JavaScript

Introduction

- JavaScript came about as a joint effort between Netscape Communications Corporation and Sun Microsystems, Inc. The news release of the new language came on December 4, 1995. It was designed to add interactivity to HTML pages
- JavaScript is an object-based, client-side scripting language that you can use to make Web pages more dynamic.

Object Based

- *Object based* means that JavaScript can use items called *objects*.
- However, the objects are not *class based* (meaning no distinction is made between a class and an instance); instead, they are just general objects.

Client Side

- *Client side* means that JavaScript runs in the *client* (software) that the viewer is using, rather than on the Web server of the site serving the page. In this case, the client would be a Web browser.
- To make more sense of this, let’s take a look at how a server-side language works and how a client-side language works.

Server-Side Languages

- A server-side language needs to get information from the Web page or the Web browser, send it to a program that is run on the host’s server, and then send the information back to the browser.
- A server-side language often gives the programmer options that a client-side language doesn’t have, such as saving information on the Web server for later use, or using the new information to update a Web page and save the updates.
- However, a server-side language is likely to be limited in its ability to deal with special features of the browser window that can be accessed with a client-side language (like the content in a particular location on a Web page or the contents of a form before it’s submitted to the server).

Client-Side Languages

- A client-side language is run directly through the client being used by the viewer. In the case of JavaScript, the client is a Web browser. Therefore, JavaScript is run directly in the Web browser and doesn’t need to go through the extra step of sending and retrieving information from the Web server.
- With a client-side language, the browser reads and interprets the code, and the results can be given to the viewer without getting information from the server first. This process can make certain tasks run more quickly.
- A client-side language can also access special features of a browser window that may not be accessible with a server-side language.
- However, a client-side language lacks the ability to save files or updates to files on a Web server like a server-side language can.

Scripting Language

- A scripting language doesn’t require a program to be compiled before it is run.
- With a regular programming language, before we can run a program that we have written, we must compile it using a special compiler to be sure there are no syntax errors. With a scripting language, the code is interpreted as it is loaded in the client.

It’s Not Java

- JavaScript and Java are two different language
- Java is a full programming language that must be compiled (running a program through software that converts the higher-level code to machine language) before a program can be executed.
- Java is more powerful but also more complex
- JavaScript doesn’t need a compiler and is more lenient in a number of areas, such as syntax.

What Can JavaScript do?

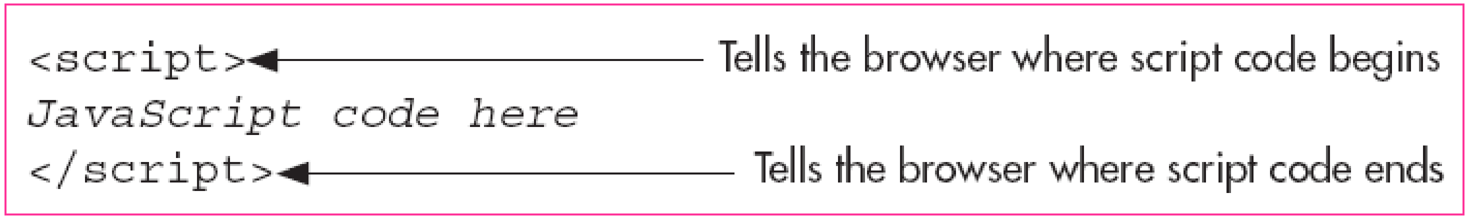
- **JavaScript can react to events -** A JavaScript can be set to execute when something happens, like when a page has finished loading or when a user clicks on an HTML element
- **JavaScript can read and write HTML elements -** A JavaScript can read and change the content of an HTML element
- **JavaScript can be used to validate data -** A JavaScript can be used to validate form data before it is submitted to a server. This saves the server from extra processing
- **JavaScript can be used to detect the visitor's browser** - A JavaScript can be used to detect the visitor's browser, and - depending on the browser - load another page specifically designed for that browser
- **JavaScript can be used to create cookies** - A JavaScript can be used to store and retrieve information on the visitor's computer

How JavaScript Run in a Browser.

- JavaScript runs in the browser by being added into an existing HTML document (either directly or by referring to an external script file).
- We can add special tags and commands to the HTML code that will tell the browser that it needs to run a script. Once the browser sees these special tags, it interprets the JavaScript commands and will do what we have directed it to do with our code.
- Thus, by simply editing an HTML document, we can begin using JavaScript on our Web pages and see the results.

Script Tag

- Script tags are used to tell the browser where some type of scripting language will begin and end in an HTML document.

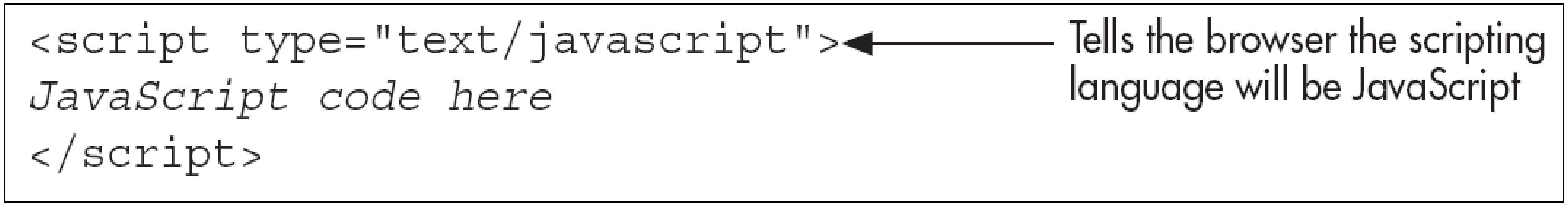


```
<script>
JavaScript code here
</script>
```

Tells the browser where script code begins
Tells the browser where script code ends

- The scripting language between the opening and closing script tags could be JavaScript, VBScript, or some other language.
- Even though JavaScript is usually set as the default scripting language in browsers, there may be some browsers that do not default to JavaScript.

- To be safe, it is better to explicitly identify the language as JavaScript. We do this by adding the type attribute with the value of “text/javascript” to the opening script tag.



```
<script type="text/javascript">
JavaScript code here
</script>
```

Calling External Scripts

- Script tags are also useful if we wish to call an external JavaScript file in our document.
- An *external JavaScript file* is a text file that contains JavaScript code, and it is saved with the .js file extension.
- By calling an external file, we can save the time of coding or copying a long script into each page in which the script is needed.
- We can call external scripts by adding an src (source) attribute to the opening script tag:

```
<script type="text/javascript" src="yourfile.js"></script>
```

Statements

- A JavaScript statement is a command to a browser. The purpose of the command is to tell the browser what to do.
- A statement can be used to perform a single task, to perform multiple tasks, or to make calls to other parts of the script that perform several statements
- JavaScript statements can be grouped together in blocks. Blocks start with a left curly bracket {, and end with a right curly bracket }. The purpose of a block is to make the sequence of statements execute together.
- Most JavaScript statements end with a semicomma.

Comments

- We may need to make notes in our JavaScript code, such as to describe what a line of code is supposed to do.
- It’s also possible that we will want to disable a line of the script for some reason. For instance, if we are looking for an error in a script, we may want to disable a line in the script to see if it is the line causing the error.
- We can accomplish these tasks by using JavaScript comments. We can insert comments that appear on one line or run for numerous lines.

Single-Line Comments

- If we want to add comment on a single line in our code then we place a pair of forward slashes before the text of the comment

```
// Your comment here
```

- In this format, anything preceding the two slashes on that line is “live” code (code that will be executed) and anything after the slashes on that line is ignored. For example,

```
document.write("This is cool!"); // writes out my opinion
// document.write("This is cool!"); writes out my opinion
```

Multiple-Line Comments

- To add comments that span any number of lines, we use: a forward slash followed by an asterisk at the beginning of the comment, then the text of the comment, and then an asterisk followed by a forward slash at the end of the comment.
- We need to be careful to close comment, otherwise, we might accidentally comment out code we need to execute!
- Comments can be used to provide some documentation of what to expect from each script.

```
<script type="text/javascript">
/*
The JavaScript code is now working! This text is hidden.
*/
document.write("Now everyone can see me!");
</script>
```

Variables

- A *variable* represents or holds a value. The actual value of a variable can be changed at any time.
- They can be used in places where the value they represent is unknown when the code is written.
- They can save time in writing and updating scripts. When we assign a value to a variable at the beginning of a script, the rest of the script can simply use the variable in its place. If we decide to change the value later, we need to change the code in only one place (where you assigned a value to the variable) rather than in numerous places.
- They are code clarifiers. Since variables represent something, and we can give them meaningful names, they are often easier to recognize when we read our scripts
  - **TotalPrice=2.42+4.33;**
  - **TotalPrice=CandyPrice+OilPrice**;

Declaring Variables

- To declare text as a variable, we use the var keyword, which tells the browser that the text to follow will be the name of a new variable:

```
var variablename;
```

- For example, to name variable coolcar, the declaration looks like this:

```
var coolcar;
```

- The semicolon ends the statement. The variable does not have a value assigned to it yet.

Assigning Values to Variables

- To assign a value to a variable, we use the assignment operator, which is the equal to (=) symbol. If you want to declare a variable and assign a value to it on the same line, use this format:

```
var variablename=variablevalue;
```

- For example, to name your variable paycheck and give it the numeric value 1200, use this statement:

```
var paycheck=1200;
```

Naming Variables

- Using Case in Variables
- Using Allowed Characters
- Avoiding Reserved Words
- Giving Variables Meaningful Names

Using Case in Variables

- JavaScript variables are case sensitive—paycheck, PAYCHECK, Paycheck, and PaYcHeCk are four different variables.

- When you create a variable, you need to be sure to use the same case when you write that variable's name later in the script. If you change the capitalization , JavaScript sees it as a new variable or returns an error.

- Here are a couple of suggestions for using case in variable names:
  - If you are using a variable name that consists of only one word, it is probably easiest to use lowercase for the entire name. It will be quicker to type, and you will know when you use it later to type it all in lowercase.
  - For a variable name with two words, you might decide to capitalize the first letter of each word. For example, you may name a variable MyCar or My_Car

Using Allowed Characters

- An important rule is that a variable name must begin with a letter or an underscore character (_).
- The variable name cannot begin with a number or any other character that is not a letter (other than the underscore).
- The other characters in the variable name can be letters, numbers, or underscores.
- Blank spaces are not allowed in variable names.

Valid

- paycheck
- _paycheck
- pay2check
- pay_check
- pay_245

Invalid

- #paycheck
- 1paycheck
- pay check
- pay_check 2
- _pay check

Avoiding Reserved Words

- Another rule is when naming variables is to avoid the use of JavaScript reserved words. These are special words that are used for a specific purpose in JavaScript.
- For instance, the reserved word var is used to declare a variable. Using it as a variable name can cause numerous problems in your script, since this word is meant to be used in a different way.

abstract	delete	goto	null	throws
as	do	if	package	transient
boolean	double	implements	private	true
break	else	import	protected	try
byte	enum	in	public	typeof
case	export	instanceof	return	use
catch	extends	int	short	var
char	false	interface	static	void
class	final	is	super	volatile
const	finally	long	switch	while
continue	float	namespace	synchronized	with
debugger	for	native	this	
default	function	new	throw	

JavaScript Reserved Words

Giving Variables Meaningful Names

- You should try to give variables names that describe what they represent as clearly as possible.
- Suppose that we want to use a variable to hold a number of an example on a page. Rather than use *x*, *ex*, or another short variable, we must use something more descriptive:

```
var example_number=2;
```

Variable Types

- In JavaScript, the variable values, or *types*, can include
  - Number
  - String
  - Boolean and
  - Null
- Unlike stricter programming languages, JavaScript does not force to declare the type of variable when we define it.

