2

BASIC JAVASCRIPT INSTRUCTIONS

In this chapter, you will start learning to read and write JavaScript. You will also learn how to give a web browser instructions you want it to follow.

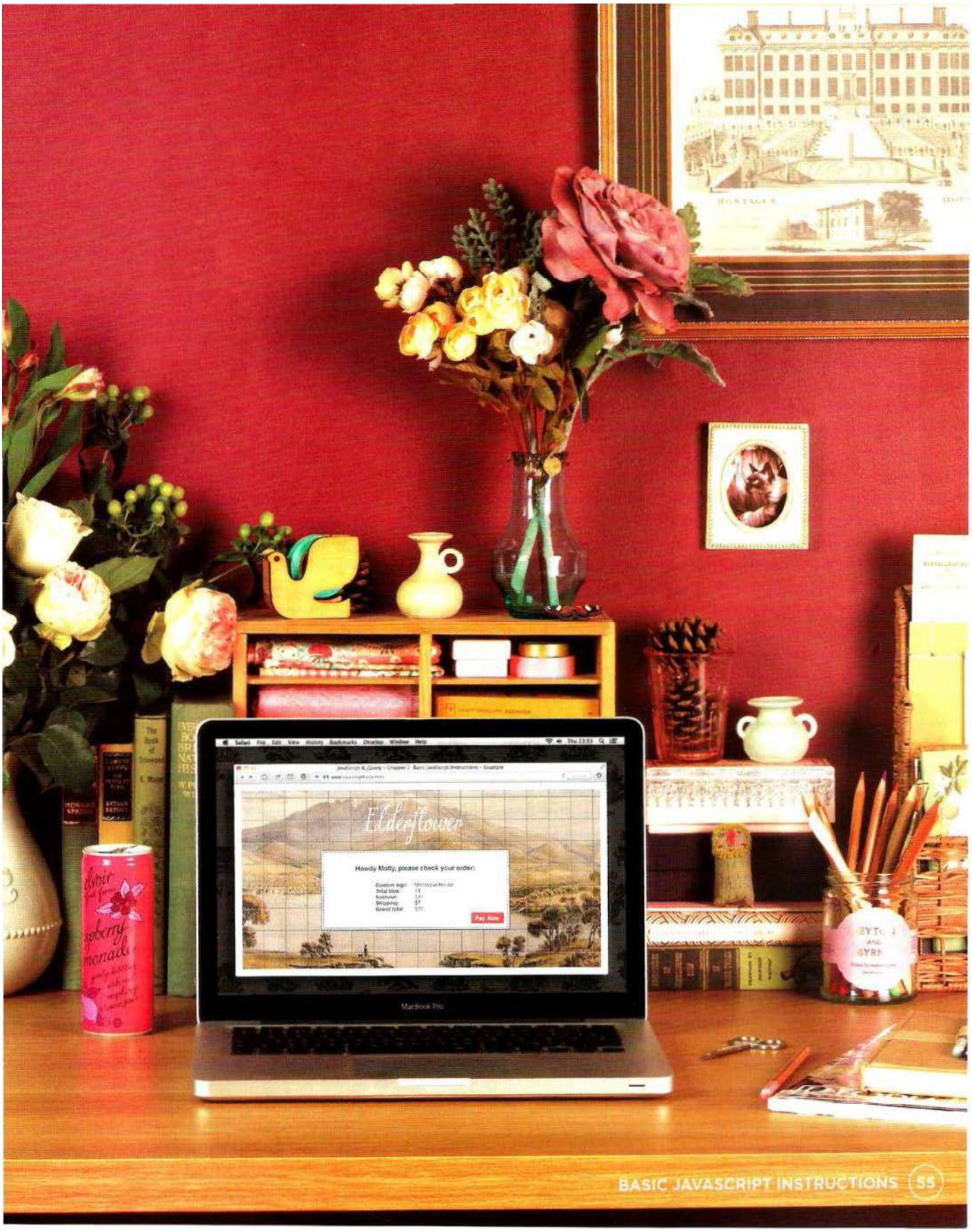
THE LANGUAGE: SYNTAX AND GRAMMAR

Like any new language, there are new words to learn (the vocabulary) and rules for how these can be put together (the grammar and syntax of the language).

GIVING INSTRUCTIONS: FOR A BROWSER TO FOLLOW

Web browsers (and computers in general) approach tasks in a very different way than a human might. Your instructions need to reflect how computers get things done.

We will start with a few of the key building blocks of the language and look at how they can be used to write some very basic scripts (consisting of a few simple steps) before going on to look at some more complex concepts in subsequent chapters.


Elderflower
Howdy Molly, please check your order:
Pay Now
MacBook Pro

STATEMENTS

A script is a series of instructions that a computer can follow one-by-one. Each individual instruction or step is known as a **statement**. Statements should end with a semicolon.

We will look at what the code on the right does shortly, but for the moment note that:

- Each of the lines of code in **green** is a **statement**.
- The **pink** curly braces indicate the start and end of a **code block**. (Each code block could contain many more statements.)
- The code in **purple** determines which code should run (as you will see on p149).

```
var today = new Date();
var hourNow = today.getHours();
var greeting;

if (hourNow > 18) {
  greeting = 'Good evening';
} else if (hourNow > 12) {
  greeting = 'Good afternoon';
} else if (hourNow > 0) {
  greeting = 'Good morning';
} else {
  greeting = 'Welcome';
}
document.write(greeting);
```

JAVASCRIPT IS CASE SENSITIVE

JavaScript is case sensitive so `hourNow` means something different to `HourNow` or `HOURNOW`.

STATEMENTS ARE INSTRUCTIONS AND EACH ONE STARTS ON A NEW LINE

A statement is an individual instruction that the computer should follow. Each one should start on a new line and end with a semicolon. This makes your code easier to read and follow.

The semicolon also tells the JavaScript interpreter when a step is over, indicating that it should move to the next step.

STATEMENTS CAN BE ORGANIZED INTO CODE BLOCKS

Some statements are surrounded by curly braces; these are known as **code blocks**. The closing curly brace is not followed by a semicolon.

Above, each code block contains one statement related to what the current time is. Code blocks will often be used to group together many more statements. This helps programmers organize their code and makes it more readable.

