Chapter 1. A Quick dip into javascript: *Getting your feet wet*

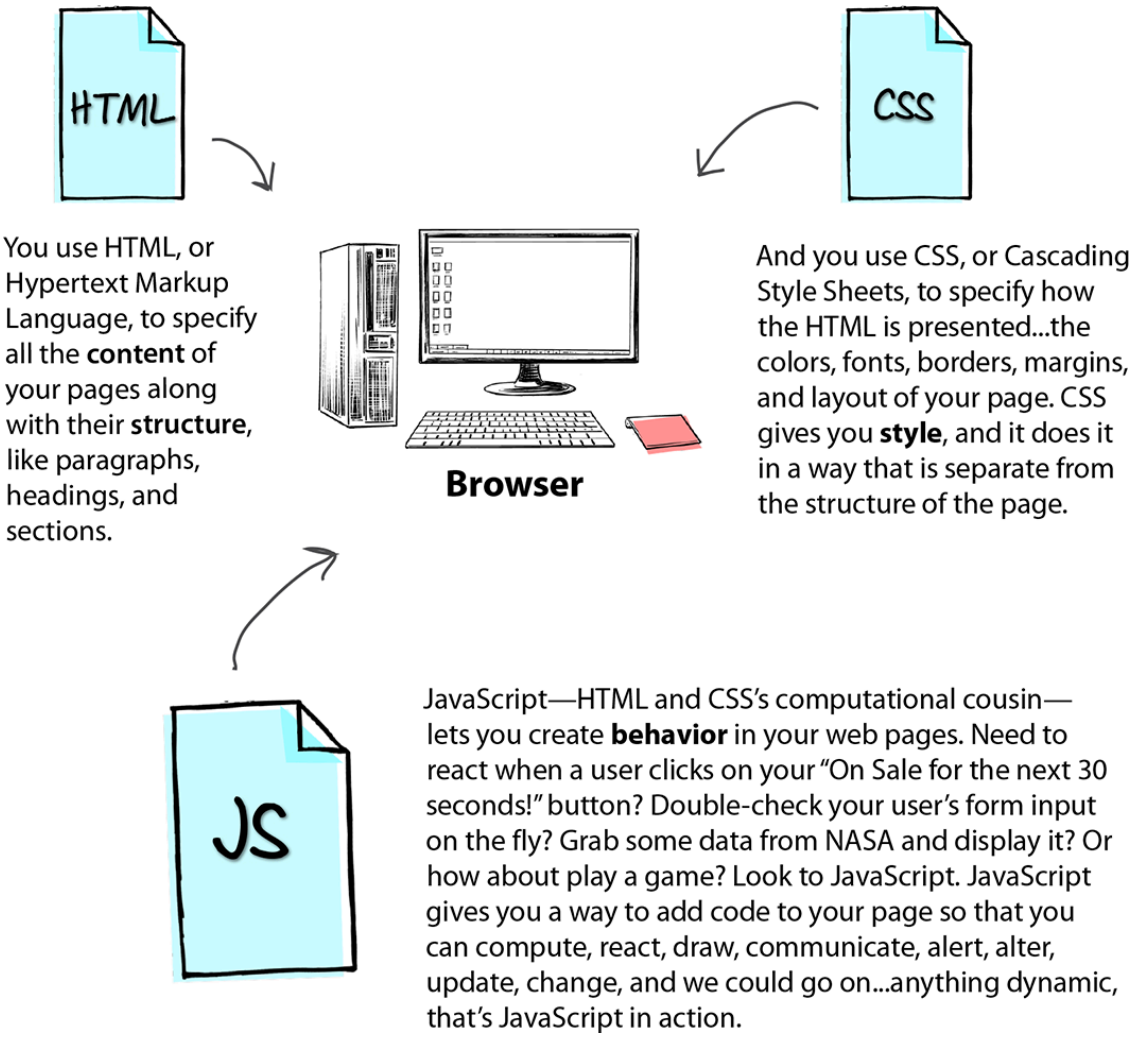
JavaScript gives you superpowers. The **true programming language** of the web, JavaScript lets you **add behavior** to your web pages. No more dry, boring, static pages that just sit there looking at you—with JavaScript, you'll be able to reach out and touch your users, react to interesting events, grab data from the web to use in your pages, draw graphics right into those 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 browsers

and is used in many environments outside of 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 you added a little behavior as well? After all, there's no need for the page to just *sit there*. Great pages should be *interactive* and *dynamic*. That's where JavaScript comes in. Let's start by taking a look at how JavaScript fits into the *web page ecosystem*:



How you're going to write JavaScript

JavaScript is *the* programming language for the web browser. With your typical programming language, you have to write the code, compile it,

link it, and deploy it. With JavaScript, all you need to do is write code 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:

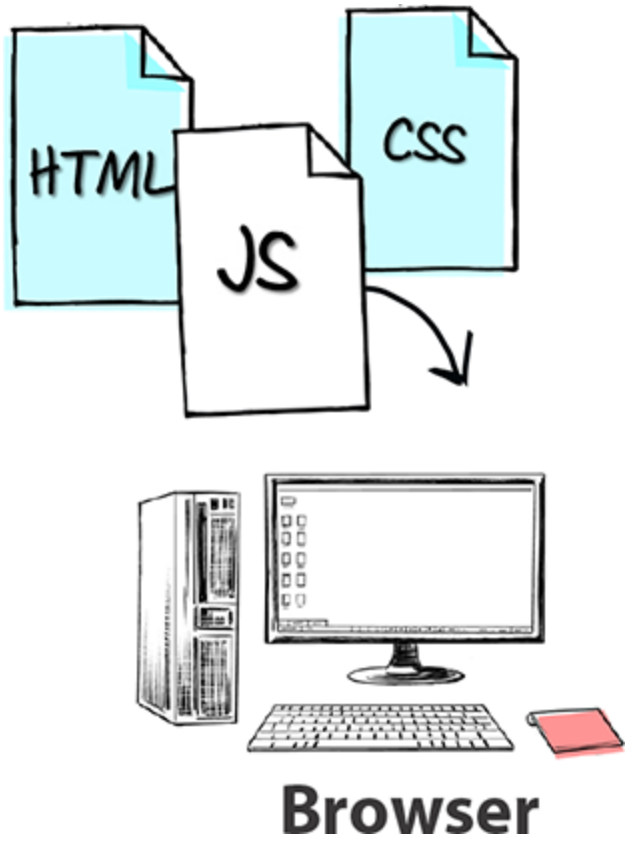
```
<html>
<head>
<title>Ice Cream</title>
<script>
 let x = 49;
</script>
</head>
<body>
<h1>Ice Cream Flavors</h1>
<h2><em>49 flavors</em></h2>
<p>All your favorite
flavors!</p>
</body>
</html>
```