Number

- JavaScript does not require numbers to be declared as integers, floating-point (decimal) numbers, or any other number type.
- We can define a number variable by using the keyword var:

```
var variablename=number;
```

```
var paycheck=1200;
var phonebill=29.99;
var savings=0;
var sparetime=-24.5;
```

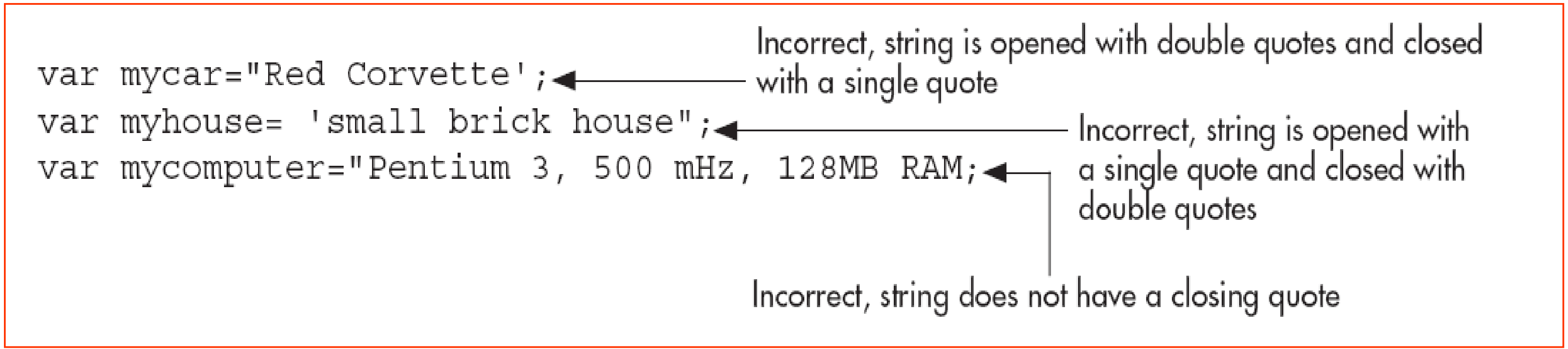
- For a long number, JavaScript has exponential notation. To denote the exponent, we use a letter *e* right after the base number and before the exponent. For example, to create a variable named bignumber and assign it a value of 4.52×10^5 (452,000),

```
var bignumber=4.52e5;
```

String

- *String variables* are variables that represent a string of text.
- The string may contain letters, words, spaces, numbers, symbols, or anything
- JavaScript strings are case sensitive. This may not seem important now, but it matters when we need to compare strings for a match
- Strings are defined in a slightly different way than numbers, using this format:

```
var variablename="stringtext";
var mycar="Corvette";
var oldcar="Big Brown Station Wagon";
var mycomputer="Pentium 3, 500 MHz, 128MB RAM";
var oldcomputer="386 SX, 40 mHz, 8MB RAM";
var jibberish="what? cool! I am @ home 4 now. (cool, right?)";
```



```
var mycar="Red Corvette';
var myhouse= 'small brick house";
var mycomputer="Pentium 3, 500 mHz, 128MB RAM;
```

Incorrect, string is opened with double quotes and closed with a single quote

Incorrect, string is opened with a single quote and closed with double quotes

Incorrect, string does not have a closing quote

Boolean

- A *Boolean* variable is one with a value of true or false.

```
var JohnCodes=true;
var JohnIsCool=false;
```

- The words *true* and *false* do not need to be enclosed in quotes. This is because they are reserved words, which JavaScript recognizes as Boolean values.

- Instead of using the words true and false, JavaScript also allows to use the number 1 for true and the number 0 for false, as shown here:

```
var JohnCodes=1;  ←——— Using the number 1 is the same as using the value of true
var JohnIsCool=0; ←——— Using the number 0 is the same as using the value of false
```

Null

- *Null* means that the variable has no value.
- If you need to define a variable with a value of null, use a declaration like this:

```
var variablename=null;
```

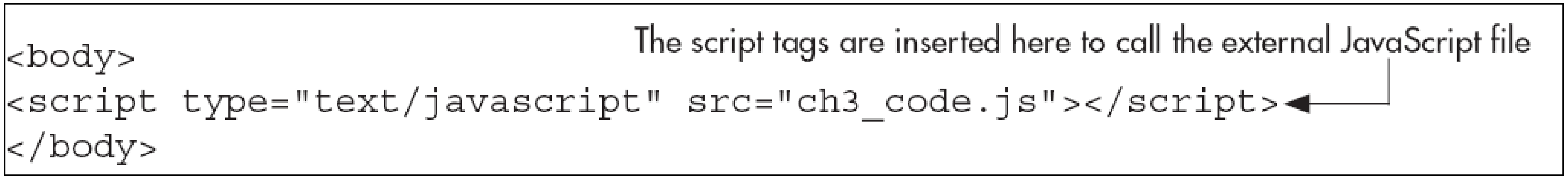
- Like Boolean variables, we do not need to enclose this value in quotation marks as we do with string values, because JavaScript recognizes null as a keyword with a predefined value (nothing).
- Null variables are useful when we test for input in scripts

Writing a Page of JavaScript

- Creating the Framework
- Defining the Variables
- Adding the Commands

Creating the Framework

- The first thing we need is a basic framework for the page so that we know where to insert your script.



The script tags are inserted here to call the external JavaScript file

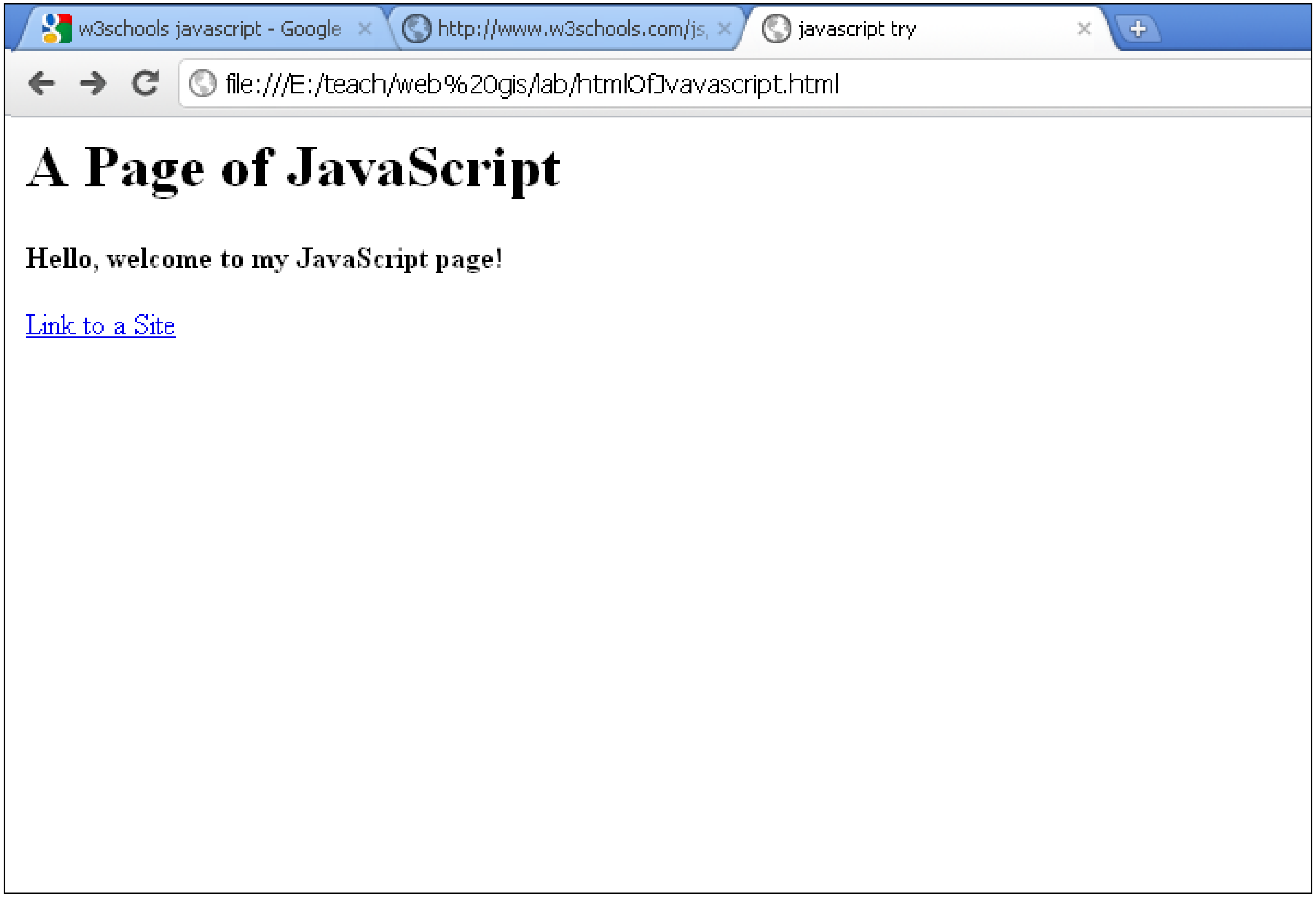
```
<body>
<script type="text/javascript" src="ch3_code.js"></script>
</body>
```

Defining the Variables

```
var headingtext="<h1>A Page of JavaScript</h1>";
var myintro="Hello, welcome to my JavaScript page!";
var linktag="<a href=\"http://www.pageresource.com\">Link to a Site</a>";
var begineffect="<strong>";
var endeffect= "</strong>";
var beginpara="<p>";
var endpara="</p>";
```

Adding the Commands

```
document.write(headingtext);
document.write(begineffect+myintro+endeffect);
document.write(beginpara);
document.write(linktag);
document.write(endpara);
document.write(beginpara);
document.write(endpara);
```


w3schools javascript - Google
http://www.w3schools.com/js,
javascript try
file:///E:/teach/web%20gis/lab/htmlOfJvavascript.html

A Page of JavaScript

Hello, welcome to my JavaScript page!

Link to a Site

JavaScript (2)

Function

- Its purpose is to perform a single task or a series of tasks.
- What a function does depends on what code we place inside it. For instance, a function might write a line of text to the browser or calculate a numeric value and return that value to the main script.
- Functions help to organize the various parts of a script into the different tasks that must be accomplished. By using one function for writing text and another for making a calculation, we make it easier for our self and others to see the purpose of each section of the script, and thus debug it more easily.

- Reusability is an important feature of functions. They can be used more than once within a script to perform their task. Rather than rewriting the entire block of code, we can simply call the function again.

Structuring Functions

- Declaring Functions
- Defining the Code for Functions
- Naming Functions
- Adding Parameters to Functions
- Adding Return Statements to Functions

Declaring Functions

- To declare a function, we use the reserved word function, followed by its name, and then a set of parentheses. This line does not end with a semicolon

```
function functionname()
```

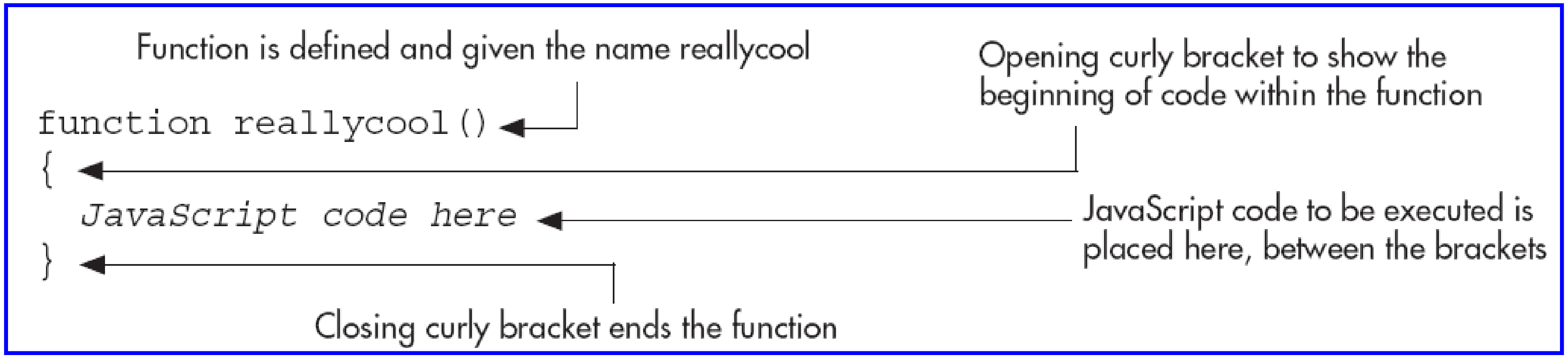
- For example, to name function reallycool and indicate that it does not use any parameters, the first line looks like this:

```
function reallycool()
```

  - Because the function does not use any parameters, the parentheses are left empty.

