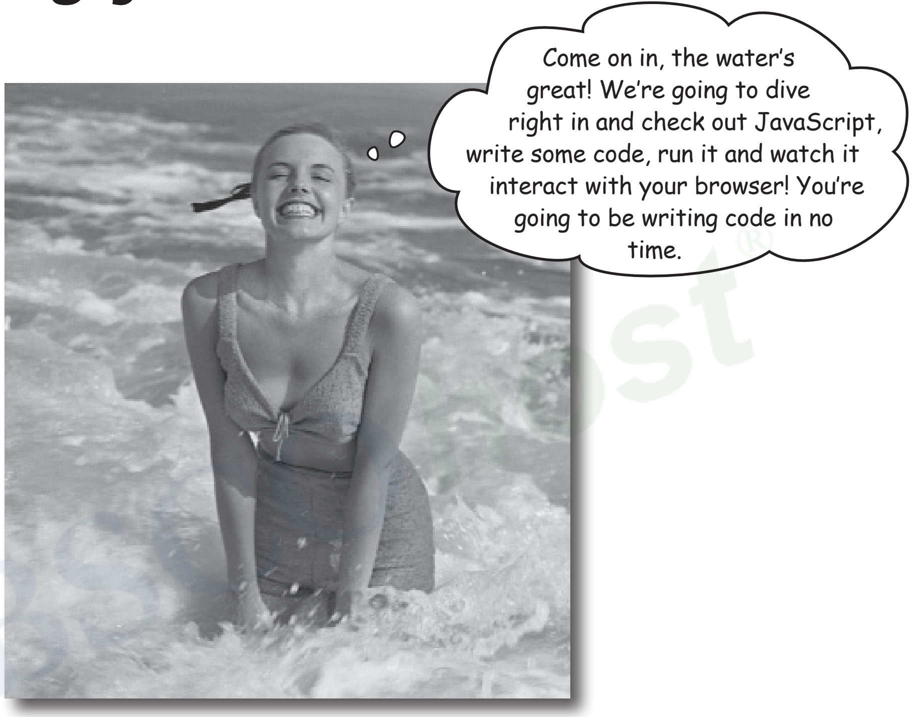
1 a quick dip into javascript

Getting your feet wet

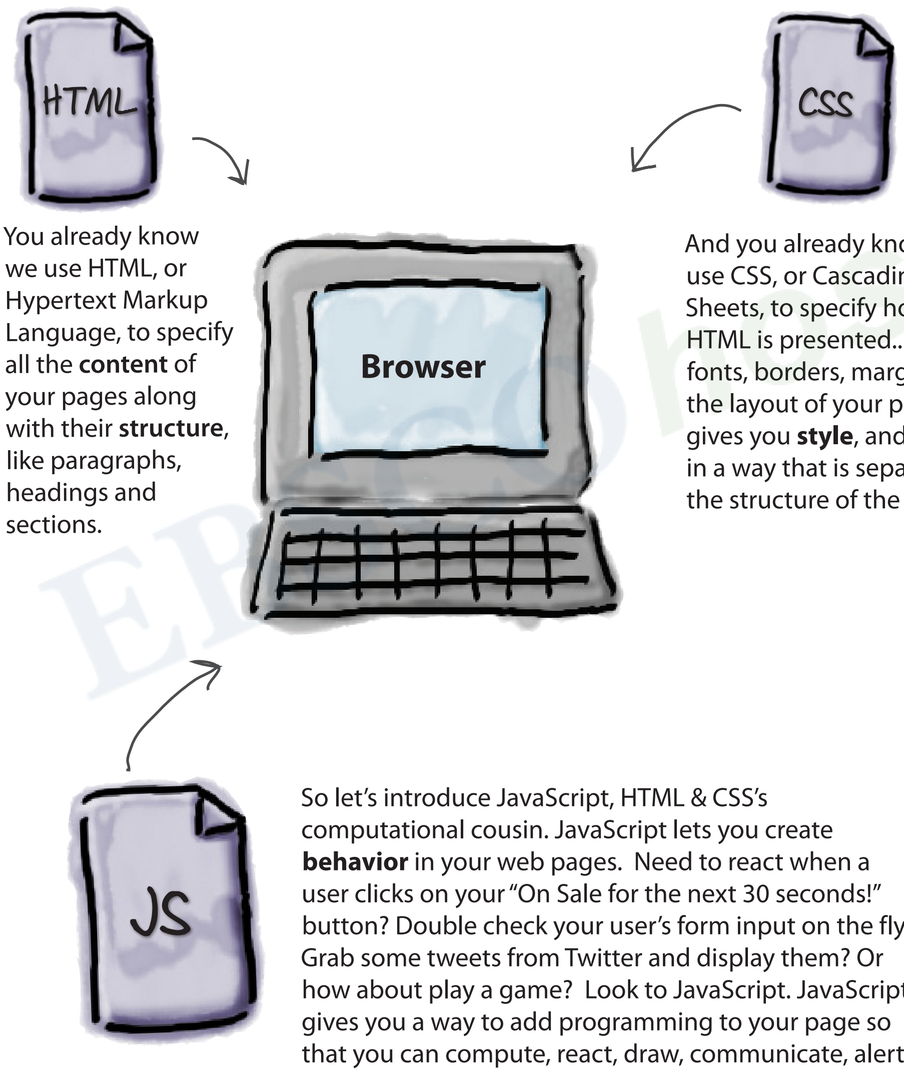

Come on in, the water's great! We're going to dive right in and check out JavaScript, write some code, run it and watch it interact with your browser! You're going to be writing code in no time.

JavaScript gives you superpowers. The **true programming language** of the web, JavaScript lets you **add behavior** to your web pages. No more dry, boring, static pages that just sit there looking at you—with JavaScript you're going to be able to reach out and touch your users, react to interesting events, grab data from the web to use in your pages, draw graphics right in your web pages and a lot more. And once you know JavaScript you'll also be in a position to create **totally new** behaviors for your users.

You'll be in good company too, JavaScript's not only one of the **most popular** programming languages, it's also **supported** in all modern (and most ancient) browsers; JavaScript's even branching out and being **embedded** in a lot of environments outside the browser. More on that later; for now, let's get started!

The way JavaScript works

If you're used to creating structure, content, layout and style in your web pages, isn't it time to add a little behavior as well? These days, there's no need for the page to just *sit there*. Great pages should be dynamic, interactive, and they should work with your users in new ways. That's where JavaScript comes in. Let's start by taking a look at how JavaScript fits into the *web page ecosystem*:



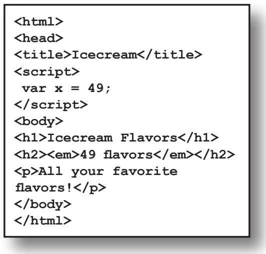
You already know we use HTML, or Hypertext Markup Language, to specify all the **content** of your pages along with their **structure**, like paragraphs, headings and sections.

And you already know that we use CSS, or Cascading Style Sheets, to specify how the HTML is presented...the colors, fonts, borders, margins, and the layout of your page. CSS gives you **style**, and it does it in a way that is separate from the structure of the page.

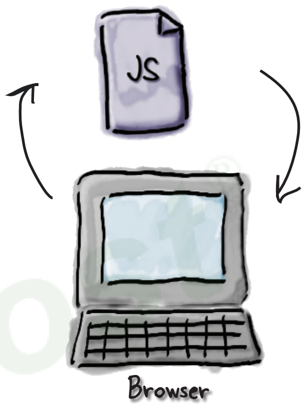
So let's introduce JavaScript, HTML & CSS's computational cousin. JavaScript lets you create **behavior** in your web pages. Need to react when a user clicks on your "On Sale for the next 30 seconds!" button? Double check your user's form input on the fly? Grab some tweets from Twitter and display them? Or how about play a game? Look to JavaScript. JavaScript gives you a way to add programming to your page so that you can compute, react, draw, communicate, alert, alter, update, change, and we could go on... anything dynamic, that's JavaScript in action.

How you're going to write JavaScript

JavaScript is fairly unique in the programming world. With your typical programming language you have to write it, compile it, link it and deploy it. JavaScript is much more fluid and flexible. With JavaScript all you need to do is write JavaScript right into your page, and then load it into a browser. From there, the browser will happily begin executing your code. Let's take a closer look at how this works:



```
<html>
<head>
<title>Icecream</title>
<script>
 var x = 49;
</script>
<body>
<h1>Icecream Flavors</h1>
<h2><em>49 flavors</em></h2>
<p>All your favorite
flavors!</p>
</body>
</html>
```



Writing

1

You create your page just like you always do, with HTML content and CSS style. And you also include JavaScript in your page. As you'll see, just like HTML and CSS, you can put everything together in one file, or you can place JavaScript in its own file, to be included in your page.

We'll talk about the best way in a bit...

Loading

2

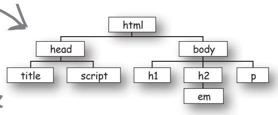
Point your browser to your page, just as you always do. When the browser sees code, it starts parsing it immediately, getting ready to execute it. Note that like HTML and CSS, if the browser sees errors in your code, it will do its best to keep moving and reading more JavaScript, HTML and CSS. The last thing it wants to do is not be able to give the user a page to see.

Executing

3

The browser starts executing your code as soon as it encounters it in your page, and continues executing it for the lifetime of your page. Unlike early versions of JavaScript, today's JavaScript is a powerhouse, using advanced compilation techniques to execute your code at nearly the same speed as many native programming languages.

For future reference, the browser also builds an "object model" of your HTML page that JavaScript can make use of. Put that in the back of your brain, we'll come back to it later...



How to get JavaScript into your page

First things first. You can't get very far with JavaScript if you don't know how to get it into a page. So, how do you do that? Using the `<script>` element of course!

Let's take a boring old, garden-variety web page and add some dynamic behavior using a `<script>` element. Now, at this point, don't worry too much about the details of what we're putting into the `<script>` element—your goal right now is to get some JavaScript working.

Here's our standard HTML5 doctype, and <html> and <head> elements.

And we've got a pretty generic <body> for this page as well.

