you don't have a mobile device that's ideal for this book, consider installing the free Kindle reading app on your laptop.
- **If you're reading on a mobile device, go horizontal**. For some reason, I resist doing this on my iPad unless I'm watching a video. But even I, Vern Vertical, put my tablet into horizontal mode to proof this book. So please: starting with Chapter 1, do yourself a favor and rotate your tablet, reader, or phone to give yourself a longer line of text. It'll help prevent the unpleasant code jumble mentioned above.
- **Do the coding exercises on a physical keyboard.** A mobile device can be ideal for reading, but it's no way to code. Very, very few Web developers would attempt to do their work on a phone. The same thing goes for *learning* to code. Theoretically, most of the interactive exercises *could* be done on a mobile device. But the idea seems so perverse that I've disabled online practice on tablets, readers, and phones. Read the book on your mobile device if you like. But practice on your laptop.
- **If you have an authority problem, try to get over it.** When you start doing the exercises, you'll find that I can be

a pain about insisting that you get every little detail right. For example, if you indent a line one space instead of two spaces, the program monitoring your work will tell you the code isn't correct, even though it would still run perfectly. Do I insist on having everything just so because I'm a control freak? No, it's because I have to place a limit on harmless maverick behavior in order to automate the exercises. If I were to grant you as much freedom as you might like, creating the algorithms that check your work would be, for me, a project of driverless-car proportions. Besides, learning to write code with fastidious precision helps you learn to pay close attention to details, a fundamental requirement for coding in any language.

- **Subscribe, temporarily, to my formatting biases.** Current code formatting is like seventeenth-century spelling. Everyone does it his own way. There are no universally accepted standards. But the algorithms that check your work when you do the interactive exercises need standards. They can't grant you the latitude that a human teacher could, because, let's face it, they aren't that bright. So I've had to settle on certain conventions. All of the conventions I teach are embraced by a large segment of the coding community, so you'll be in good company. But that doesn't mean you'll be married to my formatting biases forever. When you begin coding projects, you'll soon develop your own opinions or join an organization that has a stylebook. Until then, I'll ask you to make your code look like my code.
- **Email me with any problems or questions.** The book and exercises have been tested on many learners, but haven't been tested on you. If you hit a snag, if you're just curious about something, or if I've found some way to give you fits, email me at mark@ASmarterWayToLearn.com. I'll be happy to hear from you. I'll reply promptly. And, with

your help, I'll probably learn something that improves the next edition.

1
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.

- 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.
- In the example above, some coders would use single rather than double quotation marks. This is legal, as long as it's a matching pair. But in a case like this, I'll ask you to use double quotation marks.

Find the interactive coding exercises for this chapter at:
http://www.ASmarterWayToLearn.com/js/1.html

2
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."

If 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.

In these examples, the terms "my name" and "my nationality" work the same way JavaScript *variables* do. *My name* is a term that refers to a particular *value*, "Mark." In the same way, a variable is a word that refers to a particular value.

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, within limits.

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...

Mark

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 used to refer to (Mark), but refers to a new 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 nationality* doesn't refer to the original value, "U.S.", but now refers to a new value.

JavaScript variables can also change.

If I code...

```
var name = "Mark";
```

name refers to "Mark". Then I come along and code the line...

```
name = "Ace";
```

Before I coded the new line, if I asked JavaScript to print **name**, it printed...

Mark

But that was then.

Now if I ask JavaScript to print **name**, it prints...

Ace

A variable can have any number of values, but only one at a time.

You may be wondering why, in the statement above that assigns "Ace" to **name**, the keyword **var** is missing. It's because the variable was declared earlier, in the original statement that assigned "Mark" to it. Remember, **var** is the keyword that

creates a variable—the keyword that *declares* it. Once a variable has been declared, you don't have to declare it again. You can just assign the new value to it.