Defining the Code for Functions

- Curly brackets ({ }) surround the code inside the function. The opening curly bracket marks the beginning of the function’s code; then comes the code; and, finally, the closing curly bracket marks the end of the function.
- The browser will execute all of the code inside the curly brackets when the function is called.
- There are several common ways to place the curly brackets into a script.



Function is defined and given the name reallycool

Opening curly bracket to show the beginning of code within the function

```
function reallycool()
{
  JavaScript code here
}
```

JavaScript code to be executed is placed here, between the brackets

Closing curly bracket ends the function

```
function reallycool() {
  JavaScript code here
}
```

Opening bracket is on the first line with the declaration of the function

```
function reallycool() { JavaScript code here }
```

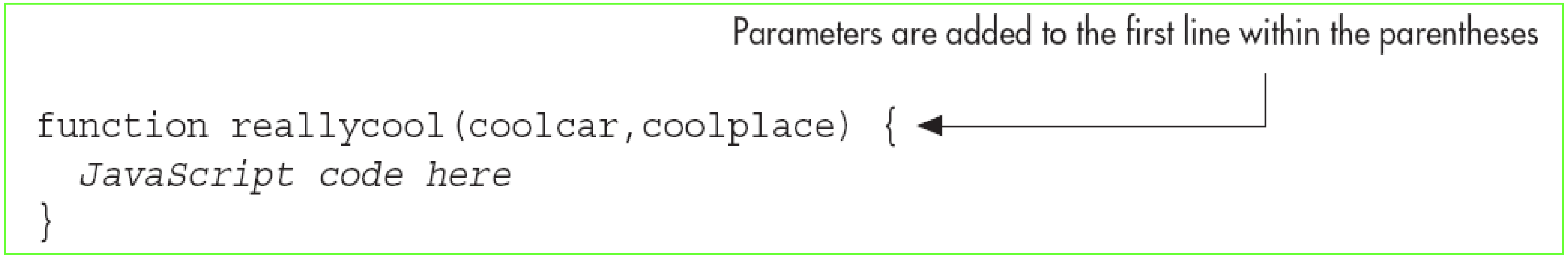
Naming Functions

- As with variables, functions need to be named carefully to avoid problems with scripts.
- The same basic rules that applied to variables, apply to the naming of functions:
    - case sensitivity,
    - using allowed characters,
    - avoiding reserved words, and
    - giving functions memorable and meaningful names.

Adding Parameters to Functions

- *Parameters* are used to allow a function to import one or more values from somewhere outside the function.
- Parameters are set on the first line of the function inside the set of parentheses, in this format:

```
function functionname(variable1,variable2)
```



```
function reallycool(coolcar,coolplace) {
  JavaScript code here
}
```

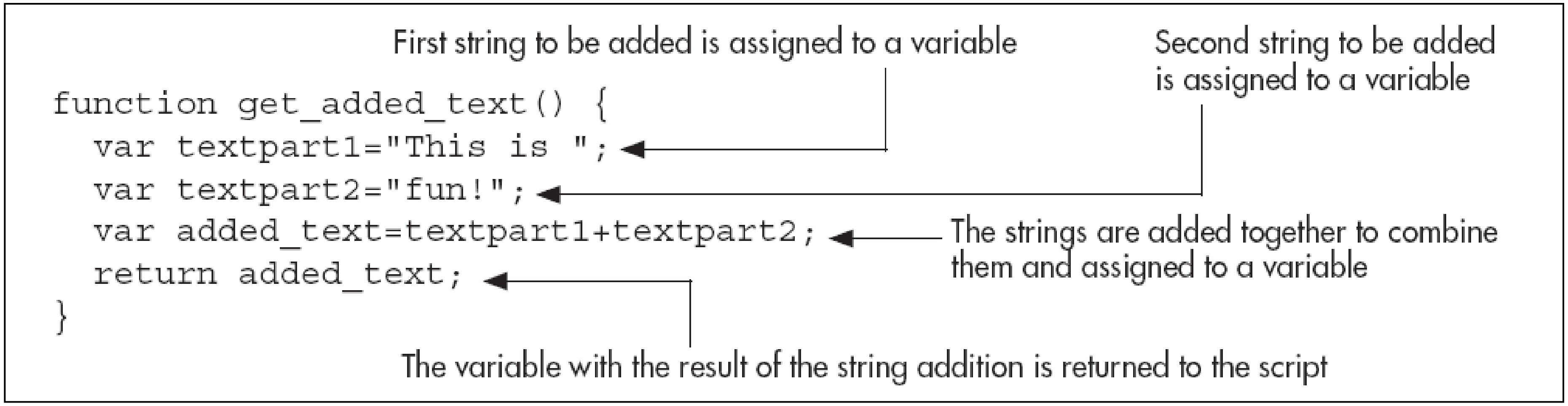
Parameters are added to the first line within the parentheses

- Any value brought in as a parameter becomes a variable within the function, using the name we give it inside the parentheses.

Adding Return Statements to Functions

- A return statement is used to be sure that a function returns a specific value to the main script, to be used in the main script.
- We place the return statement as the last line of the function before the closing curly bracket and end it with a semicolon.
- Most often, the value returned is the value of a variable, using the following format:

```
return variablename;
```



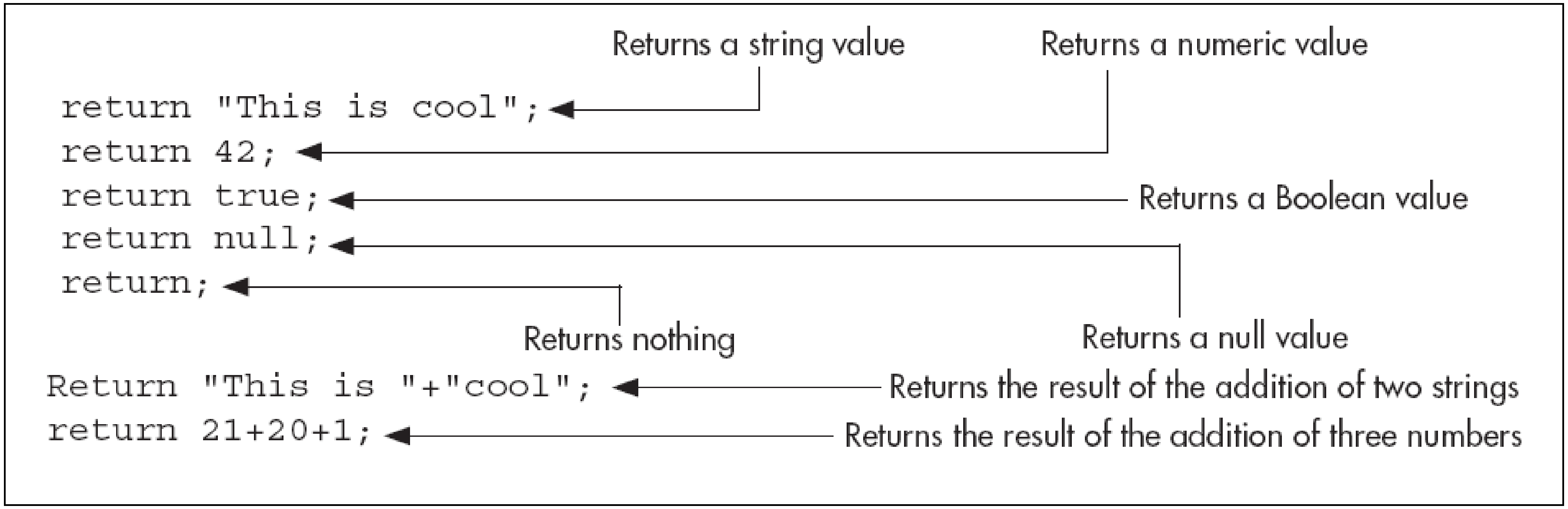
```
function get_added_text() {
  var textpart1="This is ";
  var textpart2="fun!";
  var added_text=textpart1+textpart2;
  return added_text;
}
```

First string to be added is assigned to a variable

Second string to be added is assigned to a variable

The strings are added together to combine them and assigned to a variable

The variable with the result of the string addition is returned to the script



```
 return "This is cool";
 return 42;
 return true;
 return null;
 return;
Return "This is "+"cool";
return 21+20+1;
```

Returns a string value

Returns a numeric value

Returns a Boolean value

Returns a null value

Returns nothing

Returns the result of the addition of two strings

Returns the result of the addition of three numbers

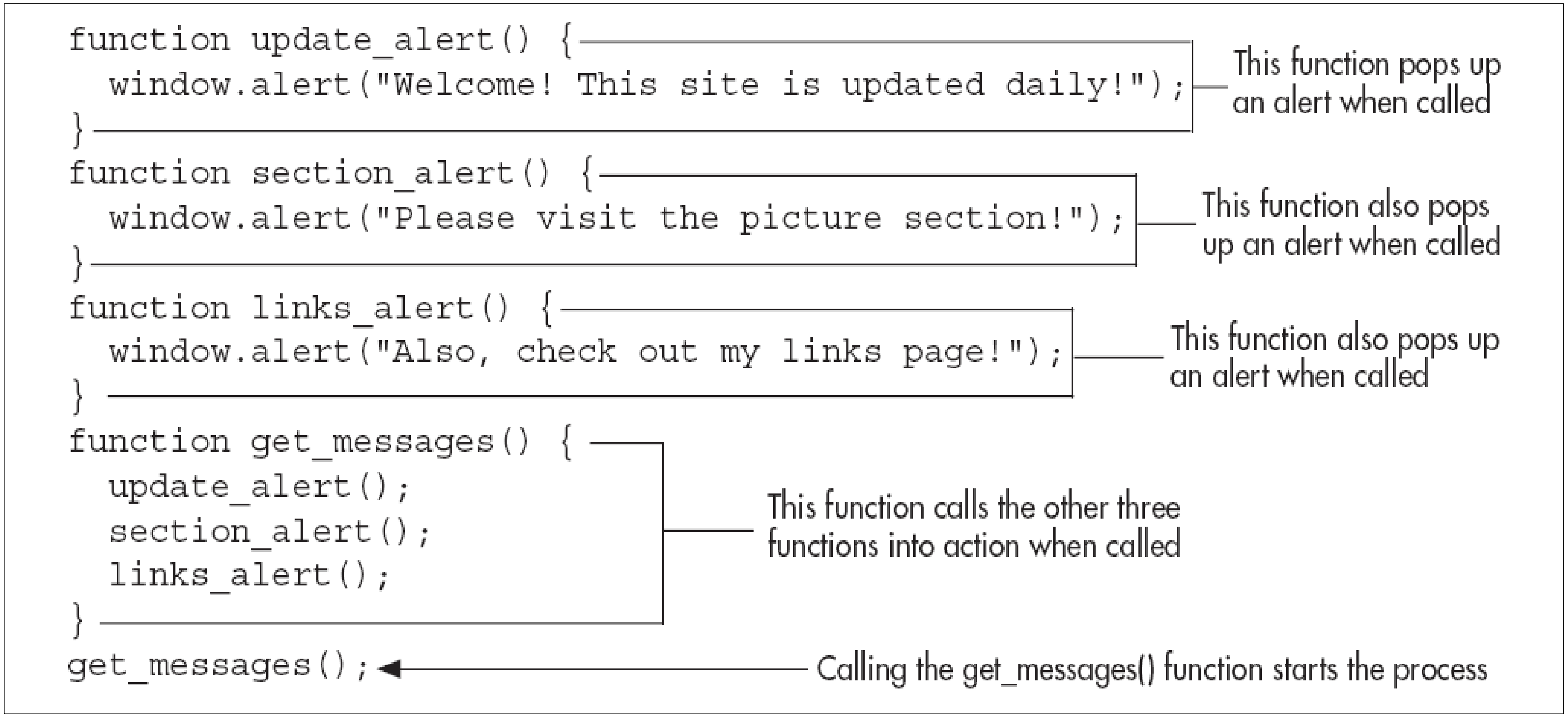
Calling Functions in Scripts

- A call to a function in JavaScript is simply the function name along with the set of parentheses (with or without parameters between the opening and closing parentheses), ending with a semicolon

```
functionname();
```

- We can call functions anywhere in our script code. We can even call a function inside of another function.
- A good rule to follow is to have the function definition come before the function call in the script. The easiest way to be sure that function definition comes before function call is to place all of function definitions as close to the beginning of the script as possible.
- Defining a function before calling it is a suggestion for good coding practice, not a strict rule.