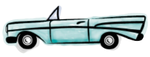
Ah, but we've added a script element to the <head> of the page.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>Just a Generic Page</title>
    <script>
      setTimeout(wakeUpUser, 5000);
      function wakeUpUser() {
         alert("Are you going to stare at this boring page forever?");
      }
    </script>
  </head>
  <body>
       <h1>Just a generic heading</h1>
       <p>Not a lot to read about here. I'm just an obligatory paragraph living in
an example in a JavaScript book. I'm looking for something to make my life more
exciting.</p>
  </body>
</html>
```

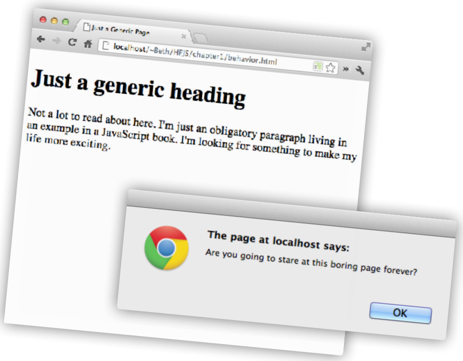
And we've written some JavaScript code inside it.

Again, don't worry too much about what this code does. Then again, we bet you'll want to take a look at the code and see if you can think through what each part might do.

A little test drive



Go ahead and type this page into a file named "behavior.html". Drag the file to your browser (or use File > Open) to load it. What does it do? Hint, you'll need to wait five seconds to find out.



Just a generic heading

Not a lot to read about here. I'm just an obligatory paragraph living in an example in a JavaScript book. I'm looking for something to make my life more exciting.

The page at localhost says:

Are you going to stare at this boring page forever?

Just relax. At this point we don't expect you to read JavaScript like you grew up with it. In fact, all we want you to do right now is get a feel for what JavaScript looks like.

That said, you're not totally off the hook because we need to get your brain revved up and working. Remember that code on the previous page? Let's just walk through it to get a feel for what it might do:

```
setTimeout(wakeUpUser, 5000);
function wakeUpUser() {
    alert("Are you going to stare at this boring page forever?");
}
```

Perhaps a way to count five seconds of time? Hint: 1000 milliseconds = 1 second.

A way to create reusable code and call it "wakeUpUser"?

Clearly a way to alert the user with a message.

there are no Dumb Questions

Q: I've heard JavaScript is a bit of a wimpy language. Is it?

A: JavaScript certainly wasn't a power lifter in its early days, but its importance to the web has grown since then, and as a result, many resources (including brain power from some of the best minds in the business) have gone into supercharging the performance of JavaScript. But, you know what? Even before JavaScript was super fast, it was always a brilliant language. As you'll see, we're going to do some very powerful things with it.

Q: Is JavaScript related to Java?

A: Only by name. JavaScript was created during a time when Java was a red hot popular language, and the inventors of JavaScript capitalized on that popularity by making use of the Java name. Both languages borrow some syntax from programming languages like C, but other than that, they are quite different.

Q: Is JavaScript the best way to create dynamic web pages? What about solutions like Flash?

A: There was a time when Flash may have been the preferred choice for many to create interactive and more dynamic web pages, but the industry direction is moving strongly in favor of HTML5 with JavaScript. And, with HTML5, JavaScript is now the standard scripting language for the Web. Many resources are going into making JavaScript fast and efficient, and creating JavaScript APIs that extend the functionality of the browser.

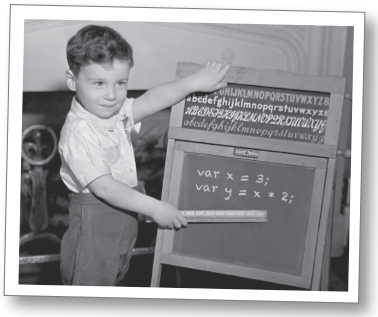
Q: My friend is using JavaScript inside Photoshop, or at least he says he is. Is that possible?

A: Yes, JavaScript is breaking out of the browser as a general scripting language for many applications from graphics utilities to music applications and even to server-side programming. Your investment in learning JavaScript is likely to pay off in ways beyond web pages in the future.

Q: You say that many other languages are compiled. What exactly does that mean and why isn't JavaScript?

A: With conventional programming languages like C, C++ or Java, you compile the code before you execute it. Compiling takes your code and produces a machine efficient representation of it, usually optimized for runtime performance. Scripting languages are typically interpreted, which means that the browser runs each line of JavaScript code as it gets to it. Scripting languages place less importance on runtime performance, and are more geared towards tasks like prototyping, interactive coding and flexibility. This was the case with early JavaScript, and was why, for many years, the performance of JavaScript was not so great. There is a middle ground however; an interpreted language can be compiled on the fly, and that's the path browser manufacturers have taken with modern JavaScript. In fact, with JavaScript you now have the conveniences of a scripting language, while enjoying the performance of a compiled language. By the way, we'll use the words *interpret*, *evaluate* and *execute* in this book. They have slightly different meanings in various contexts, but for our purposes, they all basically mean the same thing.

JavaScript, you've come a long way baby...



JavaScript 1.0

Netscape might have been before your time, but it was the first *real* browser company. Back in the mid-1990s browser competition was fierce, particularly with Microsoft, and so adding new, exciting features to the browser was a priority.

And towards that goal, Netscape wanted to create a scripting language that would allow anyone to add scripts to their pages. Enter LiveScript, a language developed in short order to meet that need. Now if you've never heard of LiveScript, that's because this was all about the time that Sun Microsystems introduced Java, and, as a result, drove their own stock to stratospheric levels. So, why not capitalize on that success and rename LiveScript to JavaScript? After all, who cares if they don't actually have anything to do with each other? Right?

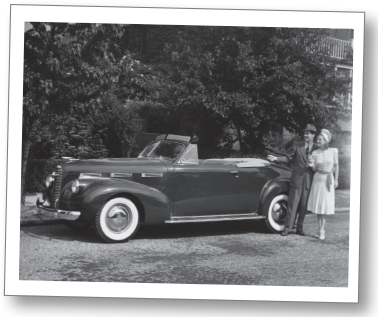
Did we mention Microsoft? They created their own scripting language soon after Netscape did, named, um, JScript, and it was, um, quite similar to JavaScript. And so began the browser wars.

JavaScript 1.3

Between 1996 and 2000, JavaScript grew up. In fact, Netscape submitted JavaScript for standardization and ECMAScript was born. Never heard of ECMAScript? That's okay, now you have; just know that ECMAScript serves as the standard language definition for all JavaScript implementations (in and out of the browser).

During this time developers continued struggling with JavaScript as casualties of the browser wars (because of all the differences in browsers), although the use of JavaScript became common-place in any case. And while subtle differences between JavaScript and JScript continued to give developers headaches, the two languages began to look more and more like each other over time.

JavaScript still hadn't outgrown its reputation as an amateurish language, but that was soon to change...



JavaScript 1.8.5

Finally, JavaScript comes of age and gains the respect of professional developers! While you might say it's all due to having a solid standard, like ECMAScript 5, which is now implemented in all modern browsers, it's really Google that pushed JavaScript usage into the professional limelight, when in 2005 they released Google Maps and showed the world what could really be done with JavaScript to create dynamic web pages.

With all the new attention, many of the best programming language minds focused on improving JavaScript's interpreters and made vast improvements to its runtime performance. Today, JavaScript stands with only a few changes from the early days, and despite its rushed birth into the world, is showing itself to be a powerful and expressive language.

1995
2000
2012

Sharpen your pencil

Look how easy it is to write JavaScript

You don't know JavaScript yet, but we bet you can make some good guesses about how JavaScript code works. Take a look at each line of code below and see if you can guess what it does. Write in your answers below. We've done one for you to get you started. If you get stuck, the answers are on the next page.

```
var price = 28.99;
var discount = 10;
var total =
      price - (price * (discount / 100));
if (total > 25) {
   freeShipping();
}

var count = 10;
while (count > 0) {
   juggle();
   count = count - 1;
}

var dog = {name: "Rover", weight: 35};
if (dog.weight > 30) {
    alert("WOOF WOOF");
} else {
    alert("woof woof");
}

var circleRadius = 20;
var circleArea =
   Math.PI * (circleRadius * circleRadius);
```

Create a variable named price, and assign the value 28.99 to it.

Sharpen your pencil Solution

Look how easy it is to write JavaScript

You don't know JavaScript yet, but we bet you can make some good guesses about how JavaScript code works. Take a look at each line of code below and see if you can guess what it does. Write in your answers below. We've done one for you to get you started. Here are our answers.

`var price = 28.99;`	Create a variable named price, and assign the value 28.99 to it.
`var discount = 10;`	Create a variable named discount, and assign the value 10 to it.
`var total = price - (price * (discount / 100));`	Compute a new price by applying a discount and then assign it to the variable total.
`if (total > 25) {`	Compare the value in the variable total to 25. If it's greater...
`freeShipping();`	...then do something with freeShipping.
`}`	End the if statement
`var count = 10;`	Create a variable named count, and assign the value 10 to it.
`while (count > 0) {`	As long as the variable count is greater than 0...
`juggle();`	...do some juggling, and...
`count = count - 1;`	...reduce the value of count by 1 each time.
`}`	End the while loop
`var dog = {name: "Rover", weight: 35};`	Create a dog with a name and weight.
`if (dog.weight > 30) {`	If the dog's weight is greater than 30...
`alert("WOOF WOOF");`	...alert "WOOF WOOF" to the browser's web page
`} else {`	Otherwise...
`alert("woof woof");`	...alert "woof woof" to the browser's web page
`}`	End the if/else statement
`var circleRadius = 20;`	Create a variable, circleRadius, and assign the value 20 to it.
`var circleArea =`	Create a variable named circleArea...
`Math.PI * (circleRadius * circleRadius);`	...and assign the result of this expression to it (1256.6370614359173)

Look, if you want to go beyond creating *just* static web pages, you gotta have JavaScript chops.

It's True.

With HTML and CSS you can create some great looking pages. But once you know JavaScript, you can really expand on the kinds of pages you can create. So much so, in fact, you might actually start thinking of your pages as applications (or even experiences!) rather than mere pages.

And usually increase the size of your paycheck too!

Now, you might be saying, "Sure, I know that. Why do you think I'm reading this book?" Well, we actually wanted to use this opportunity to have a little chat about learning JavaScript. If you already have a programming language or scripting language under your belt, then you have some idea of what lies ahead. However, if you've mostly been using HTML & CSS to date, you should know that there is something fundamentally different about learning a programming language.

With HTML & CSS what you're doing is largely declarative—for instance, you're declaring, say, that some text is a paragraph or that all elements in the "sale" class should be colored red. With JavaScript you're adding *behavior* to the page, and to do that you need to describe computation. You need to be able to describe things like, "compute the user's score by summing up all the correct answers" or "do this action ten times" or "when the user clicks on that button play the you-have-won sound" or even "go off and get my latest tweet, and put it in this page."

To do those things you need a language that is quite different from HTML or CSS. Let's see how…

How to make a statement

When you create HTML you usually **mark up** text to give it structure; to do that you add elements, attributes and values to the text:

```
<h1 class="drink">Mocha Caffe Latte</h1>
<p>Espresso, steamed milk and chocolate syrup,
just the way you like it.</p>
```

With HTML we mark up text to create structure. Like, "I need a large heading called Mocha Cafe Latte; it's a heading for a drink. And I need a paragraph after that."

CSS is a bit different. With CSS you're writing a set of **rules**, where each rule selects elements in the page, and then specifies a set of styles for those elements:

```
h1.drink {
    color: brown;
}
p {
    font-family: sans-serif;
}
```

With CSS we write rules that use selectors, like h1.drink and p, to determine what parts of the HTML the style is applied to.

Let's make sure all drink headings are colored brown...

...and we want all the paragraphs to have a sans-serif type font.

With JavaScript you write **statements**. Each statement specifies a small part of a computation, and together, all the statements create the behavior of the page:

A set of statements.

Each statement does a little bit of work, like declaring some variables to contain values for us.

```
var age = 25;
var name = "Owen";

if (age > 14) {
    alert("Sorry this page is for kids only!");
} else {
    alert("Welcome " + name + "!");
}
```

Here we create a variable to contain an age of 25, and we also need a variable to contain the value "Owen".

Or making decisions, such as: Is the age of the user greater than 14?

And if so alerting the user they are too old for this page.

Otherwise, we welcome the user by name, like this: "Welcome Owen!" (but since Owen is 25, we don't do that in this case.)

Variables and values

You might have noticed that JavaScript statements usually involve variables. Variables are used to store values. What kinds of values? Here are a few examples:

```
var winners = 2;
```

This statement declares a variable named winners and assigns a numeric value of 2 to it.

```
var name = "Duke";
```

This one assigns a string of characters to the variable name (we call those "strings," for short).