1 Writing

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

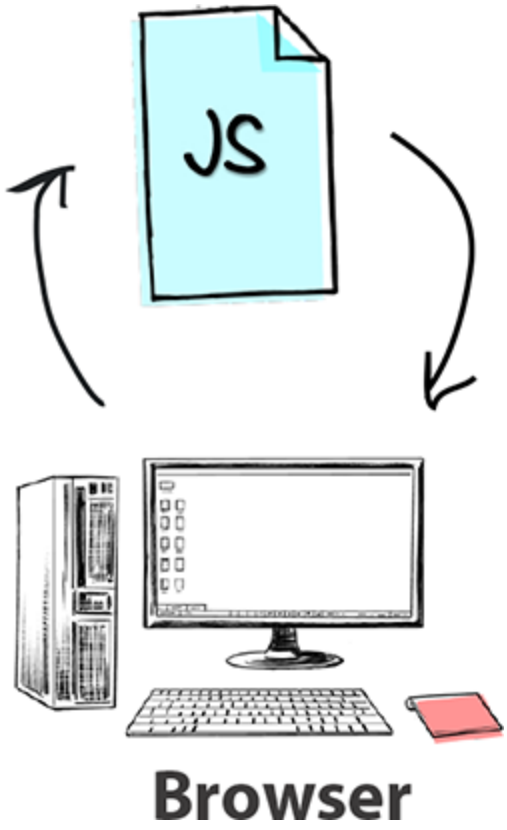
NOTE

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



❷ Loading

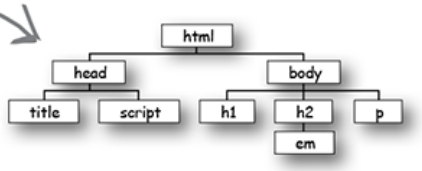
Point your browser to your page, just as you always do. The browser sees the code and begins parsing it immediately, getting ready to execute it. Note that like with HTML and CSS, if the browser sees errors in your code, it will do its best to keep moving and reading more JavaScript, HTML, and CSS. The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



3 Executing

The browser starts executing your code as soon as it encounters the code in your page, and continues executing it for the lifetime of your page. Some code executes once and is not executed again (until you reload the page); other code is executed whenever the user does something, like click a button or move the mouse. You'll see examples of both in this book.

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 web 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. 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 HTML 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.

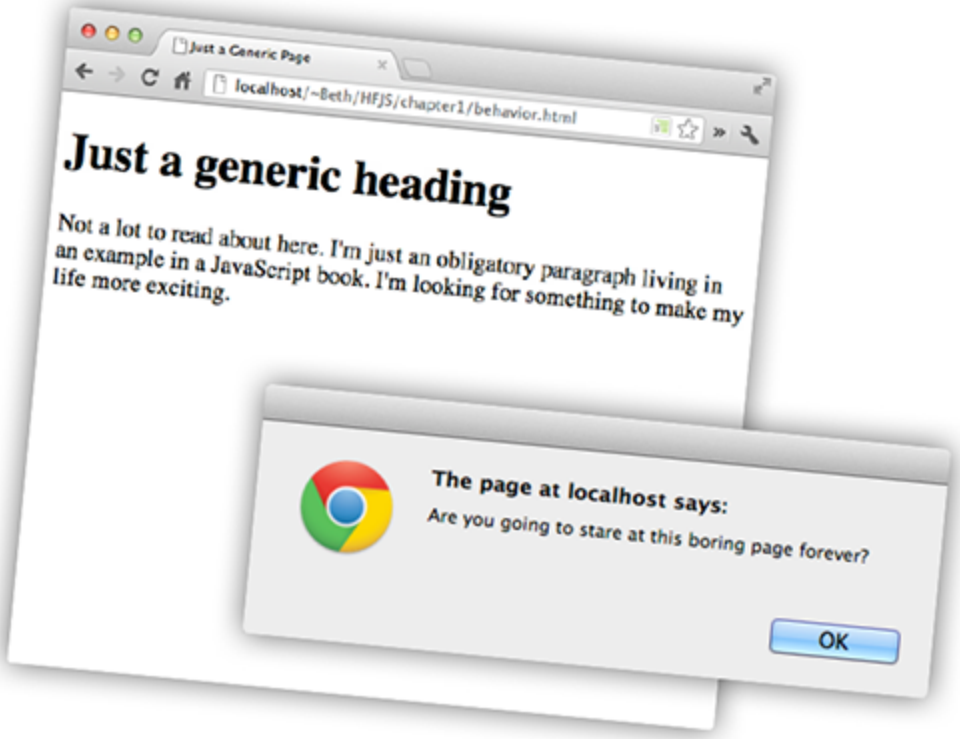
And we've written some JavaScript code inside it.