COMMENTS

You should write **comments** to explain what your code does. They help make your code easier to read and understand. This can help you and others who read your code.

```
/* This script displays a greeting to the user based upon the current time.
   It is an example from JavaScript & jQuery book */

var today = new Date();          // Create a new date object
var hourNow = today.getHours();  // Find the current hour
var greeting;

// Display the appropriate greeting based on the current time
if (hourNow > 18) {
  greeting = 'Good evening';
} else if (hourNow > 12) {
  greeting = 'Good afternoon';
} else if (hourNow > 0) {
  greeting = 'Good morning';
} else {
  greeting = 'Welcome';
}
document.write(greeting);
```

JavaScript code is **green**
Multi-line comments are **pink**
Single-line comments are **gray**

MULTI-LINE COMMENTS

To write a comment that stretches over more than one line, you use a **multi-line** comment, starting with the /* characters and ending with the */ characters. Anything between these characters is not processed by the JavaScript interpreter.

Multi-line comments are often used for descriptions of how the script works, or to prevent a section of the script from running when testing it.

SINGLE-LINE COMMENTS

In a **single-line** comment, anything that follows the two forward slash characters // on that line will not be processed by the JavaScript interpreter. Single-line comments are often used for short descriptions of what the code is doing.

Good use of comments will help you if you come back to your code after several days or months. They also help those who are new to your code.

WHAT IS A VARIABLE?

A script will have to temporarily store the bits of information it needs to do its job. It can store this data in **variables**.

When you write JavaScript, you have to tell the interpreter every individual step that you want it to perform. This sometimes involves more detail than you might expect.

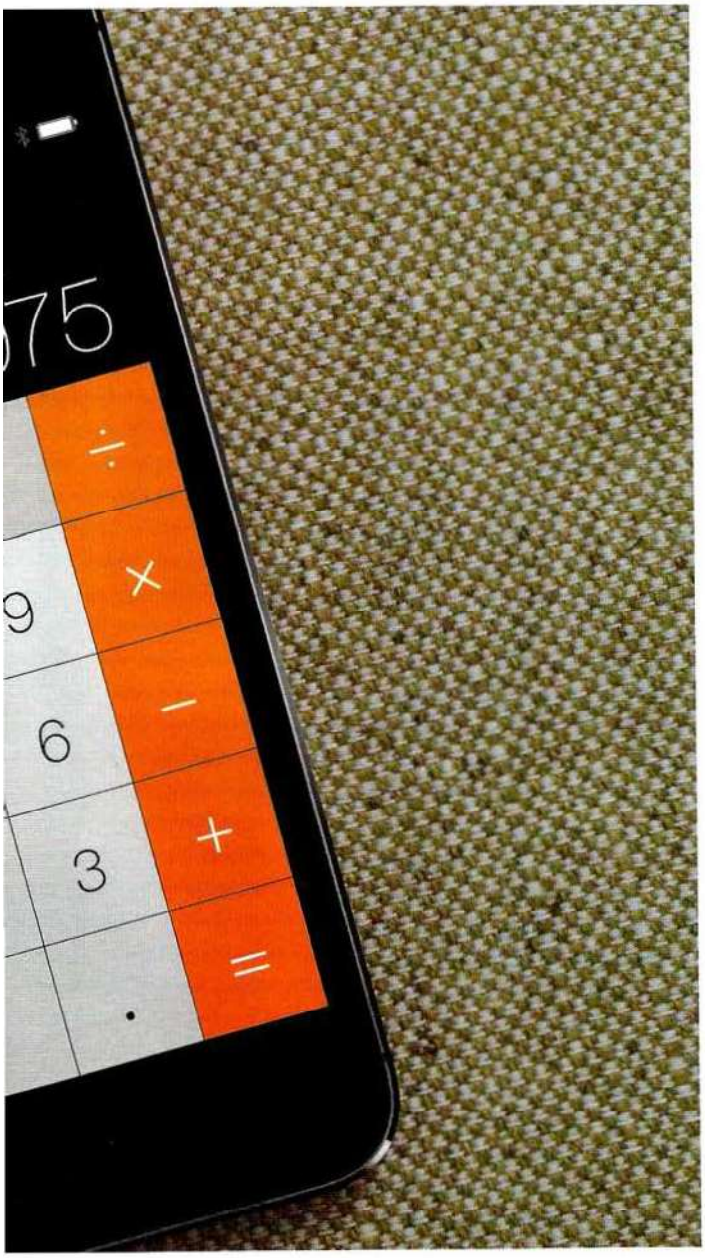
Think about calculating the area of a wall; in math the area of a rectangle is obtained by multiplying two numbers:

width × *height* = *area*

You may be able to do calculations like this in your head, but when writing a script to do this calculation, you need to give the computer very detailed instructions. You might tell it to perform the following four steps in order:

1. Remember the value for *width*
2. Remember the value for *height*
3. Multiply *width* by *height* to get the *area*
4. Return the result to the user

In this case, you would use variables to "remember" the values for *width* and *height*. (This also illustrates how a script contains very explicit instructions about exactly what you want the computer to do.)
You can compare variables to short-term memory, because once you leave the page, the browser will forget any information it holds.



A variable is a good name for this concept because the data stored in a variable can change (or vary) each time a script runs.

No matter what the dimensions of any individual wall are, you know that you can find its *area* by multiplying the *width* of that wall by its *height*. Similarly, scripts often need to achieve the same goal even when they are run with different data, so variables can be used to represent values in your scripts that are likely to change. The result is said to be **calculated** or **computed** using the data stored in the variables.

The use of variables to represent numbers or other kinds of data is very similar to the concept of algebra (where letters are used to represent numbers). There is one key difference, however. The equals sign does something very different in programming (as you will see on the next two pages).

VARIABLES: HOW TO DECLARE THEM

Before you can use a variable, you need to announce that you want to use it. This involves creating the variable and giving it a name. Programmers say that you **declare** the variable.

```
var quantity;
```

VARIABLE KEYWORD — VARIABLE NAME

var is an example of what programmers call a **keyword**. The JavaScript interpreter knows that this keyword is used to create a variable.

In order to use the variable, you must give it a name. (This is sometimes called an identifier.) In this case, the variable is called **quantity**.

If a variable name is more than one word, it is usually written in camelCase. This means the first word is all lowercase and any subsequent words have their first letter capitalized.

VARIABLES: HOW TO ASSIGN THEM A VALUE

Once you have created a variable, you can tell it what information you would like it to store for you. Programmers say that you **assign a value** to the variable.