Calling a Function from Another Function



```
function update_alert() {
  window.alert("Welcome! This site is updated daily!");
}
function section_alert() {
  window.alert("Please visit the picture section!");
}
function links_alert() {
  window.alert("Also, check out my links page!");
}
function get_messages() {
  update_alert();
  section_alert();
  links_alert();
}
get_messages();
```

This function pops up an alert when called
This function also pops up an alert when called
This function also pops up an alert when called
This function calls the other three functions into action when called
Calling the get_messages() function starts the process

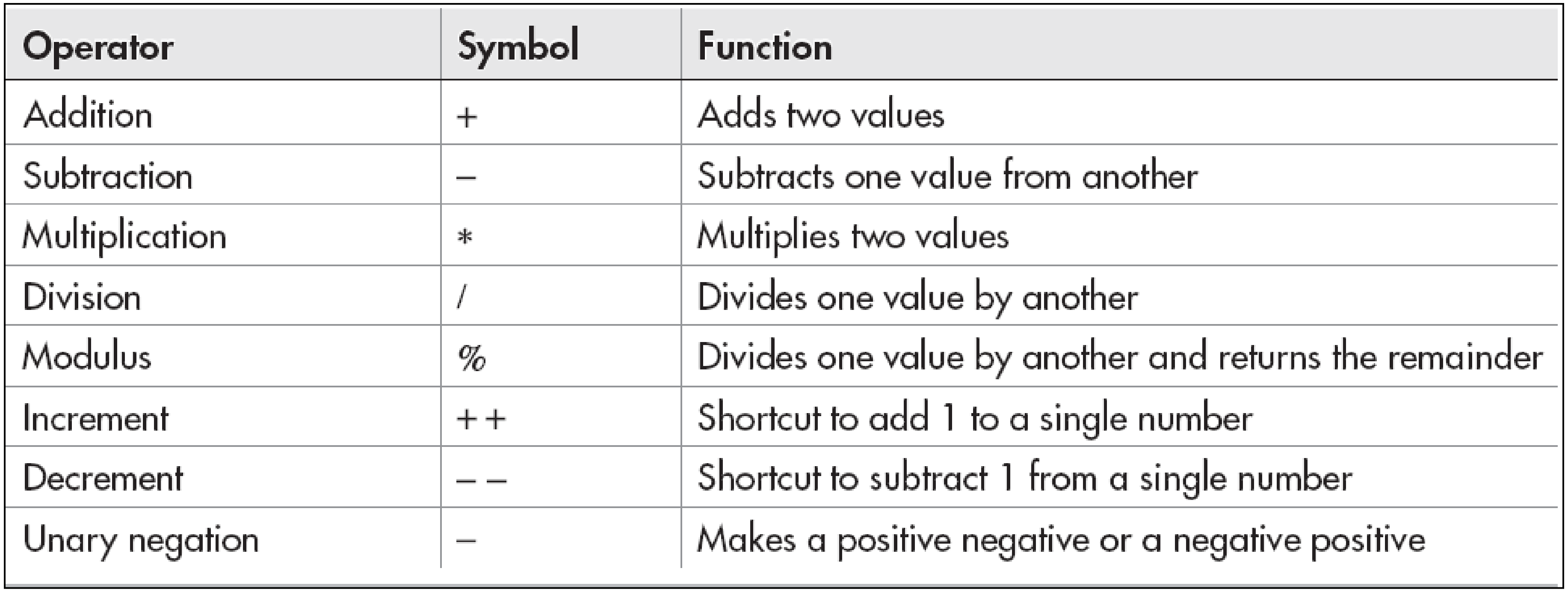
JavaScript Operators

Operator

- An *operator* is a symbol or word in JavaScript that performs some sort of calculation, comparison, or assignment on one or more values.
- JavaScript uses several different types of operators:
  - **Mathematical**
  - **Assignment**
  - **Comparison**
  - **Logical**

Mathematical Operators

- For a mathematical calculation, we use a mathematical operator.
- The values that we use can be any sort of values we like. For instance, we could use two variables, two numbers, or a variable and a number.

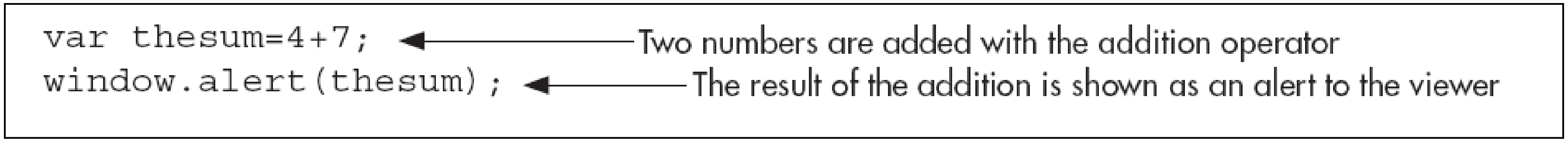


Operator	Symbol	Function
Addition	+	Adds two values
Subtraction	–	Subtracts one value from another
Multiplication	*	Multiplies two values
Division	/	Divides one value by another
Modulus	%	Divides one value by another and returns the remainder
Increment	+ +	Shortcut to add 1 to a single number
Decrement	– –	Shortcut to subtract 1 from a single number
Unary negation	–	Makes a positive negative or a negative positive

The Mathematical Operators

Addition Operator (+)

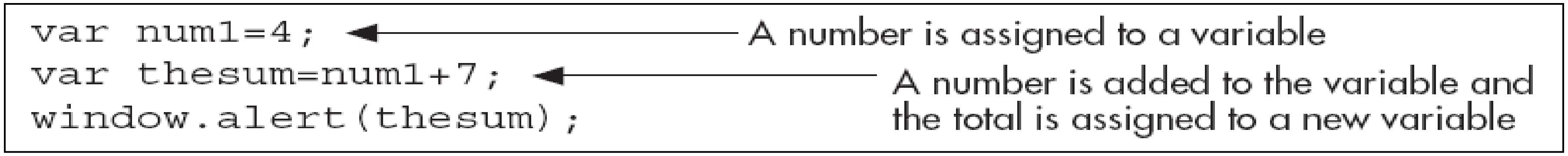
- The addition operator is used to add numbers in mathematical calculations.
- It can be used to combine two strings.



```
var thesum=4+7;
window.alert(thesum);
```

Two numbers are added with the addition operator

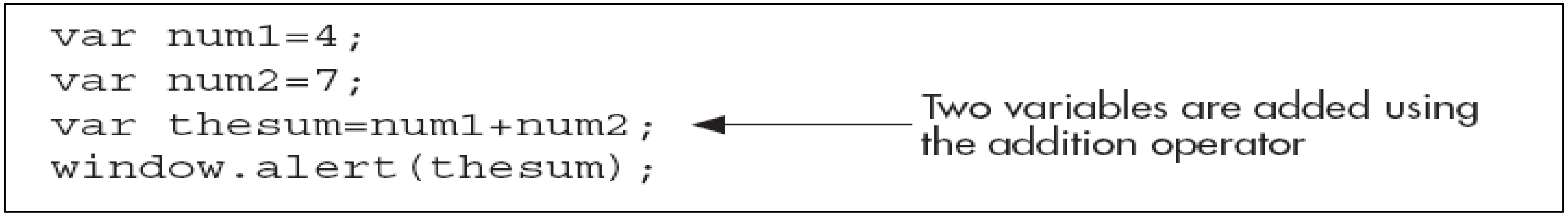
The result of the addition is shown as an alert to the viewer



```
var num1=4;
var thesum=num1+7;
window.alert(thesum);
```

A number is assigned to a variable

A number is added to the variable and the total is assigned to a new variable

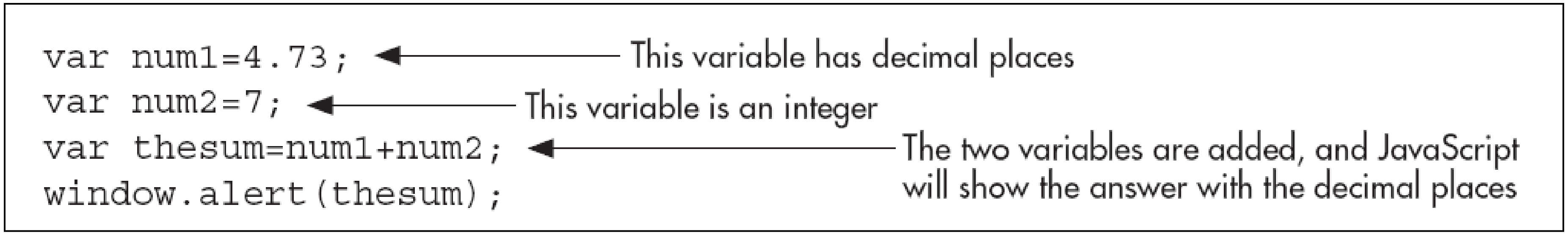


```
var num1=4;
var num2=7;
var thesum=num1+num2;
window.alert(thesum);
```

Two variables are added using the addition operator

The result is an alert that says 11.