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.

```
<!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>
```

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 Page
localhost/~Beth/HFJS/chapter1/behavior.html
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?
OK

RELAX

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?");
}
```

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

Perhaps a way to count five seconds of time? Hint: 1,000 milliseconds = 1 second.

Clearly a way to alert the user with a message.

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: I've heard JavaScript is a bit of an underpowered language. Is it?

A: JavaScript certainly wasn't a powerlifter 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 the best way to create dynamic web pages?

A: JavaScript is **the** language for programming in the browser. There are variants of JavaScript, like TypeScript, but the TypeScript code you write will be translated into JavaScript before it runs in the browser. With today's super-fast JavaScript environments and sophisticated APIs, JavaScript is here to stay and is the standard for programming in the browser.

Q: My friend is using JavaScript inside a music application...or at least he says he is. Is that possible?

A: Yes! JavaScript has broken 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.

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 toward tasks like prototyping, interactive coding, and flexibility. This was the case with early JavaScript, and it 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...

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.

Toward 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 its 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? It created its own scripting language soon after Netscape did, named, um, JScript, and it was, um, quite similar to JavaScript. And so began the browser wars and a frustrating time for developers.

JavaScript 2015

After all that confusion, JavaScript finally grew up. ECMAScript, an official language definition for JavaScript was born, and now serves as the standard language definition for all JavaScript implementations (in and out of the browser).

By 2015, JavaScript had finally come of age and gained the respect of professional developers. Having a solid standard helped, along with serious commitment from browser makers like Google, which pushed JavaScript into the professional limelight with Google Maps and other complex browser-based applications.

With all the new attention, many of the best programming language minds focused on improving JavaScript's interpreters and made improvements to its runtime performance. After JavaScript 2015, a major language update, we switched from official version numbers for language releases to yearly updates.

JavaScript 2024

Today, JavaScript is *the* language of the web. Pre-compilers make interpreting JavaScript code in webpages blindingly fast. Syntax checkers, syntax-aware code editors, and robust browser-based debugging tools have professionalized web development. JavaScript is one of the most popular languages in the world and has been implemented in environments as diverse as embedded systems, web servers, and music-making applications.

The language is fairly mature at this point. Most language updates are small and incorporated quickly into browsers. Numerous JavaScript libraries and frameworks mean that you can do almost anything with JavaScript that you can do in any other language.

After a strange and contentious history, JavaScript has made it!

LOOK HOW EASY IT IS TO WRITE JAVASCRIPT

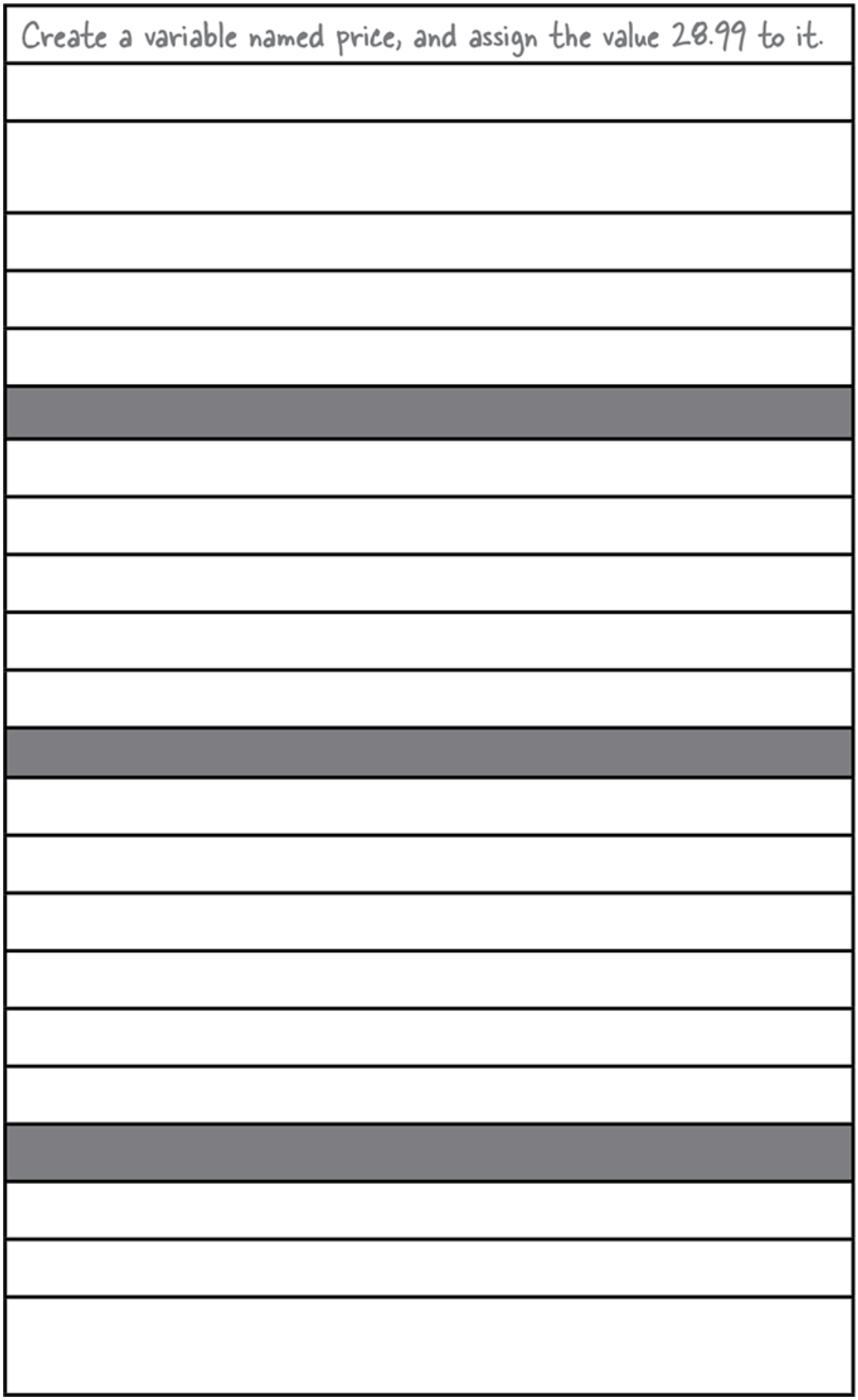
You don't know JavaScript yet, but we bet you can make some good guesses about how JavaScript code works. Take a look at each line of code and see if you can guess what it does. Write in your answers below. We've done one for you to get you started. If you get stuck, the answers are on the next page.

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

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

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

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



→ Solution in **“Look how easy it is to write JavaScript Solution”**

LOOK HOW EASY IT IS TO WRITE JAVASCRIPT SOLUTION

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 and see if you can guess what it does. Here are our answers.

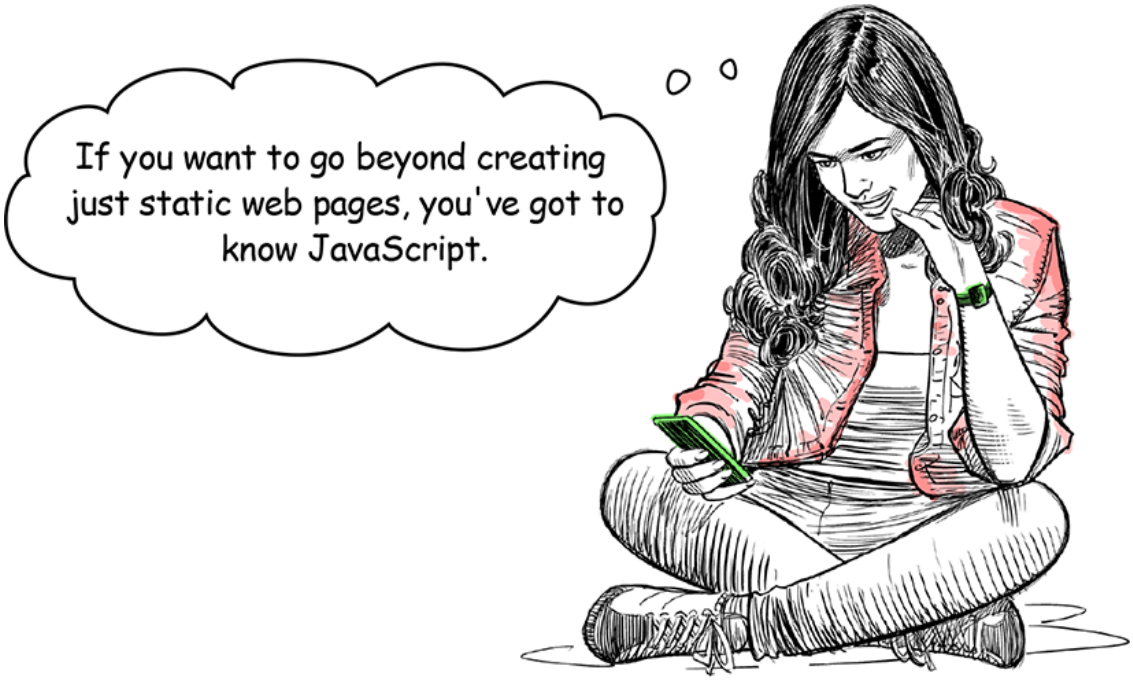
```
let price = 28.99;
let discount = 10;
let total =
      price - (price * (discount / 100));
if (total > 25) {
   freeShipping();
}