You can now use the variable by its name. Here we set a value for the variable called **quantity**. Where possible, a variable's name should describe the kind of data the variable holds.

The equals sign (=) is an **assignment operator**. It says that you are going to assign a value to the variable. It is also used to update the value given to a variable (see p68).

Until you have assigned a value to a variable, programmers say the value is **undefined**.

Where a variable is declared can have an effect upon whether the rest of the script can use it. Programmers call this the **scope** of a variable and it is covered on p98.

DATA TYPES

JavaScript distinguishes between numbers, strings, and `true` or `false` values known as Booleans.

NUMERIC DATA TYPE

The numeric data type handles numbers.

```
0.75
```

For tasks that involve counting or calculating sums, you will use numbers 0-9. For example, five thousand, two hundred and seventy-two would be written 5272 (note there is no comma between the thousands and the hundreds). You can also have negative numbers (such as -23678) and decimals (three quarters is written as 0.75).

STRING DATA TYPE

The strings data type consists of letters and other characters.

```
'Hi, Ivy!'
```

Note how the string data type is enclosed within a pair of quotes. These can be single or double quotes, but the opening quote must match the closing quote.

Strings can be used when working with any kind of text. They are frequently used to add new content into a page and they can contain HTML markup.

BOOLEAN DATA TYPE

Boolean data types can have one of two values: `true` or `false`.

```
true
```

It might seem a little abstract at first, but the Boolean data type is actually very helpful.

You can think of it a little like a light switch - it is either on or off. As you will see in Chapter 4, Booleans are helpful when determining which part of a script should run.

Numbers are not only used for things like calculators; they are also used for tasks such as determining the size of the screen, moving the position of an element on a page, or setting the amount of time an element should take to fade in.

In addition to these three data types, JavaScript also has others (arrays, objects, `undefined`, and `null`) that you will meet in later chapters.

Unlike some other programming languages, when declaring a variable in JavaScript, you do not need to specify what type of data it will hold.

USING A VARIABLE TO STORE A NUMBER

JAVASCRIPT c02/js/numeric-variable.js

```
var price;
var quantity;
var total;

price = 5;
quantity = 14;
total = price * quantity;

var el = document.getElementById('cost');
el.textContent = '$' + total;
```

HTML c02/numeric-variable.html

```
<h1>Elderflower</h1>
<div id="content">
  <h2>Custom Signage</h2>
  <div id="cost">Cost: $5 per tile</div>
  <img src="images/preview.jpg" alt="Sign" />
</div>
<script src="js/numeric-variable.js"></script>
```

RESULT

Here, three variables are created and values are assigned to them.

- `price` holds the price of an individual tile
- `quantity` holds the number of tiles a customer wants
- `total` holds the total cost of the tiles

Note that the numbers are not written inside quotation marks. Once a value has been assigned to a variable, you can use the variable name to represent that value (much like you might have done in algebra). Here, the total cost is calculated by multiplying the price of a single tile by the number of tiles the customer wants.

The result is then written into the page on the final two lines. You see this technique in more detail on p194 and p216.
The first of these two lines finds the element whose `id` attribute has a value of `cost`, and the final line replaces the content of that element with new content.

Note: There are many ways to write content into a page, and several places you can place your script. The advantages and disadvantages of each technique are discussed on p226. This technique will not work in IE8.

USING A VARIABLE TO STORE A STRING

For the moment, concentrate on the first four lines of JavaScript. Two variables are declared (`username` and `message`), and they are used to hold strings (the user's name and a message for that user).

The code to update the page (shown in the last four lines) is discussed fully in Chapter 5. This code selects two elements using the values of their `id` attributes. The text in those elements is updated using the values stored in these variables.

Note how the string is placed inside quote marks. The quotes can be single or double quotes, but they must match. If you start with a single quote, you must end with a single quote, and if you start with a double quote, you must end with a double quote:

- ✔ `"hello"` ✘ `"hello'`
- ✔ `'hello'` ✘ `'hello"`

Quotes should be straight (not curly) quotes:

- ✔ " " ✘ “ ”
- ✔ ' ' ✘ ‘ ’

Strings must always be written on one line:

- ✔ `'See our upcoming range'`
- ✘ `'See our upcoming range'`

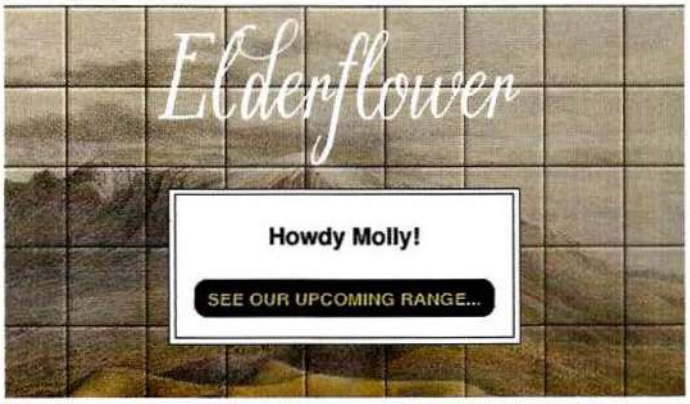
c02/js/string-variable.js **JAVASCRIPT**

```
var username;
var message;
username = 'Molly';
message = 'See our upcoming range';

var elName = document.getElementById('name');
elName.textContent = username;
var elNote = document.getElementById('note');
elNote.textContent = message;
```

c02/string-variable.html **HTML**

```
<h1>Elderflower</h1>
<div id="content">
  <div id="title">Howdy
    <span id="name">friend</span>!</div>
  <div id="note">Take a look around...</div>
</div>
<script src="js/string-variable.js"></script>
```

RESULT



USING QUOTES INSIDE A STRING

JAVASCRIPT c02/js/string-with-quotes.js

```
var title;
var message;
title = "Molly's Special Offers";
message = '<a href=\"sale.html\">25% off!</a>';

var elTitle = document.getElementById('title');
elTitle.innerHTML = title;
var elNote = document.getElementById('note');
elNote.innerHTML = message;
```

HTML c02/string-with-quotes.html

```
<h1>Elderflower</h1>
<div id="content">
  <div id="title">Special Offers</div>
  <div id="note">Sign-up to receive personalized
    offers!</div>
</div>
<script src="js/string-with-quotes.js"></script>
```

RESULT

Sometimes you will want to use a double or single quote mark *within* a string.