```
var isEligible = false;
```

And this statement assigns the value false to the variable isEligible. We call true/false values "booleans."

Pronounced "boo-lee-ans."

2
winners
"Duke"
name
false
isEligible

Notice we don't put quotes around boolean values.

There are other values that variables can hold beyond numbers, strings and booleans, and we'll get to those soon enough, but, no matter what a variable contains, we create all variables the same way. Let's take a little closer look at how to declare a variable:

We always start with the var keyword when declaring a variable.

NO EXCEPTIONS! Even if JavaScript doesn't complain when you leave off the var. We'll tell you why later...

Next we give the variable a name.

```
var winners = 2;
```

We always end an assignment statement with a semicolon.

And, optionally, we assign a value to the variable by adding an equals sign followed by the value.

We say optionally, because if you want, you can create a variable without an initial value, and then assign it a value later. To create a variable without an initial value, just leave off the assignment part, like this:

```
var losers;
```

By leaving off the equals sign and value you're just declaring the variable for later use.

No value?! What am I supposed to do now?! I'm so humiliated.
losers

Back away from that keyboard!

You know variables have a name, and you know they have a value.

You also know some of the things a variable can hold are numbers, strings and boolean values.

But what can you call your variables? Is any name okay? Well no, but the rules around creating variable names are simple: just follow the two rules below to create valid variable names:

1. **Start your variables with a letter, an underscore or a dollar sign.**
2. **After that, use as many letters, numeric digits, underscores or dollar signs as you like.**

Oh, and one more thing; we really don't want to confuse JavaScript by using any of the built-in *keywords*, like **var** or **function** or **false**, so consider those off limits for your own variable names. We'll get to some of these keywords and what they mean throughout the rest of the book, but here's a list to take a quick look at:

break	delete	for	let	super	void
case	do	function	new	switch	while
catch	else	if	package	this	with
class	enum	implements	private	throw	yield
const	export	import	protected	true	
continue	extends	in	public	try	
debugger	false	instanceof	return	typeof	
default	finally	interface	static	var	

there are no Dumb Questions

Q: What's a keyword?

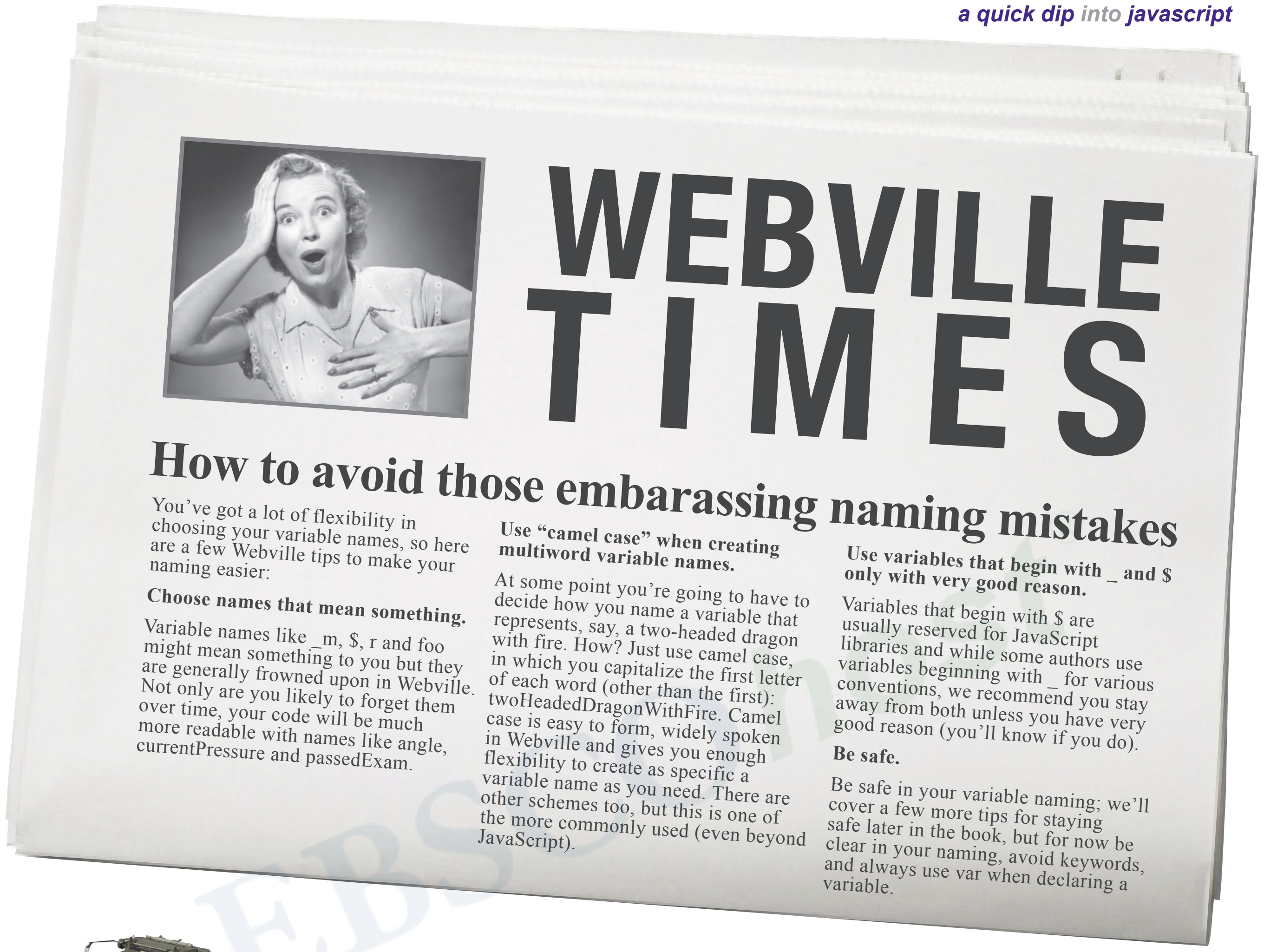
A: A keyword is a reserved word in JavaScript. JavaScript uses these reserved words for its own purposes, and it would be confusing to you and the browser if you started using them for your variables.

Q: What if I used a keyword as part of my variable name? For instance, can I have a variable named ifOnly (that is, a variable that contains the keyword if)?

A: You sure can, just don't match the keyword exactly. It's also good to write clear code, so in general you wouldn't want to use something like `elze`, which might be confused with `else`.

Q: Is JavaScript case sensitive? In other words, are myvariable and MyVariable the same thing?

A: If you're used to HTML markup you might be used to case insensitive languages; after all, <head> and <HEAD> are treated the same by the browser. With JavaScript however, case matters for variables, keywords, function names and pretty much everything else, too. So pay attention to your use of upper- and lowercase.



WEBVILLE TIMES

How to avoid those embarassing naming mistakes

You've got a lot of flexibility in choosing your variable names, so here are a few Webville tips to make your naming easier:

Choose names that mean something.

Variable names like _m, $, r and foo might mean something to you but they are generally frowned upon in Webville. Not only are you likely to forget them over time, your code will be much more readable with names like angle, currentPressure and passedExam.

Use "camel case" when creating multiword variable names.

At some point you're going to have to decide how you name a variable that represents, say, a two-headed dragon with fire. How? Just use camel case, in which you capitalize the first letter of each word (other than the first): twoHeadedDragonWithFire. Camel case is easy to form, widely spoken in Webville and gives you enough flexibility to create as specific a variable name as you need. There are other schemes too, but this is one of the more commonly used (even beyond JavaScript).

Use variables that begin with _ and $ only with very good reason.

Variables that begin with $ are usually reserved for JavaScript libraries and while some authors use variables beginning with _ for various conventions, we recommend you stay away from both unless you have very good reason (you'll know if you do).

Be safe.

Be safe in your variable naming; we'll cover a few more tips for staying safe later in the book, but for now be clear in your naming, avoid keywords, and always use var when declaring a variable.

Syntax Fun

- Each statement ends in a semicolon.
  ```
  x = x + 1;
  ```
- A single line comment begins with two forward slashes. Comments are just notes to you or other developers about the code.They aren't executed.
  ```
  // I'm a comment
  ```
- Whitespace doesn't matter (almost everywhere).
  ```
  x     =     2233;
  ```
- Surround strings of characters with double quotes (or single, both work, just be consistent).
  ```
  "You rule!"
  'And so do you!'
  ```
- Don't use quotes around the boolean values true and false.
  ```
  rockin = true;
  ```
- Variables don't have to be given a value when they are declared:
  ```
  var width;
  ```
- JavaScript, unlike HTML markup, is case sensitive, meaning upper- and lowercase matters. The variable `counter` is different from the variable `Counter`.

BE the Browser

Below, you'll find JavaScript code with some mistakes in it. Your job is to play like you're the browser and find the errors in the code. After you've done the exercise look at the end of the chapter to see if you found them all.

Don't worry too much about what this JavaScript does for now; just focus on looking for errors in variables and syntax.

A

```
// Test for jokes
var joke = "JavaScript walked into a bar....';
var toldJoke = "false";
var $punchline =
  "Better watch out for those semi-colons."
var %entage = 20;
var result

if (toldJoke == true) {
    Alert($punchline);
} else
    alert(joke);
}
```

B

```
\\ Movie Night
var zip code = 98104;
var joe'sFavoriteMovie = Forbidden Planet;
var movieTicket$    =    9;

if (movieTicket$ >= 9) {
    alert("Too much!");
} else {
    alert("We're going to see " + joe'sFavoriteMovie);
}
```

Express yourself

To truly express yourself in JavaScript you need ***expressions***. Expressions evaluate to values. You've already seen a few in passing in our code examples. Take the expression in this statement for instance:

Here's a JavaScript statement that assigns the result of evaluating an expression to the variable total.

We use * for multiply and / for divide.

```
var total = price - (price * (discount / 100));
```

Here's our variable total.

And the assignment.

And this whole thing is an expression.

This expression evaluates to a price reduced by a discount that is a percent of the price. So if your price is 10 and the discount is 20, we get 8 as a result.

If you've ever taken a math class, balanced your checkbook or done your taxes, we're sure these kinds of numeric expressions are nothing new.

There are also string expressions; here are a few:

```
"Dear " + "Reader" + ","
```

This adds together, or concatenates, these strings to form a new string "Dear Reader,".

```
"super" + "cali" + youKnowTheRest
```

Same here, except we have a variable that contains a string as part of the expression. This evaluates to "supercalifragilisticexpialidocious".*

```
phoneNumber.substring(0,3)
```

Just another example of an expression that results in a string. We'll get to exactly how this works later, but this returns the area code of a US phone number string.

We also have expressions that evaluate to **true** or **false**, otherwise known as boolean expressions. Work through each of these to see how you get true or false from them:

```
age < 14
```

If a person's age is less than 14 this is true, otherwise it is false. We could use this to test if someone is a child or not.

```
cost >= 3.99
```

If the cost is 3.99 or greater, this is true. Otherwise it's false. Get ready to buy on sale when it's false!