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

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

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

Create a variable named price, and assign the value 28.99 to it.
Create a variable named discount, and assign the value 10 to it.
Compute a new price by applying a discount and then assign it to the variable total.
Compare the value in the variable total to 25. If it's greater...
...then do something with freeShipping.
End the if statement.
Create a variable named count, and assign the value 10 to it.
As long as the variable count is greater than 0...
...do some juggling, and...
...reduce the value of count by 1 each time.
End the while loop.
Create a constant named dog with a name and weight.
If the dog's weight is greater than 30...
...alert "WOOF WOOF" to the browser's web page.
Otherwise...
...alert "woof woof" to the browser's web page.
End the if/else statement.
Create a variable, circleRadius, and assign the value 20 to it.
Create a variable named circleArea...
...and assign the result of this expression to it (1256.637061435917 3).



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.

NOTE

Knowing JavaScript might increase the size of your paycheck too!

So much so, in fact, you might actually start thinking of your pages as applications (or even experiences!) rather than mere pages.

Now, you might be saying, “I already 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 and CSS to date, you should know that there is something fundamentally different about learning a programming language.

With HTML and CSS, what you’re doing is largely declarative—for instance, you’re declaring, say, that some text is a paragraph or that all the HTML 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'll need to be able to describe things like "calculate the user's score by summing up all the correct answers" or "do this action 10 times" or "when the user clicks on that button play the you-have-won sound" or even "go off and get the current temperature, 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:

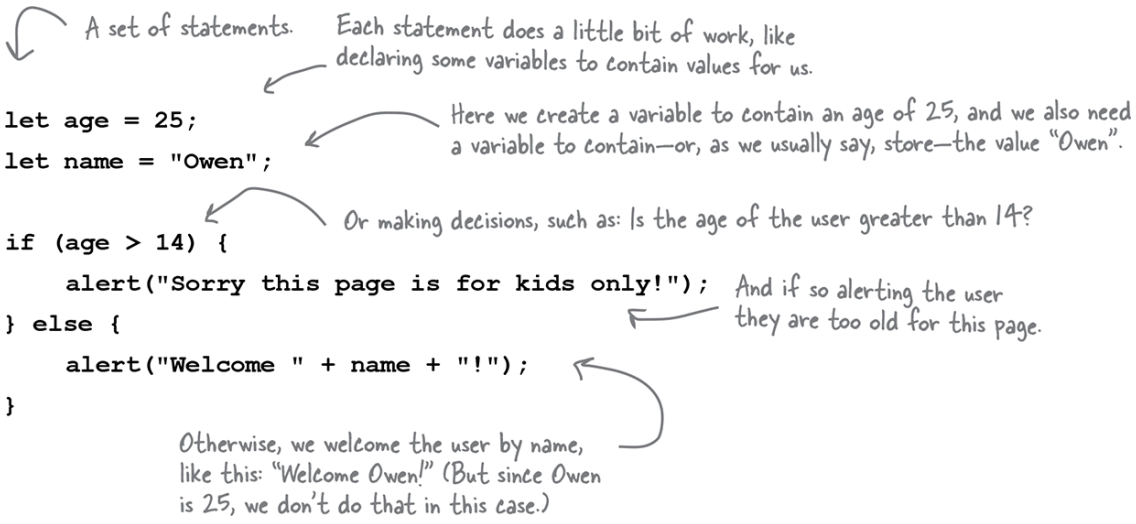
```
<h1 class="drink">Mocha Caffe Latte</h1>
<p>Espresso, steamed milk, and chocolate syrup,
just the way you like it.</p>
```

NOTE

We're using HTML to mark up text to create structures like headings and paragraphs. Like, "I need a large heading called Mocha Caffe Latte; it's a heading for a drink. And I need a paragraph after that."