Because strings can live in single or double quotes, if you just want to use double quotes in the string, you could surround the entire string in single quotes.

If you just want to use single quotes in the string, you could surround the string in double quotes (as shown in the third line of this code example).

You can also use a technique called **escaping** the quotation characters. This is done by using a **backwards slash** (or "backslash") *before* any type of quote mark that appears within a string (as shown on the fourth line of this code sample). The backwards slash tells the interpreter that the following character is part of the string, rather than the end of it.

Techniques for adding content to a page are covered in Chapter 5. This example uses a property called `innerHTML` to add HTML to the page. In certain cases, this property *can* pose a security risk (discussed on p228 - p231).

USING A VARIABLE TO STORE A BOOLEAN

A Boolean variable can only have a value of `true` or `false`, but this data type is very helpful.

In the example on the right, the values `true` or `false` are used in the `class` attributes of HTML elements. These values trigger different CSS class rules: `true` shows a check, `false` shows a cross. (You learn how the `class` attribute is set in Chapter 5.)

It is rare that you would want to write the words `true` or `false` into the page for the user to read, but this data type does have two very popular uses:

First, Booleans are used when the value can only be `true`/`false`. You could also think of these values as `on`/`off` or `0`/`1`: `true` is equivalent to `on` or `1`, `false` is equivalent to `off` or `0`

Second, Booleans are used when your code can take more than one path. Remember, different code may run in different circumstances (as shown in the flowcharts throughout the book).

The path the code takes depends on a test or condition.

c02/js/boolean-variable.js — JAVASCRIPT

```
var inStock;
var shipping;
inStock = true;
shipping = false;

var elStock = document.getElementById('stock');
elStock.className = inStock;

var elShip = document.getElementById('shipping');
elShip.className = shipping;
```

c02/boolean-variable.html — HTML

```
<h1>Elderflower</h1>
<div id="content">
  <div class="message">Available:
    <span id="stock"></span></div>
  <div class="message">Shipping:
    <span id="shipping"></span></div>
</div>
<script src="js/boolean-variable.js"></script>
```

RESULT

SHORTHAND FOR CREATING VARIABLES

JAVASCRIPT c02/js/shorthand-variable.js

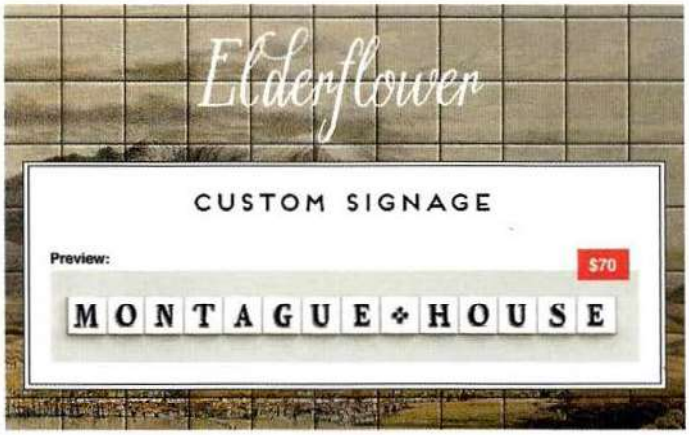
```
① var price = 5;
  var quantity = 14;
  var total = price * quantity;
```

```
② var price, quantity, total;
  price = 5;
  quantity = 14;
  total = price * quantity;
```

```
③ var price = 5, quantity = 14;
  var total = price * quantity;
```

```
④ // Write total into the element with id of cost
  var el = document.getElementById('cost');
  el.textContent = '$' + total;
```

RESULT



Programmers sometimes use shorthand to create variables. Here are three variations of how to declare variables and assign them values:

1. Variables are declared and values assigned in the same statement.

2. Three variables are declared on the same line, then values assigned to each.

3. Two variables are declared and assigned values on the same line. Then one is declared and assigned a value on the next line.

(The third example shows two numbers, but you can declare variables that hold different types of data on the same line, e.g., a string and a number.)

4. Here, a variable is used to hold a reference to an element in the HTML page. This allows you to work directly with the element stored in that variable. (See more about this on p190.)

While the shorthand might save you a little bit of typing, it can make your code a little harder to follow. So, when you are starting off, you will find it easier to spread your code over a few more lines to make it easier to read and understand.

CHANGING THE VALUE OF A VARIABLE

Once you have assigned a value to a variable, you can then change what is stored in the variable later in the same script.

Once the variable has been created, you do not need to use the `var` keyword to assign it a new value. You just use the variable name, the equals sign (also known as the assignment operator), and the new value for that attribute.

For example, the value of a `shipping` variable might start out as being `false`. Then something in the code might change the ability to ship the item and you could therefore change the value to `true`.

In this code example, the values of the two variables are both swapped from being `true` to `false` and vice versa.

c02/js/update-variable.js

JAVASCRIPT

```
var inStock;
var shipping;

inStock = true;
shipping = false;

/* Some other processing might go here and, as
a result, the script might need to change these
values */

inStock = false;
shipping = true;

var elStock = document.getElementById('stock');
elStock.className = inStock;
var elShip = document.getElementById('shipping');
elShip.className = shipping;
```

RESULT

RULES FOR NAMING VARIABLES

Here are six rules you must always follow when giving a variable a name:

1

The name must begin with a letter, dollar sign ($), or an underscore (_). It must **not** start with a number.

2

The name can contain letters, numbers, dollar sign ($), or an underscore (_). Note that you must not use a dash (-) or a period (.) in a variable name.

3

You cannot use **keywords** or **reserved** words. Keywords are special words that tell the interpreter to do something. For example, `var` is a keyword used to declare a variable. Reserved words are ones that may be used in a *future* version of JavaScript.

ONLINE EXTRA

View a full list of keywords and reserved words in JavaScript.

4

All variables are case sensitive, so `score` and `Score` would be different variable names, but it is bad practice to create two variables that have the same name using different cases.

5

Use a name that describes the kind of information that the variable stores. For example, `firstName` might be used to store a person's first name, `lastName` for their last name, and `age` for their age.

6

If your variable name is made up of more than one word, use a capital letter for the first letter of every word *after* the first word. For example, `firstName` rather than `firstname` (this is referred to as camel case). You can also use an underscore between each word (you cannot use a dash).

ARRAYS

An array is a special type of variable. It doesn't just store one value; it stores a list of values.

You should consider using an array whenever you are working with a **list** or a set of values that are **related** to each other.