You can declare a variable in one statement, leaving it *undefined*. Then you can assign a value to it in a later statement, without declaring it again.

```
var nationality;
nationality = "U.S.";
```

In the example above, the assignment statement follows the declaration statement immediately. But any amount of code can separate the two statements, as long as the declaration statement comes first. In fact, there's no law that says you have to *ever* define a variable that you've declared.

JavaScript variable names have no inherent meaning to JavaScript.

In English, words have meaning. You can't use just any word to communicate. I can say, "My name is Mark," but, if I want to be understood, I can't say, "My floogle is Mark." That's nonsense.

But with variables, JavaScript is blind to semantics. You *can* use just any word (as long as it doesn't break the rules of variable-naming). From JavaScript's point of view...

```
var floogle = "Mark";
```

...is just as good as...

```
var name = "Mark";
```

If you write...

```
var floogle = "Mark";
```

...then ask JavaScript to print `floogle`, JavaScript prints...

Mark

Within limits, you can name variables anything you want, and JavaScript won't care.

```
var lessonAuthor = "Mark";
var guyWhoKeepsSayingHisOwnName = "Mark";
var x = "Mark";
```

JavaScript's blindness to meaning notwithstanding, when it comes to variable names, you'll want to give your variables meaningful names, because it'll help you and other coders understand your code.

Again, the syntactic difference between variables and text strings is that variables are never enclosed in quotes, and text strings are always enclosed in quotes.

It's always...

```
var lastName = "Smith";
var cityOfOrigin = "New Orleans";
var aussieGreeting = "g'Day";
```

If it's an alphabet letter or word, and it isn't enclosed in quotes, and it isn't a keyword that has special meaning for JavaScript, like `alert`, it's a variable.

If it's some characters enclosed in quotes, it's a *text string*.

If you haven't noticed, let me point out the spaces between the variable and the equal sign, and between the equal sign and the value.

```
var nickname = "Bub";
```

These spaces are a style choice rather than a legal requirement. But I'll ask you to include them in your code throughout the practice exercises.

In the last chapter you learned to write...

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

When the code executes, a message box displays saying "Thanks for your input!"

But what if you wrote these two statements instead:

```
1 var thanx = "Thanks for your input!"
2 alert(thanx);
```

Instead of placing a text string inside the parentheses of the *alert* statement, the code above assigns the text string to a variable. Then it places the variable, not the string, inside the parentheses. Because JavaScript always substitutes the value for the variable, JavaScript displays—not the variable name **thanx** but the text to which it refers, "Thanks for your input!" That same alert, "Thanks for your input!" displays.

Find the interactive coding exercises for this chapter at:
http://www.ASmarterWayToLearn.com/js/2.html

3
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**...

```
weight + 25
```

...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.

```
1 var originalNum = 23;
2 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...

```
1 var originalNum = 23;
2 var numToBeAdded = 7;
3 var newNum = originalNum + numToBeAdded;
```

A variable can be used in calculating its own new value.

```
1 var originalNum = 90;
2 originalNum = originalNum + 10;
```

If you enclose a number in quotation marks and add 7...

```
1 var originalNum = "23";
2 var newNum = originalNum + 7;
```

...it won't work, because JavaScript can't sum a string and a number. JavaScript interprets "23" as a word, not a number. In the second statement, it doesn't add 23 + 7 to total 30. It does something that might surprise you. I'll tell you about this in a subsequent chapter. For now, know that a number enclosed by quotation marks is not a number, but a string, and JavaScript can't do addition on it.

Note that any particular variable name can be the name of a number variable or string variable. From JavaScript's point of view, there's nothing in a name that denotes one kind of variable or another. In fact, a variable can start out as one type of variable, then become another type of variable.

Did you notice what's new in...

```
1 var originalNumber = 23;
2 var newNumber = originalNumber + 7;
```

The statement assigns to the variable `newNumber` *the result of a mathematical operation*. The result of this operation, of course, *is* a number value.

The example mixes a variable and a literal number in a math expression. But you could also use nothing but numbers or nothing but variables. It's all the same to JavaScript.

I've told you that you can't begin a variable name with a number. The statement...

```
var 1stPresident = "Washington";
```