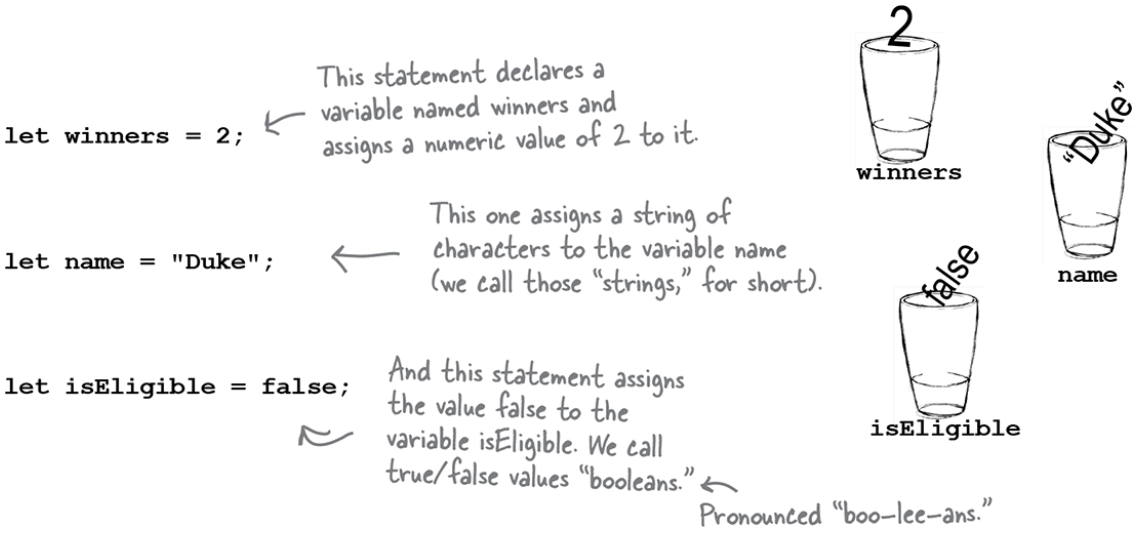
CSS is a bit different. With CSS, you're writing a set of **rules**, where each rule selects elements in the page and then specifies a set of styles for those elements:

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

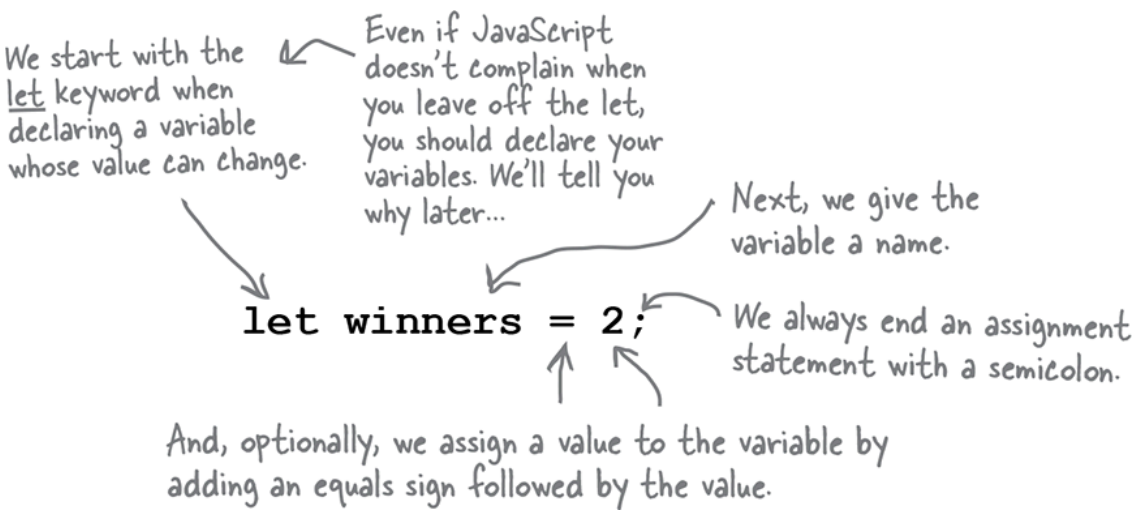


Variables and values

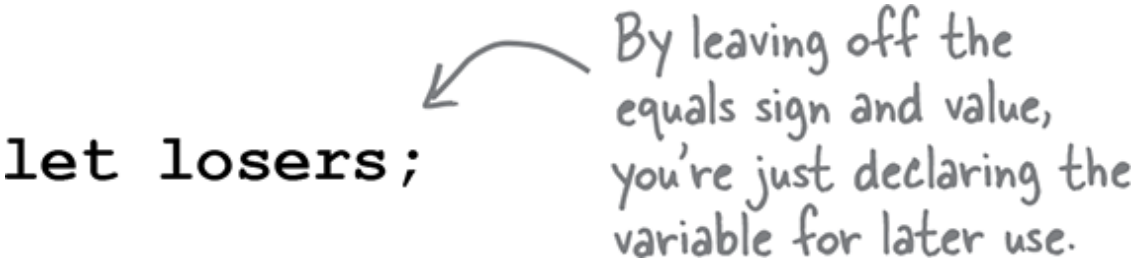
You might have noticed that JavaScript statements usually involve variables. Variables are used to store values. What kinds of values? Here are a few examples:



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 closer look at how to declare a variable:



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:



BRAIN POWER

If you declare a variable without a value, such as:

what value do you think JavaScript assigns to this variable?

Constants, another kind of variable

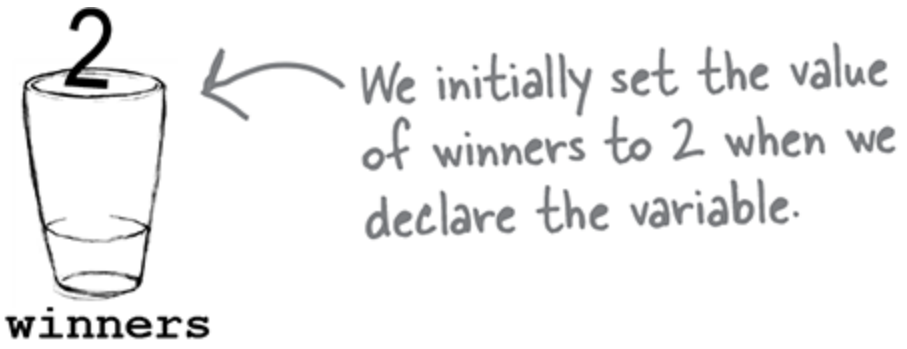
So far, we've used the keyword `let` to declare our variables. And that's typically what we want to do with variables whose values can *vary*, or in other words, change their value over time. For instance, if we use `let` to declare the variable `winners` and assign it the value 2, we can change the value in `winners` later to be 3 if another winner comes along:

```
let winners = 2;
```

Later on, more people play the game and another person wins. So, we change the value of winners.