Arrays are especially helpful when you do not know how many items a list will contain because, when you create the array, you do not need to specify how many values it will hold.

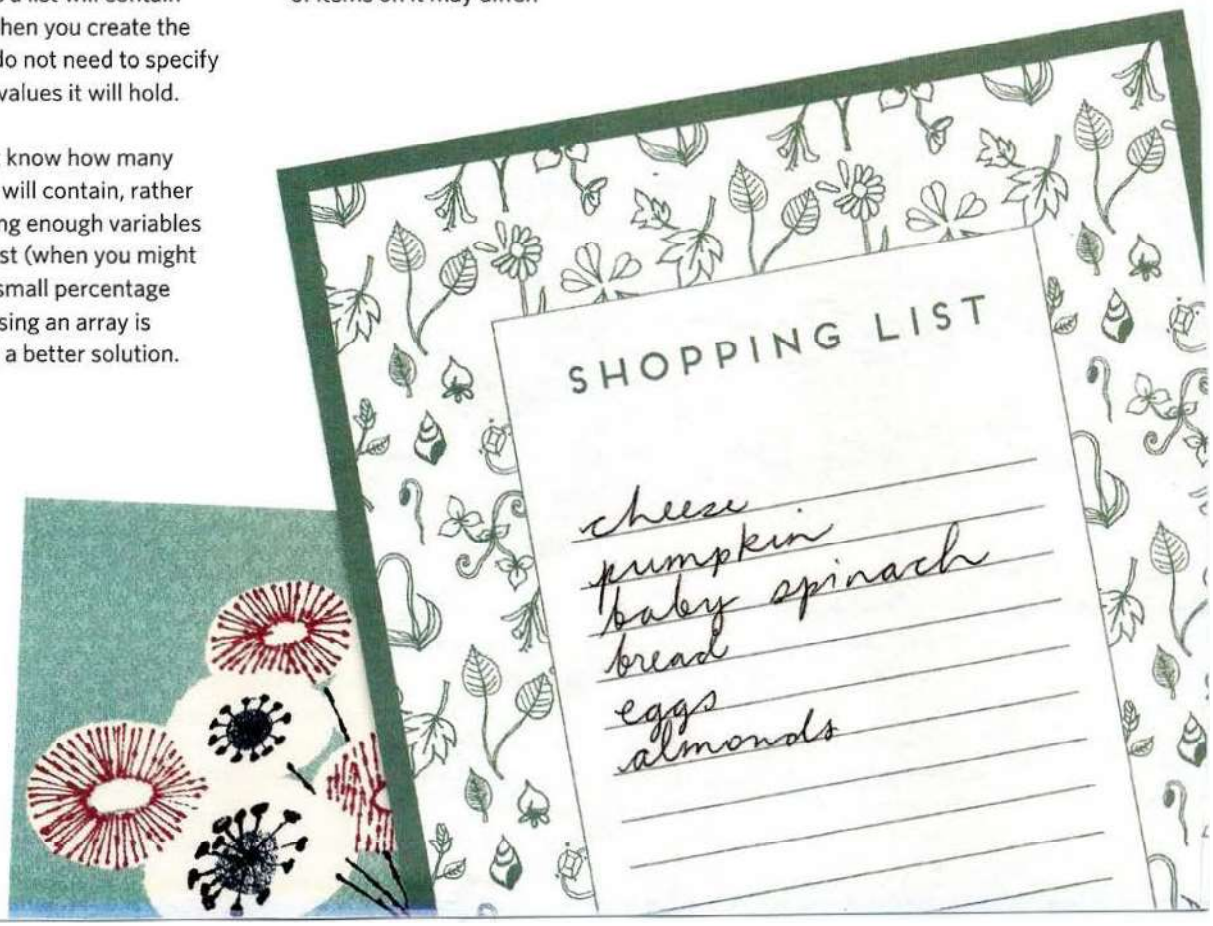
If you don't know how many items a list will contain, rather than creating enough variables for a long list (when you might only use a small percentage of them), using an array is considered a better solution.

For example, an array can be suited to storing the individual items on a shopping list because it is a list of related items.

Additionally, each time you write a new shopping list, the number of items on it may differ.

As you will see on the next page, values in an array are separated by commas.

In Chapter 12, you will see that arrays can be very helpful when representing complex data.



CREATING AN ARRAY

JAVASCRIPT c02/js/array-literal.js

```
var colors;
colors = ['white', 'black', 'custom'];

var el = document.getElementById('colors');
el.textContent = colors[0];
```

RESULT

JAVASCRIPT c02/js/array-constructor.js

```
var colors = new Array('white',
                       'black',
                       'custom');

var el = document.getElementById('colors');
el.innerHTML = colors.item(0);
```

The array literal (shown in the first code sample) is preferred over the array constructor when creating arrays.

You create an array and give it a name just like you would any other variable (using the `var` keyword followed by the name of the array).

The values are assigned to the array inside a pair of square brackets, and each value is separated by a comma. The values in the array do not need to be the same data type, so you can store a string, a number and a Boolean all in the same array.

This technique for creating an array is known as an **array literal**. It is usually the preferred method for creating an array. You can also write each value on a separate line:

```
colors = ['white',
          'black',
          'custom'];
```

On the left, you can see an array created using a different technique called an **array constructor**. This uses the `new` keyword followed by `Array();` The values are then specified in parentheses (not square brackets), and each value is separated by a comma. You can also use a method called `item()` to retrieve data from the array. (The index number of the item is specified in the parentheses.)

VALUES IN ARRAYS

Values in an array are accessed as if they are in a numbered list. It is important to know that the numbering of this list starts at zero (not one).

NUMBERING ITEMS IN AN ARRAY

Each item in an array is automatically given a number called an **index**. This can be used to access specific items in the array. Consider the following array which holds three colors:

```
var colors;
colors = ['white',
          'black',
          'custom'];
```

Confusingly, index values start at 0 (not 1), so the following table shows items from the array and their corresponding index values:

INDEX	VALUE
0	'white'
1	'black'
2	'custom'

ACCESSING ITEMS IN AN ARRAY

To retrieve the third item on the list, the array name is specified along with the index number in square brackets.

Here you can see a variable called `itemThree` is declared. Its value is set to be the third color from the `colors` array.

```
var itemThree;
itemThree = colors[2];
```

NUMBER OF ITEMS IN AN ARRAY

Each array has a property called `length`, which holds the number of items in the array.