Type Conversions in Addition Calculations

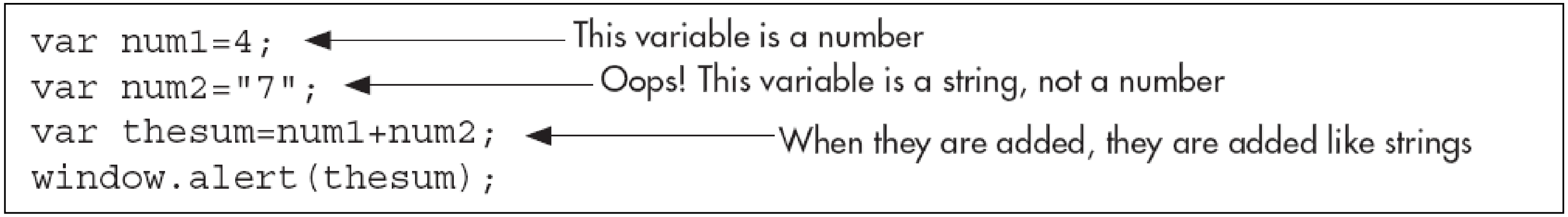


```
var num1=4.73;
var num2=7;
var thesum=num1+num2;
window.alert(thesum);
```

This variable has decimal places

This variable is an integer

The two variables are added, and JavaScript will show the answer with the decimal places



```
var num1=4;
var num2="7";
var thesum=num1+num2;
window.alert(thesum);
```

This variable is a number

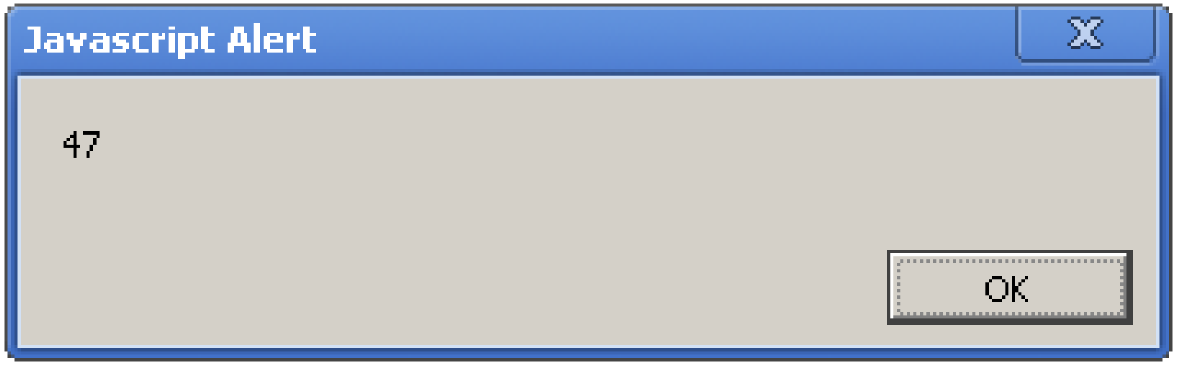
Oops! This variable is a string, not a number

When they are added, they are added like strings

Type Conversions in Addition Calculations

```
var num1=4.73;              ← This variable has decimal places
var num2=7;                 ← This variable is an integer
var thesum=num1+num2;       ← The two variables are added, and JavaScript
window.alert(thesum);         will show the answer with the decimal places
```

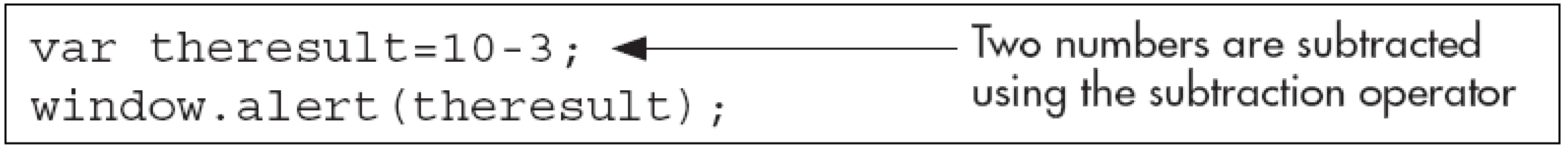
```
var num1=4;                 ← This variable is a number
var num2="7";               ← Oops! This variable is a string, not a number
var thesum=num1+num2;       ← When they are added, they are added like strings
window.alert(thesum);
```



```
47
```

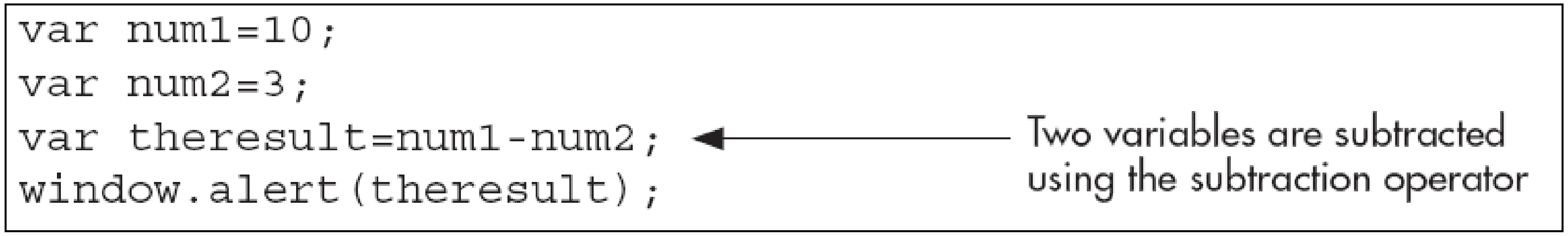
Subtraction Operator (–)

- The subtraction operator is used to subtract the value on its right side from the value on its left side, as in mathematics.



```
var theresult=10-3;
window.alert(theresult);
```

Two numbers are subtracted using the subtraction operator



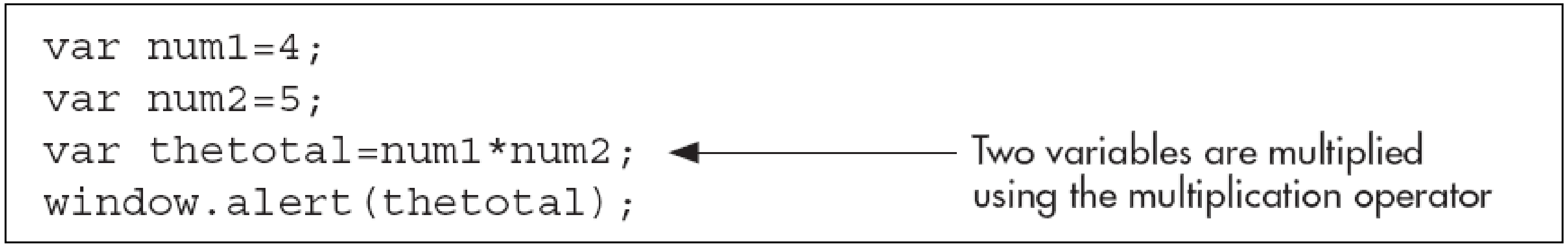
```
var num1=10;
var num2=3;
var theresult=num1-num2;
window.alert(theresult);
```

Two variables are subtracted using the subtraction operator

This code simply subtracts 3 (the number on the right of the operator) from 10 (the number on the left of the operator). The result is an alert that says 7.

Multiplication Operator (*)

- The multiplication operator is used to multiply the value on its right side by the value on its left side.

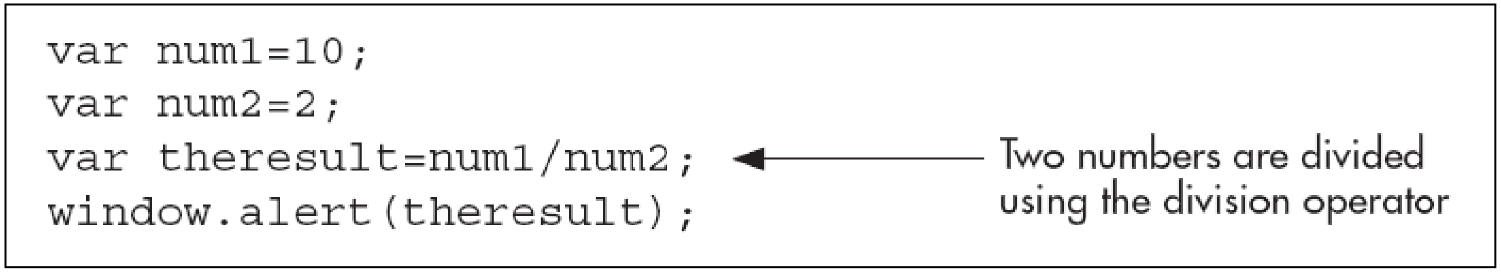


```
var num1=4;
var num2=5;
var thetotal=num1*num2;
window.alert(thetotal);
```

Two variables are multiplied using the multiplication operator

Division Operator (/)

- The division operator is used to divide the value on its left side by the value on its right side.

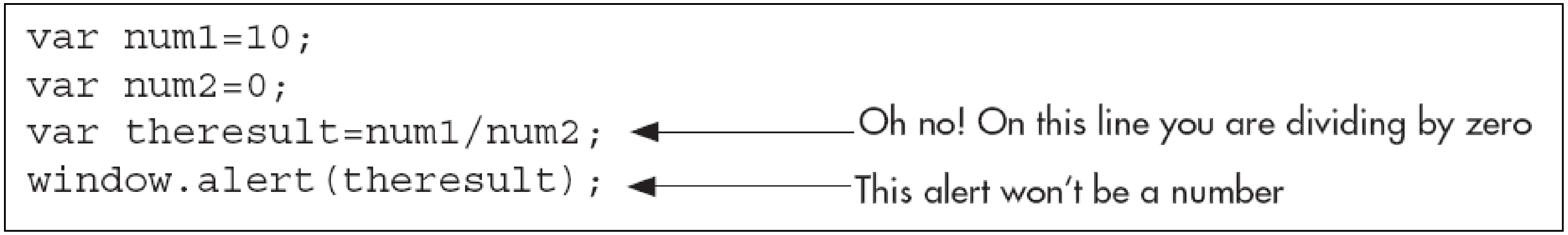


```
var num1=10;
var num2=2;
var theresult=num1/num2;
window.alert(theresult);
```

Two numbers are divided using the division operator

This gives an alert that says 5, the result of dividing 10 by 2.

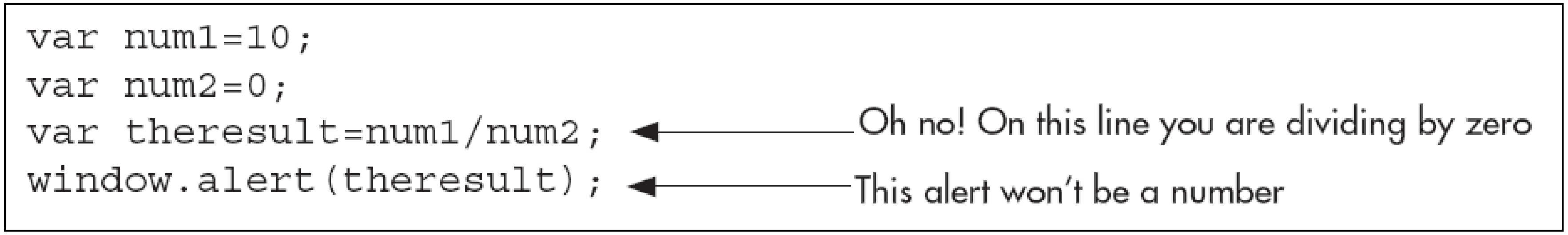
- We need to be careful that we do not divide a number by zero. If we do, the result is going to be either infinity or undefined



```
var num1=10;
var num2=0;
var theresult=num1/num2;
window.alert(theresult);
```

Oh no! On this line you are dividing by zero

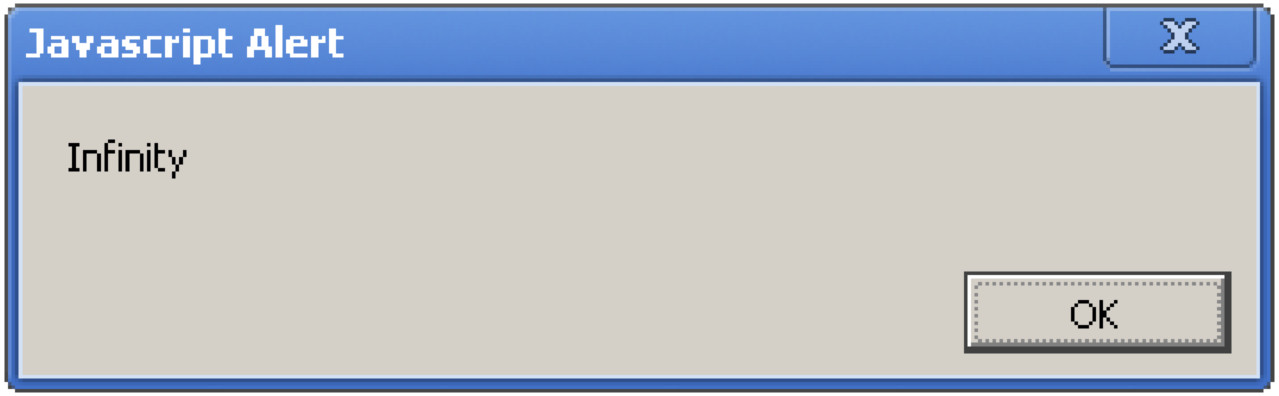
This alert won't be a number



```
var num1=10;
var num2=0;
var theresult=num1/num2;
window.alert(theresult);
```