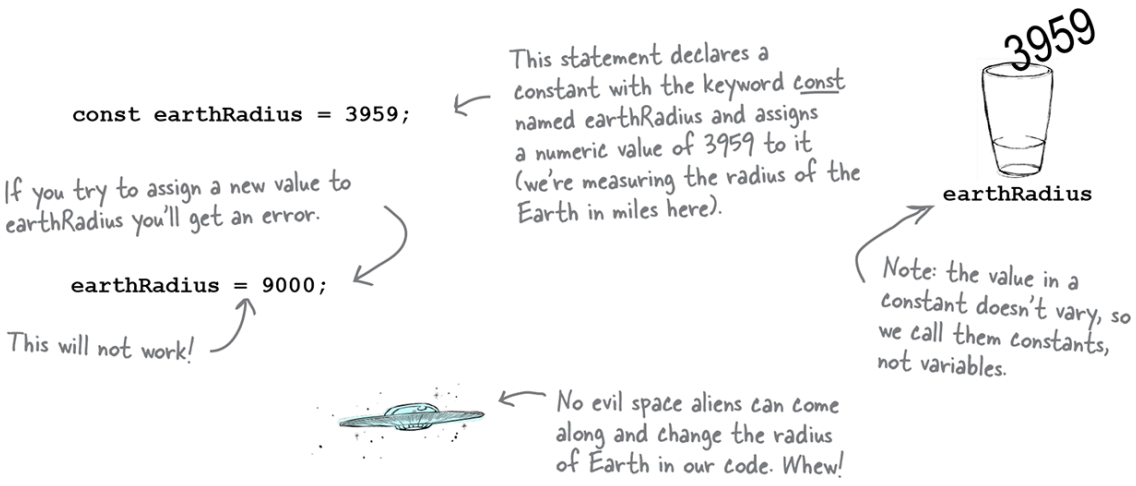
```
winners = 3;
```

Notice we don't redeclare winners here, we only assign a new value. If you try to redeclare winners, you'll get an error.

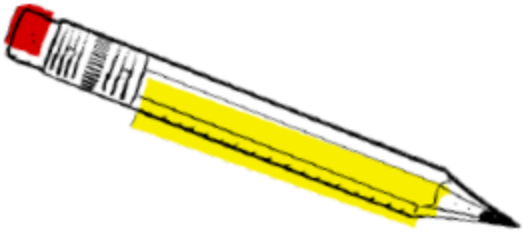


Sometimes, however, we do not want the values in our variables to vary at all. There are situations in which we might want to give a name to a value that we'll use in our code, but we don't ever want that value to change. Here's a good example: the radius of planet Earth. It might be handy to assign this value to a variable so we can use `earthRadius` instead of the number in our code. We don't want anyone to come along

and change this value accidentally, so how can we make sure the value of `earthRadius` never changes? We can use a *constant* instead of a variable, like this:



SHARPEN YOUR PENCIL



Identify the declarations below that you think are best suited for let and for const. We've done a couple for you. Check our solution at the end of the chapter before you go on.

```
const ____  DISTANCE_TO_MOON = 238900;
let   ____  last_guess = 0;
      ____  SCREEN_WIDTH = 1024;
      ____  counter = 100;
      ____  firstUSPresident = "Washington";
      ____  fluxCapacitorReading = "System Normal";
```

→ **Solution in "Sharpen your pencil Solution"**

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:

❶ **Start your variable names with a letter or an underscore.**

❷ **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 **let** 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 later in the book, but here's a list to take a quick look at:

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


STOP

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 and constants.

Q:Why aren't we using var to declare our variables?

A: Earlier versions of JavaScript used a single keyword, var, to declare all variables. Now, the var keyword is no longer recommended and has been largely replaced by let and const. These keywords work slightly differently to var and have some benefits over var that we'll come back to later.

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 variable, constant, keyword, and function names, and pretty much everything else, too. So pay attention to your use of upper- and lowercase.

How to avoid those embarassing naming mistakes

You've got a lot of flexibility in choosing your variable and constant names, so here are a few Webville tips to make 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, but 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. 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). For constants, use an underscore between words.

Use variable names 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 let when declaring a variable and const when declaring a constant.

- 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.

```
let width;
```

- JavaScript, unlike HTML markup, is case sensitive, meaning upper- and lowercase matters. The variable `counter` is different from the

variable `Counter`.

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 solution at the end of the chapter to see if you found them all.

A

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

if (toldJoke == true) {
    Alert($punchline);
} else
    alert(joke);
}
joke = "Knock, knock. Who's there? JavaScript....";
```

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

B

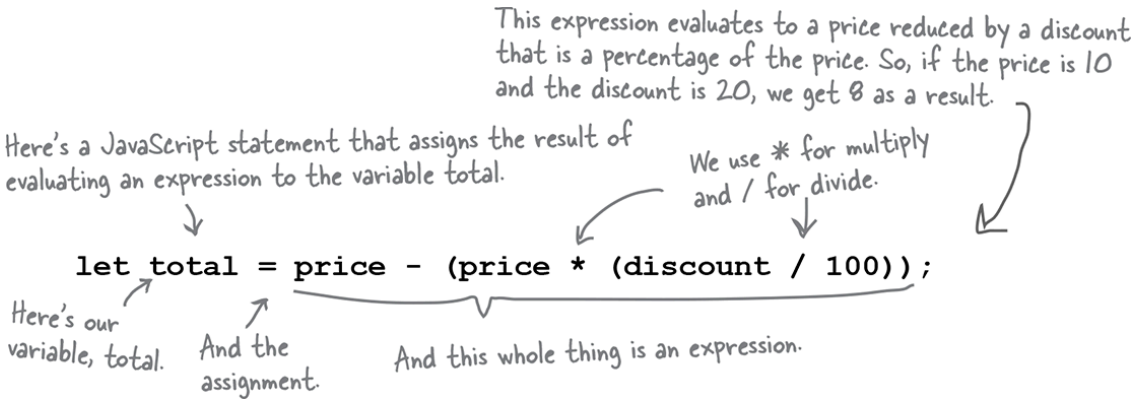
```
\\ Movie Night
let zip code = 98104;
const joe'sFavoriteMovie = Forbidden Planet;
let movieTicket$    =    9;