Below you can see that a variable called `numColors` is declared. Its value is set to be the number of the items in the array.

The name of the array is followed by a period symbol (or full stop) which is then followed by the `length` keyword.

```
var numColors;
numColors = colors.length;
```

Throughout the book (especially in Chapter 12) you meet more features of arrays, which are a very flexible and powerful feature of JavaScript.

ACCESSING & CHANGING VALUES IN AN ARRAY

JAVASCRIPT c02/js/update-array.js

```
// Create the array
var colors = ['white',
              'black',
              'custom'];

// Update the third item in the array
colors[2] = 'beige';

// Get the element with an id of colors
var el = document.getElementById('colors');

// Replace with third item from the array
el.textContent = colors[2];
```

RESULT

The first lines of code on the left create an array containing a list of three colors. (The values can be added on the same line or on separate lines as shown here.)

Having created the array, the third item on the list is changed from `'custom'` to `'beige'`.

To access a value from an array, after the array name you specify the index number for that value inside square brackets.

You can change the value of an item an array by selecting it and assigning it a new value just as you would any other variable (using the equals sign and the new value for that item).

In the last two statements, the newly updated third item in the array is added to the page.

If you wanted to write out *all* of the items in an array, you would use a loop, which you will meet on p170.

EXPRESSIONS

An **expression** evaluates into (results in) a single value. Broadly speaking there are two types of expressions.

1

EXPRESSIONS THAT JUST ASSIGN A VALUE TO A VARIABLE

In order for a variable to be useful, it needs to be given a value. As you have seen, this is done using the assignment operator (the equals sign).

```
var color = 'beige';
```

The value of `color` is now `beige`.

When you first declare a variable using the `var` keyword, it is given a special value of `undefined`. This will change when you assign a value to it. Technically, undefined is a data type like a number, string, or Boolean.

2

EXPRESSIONS THAT USE TWO OR MORE VALUES TO RETURN A SINGLE VALUE

You can perform operations on any number of individual values (see next page) to determine a single value. For example:

```
var area = 3 * 2;
```

The value of `area` is now 6.

Here the expression 3 * 2 evaluates into 6. This example also uses the assignment operator, so the result of the expression 3 * 2 is stored in the variable called `area`.

Another example where an expression uses two values to yield a single value would be where two strings are joined to create a single string.

OPERATORS

Expressions rely on things called **operators**; they allow programmers to create a single value from one or more values.

Covered in this chapter:

ASSIGNMENT OPERATORS

Assign a value to a variable

```
color = 'beige';
```

The value of `color` is now `beige`.
(See p61)

ARITHMETIC OPERATORS

Perform basic math

```
area = 3 * 2;
```

The value of `area` is now `6`.
(See p76)

STRING OPERATORS

Combine two strings

```
greeting = 'Hi ' + 'Molly';
```

The value of `greeting` is now `Hi Molly`.
(See p78)

Covered in Chapter 4:

COMPARISON OPERATORS

Compare two values and return `true` or `false`

```
buy = 3 > 5;
```

The value of `buy` is `false`.
(See p150)

LOGICAL OPERATORS

Combine expressions and return `true` or `false`

```
buy = (5 > 3) && (2 < 4);
```

The value of `buy` is now `true`.
(See p156)

ARITHMETIC OPERATORS

JavaScript contains the following mathematical operators, which you can use with numbers. You may remember some from math class.

NAME	OPERATOR	PURPOSE & NOTES	EXAMPLE	RESULT
ADDITION	+	Adds one value to another	`10 + 5`	15
SUBTRACTION	-	Subtracts one value from another	`10 - 5`	5
DIVISION	/	Divides two values	`10 / 5`	2
MULTIPLICATION	*	Multiplies two values using an asterisk (Note that this is not the letter x)	`10 * 5`	50
INCREMENT	++	Adds one to the current number	`i = 10; i++;`	11
DECREMENT	--	Subtracts one from the current number	`i = 10; i--;`	9
MODULUS	%	Divides two values and returns the remainder	`10 % 3`	1

ORDER OF EXECUTION

Several arithmetic operations can be performed in one expression, but it is important to understand how the result will be calculated. Multiplication and division are performed *before* addition or subtraction. This can affect the number that you expect to see. To illustrate this effect, look at the following examples.

Here the numbers are calculated left to right, so the total is 16:

```
total = 2 + 4 + 10;
```

But in the following example the total is 42 (not 60):

```
total = 2 + 4 * 10;
```

This is because multiplication and division happen *before* addition and subtraction.

To change the order in which operations are performed, place the calculation you want done *first* inside parentheses. So for the following, the total is 60:

```
total = (2 + 4) * 10;
```

The parentheses indicate that the 2 is added to the 4, and *then* the resulting figure is multiplied by 10.

USING ARITHMETIC OPERATORS

JAVASCRIPT c02/js/arithmetic-operator.js

```
var subtotal = (13 + 1) * 5;      // Subtotal is 70
var shipping = 0.5 * (13 + 1);    // Shipping is 7

var total = subtotal + shipping; // Total is 77

var elSub = document.getElementById('subtotal');
elSub.textContent = subtotal;

var elShip = document.getElementById('shipping');
elShip.textContent = shipping;

var elTotal = document.getElementById('total');
elTotal.textContent = total;
```

RESULT

This example demonstrates how mathematical operators are used with numbers to calculate the combined values of two costs.

The first couple of lines create two variables: one to store the subtotal of the order, the other to hold the cost of shipping the order; so the variables are named accordingly: `subtotal` and `shipping`.

On the third line, the total is calculated by adding together these two values.

This demonstrates how the mathematical operators can use variables that represent numbers. (That is, the numbers do not need to be written explicitly into the code.)

The remaining six lines of code write the results to the screen.

STRING OPERATOR

There is just one string operator: the + symbol. It is used to join the strings on either side of it.

There are many occasions where you may need to join two or more strings to create a single value. Programmers call the process of joining together two or more strings to create one new string **concatenation**.

For example, you might have a first and last name in two separate variables and want to join them to show a full name. In this example, the variable called `fullName` would hold the string `'Ivy Stone'`.

```
var firstName = 'Ivy ';
var lastName = 'Stone';
var fullName = firstName + lastName;
```

MIXING NUMBERS AND STRINGS TOGETHER

When you place quotes around a number, it is a string (not a numeric data type), and you cannot perform addition operations on strings.

```
var cost1 = '7';
var cost2 = '9';
var total = cost1 + cost2;
```

You would end up with a string saying `'79'`.