```
animal == "bear"
```

This is true when animal contains the string "bear". If it does, beware!

And expressions can evaluate to a few other types; we'll get to these later in the book. For now, the important thing is to realize all these expressions evaluate to something: a value that is a number, a string or a boolean. Let's keep moving and see what that gets you!

* Of course, that is assuming the variable youKnowTheRest is "fragilisticexpialidocious".

Sharpen your pencil

Get out your pencil and put some expressions through their paces. For each expression below, compute its value and write in your answer. Yes, WRITE IN... forget what your Mom told you about writing in books and scribble your answer right in this book! Be sure to check your answers at the end of the chapter.

Can you say "Celsius to Fahrenheit calculator"?

`(9 / 5) * temp + 32`

What is the result when temp is 10? __________

This is a boolean expression. The == operator tests if two values are equal to each other.

`color == "orange"`

Is this expression true or false when color has the value "pink"? __________
Or has the value "orange"? __________

`name + ", " + "you've won!"`

What value does this compute to when name is "Martha"?
____

This tests if the first value is greater than the second. You can also use >= to test if the first value is greater than or equal to the second.

`yourLevel > 5`

When yourLevel is 2, what does this evaluate to? ________
When yourLevel is 5, what does this evaluate to? ________
When yourLevel is 7, what does this evaluate to? ________

`(level * points) + bonus`

Okay, level is 5, points is 30,000 and bonus is 3300. What does this evaluate to? ________

`color != "orange"`

Is this expression true or false when color has the value "pink"? __________

The != operator tests if two values are NOT equal to each other.

Extra CREDIT!

`1000 + "108"`

Are there a few possible answers? Only one is correct. Which would you choose? ______________________

Serious Coding

Did you notice that the = operator is used in assignments, while the == operator tests for equality? That is, we use one equal sign to assign values to variables. We use two equal signs to test if two values are equal to each other. Substituting one for the other is a common coding mistake.

```
while (juggling) {
    keepBallsInAir();
}
```

Doing things more than once

You do a lot of things more than once:

Lather, rinse, repeat…

Wax on, wax off…

Eat candies from the bowl until they're all gone.

Of course you'll often need to do things in code more than once, and JavaScript gives you a few ways to repeatedly execute code in a loop: **while**, **for**, **for in** and **forEach**. Eventually, we'll look at all these ways of looping, but let's focus on **while** for now.

We just talked about expressions that evaluate to boolean values, like `scoops > 0`, and these kinds of expressions are the key to the while statement. Here's how:

A while statement starts with the keyword while.

While uses a boolean expression that we call a conditional test, or conditional for short.

If the conditional is true, everything in the code block is executed.

```
while (scoops > 0) {
    document.write("Another scoop!");
    scoops = scoops - 1;
}
```

What's a code block? Everything between the curly braces; that is, between { }.

And, if our conditional is true, then, after we execute the code block, we loop back around and do it all again. If the conditional is false, we're done.

Like we said, lather, rinse, repeat!

How the while loop works

Seeing as this is your first while loop, let's trace through a round of its execution to see exactly how it works. Notice we've added a declaration for scoops to declare it, and initialize it to the value 5.

Now let's start executing this code. First we set scoops to five.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

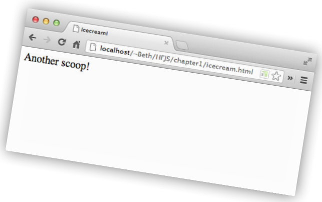
After that we hit the while statement. When we evaluate a while statement the first thing we do is evaluate the conditional to see if it's true or false.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

Is scoops greater than zero? Looks like it to us!

Because the conditional is true, we start executing the block of code. The first statement in the body writes the string "Another scoop! <br>" to the browser.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



Another scoop!

The next statement subtracts one from the number of scoops and then sets scoops to that new value, four.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

1 scoop gone, 4 left!

That's the last statement in the block, so we loop back up to the conditional and start over again.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

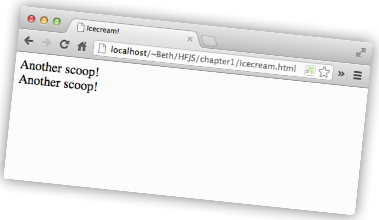
Evaluating our conditional again, this time scoops is four. But that's still more than zero.

Still plenty left!

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

Once again we write the string "Another scoop! <br>" to the browser.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



```
Another scoop!
Another scoop!
```

The next statement subtracts one from the number of scoops and sets scoops to that new value, which is three.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

2 scoops gone, 3 left!

That's the last statement in the block, so we loop back up to the conditional and start over again.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

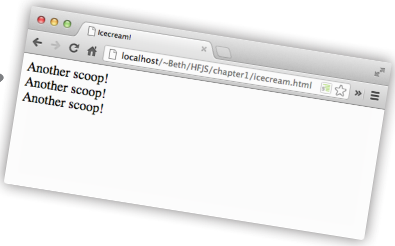
Evaluating our conditional again, this time scoops is three. But that's still more than zero.

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

Still plenty left!

Once again we write the string "Another scoop! <br>" to the browser.

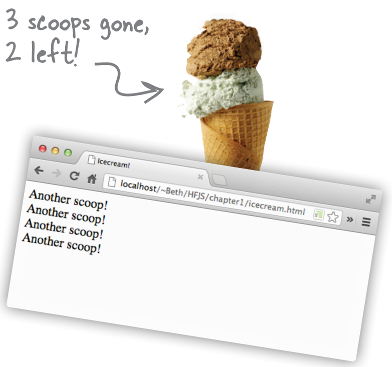
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



Another scoop!
Another scoop!
Another scoop!

And as you can see, this continues... each time we loop, we decrement (reduce scoops by 1), write another string to the browser, and keep going.

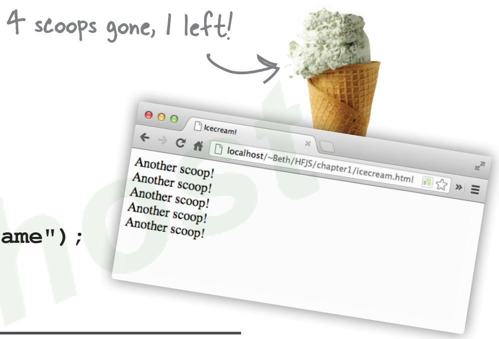
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



Another scoop!
Another scoop!
Another scoop!
Another scoop!

And continues...

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



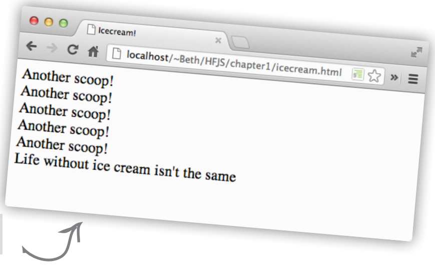
Another scoop!
Another scoop!
Another scoop!
Another scoop!
Another scoop!

Until the last time... this time something's different. Scoops is zero, and so our conditional returns false. That's it folks; we're not going to go through the loop anymore, we're not going to execute the block. This time, we bypass the block and execute the statement that follows it.

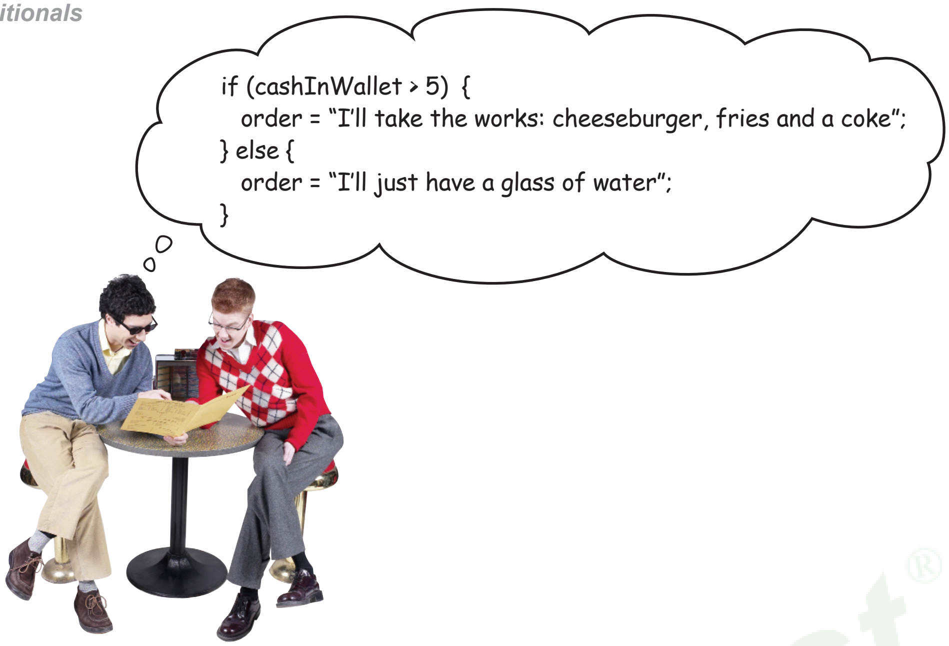
```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```

Now we execute the other document.write, and write the string "Life without ice cream isn't the same". We're done!

```
var scoops = 5;
while (scoops > 0) {
   document.write("Another scoop!<br>");
   scoops = scoops - 1;
}
document.write("Life without ice cream isn't the same");
```



Another scoop!
Another scoop!
Another scoop!
Another scoop!
Another scoop!
Life without ice cream isn't the same


if (cashInWallet > 5) {
order = "I'll take the works: cheeseburger, fries and a coke";
} else {
order = "I'll just have a glass of water";
}

Making decisions with JavaScript

You've just seen how you use a conditional to decide whether to continue looping in a `while` statement. You can also use boolean expressions to make decisions in JavaScript with the `if` statement. The `if` statement executes its code block only if a conditional test is true. Here's an example:

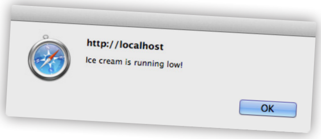
Here's the if keyword, followed by a conditional and a block of code.

This conditional tests to see if we're down to fewer than three scoops.

```
if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

And if we've got fewer than three left, then we execute the if statement's code block.

alert takes a string and displays it in a popup dialog in your browser. Give it a try!


http://localhost

Ice cream is running low!

With an `if` statement we can also string together multiple tests by adding on one or more `else if`'s, like this:

We can have one test, and then another test with if/else if

```
if (scoops >= 5) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops < 3) {
    alert("Ice cream is running low!");
}
```

Add as many tests with "else if" as you need, each with its own associated code block that will be executed when the condition is true.

And, when you need to make LOTS of decisions

You can string together as many `if/else` statements as you need, and if you want one, even a final catch-all `else`, so that if all conditions fail, you can handle it. Like this:

```
if (scoops >= 5) {
    alert("Eat faster, the ice cream is going to melt!");
} else if (scoops == 3) {
    alert("Ice cream is running low!");
} else if (scoops == 2) {
    alert("Going once!");
} else if (scoops == 1) {
    alert("Going twice!");
} else if (scoops == 0) {
    alert("Gone!");
} else {
    alert("Still lots of ice cream left, come and get it.");
}
```

In this code we check to see if there are five or more scoops left...

...or if there are precisely three left...

...or if there are 2, 1 or 0, and then we provide the appropriate alert.

And if none of the conditions above are true, then this code is executed.

there are no Dumb Questions

Q: What exactly is a block of code?

A: Syntactically, a block of code (which we usually just call a block) is a set of statements, which could be one statement, or as many as you like, grouped together between curly braces. Once you've got a block of code, all the statements in that block are treated as a group to be executed together. For instance, all the statements within the block in a while statement are executed if the condition of the while is true. The same holds for a block in an if or else if.

Q: I've seen code where the conditional is just a variable that is sometimes a string, not a boolean. How does that work?

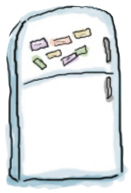
A: We'll be covering that a little later, but the short answer is JavaScript is quite flexible in what it thinks is a true or false value. For instance, any variable that holds a (non-empty) string is considered true, but a variable that hasn't been set to a value is considered false. We'll get into these details soon enough.

Q: You've said that expressions can result in things other than numbers, strings and booleans. Like what?

A: Right now we're concentrating on what are known as the *primitive types*, that is, numbers, strings and booleans. Later we'll take a look at more complex types, like arrays, which are collections of values, objects and functions.

Q: Where does the name boolean come from?

A: Booleans are named after George Boole, an English mathematician who invented Boolean logic. You'll often see boolean written "Boolean," to signify that these types of variables are named after George.



Code Magnets

A JavaScript program is all scrambled up on the fridge. Can you put the magnets back in the right places to make a working JavaScript program to produce the output shown below?. Check your answer at the end of the chapter before you go on.

Arrange these magnets to make a working JavaScript program.