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

Solution in "BE the Browser Solution"

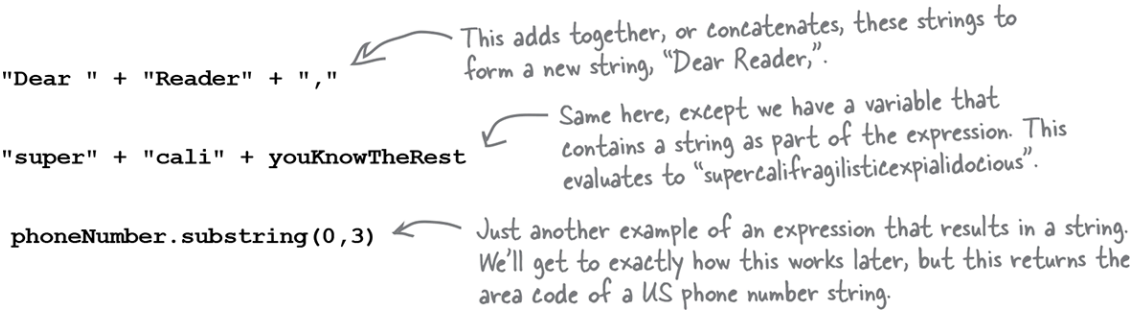
Express yourself

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:

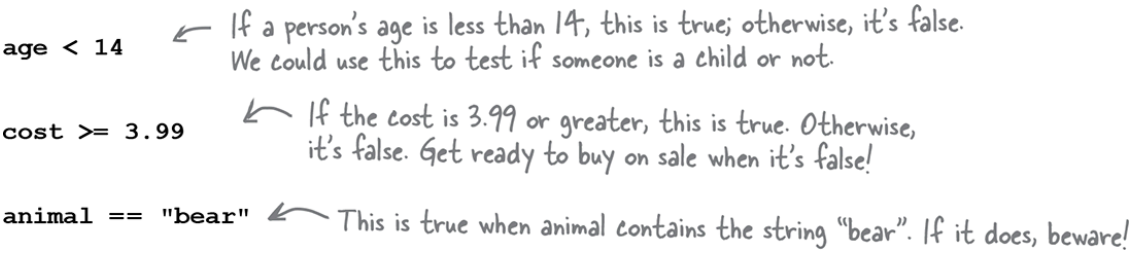


If you've ever taken a math class, balanced your checkbook, or done your taxes, we're sure these kinds of numeric expressions are nothing new.

There are also string expressions; here are a few:

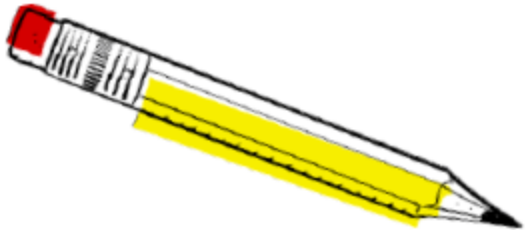


We also have expressions that evaluate to **true** or **false**, otherwise known as boolean expressions. Work through each of these to see how you get true or false from them:



Expressions can evaluate to a few other types, too; 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!

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 3,300. 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!

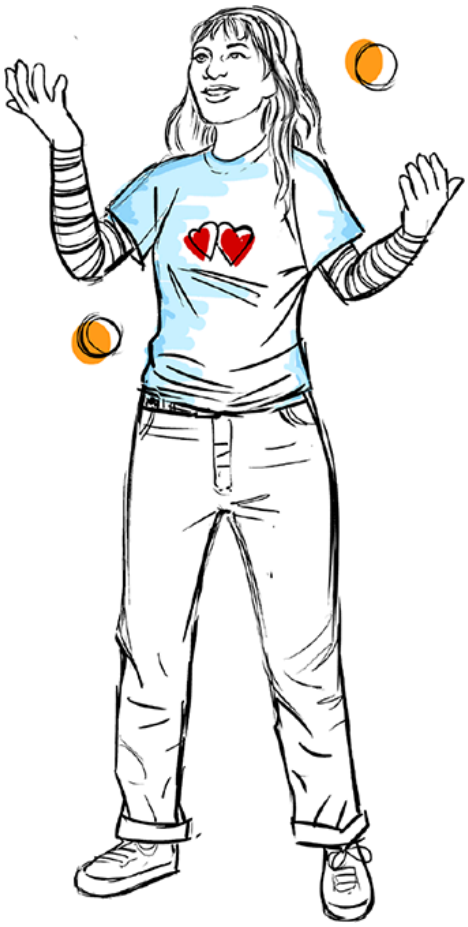
```
1000 + "108"
```

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

Solution in "Sharpen your pencil Solution"

Did you notice that the = operator is used in assignments, while the == operator tests for equality? That is, we use one equals sign to assign values to variables. We use two equals 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:

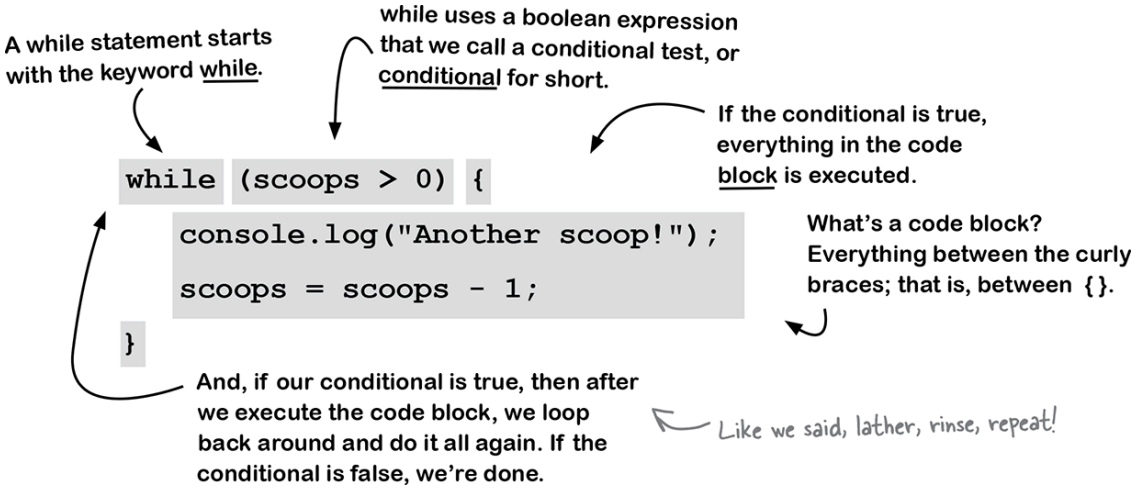
Lather, rinse, repeat...

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**, **for of**, 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:



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:

```
let scoops = 5;
while (scoops > 0) {
   console.log("Another scoop!");
   scoops = scoops - 1;
```

```
}
console.log("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:

Because the conditional is true, we start executing the block of code. The first statement in the body writes the string “Another scoop!” to the browser’s console:*

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

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

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

Once again, we write the string “Another scoop!” to the browser:

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

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

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

Once again, we write the string "Another scoop!" to the browser:

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:

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

And continues...

```
let scoops = 5;
while (scoops > 0) {
   console.log("Another scoop!");
   scoops = scoops - 1;
}
console.log("Life without ice cream isn't the same");
```

Until the last time…this time, something's different. Scoops is zero, 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:

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

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 the code inside its curly braces (its *code block*) only if a conditional test is true. Here's an example:

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

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:

THERE ARE NO DUMB QUESTIONS

Q: What exactly is a code block?

A: Syntactically, a code block (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 in sequence. 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 a variable with a value that isn't a boolean, like a string or a number. 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 (nonempty) 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 program to produce the output shown below? Check your answer at the end of the chapter before you go on.

Solution in "Code Magnets Solution"

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 a couple 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.

NOTE

We're using these two 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 called `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 in old code.

Use the console.

Every JavaScript environment also has a console that can log messages from your code. We've been using this in our examples so far. 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 the console log in just a moment). You can view `console.log` as a great tool to get started with JavaScript and 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.

NOTE

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 and change its content, and even alter its structure and style! This all happens by making use of your browser's *document object model*. As you'll see, this is the best way to communicate with your users. 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.

NOTE

This is what we're working toward. When you get there, you'll be able to read, alter, and manipulate your pages 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.

Solution in "Who Does What? Solution"

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:

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" separated 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'll come 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 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 a console is something all modern browsers supply, but it isn't part of any formal specification.

Q: Uh, the 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.

We're going to show you how to access the console in the Chrome browser (version 123) on macOS, and we'll put instructions on how to access the console in all the major browsers online at

https://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.

NOTE

Note: you don't need to type the Howdy code in. We're just showing where the console is. We'll start typing in code in just a sec...

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, and 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 rootbeer."

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

```
    }
}
```

BRAIN POWER

There's 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.

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 at the beginning of this chapter that you add JavaScript to your HTML just like you add CSS; that is, you just add it in-line with the appropriate `<script>` tags around it.

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.

TEST DRIVE

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

NOTE

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

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 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 VS Code or WebStorm, you can use that too.

Save the file as "index.html".

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.

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.

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):

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...

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

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

You 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".

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 src="code.js", we're assuming that the code file is in the same directory as the HTML file.

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:

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

WATCH IT!

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">
  let x = "quick hack";
</script>
```

WRONG

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, 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 scripting language." 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 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 hap-

pen—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 "rough spots."

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.

BULLET POINTS

- JavaScript is used to add **behavior** to web pages.
- JavaScript executes fast in modern browsers.
- 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 **let** keyword to declare a new variable and the assignment operator, **=**, to assign a value to it.
- Use **const** to assign a value that shouldn't change.
- The value of a constant doesn't vary, so we call them constants, not variables.
- There are just a few rules and guidelines for naming JavaScript variables and constants, 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.
- You can group statements together into a code **block** by enclosing them in curly braces.
- 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 them.
- JavaScript is most commonly found adding behavior to web pages, but it's also used to script many creative applications and is used as a server-side programming language with Node.js.

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

ACROSS

1. Variables are used to store these.

4. Use _____ to troubleshoot your code.

7. Today's JavaScript runs a lot _____ than it used to.

8. There are 99 _____ of rootbeer on the wall.

9. To link to an external JavaScript file from HTML, you need the _____ attribute for your <script> element.

10. Each time through a loop, we evaluate a _____ expression.

13. The if/else statement is used to make a _____.

14. All JavaScript statements end with a _____.

16. You put your JavaScript inside a _____ element.

DOWN

2. You can concatenate _____ together with the + operator.

3. Store values that don't change in this.

5. 3 + 4 is an example of an _____.

6. JavaScript adds _____ to your web pages.

9. Each line of JavaScript code is called a _____.

10. To avoid embarrassing naming mistakes, use _____ case.

11. Do things more than once in a JavaScript program with the _____ loop.

12. JavaScript variable names are _____ sensitive.

15. To declare a variable, use this keyword.

Solution in "JavaScript cross Solution"

SHARPEN YOUR PENCIL SOLUTION

From "Sharpen your pencil"

Identify the declarations below that you think are best suited for let and for const. Here's our solution.

BE THE BROWSER SOLUTION

From "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. Here's our solution.

SHARPEN YOUR PENCIL SOLUTION

From "Sharpen your pencil"

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.

CODE MAGNETS SOLUTION

From "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 program to produce the output shown below? Here's our solution.

JAVASCRIPT CROSS SOLUTION

From "javascript cross"

WHO DOES WHAT? SOLUTION

From "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. Here's our solution.