If you try to add a numeric data type to a string, then the number becomes part of the string, e.g., adding a house number to a street name:

```
var number = 12;
var street = 'Ivy Road';
var add = number + street;
```

You would end up with a string saying `'12Ivy Road'`.

If you try to use any of the other arithmetic operators on a string, then the value that results is usually a value called `NaN`. This means "not a number."

```
var score = 'seven';
var score2 = 'nine';
var total = score * score2;
```

You would end up with the value `NaN`.

USING STRING OPERATORS

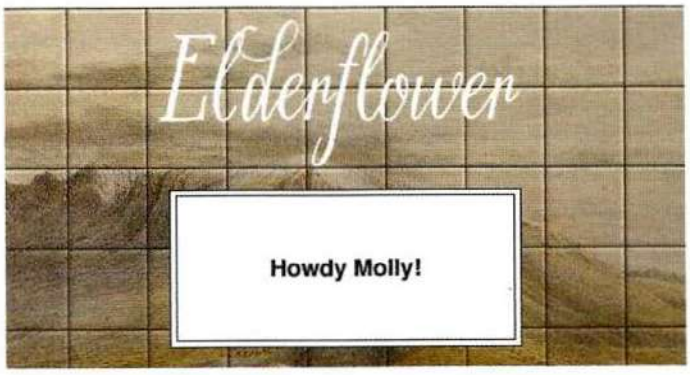
JAVASCRIPT c02/js/string-operator.js

```
var greeting = 'Howdy ';
var name = 'Molly';

var welcomeMessage = greeting + name + '!';

var el = document.getElementById('greeting');
el.textContent = welcomeMessage;
```

HTML c02/string-operator.html

```
<h1>Elderflower</h1>
<div id="content">
  <div id="greeting" class="message">Hello
    <span id="name">friend</span>!
  </div>
</div>
<script src="js/string-operator.js"></script>
```

RESULT



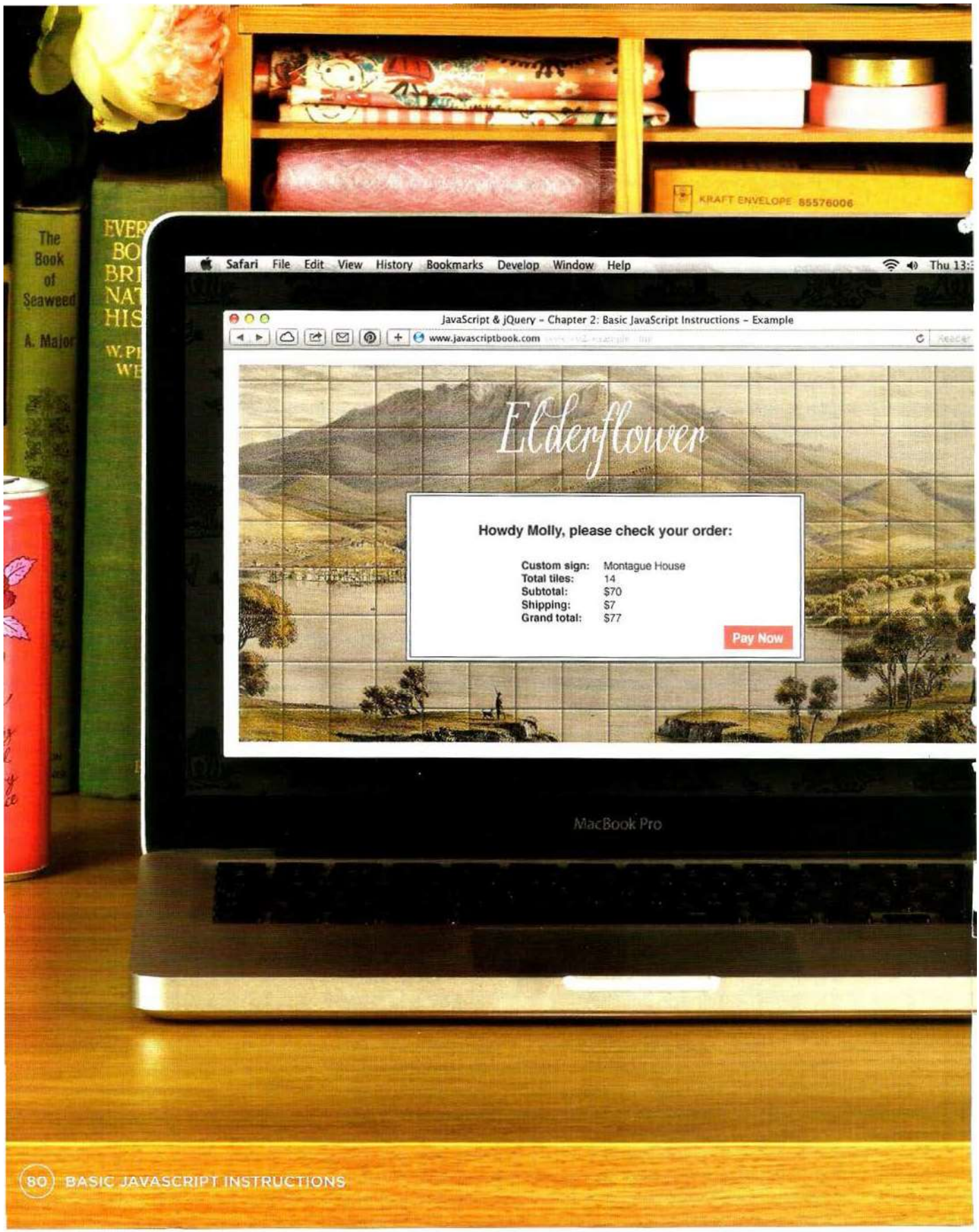
This example will display a personalized welcome message on the page.

The first line creates a variable called `greeting`, which stores the message for the user. Here the greeting is the word `Howdy`.

The second line creates a variable that stores the name of the user. The variable is called `name`, and the user in this case is `Molly`.

The personal welcome message is created by concatenating (or joining) these two variables, adding an exclamation mark, and storing them in a new variable called `welcomeMessage`.