Oh no! On this line you are dividing by zero

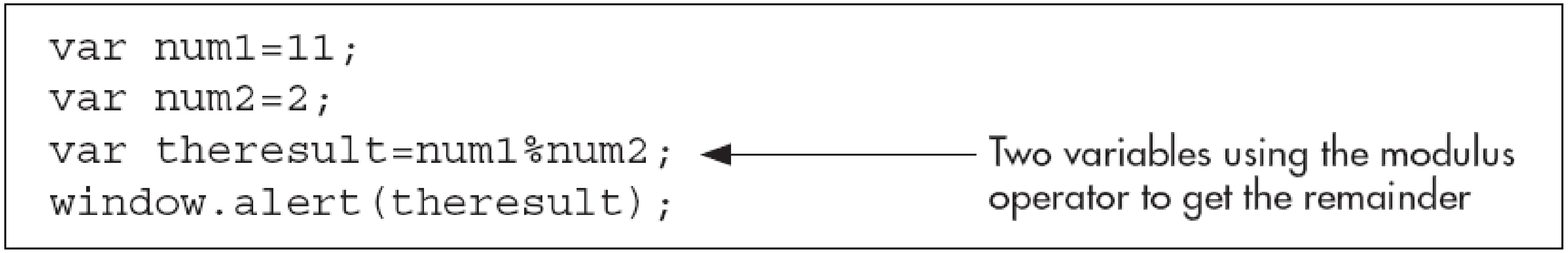
This alert won't be a number



Infinity

Modulus Operator (%)

- The modulus operator is used to divide the number on its left side by the number on its right side, and then give a result that is the integer remainder of the division.
- If the calculation had no remainder, the result would be 0.



```
var num1=11;
var num2=2;
var theresult=num1%num2;
window.alert(theresult);
```

Two variables using the modulus operator to get the remainder

The result is an alert box that shows the value of the remainder, which is 1.

Increment Operator (++)

- It increases the value by 1, just like adding 1 to the value.
- The increment operator can be used on either side of the value on which it operates.
- The actual result depends on whether the operator is used before or after the value it works on, called the *operand*.
- This operator is often used with variables, and often within loops

Increment Operator Before the Operand

- When the increment operator is placed before the operand, it increases the value of the operand by 1, and then the rest of the statement is executed.

```
var num1=2;
var theresult=++num1;
```

  - In this case, the variable num1 begins with a value of 2.
  - However, when the code assigns the value to the variable theresult, it increments the value of num1 before the assignment takes place.
  - The increment occurs first because the increment operator is in front of the operand.
  - So, the value of num1 is set to 3 (2+1) and is then assigned to the variable theresult, which gets a value of 3.

Increment Operator After the Operand

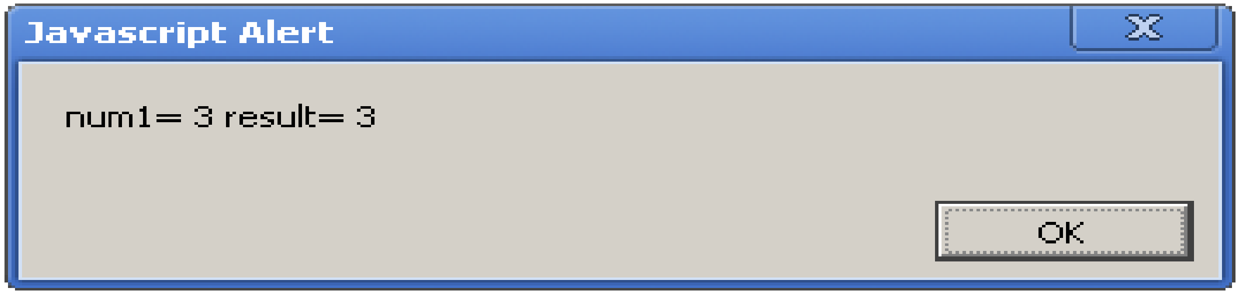
- If you place the increment operator after the operand, it changes the value of the operand after the assignment.

```
var num1=2;
var theresult=num1++;
```

  - num1 begins with the value of 2.
  - On the next line, the increment operator is used after the operand. This means that the code assigns the current value of num1 to the variable theresult, and *after* that is done, it increments the value of num1.
  - So, only after this assignment is complete do we have a new value for num1.
  - The variable theresult is given a value of 2, and then num1 is changed to 3. If we use num1 after this, it will have a value of 3.

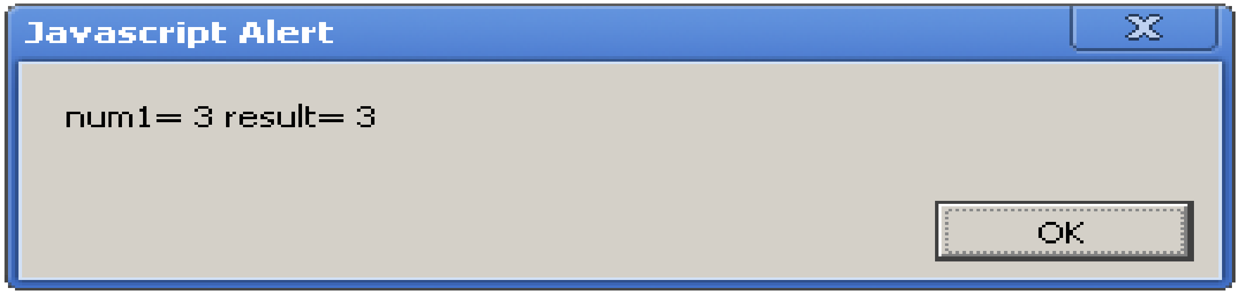
```
<script type="text/javascript">
num1=2;
result= ++num1;
alert("num1= "+num1+" result= "+result);
num1=2;
result= num1++;
alert("num1= "+num1+" result= "+result);
</script>
```

```
<script type="text/javascript">
num1=2;
result= ++num1;
alert("num1= "+num1+" result= "+result);
num1=2;
result= num1++;
alert("num1= "+num1+" result= "+result);
</script>
```

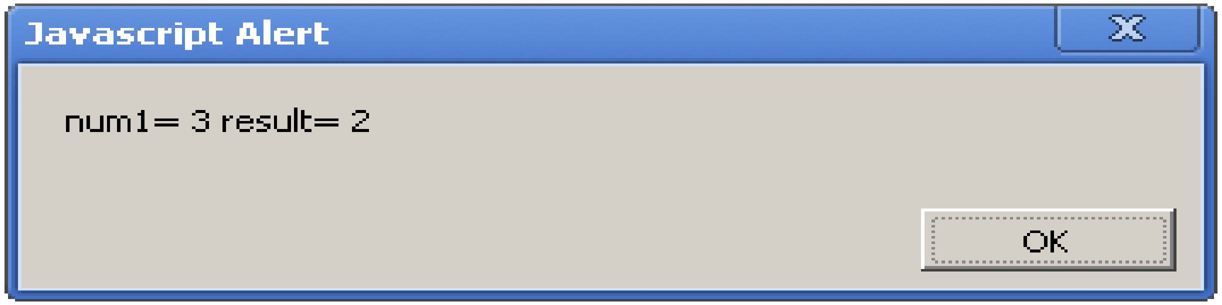


```
num1= 3 result= 3
```

```
<script type="text/javascript">
num1=2;
result= ++num1;
alert("num1= "+num1+" result= "+result);
num1=2;
result= num1++;
alert("num1= "+num1+" result= "+result);
</script>
```



```
num1= 3 result= 3
```

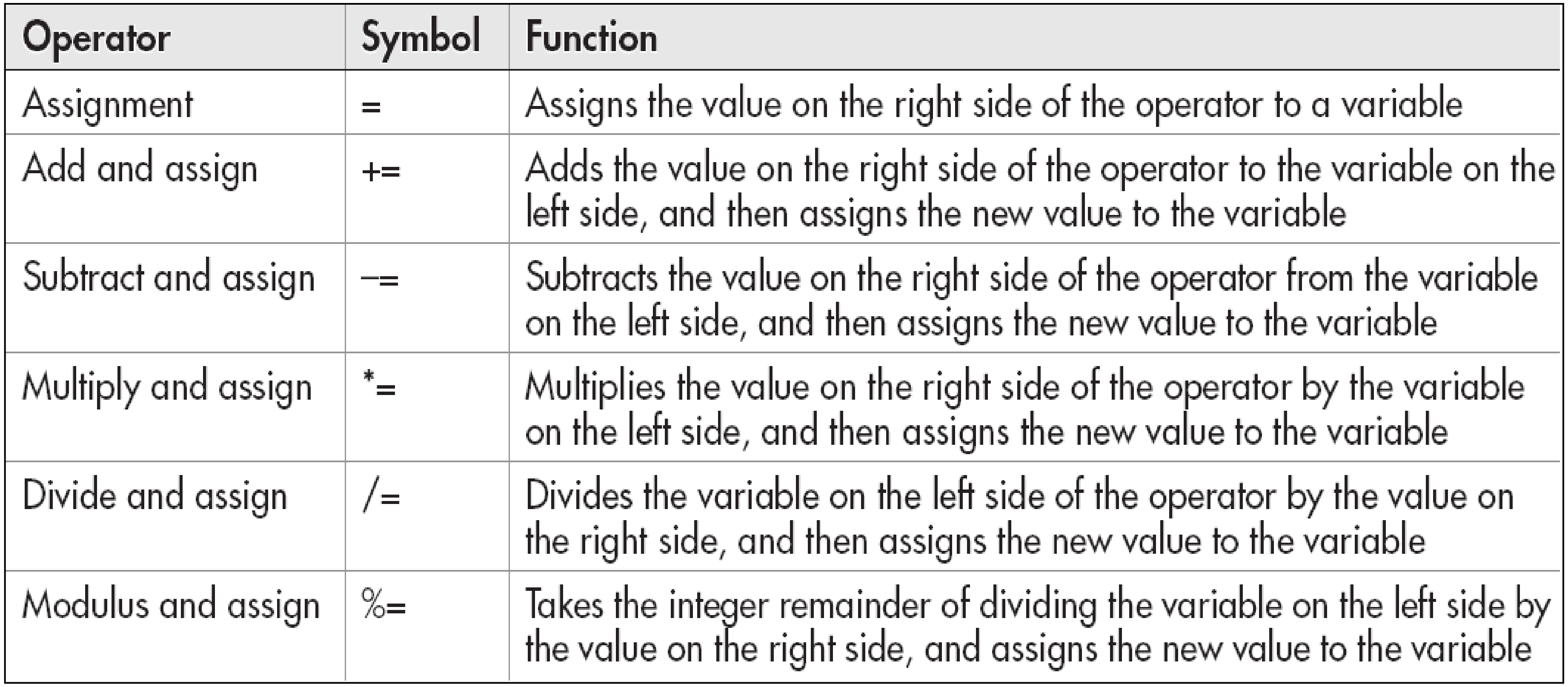


```
num1= 3 result= 2
```

The Decrement Operator (– –)

- The decrement operator works in the same way as the increment operator, but it subtracts 1 from the operand rather than adding 1 to it.
- As with the increment operator, its placement before or after the operand is important.