```
document.write("Happy Birthday dear " + name + ",<br>");
```

```
document.write("Happy Birthday to you.<br>");
```

```
var i = 0;
```

```
var name = "Joe";
```

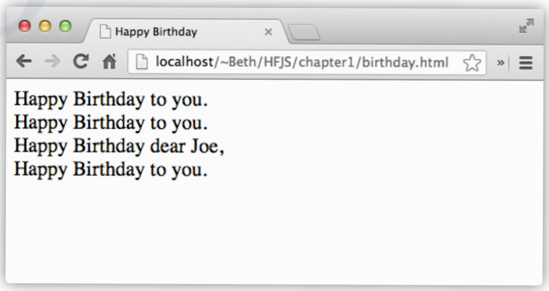
```
i = i + 1;
```

```
}
```

```
document.write("Happy Birthday to you.<br>");
```

```
while (i < 2) {
```

Your unscrambled program should produce this output.



```
Happy Birthday to you.
Happy Birthday to you.
Happy Birthday dear Joe,
Happy Birthday to you.
```

Use this space for your re-arranged magnets.

Reach out and communicate with your user

We've been talking about making your pages more interactive, and to do that you need to be able to communicate with your user. As it turns out there are a few ways to do that, and you've already seen some of them. Let's get a quick overview and then we'll dive into these in more detail throughout the book:

Create an alert.

As you've seen, the browser gives you a quick way to alert your users through the `alert` function. Just call `alert` with a string containing your alert message, and the browser will give your user the message in a nice dialog box. A small confession though: we've been overusing this because it's easy; `alert` really should be used only when you truly want to stop everything and let the user know something.

We're using these three methods in this chapter.

Write directly into your document.

Think of your web page as a document (that's what the browser calls it). You can use a function `document.write` to write arbitrary HTML and content into your page at any point. In general, this is considered bad form, although you'll see it used here and there. We've used it a bit in this chapter too because it's an easy way to get started.

Use the console.

Every JavaScript environment also has a console that can log messages from your code. To write a message to the console's log you use the function `console.log` and hand it a string that you'd like printed to the log (more details on using console log in a second). You can view `console.log` as a great tool for troubleshooting your code, but typically your users will never see your console log, so it's not a very effective way to communicate with them.

The console is a really handy way to help find errors in your code! If you've made a typing mistake, like missing a quote, JavaScript will usually give you an error in the console to help you track it down.

Directly manipulate your document.

This is the big leagues; this is the way you want to be interacting with your page and users—using JavaScript you can access your actual web page, read & change its content, and even alter its structure and style! This all happens by making use of your browser's *document object model* (more on that later). As you'll see, this is the best way to communicate with your user. But, using the document object model requires knowledge of how your page is structured and of the programming interface that is used to read and write to the page. We'll be getting there soon enough. But first, we've got some more JavaScript to learn.

This is what we're working towards. When you get there you'll be able to read, alter and manipulate your page in any number of ways.

WHO DOES WHAT?

All our methods of communication have come to the party with masks on. Can you help us unmask each one? Match the descriptions on the right to the names on the left. We've done one for you.

document.write	I'll stop your user in his tracks and deliver a short message. The user has to click on "ok" to go further.
console.log	I can insert a little HTML and text into a document. I'm not the most elegant way to get a message to your users, but I work on every browser.
alert	Using me you can totally control a web page: get values that a user typed in, alter the HTML or the style, or update the content of your page.
document object model	I'm just here for simple debugging purposes. Use me and I can write out information to a special developer's console.

A closer look at console.log

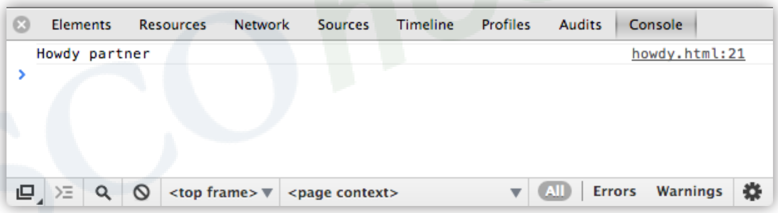
Let's take a closer look at how `console.log` works so we can use it in this chapter to see the output from our code, and throughout the book to inspect the output of our code and debug it. Remember though, the console is not a browser feature most casual users of the web will encounter, so you won't want to use it in the final version of your web page. Writing to the console log is typically done to troubleshoot as you develop your page. That said, it's a great way to see what your code is doing while you're learning the basics of JavaScript. Here's how it works:

Take any old string...

```
var message = "Howdy" + " " + "partner";
console.log(message);
```

...and give it to console.log, and it will be shown in the browser's console, like this.

The console contains all the output logged by your code.



```
Howdy partner                                              howdy.html:21
>
```

there are no Dumb Questions

Q: I get that console.log can be used to output strings, but what exactly is it? I mean why are the "console" and the "log" seperated by a period?

A: Ah, good point. We're jumping ahead a bit, but think of the console as an object that does things, console-like things. One of those things is logging, and to tell the console to log for us, we use the syntax "console.log" and pass it our output in between parentheses. Keep that in the back of your mind; we're coming back to talk a lot more about objects a little later in the book. For now, you've got enough to use console.log.

Q: Can the console do anything other than just log?

A: Yes, but typically people just use it to log. There are a few more advanced ways to use log (and console), but they tend to be browser-specific. Note that console is something all modern browsers supply, but it isn't part of any formal specification.

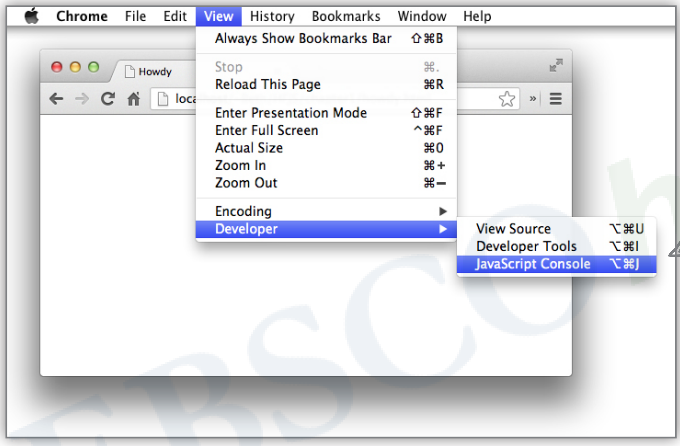
Q: Uh, console looks great, but where do I find it? I'm using it in my code and I don't see any output!

A: In most browsers you have to explicitly open the console window. Check out the next page for details.

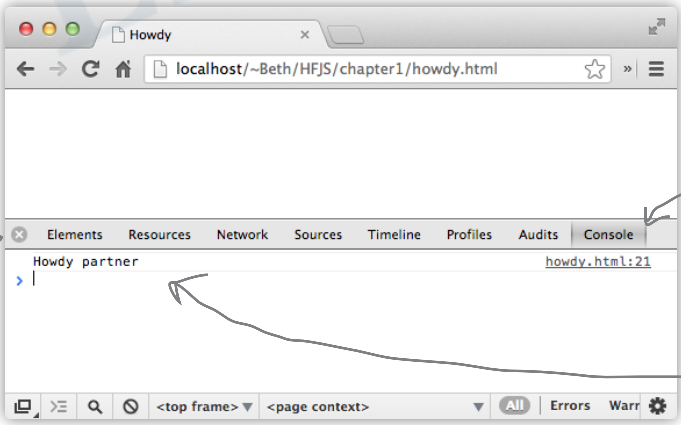
Opening the console

Every browser has a slightly different implementation of the console. And, to make things even more complicated, the way that browsers implement the console changes fairly frequently—not in a huge way, but enough so that by the time you read this, your browser's console might look a bit different from what we're showing here.

So, we're going to show you how to access the console in the Chrome browser (version 25) on the Mac, and we'll put instructions on how to access the console in all the major browsers online at http://wickedlysmart.com/hfjsconsole. Once you get the hang of the console in one browser, it's fairly easy to figure out how to use it in other browsers too, and we encourage you to try using the console in at least two browsers so you're familiar with them.



To access the console in Chrome (on the Mac), use the View > Developer > JavaScript Console menu.



```
Howdy partner                                   howdy.html:21
> |
```

The console will appear in the bottom part of your browser window.

Make sure the Console tab is selected in the tab bar along the top of the console.

You should see any messages you give to console.log in your code displayed in the window here.

Don't worry about what these other tabs are for. They're useful, but the most important one now is Console, so we can see console.log messages from our code.

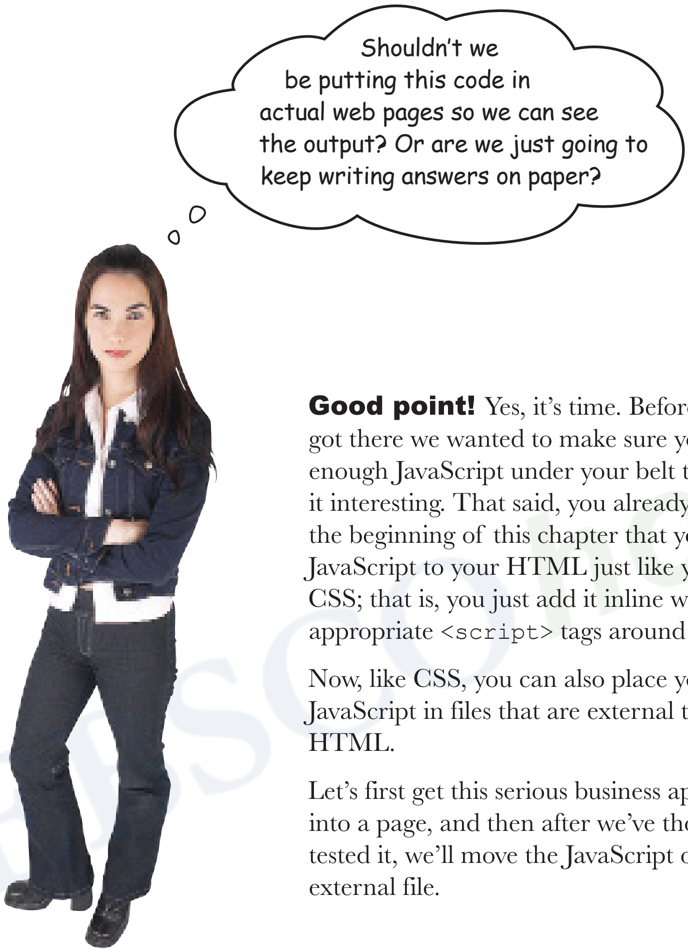
Coding a Serious JavaScript Application

Let's put all these new JavaScript skills and `console.log` to good use with something practical. We need some variables, a `while` statement, some `if` statements with `else`s. Add a little more polish and we'll have a super-serious business application before you know it. But, before you look at the code, think to yourself how you'd code that classic favorite, "99 bottles of beer."