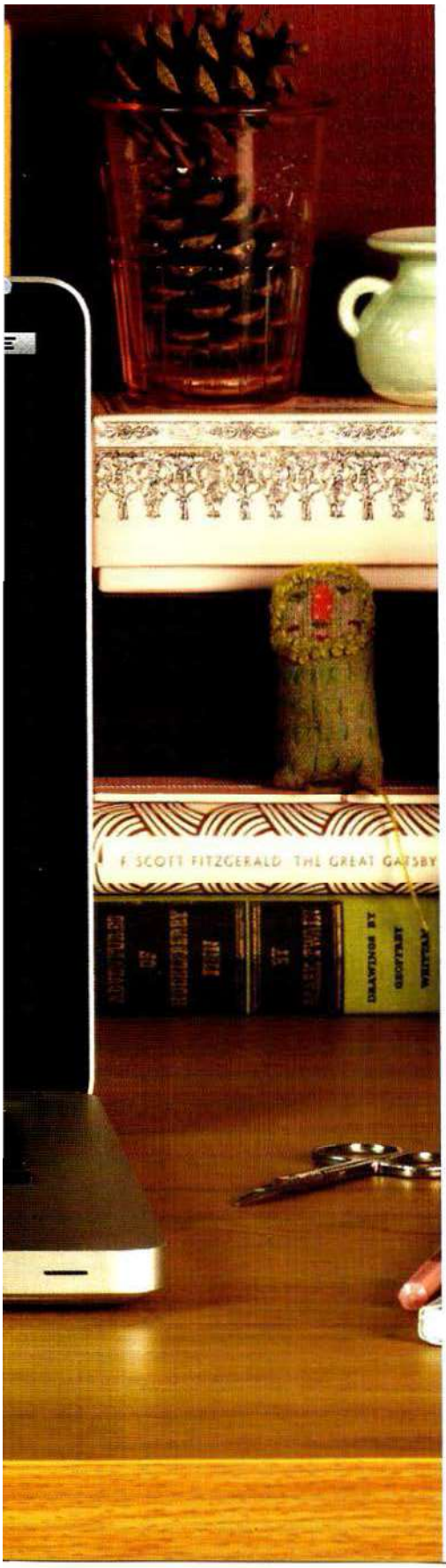
Look back at the `greeting` variable on the first line, and note how there is a space after the word `Howdy`. If the space was omitted, the value of `welcomeMessage` would be `"HowdyMolly!"`


KRAFT ENVELOPE 85576006
The Book of Seaweed
A. Major
Safari File Edit View History Bookmarks Develop Window Help
Thu 13:
JavaScript & jQuery - Chapter 2: Basic JavaScript Instructions - Example
www.javascriptbook.com
Elderflower
Howdy Molly, please check your order:
Custom sign: Montague House
Total tiles: 14
Subtotal: $70
Shipping: $7
Grand total: $77
Pay Now
MacBook Pro

EXAMPLE
BASIC JAVASCRIPT INSTRUCTIONS

c02/example.html HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript &amp;amp; jQuery - Chapter 2: Basic JavaScript Instructions -
      Example</title>
    <link rel="stylesheet" href="css/c02.css" />
  </head>
  <body>
    <h1>Elderflower</h1>
    <div id="content">
      <div id="greeting" class="message">Hello!</div>
      <table>
        <tr>
          <td>Custom sign: </td>
          <td id="userSign"></td>
        </tr>
        <tr>
          <td>Total tiles: </td>
          <td id="tiles"></td>
        </tr>
        <tr>
          <td>Subtotal: </td>
          <td id="subTotal">$</td>
        </tr>
        <tr>
          <td>Shipping: </td>
          <td id="shipping">$</td>
        </tr>
        <tr>
          <td>Grand total: </td>
          <td id="grandTotal">$</td>
        </tr>
      </table>
      <a href="#" class="action">Pay Now</a>
    </div>
    <script src="js/example.js"></script>
  </body>
</html>
```



EXAMPLE
BASIC JAVASCRIPT INSTRUCTIONS

This example combines several techniques that you have seen throughout this chapter.

You can see the code for this example on the next two pages. Single line comments are used to describe what each section of the code does.

To start, three variables are created that store information that is used in the welcome message. These variables are then concatenated (joined together) to create the full message the user sees.

The next part of the example demonstrates how basic math is performed on numbers to calculate the cost of a sign.

- A variable called `sign` holds the text the sign will show.
- A property called `length` is used to determine how many characters are in the string (you will meet this property on p128).
- The cost of the sign (the subtotal) is calculated by multiplying the number of tiles by the cost of each one.
- The grand total is created by adding $7 for shipping.

Finally, the information is written into the page by selecting elements and then replacing the content of that element (using a technique you meet fully in Chapter 5). It selects elements from the HTML page using the value of their `id` attributes and then updates the text inside those elements.

Once you have worked your way through this example, you should have a good basic understanding of how data is stored in variables and how to perform basic operations with the data in those variables.

EXAMPLE
BASIC JAVASCRIPT INSTRUCTIONS

c02/example.html HTML

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript &amp;amp; jQuery - Chapter 2: Basic JavaScript Instructions -
      Example</title>
    <link rel="stylesheet" href="css/c02.css" />
  </head>
  <body>
    <h1>Elderflower</h1>
    <div id="content">
      <div id="greeting" class="message">Hello!</div>
      <table>
        <tr>
          <td>Custom sign: </td>
          <td id="userSign"></td>
        </tr>
        <tr>
          <td>Total tiles: </td>
          <td id="tiles"></td>
        </tr>
        <tr>
          <td>Subtotal: </td>
          <td id="subTotal">$</td>
        </tr>
        <tr>
          <td>Shipping: </td>
          <td id="shipping">$</td>
        </tr>
        <tr>
          <td>Grand total: </td>
          <td id="grandTotal">$</td>
        </tr>
      </table>
      <a href="#" class="action">Pay Now</a>
    </div>
    <script src="js/example.js"></script>
  </body>
</html>
```

SUMMARY

BASIC JAVASCRIPT INSTRUCTIONS

- A script is made up of a series of statements. Each statement is like a step in a recipe.
- Scripts contain very precise instructions. For example, you might specify that a value must be remembered before creating a calculation using that value.
- Variables are used to temporarily store pieces of information used in the script.
- Arrays are special types of variables that store more than one piece of related information.
- JavaScript distinguishes between numbers (0-9), strings (text), and Boolean values (true or false).
- Expressions evaluate into a single value.
- Expressions rely on operators to calculate a value.