Assignment Operators



Operator	Symbol	Function
Assignment	=	Assigns the value on the right side of the operator to a variable
Add and assign	+=	Adds the value on the right side of the operator to the variable on the left side, and then assigns the new value to the variable
Subtract and assign	−=	Subtracts the value on the right side of the operator from the variable on the left side, and then assigns the new value to the variable
Multiply and assign	*=	Multiplies the value on the right side of the operator by the variable on the left side, and then assigns the new value to the variable
Divide and assign	/=	Divides the variable on the left side of the operator by the value on the right side, and then assigns the new value to the variable
Modulus and assign	%=	Takes the integer remainder of dividing the variable on the left side by the value on the right side, and assigns the new value to the variable

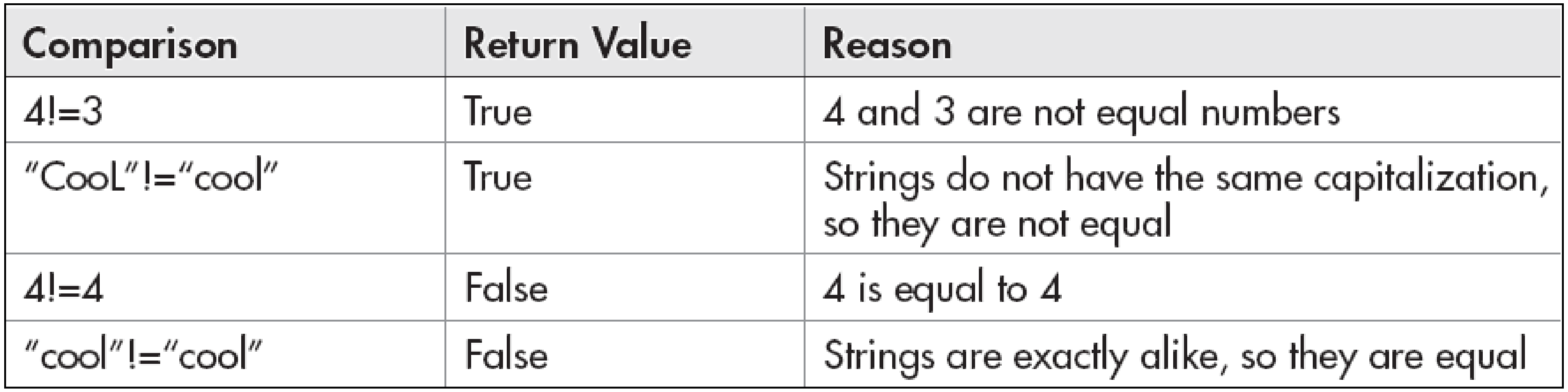
Comparison Operators

Operator	Symbol	Function
Is equal to	==	Returns true if the values on both sides of the operator are equal to each other
Is not equal to	!=	Returns true if the values on both sides of the operator are not equal to each other
Is greater than	>	Returns true if the value on the left side of the operator is greater than the value on the right side
Is less than	<	Returns true if the value on the left side of the operator is less than the value on the right side
Is greater than or equal to	>=	Returns true if the value on the left side of the operator is greater than or equal to the value on the right side
Is less than or equal to	<=	Returns true if the value on the left side of the operator is less than or equal to the value on the right side
Strict is equal to	===	Returns true if the values on both sides are equal and of the same type
Strict is not equal to	!==	Returns true if the values on both sides are not equal or not of the same type

Is-Equal-To Operator (==)

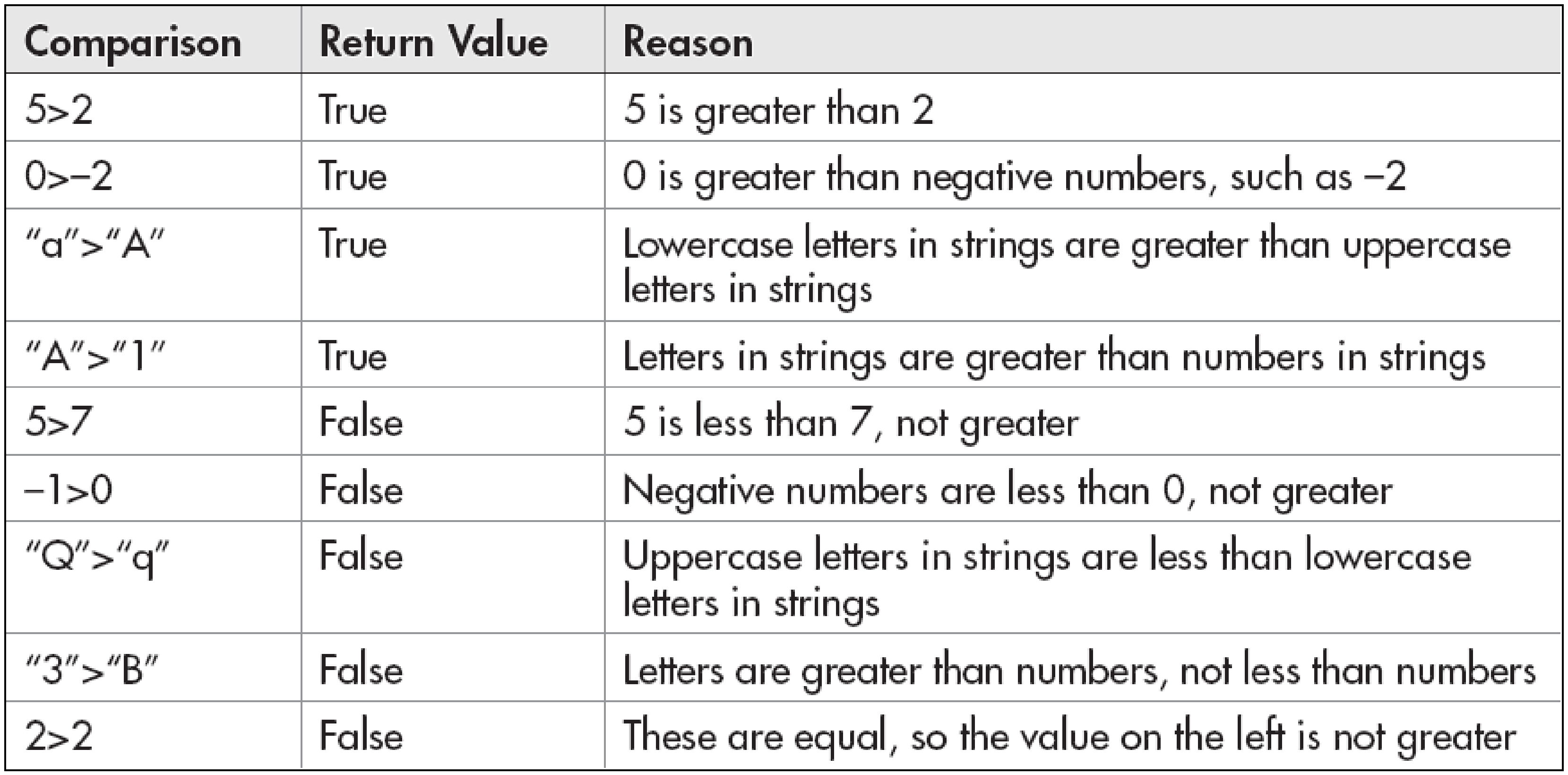
Comparison	Return Value	Reason
4==4	True	Two equal numbers
(4+2)==(3+3)	True	Result on both sides is 6, and 6 is equal to 6
“my socks”==“my socks”	True	Both strings are exactly the same
(“my ”+“socks”)==(“my”+ “ socks”)	True	Results of string additions return equal string values
4==5	False	4 and 5 are not equal numbers
(4+3)==(2+2)	False	Result on left is 7, result on right is 4, and these are not equal
“My socks”==“my socks”	False	Strings are not exactly alike (capitalization)
(“my”+ “socks”)==(“my ” +“socks”)	False	Result on left has no space character; result on right does, causing the strings to be unequal

Is-Not-Equal-To Operator (!=)



Comparison	Return Value	Reason
4!=3	True	4 and 3 are not equal numbers
"CooL"!="cool"	True	Strings do not have the same capitalization, so they are not equal
4!=4	False	4 is equal to 4
"cool"!="cool"	False	Strings are exactly alike, so they are equal

Is-Greater-Than Operator (>)



Comparison	Return Value	Reason
5>2	True	5 is greater than 2
0>−2	True	0 is greater than negative numbers, such as −2
"a">"A"	True	Lowercase letters in strings are greater than uppercase letters in strings
"A">"1"	True	Letters in strings are greater than numbers in strings
5>7	False	5 is less than 7, not greater
−1>0	False	Negative numbers are less than 0, not greater
"Q">"q"	False	Uppercase letters in strings are less than lowercase letters in strings
"3">"B"	False	Letters are greater than numbers, not less than numbers
2>2	False	These are equal, so the value on the left is not greater

Is-Less-Than Operator (<)

Comparison	Return Value	Reason
2<10	True	2 is less than 10
"A"<"a"	True	Uppercase letters in strings are less than lowercase letters in strings
10<2	False	10 is greater than 2, not less
"a"<"A"	False	Lowercase letters in strings are greater than uppercase letters in strings, not less
10<10	False	These are equal, so the value on the left is not less

Is-Greater-Than-or-Equal-To Operator (>=)

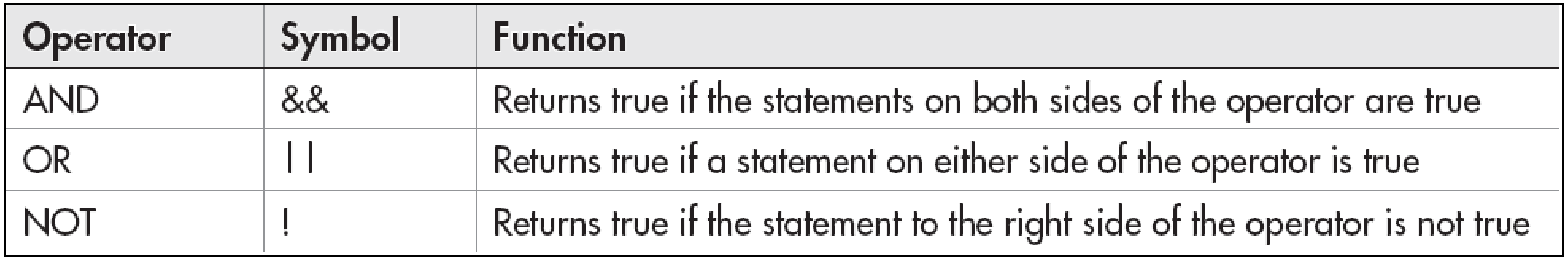
Comparison	Return Value	Reason
5>=2	True	5 is greater than 2
2>=2	True	2 is equal to 2
"a">="A"	True	Lowercase letters are greater than uppercase letters
"A">="A"	True	The strings are equal
1>=2	False	1 is less than 2
"A">="a"	False	Uppercase letters are less than lowercase letters, not greater or equal to

Is-Less-Than-or-Equal-To Operator (<=)

Comparison	Return Value	Reason
2<=5	True	2 is less than 5
2<=2	True	2 is equal to 2
"A"<="a"	True	Uppercase letters are less than lowercase letters
"A"<="A"	True	The strings are equal
5<=2	False	5 is greater than 2, not less than or equal to
"a"<="A"	False	Lowercase letters are greater than uppercase letters, not less than or equal to