```
var word = "bottles";

var count = 99;

while (count > 0) {

    console.log(count + " " + word + " of beer on the wall");

    console.log(count + " " + word + " of beer,");

    console.log("Take one down, pass it around,");

    count = count - 1;

    if (count > 0) {

        console.log(count + " " + word + " of beer on the wall.");

    } else {

        console.log("No more " + word + " of beer on the wall.");

    }

}
```

BRAIN POWER

There's still a little flaw in our code. It runs correctly, but the output isn't 100% perfect. See if you can find the flaw, and fix it.


Shouldn't we be putting this code in actual web pages so we can see the output? Or are we just going to keep writing answers on paper?

Good point! Yes, it's time. Before we got there we wanted to make sure you had enough JavaScript under your belt to make it interesting. That said, you already saw in the beginning of this chapter that you add JavaScript to your HTML just like you add CSS; that is, you just add it inline with the appropriate `<script>` tags around it.

Now, like CSS, you can also place your JavaScript in files that are external to your HTML.

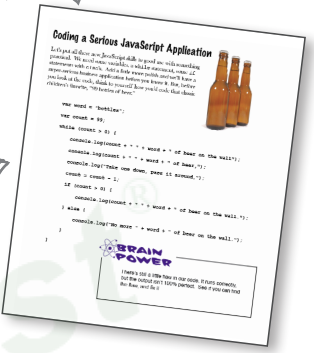
Let's first get this serious business application into a page, and then after we've thoroughly tested it, we'll move the JavaScript out to an external file.

A Test Drive

Okay, let's get some code in the browser... follow the instructions below and get your serious business app launched! You'll see our result below:

To download all the code and sample files for this book, please visit http://wickedlysmart.com/hfjs.

1. Check out the HTML below; that's where your JavaScript's going to go. Go ahead and type in the HTML and then place the JavaScript from two pages back in between the <script> tags. You can use an editor like Notepad (Windows) or TextEdit (Mac), making sure you are in plain text mode. Or, if you have a favorite HTML editor, like Dreamweaver, Coda or WebStorm, you can use that too.



Coding a Serious JavaScript Application

Let's put all these new JavaScript skills to good use with something practical. We need some variables, a while statement, some if statements with else's. Add a little more polish and we'll have a super-serious business application before you know it. But, before you look at the code, think to yourself how you'd code that classic children's favorite, "99 bottles of beer."

```
var word = "bottles";
var count = 99;
while (count > 0) {
    console.log(count + " " + word + " of beer on the wall");
    console.log(count + " " + word + " of beer,");
    console.log("Take one down, pass it around,");
    count = count - 1;
    if (count > 0) {
        console.log(count + " " + word + " of beer on the wall.");
    } else {
        console.log("No more " + word + " of beer on the wall.");
    }
}
```

BRAIN POWER

There's still a little flaw in our code. It runs correctly, but the output isn't 100% perfect. See if you can find the flaw, and fix it.

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>

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

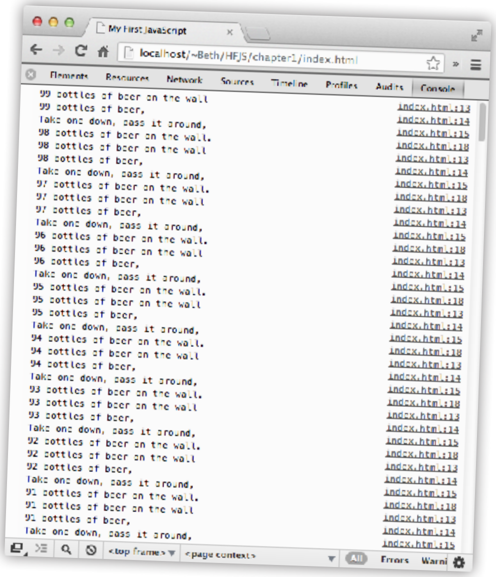
Type this in.

Here are the <script> tags. At this point you know that's where you should put your code.

2. Save the file as "index.html".

3. Load the file into your browser. You can either drag the file right on top of your browser window, or use the File > Open (or File > Open File) menu option in your favorite browser.

4. You won't see anything in the web page itself because we're logging all the output to the console, using console.log. So open up the browser's console, and congratulate yourself on your serious business application.



My First JavaScript
localhost/~Beth/HFJS/chapter1/index.html
Elements Resources Network Sources Timeline Profiles Audits Console
<top frame> <page context> All Errors Warni

```
99 bottles of beer on the wall
99 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
98 bottles of beer on the wall.                       index.html:15
98 bottles of beer on the wall                        index.html:18
98 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
97 bottles of beer on the wall.                       index.html:15
97 bottles of beer on the wall                        index.html:18
97 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
96 bottles of beer on the wall.                       index.html:15
96 bottles of beer on the wall                        index.html:18
96 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
95 bottles of beer on the wall.                       index.html:15
95 bottles of beer on the wall                        index.html:18
95 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
94 bottles of beer on the wall.                       index.html:15
94 bottles of beer on the wall                        index.html:18
94 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
93 bottles of beer on the wall.                       index.html:15
93 bottles of beer on the wall                        index.html:18
93 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
92 bottles of beer on the wall.                       index.html:15
92 bottles of beer on the wall                        index.html:18
92 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
91 bottles of beer on the wall.                       index.html:15
91 bottles of beer on the wall                        index.html:18
91 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
                                                      index.html:15
```

Here's our test run of this code. The code creates the entire lyrics for the 99 bottles of beer song and logs the text to the browser's console.

How do I add code to my page? (let me count the ways)

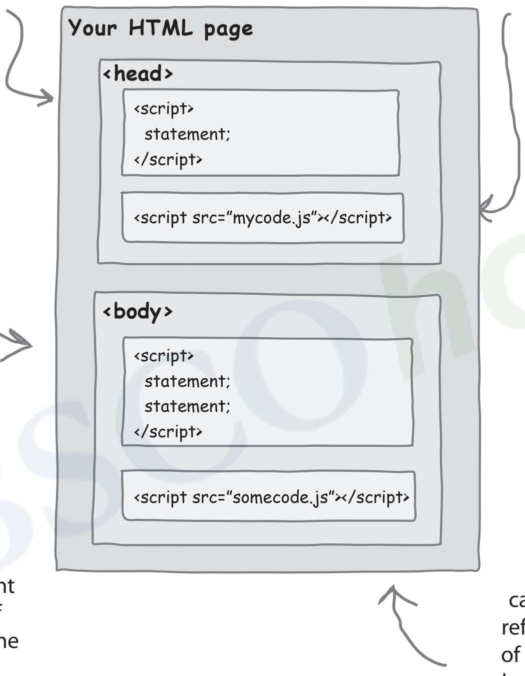
You already know you can add the `<script>` element with your JavaScript code to the `<head>` or `<body>` of your page, but there are a couple of other ways to add your code to a page. Let's check out all the places you can put JavaScript (and why you might want to put it one place over another):

You can place your code inline, in the <head> element. The most common way to add code to your pages is to put a <script> element in the <head>. Sure, it makes your code easy to find and seems to be a logical place for your code, but it's not always the best place. Why? Read on...

Or, you can add your code inline in the body of the document. To do this, enclose your JavaScript code in the <script> element and place it in the <body> of your page (typically at the end of the body).

This is a little better. Why? When your browser loads a page, it loads everything in your page's <head> before it loads the <body>. So, if your code is in the <head>, users might have to wait a while to see the page. If the code is loaded after the HTML in the <body>, users will get to see the page content while they wait for the code to load.

Still, is there a better way? Read on...

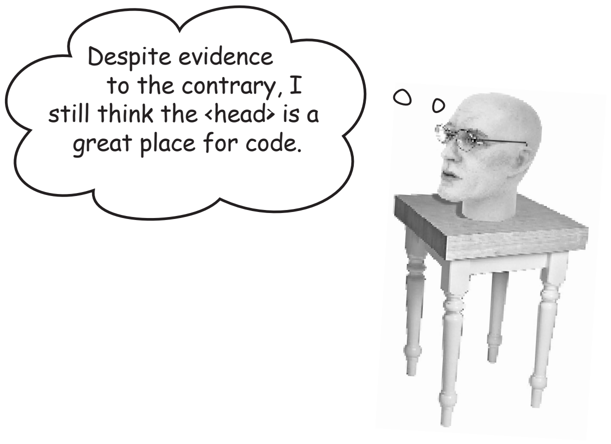


```
<script>
  statement;
</script>

<script src="mycode.js"></script>

<script>
  statement;
  statement;
</script>

<script src="somecode.js"></script>
```

Or, put your code in its own file and link to it from the <head>. This is just like linking to a CSS file. The only difference is that you use the src attribute of the <script> tag to specify the URL to your JavaScript file.

When your code is in an external file, it's easier to maintain (separately from the HTML) and can be used across multiple pages. But this method still has the drawback that all the code needs to be loaded before the body of the page. Is there a better way? Read on...

Finally, you can link to an external file in the body of your page. Ahhh, the best of both worlds. We have a nice, maintainable JavaScript file that can be included in any page, and it's referenced from the bottom of the body of the page, so it's only loaded after the body of the page. Not bad.


Despite evidence to the contrary, I still think the <head> is a great place for code.

We're going to have to separate you two

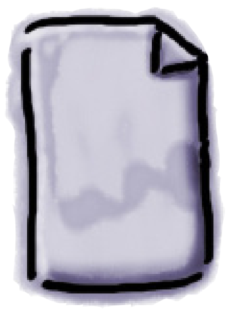
Going separate ways hurts, but we know we have to do it. It's time to take your JavaScript and move it into its own file. Here's how you do that…

1. Open index.html and select all the code; that is, everything between the <script> tags. Your selection should look like this:

```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script>
     var word = "bottles";
     var count = 99;
     while (count > 0) {
        console.log(count + " " + word + " of beer on the wall");
        console.log(count + " " + word + " of beer,");
        console.log("Take one down, pass it around,");
        count = count - 1;
        if (count > 0) {
           console.log(count + " " + word + " of beer on the wall.");
        } else {
           console.log("No more " + word + " of beer on the wall.");
        }
     }
    </script>
  </body>
</html>
```

Select just the code, not the <script> tags; you won't need those where you're going…

2. Now create a new file named "code.js" in your editor, and place the code into it. Then save "code.js".


code.js

3 Now we need to place a reference to the "code.js" file in "index.html" so that it's retrieved and loaded when the page loads. To do that, delete the JavaScript code from "index.html", but leave the <script> tags. Then add a src attribute to your opening <script> tag to reference "code.js".

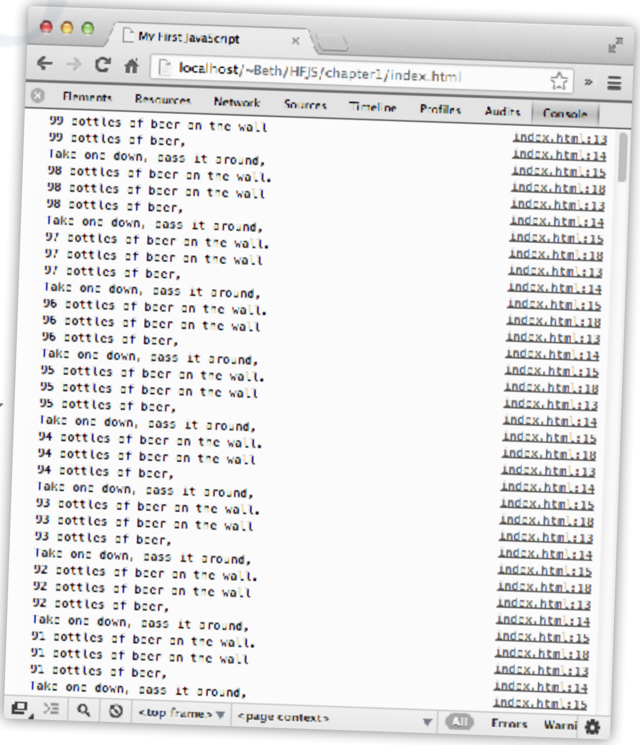
```
<!doctype html>
<html lang="en">
  <head>
    <meta charset="utf-8">
    <title>My First JavaScript</title>
  </head>
  <body>
    <script src="code.js">

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

Use the src attribute of the <script> element to link to your JavaScript file.

Where your code was.

Believe it or not we still need the ending <script> tag, even if there is no code between the two tags.

4 That's it, the surgery is complete. Now you need to test it. Reload your "index.html" page and you should see exactly the same result as before. Note that by using a src="code.js", we're assuming that the code file is in the same directory as the HTML file.

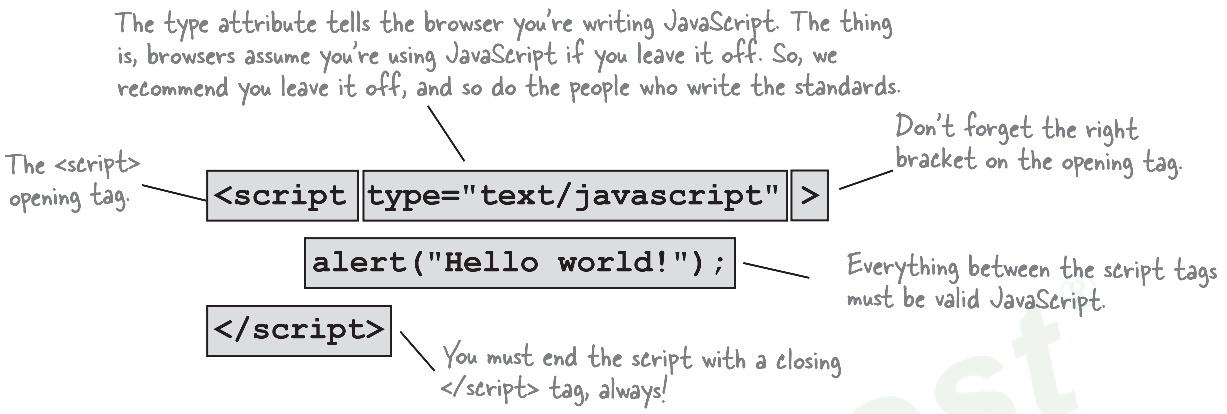


```
99 bottles of beer on the wall
99 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
98 bottles of beer on the wall.                       index.html:15
98 bottles of beer on the wall                        index.html:18
98 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
97 bottles of beer on the wall.                       index.html:15
97 bottles of beer on the wall                        index.html:18
97 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
96 bottles of beer on the wall.                       index.html:15
96 bottles of beer on the wall                        index.html:18
96 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
95 bottles of beer on the wall.                       index.html:15
95 bottles of beer on the wall                        index.html:18
95 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
94 bottles of beer on the wall.                       index.html:15
94 bottles of beer on the wall                        index.html:18
94 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
93 bottles of beer on the wall.                       index.html:15
93 bottles of beer on the wall                        index.html:18
93 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
92 bottles of beer on the wall.                       index.html:15
92 bottles of beer on the wall                        index.html:18
92 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
91 bottles of beer on the wall.                       index.html:15
91 bottles of beer on the wall                        index.html:18
91 bottles of beer,                                   index.html:13
Take one down, pass it around,                        index.html:14
                                                      index.html:15
```

You should get the same result as before. But now your HTML and JavaScript are in separate files. Doesn't that just feel cleaner, more manageable, more stress-free already?

Anatomy of a Script Element

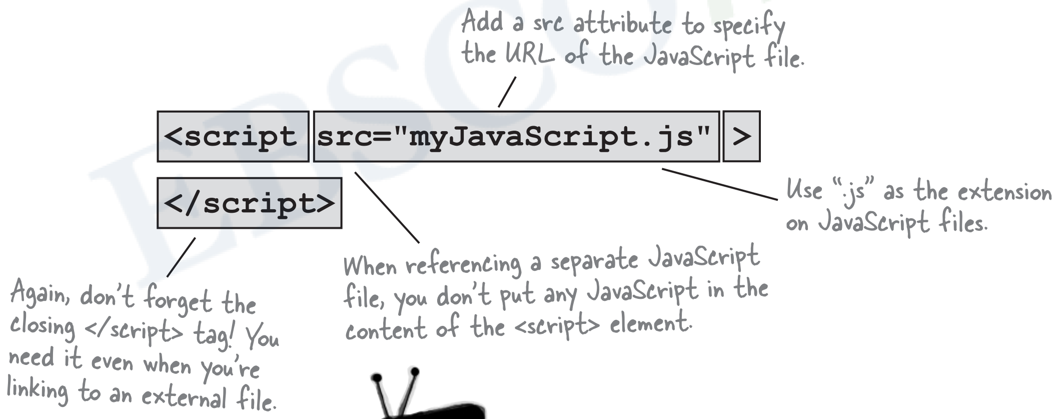
You know how to use the <script> element to add code to your page, but just to really nail down the topic, let's review the <script> element to make sure we have every detail covered:



The type attribute tells the browser you're writing JavaScript. The thing is, browsers assume you're using JavaScript if you leave it off. So, we recommend you leave it off, and so do the people who write the standards.
The <script> opening tag.
Don't forget the right bracket on the opening tag.
Everything between the script tags must be valid JavaScript.
You must end the script with a closing </script> tag, always!

```
<script type="text/javascript">
    alert("Hello world!");
</script>
```

And when you are referencing a separate JavaScript file from your HTML, you'll use the <script> element like this:



Add a src attribute to specify the URL of the JavaScript file.
Use ".js" as the extension on JavaScript files.
Again, don't forget the closing </script> tag! You need it even when you're linking to an external file.
When referencing a separate JavaScript file, you don't put any JavaScript in the content of the <script> element.

```
<script src="myJavaScript.js">
</script>
```

You can't use inline and external together.

If you try throwing some quick code in between those <script> tags when you're already using a src attribute, it won't work. You'll need two separate <script> elements.

```
<script src="goodies.js">
   var = "quick hack";
</script>
```

WRONG

JavaScript Exposed

This week's interview:
Getting to know JavaScript

Head First: Welcome JavaScript. We know you're super-busy out there, working on all those web pages, so we're glad you could take time out to talk to us.

JavaScript: No problem. And, I *am* busier than ever these days; people are using JavaScript on just about every page on the Web nowadays, for everything from simple menu effects to full blown games. It's nuts!

Head First: That's amazing given that just a few years ago, someone said that you were just a "half-baked, wimpy scripting language" and now you're everywhere.

JavaScript: Don't remind me. I've come a long way since then, and many great minds have been hard at work making me better.

Head First: Better how? Seems like your basic language features are about the same…

JavaScript: Well, I'm better in a couple of ways. First of all, I'm lightning fast these days. While I'm considered a scripting language, now my performance is close to that of native compiled languages.

Head First: And second?

JavaScript: My ability to do things in the browser has expanded dramatically. Using the JavaScript libraries available in all modern browsers you can find out your location, play video and audio, paint graphics on your web page and a lot more. But if you wanna do all that you have to know JavaScript.

Head First: But back to those criticisms of you, the language. I've heard some not so kind words… I believe the phrase was "hacked up language."

JavaScript: I'll stand on my record. I'm pretty much one of, if not *the* most widely used languages in the world. I've also fought off many competitors and won. Remember Java in the browser? Ha, what a joke. VBScript? Ha. JScript? Flash?! Silverlight? I could go on and on. So, tell me, how bad could I be?

Head First: You've been criticized as, well, "simplistic."

JavaScript: Honestly, it's my greatest strength. The fact that you can fire up a browser, type in a few lines of JavaScript and be off and running, that's powerful. And it's great for beginners too. I've heard some say there's no better beginning language than JavaScript.

Head First: But simplicity comes at a cost, no?

JavaScript: Well that's the great thing, I'm simple in the sense you can get a quick start. But I'm deep and full of all the latest modern programming constructs.

Head First: Oh, like what?

JavaScript: Well, for example, can you say dynamic types, first-class functions and closures?

Head First: I can say it but I don't know what they are.

JavaScript: Figures… that's okay, if you stay with the book you will get to know them.

Head First: Well, give us the gist.

JavaScript: Let me just say this, JavaScript was built to live in a dynamic web environment, an exciting environment where users interact with a page, where data is coming in on the fly, where many types of events happen, and the language reflects that style of programming. You'll get it a little more a bit later in the book when you understand JavaScript more.

Head First: Okay, to hear you tell it, you're the perfect language. Is that right?

JavaScript tears up…

JavaScript: You know, I didn't grow up within the ivy-covered walls of academia like most languages. I was born into the real world and had to sink or swim very fast in my life. Given that, I'm not perfect; I certainly have a few "bad parts."

Head First with a slight Barbara Walters smile: We've seen a new side of you today. I think this merits another interview in the future. Any parting thoughts?

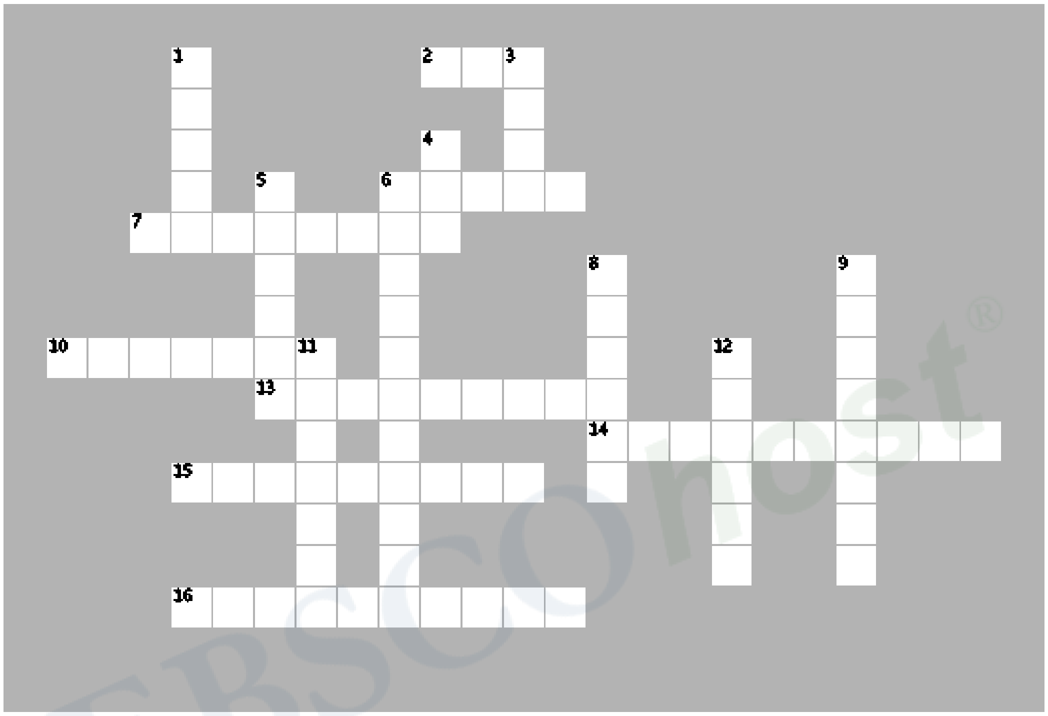
JavaScript: Don't judge me by my bad parts, learn the good stuff and stick with that!

BULLET POINTS

- JavaScript is used to add **behavior** to web pages.
- Browser engines are much faster at executing JavaScript than they were just a few years ago.
- Browsers begin executing JavaScript code as soon as they encounter the code in the page.
- Add JavaScript to your page with the **<script>** element.
- You can put your JavaScript inline in the web page, or link to a separate file containing your JavaScript from your HTML.
- Use the **src** attribute in the <script> tag to link to a separate JavaScript file.
- HTML **declares** the structure and content of your page; JavaScript **computes** values and adds behavior to your page.
- JavaScript programs are made up of a series of **statements**.
- One of the most common JavaScript statements is a variable declaration, which uses the **var** keyword to declare a new variable and the assignment operator, **=,** to assign a value to it.
- There are just a few rules and guidelines for naming JavaScript variables, and it's important that you follow them.
- Remember to avoid JavaScript keywords when naming variables.
- JavaScript expressions compute values.
- Three common types of expressions are **numeric**, **string** and **boolean** expressions.
- **if/else** statements allow you to make decisions in your code.
- **while/for** statements allow you to execute code many times by looping.
- Use **console.log** instead of **alert** to display messages to the Console.
- Console messages should be used primarily for troubleshooting as users will most likely never see console messages.
- JavaScript is most commonly found adding behavior to web pages, but is also used to script applications like Adobe Photoshop, OpenOffice and Google Apps, and is even used as a server-side programming language.

JavaScript cross

Time to stretch your dendrites with a puzzle to help it all sink in.



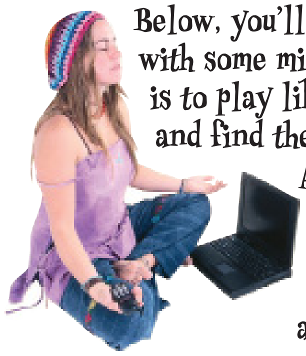
ACROSS

2. To link to an external JavaScript file from HTML, you need the _______ attribute for your <script> element.
6. To avoid embarrassing naming mistakes, use __________ case.
7. JavaScript adds _______________ to your web pages.
10. There are 99 _____________ of beer on the wall.
13. Each line of JavaScript code is called a _______________.
14. 3 + 4 is an example of an _____________.
15. All JavaScript statements end with a ___________.
16. Use _____________ to troubleshoot your code.

DOWN

1. Do things more than once in a JavaScript program with the _________ loop.
3. JavaScript variable names are _________ sensitive.
4. To declare a variable, use this keyword.
5. Variables are used to store these.
6. Each time through a loop, we evaluate a ______________ expression.
8. Today's JavaScript runs a lot ________________ than it used to.
9. The if/else statement is used to make a ____________.
11. You can concatenate _______________ together with the + operator.
12. You put your JavaScript inside a ______________ element.

BE the Browser Solution



Below, you'll find JavaScript code with some mistakes in it. Your job is to play like you're the browser and find the errors in the code. After you've done the exercise look at the end of the chapter to see if you found them all. Here's our solution.

A

```
// Test for jokes
var joke = "JavaScript walked into a bar....';
var toldJoke = "false";
var $punchline =
  "Better watch out for those semi-colons."
var %entage = 20;
var result

if (toldJoke == true) {
    Alert($punchline);
} else
    alert(joke);
}
```

Delimit your strings with two double quotes (") or two single quotes ('). Don't mix!

Don't put quotes around boolean values unless you really want a string.

It's okay, but not recommended, to begin a variable with a $.

Don't forget to end statements with a semi-colon!

Can't use % in variable names.

Another missing semi-colon.

Should be alert, not Alert. JavaScript is case-sensitive.

We're missing an opening brace here.

B

```
\\ Movie Night
var zip code = 98104;
var joe'sFavoriteMovie = Forbidden Planet;
var movieTicket$    =    9;

if (movieTicket$ >= 9) {
    alert("Too much!");
} else {
    alert("We're going to see " + joe'sFavoriteMovie);
}
```

Comments should begin with // not \\.

No spaces allowed in variable names.

No quotes allowed in variable names.

But we do need quotes around the string "Forbidden Planet".

This if/else doesn't work because of the invalid variable name here.

Sharpen your pencil Solution

Get out your pencil and let's put some expressions through their paces. For each expression below, compute its value and write in your answer. Yes, WRITE IN... forget what your Mom told you about writing in books and scribble your answer right in this book! Here's our solution.

Can you say "Celsius to Fahrenheit calculator"?

```
(9 / 5) * temp + 32
```

What is the result when temp is 10? 50

This is a boolean expression. The == operator tests if two values are equal to each other.

```
color == "orange"
```

Is this expression true or false when color has the value "pink"? false
Or, has the value "orange"? true

```
name + ", " + "you've won!"
```

What value does this compute to when name is "Martha"? "Martha, you've won!"

This tests if the first value is greater than the second. You can also use >= to test if the first value is greater than or equal to the second.

```
yourLevel > 5
```

When yourLevel is 2, what does this evaluate to? false
When yourLevel is 5, what does this evaluate to? false
When yourLevel is 7, what does this evaluate to? true

```
(level * points) + bonus
```

Okay, level is 5, points is 30,000 and bonus is 3300. What does this evaluate to? 153300

```
color != "orange"
```

Is this expression true or false when color has the value "pink"? true

The != operator tests if two values are NOT equal to each other.

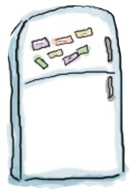
Extra CREDIT!

```
1000 + "108"
```

Are there a few possible answers? Only one is correct. Which would you choose? "1000108"

Serious Coding

Did you notice that the = operator is used in assignments, while the == operator tests for equality? That is, we use one equal sign to assign values to variables. We use two equal signs to test if two values are equal to each other. Substituting one for the other is a common coding mistake.



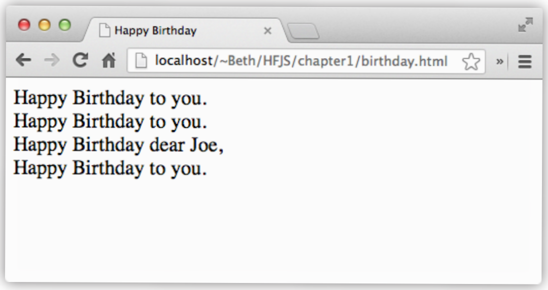
Code Magnets Solution

A JavaScript program is all scrambled up on the fridge. Can you put the magnets back in the right places to make a working JavaScript program to produce the output shown below?. Here's our solution.

Here are the unscrambled magnets!

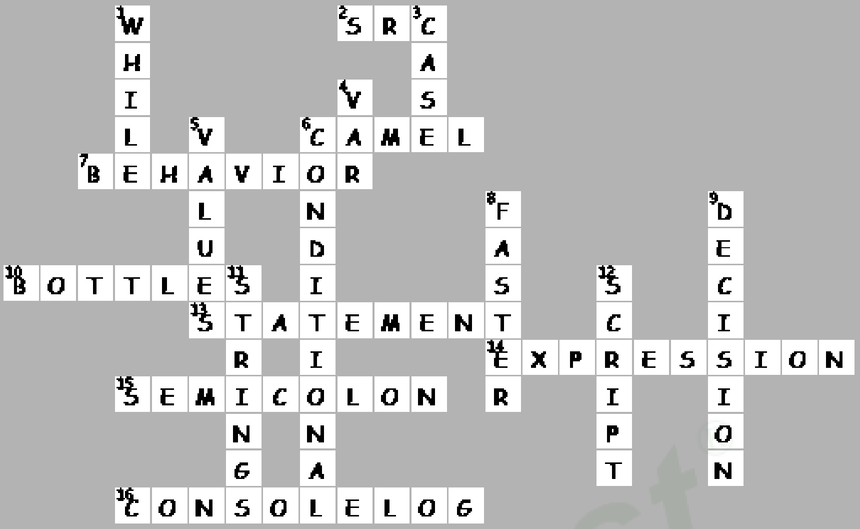
```
var name = "Joe";
var i = 0;
while (i < 2) {
    document.write("Happy Birthday to you.<br>");
    i = i + 1;
}
document.write("Happy Birthday dear " + name + ",<br>");
document.write("Happy Birthday to you.<br>");
```

Your unscrambled program should produce this output.



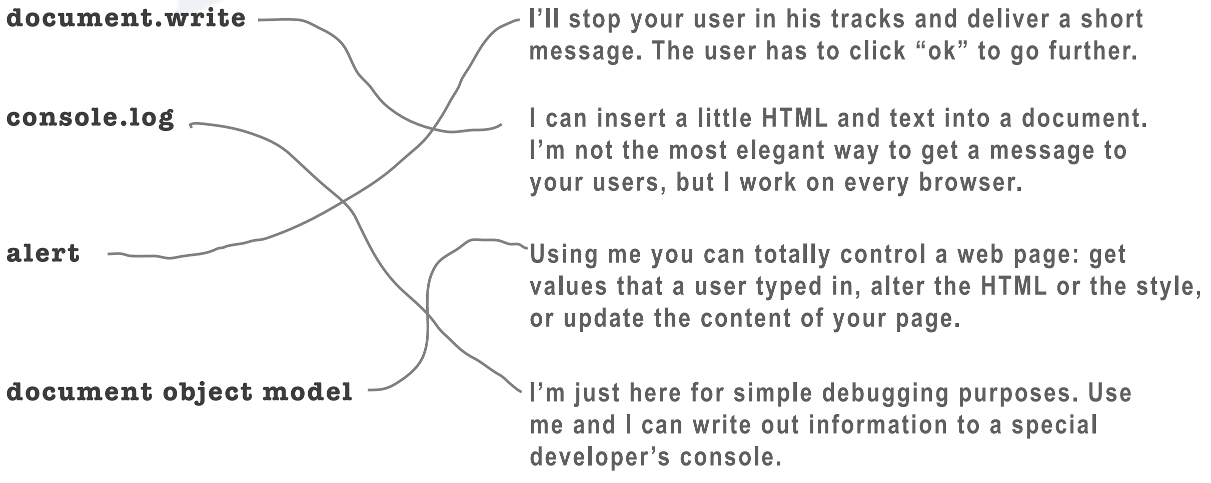
```
Happy Birthday to you.
Happy Birthday to you.
Happy Birthday dear Joe,
Happy Birthday to you.
```

JavaScript Cross Solution



WHO DOES WHAT? SOLUTION

All our methods of communication have come to the party with masks on. Can you help us unmask each one? Match the descriptions on the right to the names on the left. Here's our solution:



document.write

console.log

alert

document object model

I'll stop your user in his tracks and deliver a short message. The user has to click "ok" to go further.

I can insert a little HTML and text into a document. I'm not the most elegant way to get a message to your users, but I work on every browser.

Using me you can totally control a web page: get values that a user typed in, alter the HTML or the style, or update the content of your page.

I'm just here for simple debugging purposes. Use me and I can write out information to a special developer's console.