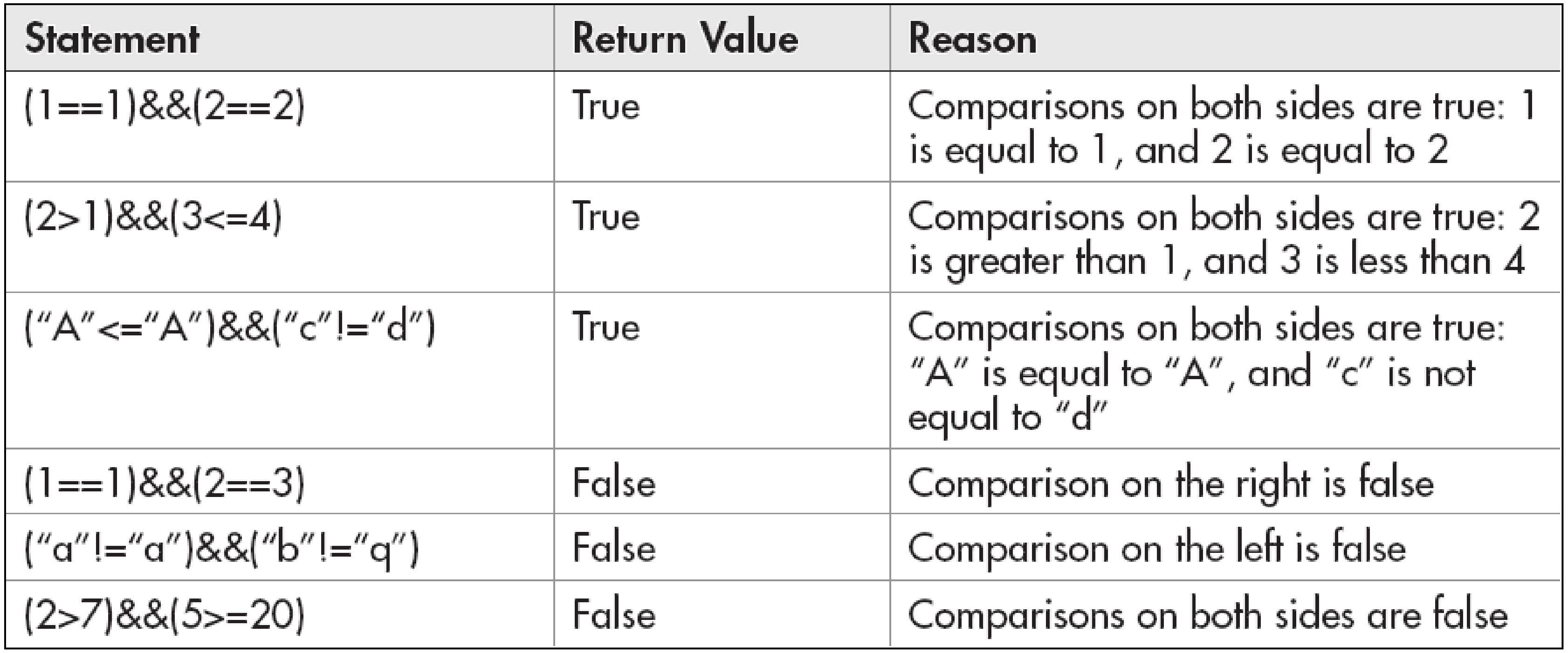
Logical Operators

- The three logical operators allow us to compare two conditional statements to see if one or both of the statements is true and to proceed accordingly.
- The logical operators can be useful if we want to check more than one condition at a time and use the results.
- Like the comparison operators, the logical operators return either true or false, depending on the values on either side of the operator.



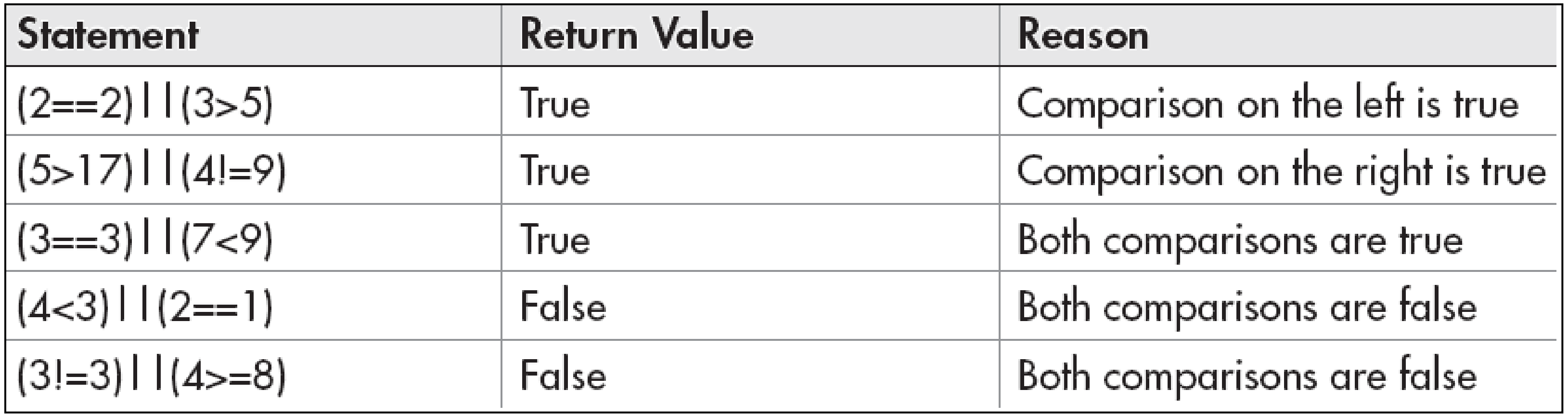
Operator	Symbol	Function
AND	&&	Returns true if the statements on both sides of the operator are true
OR	||	Returns true if a statement on either side of the operator is true
NOT	!	Returns true if the statement to the right side of the operator is not true

AND Operator (&&)



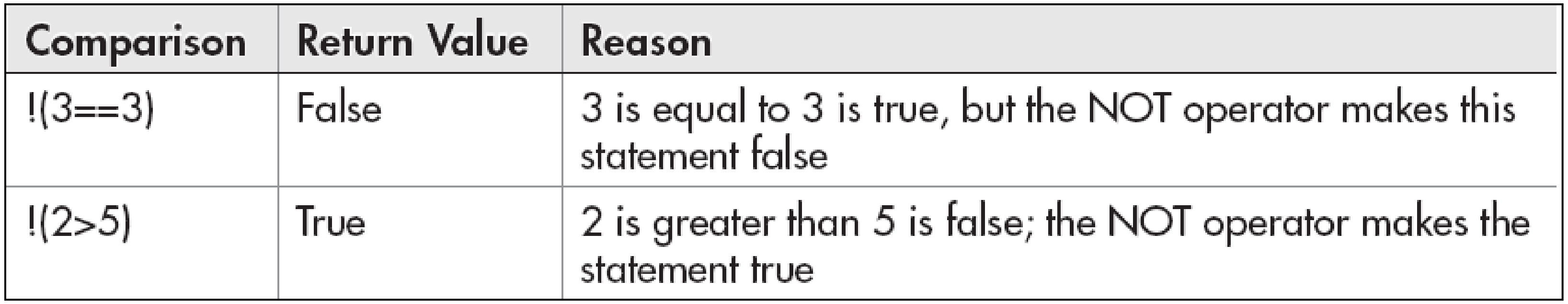
Statement	Return Value	Reason
(1==1)&&(2==2)	True	Comparisons on both sides are true: 1 is equal to 1, and 2 is equal to 2
(2>1)&&(3<=4)	True	Comparisons on both sides are true: 2 is greater than 1, and 3 is less than 4
(“A”<=“A”)&&(“c”!=“d”)	True	Comparisons on both sides are true: “A” is equal to “A”, and “c” is not equal to “d”
(1==1)&&(2==3)	False	Comparison on the right is false
(“a”!=“a”)&&(“b”!=“q”)	False	Comparison on the left is false
(2>7)&&(5>=20)	False	Comparisons on both sides are false

OR Operator (||)



Statement	Return Value	Reason
(2==2)||(3>5)	True	Comparison on the left is true
(5>17)||(4!=9)	True	Comparison on the right is true
(3==3)||(7<9)	True	Both comparisons are true
(4<3)||(2==1)	False	Both comparisons are false
(3!=3)||(4>=8)	False	Both comparisons are false

NOT Operator (!)



Comparison	Return Value	Reason
!(3==3)	False	3 is equal to 3 is true, but the NOT operator makes this statement false
!(2>5)	True	2 is greater than 5 is false; the NOT operator makes the statement true

Conditional Statements and Loops

What Is a Conditional Statement?

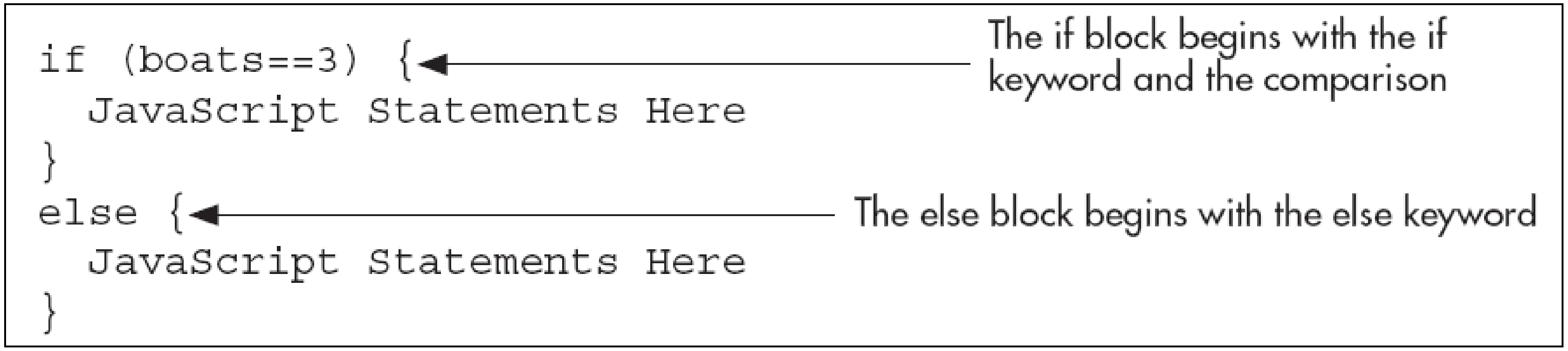
- A conditional statement is a statement that you can use to execute a bit of code based on a condition or to do something else if that condition is not met.
- You can think of a conditional statement as being a little like cause and effect.
- A conditional statement can be quite useful. Rather than executing every single line of code in the script, we could have certain sections of the script only be executed when a particular condition is met.
- We will study two types of conditional statement blocks used in JavaScript:
  - the if/else statement blocks and
  - the switch statement blocks.

Using if/else Statement Blocks

- You begin an if/else statement with the JavaScript keyword if, followed by a comparison in parentheses.

```
if (comparison here)

if (boats==3)
```

- A comparison will return a value of true or false. If the comparison returns true, the browser can go on to the next line. If it returns false, the browser begins looking for the else keyword, or the first line of script after the block following the if line is completed.



```
if (boats==3) {
  JavaScript Statements Here
}
else {
  JavaScript Statements Here
}
```

The if block begins with the if keyword and the comparison
The else block begins with the else keyword

- If the comparison of boats==3 returns true, the code we place within the brackets will be executed. If it returns false, the code inside the brackets is ignored and the line of code after the closing curly bracket is executed.

- If we wish to use an else block to execute a certain bit of code when the comparison returns false, we place the else keyword on the next line and then follow it with its own set of curly brackets.

Block Nesting

- If we *nest* something, we are basically putting one structure inside another structure of the same or a similar nature.
- With the if/else statement blocks, we are able to nest other if/else statements within the first block after the comparison (the “if block”) or within the second block after the else keyword (the “else block”).

```
if (have_cookbook=="yes") {
  if (meatloaf_recipe=="yes") {
    window.alert("Recipe found");
  }
  else {
    window.alert("Have the book but no recipe");
  }
}
else {
  window.alert("You need a cookbook");
}
```

This if/else block is nested within an outside if block

```
if (have_cookbook=="yes") {
  if (meatloaf_recipe=="yes") {
    window.alert("Recipe found");
  }
  else {
    window.alert("Have the book but no recipe");
  }
}
else {
  if (have_web_access=="yes") {
    window.alert("Find the recipe on the Web");
  }
  else {
    window.alert("You need a cookbook");
  }
}
```

This if/else block is nested within an outside if block

This if/else block is nested within an outside else block

```
if (have_cookbook=="yes") {
  if (meatloaf_recipe=="yes") {
    if (is_moms_meatloaf=="yes") {
      window.alert("Recipe found");
    }
    else {
      window.alert("Recipe found, but not like what mom makes");
    }
  }
  else {
    window.alert("Have the book but no recipe");
  }
}
else {
  if (have_web_access=="yes") {
    window.alert("Find the recipe on the Web");
  }
  else {
    window.alert("You need a cookbook");
  }
}
```

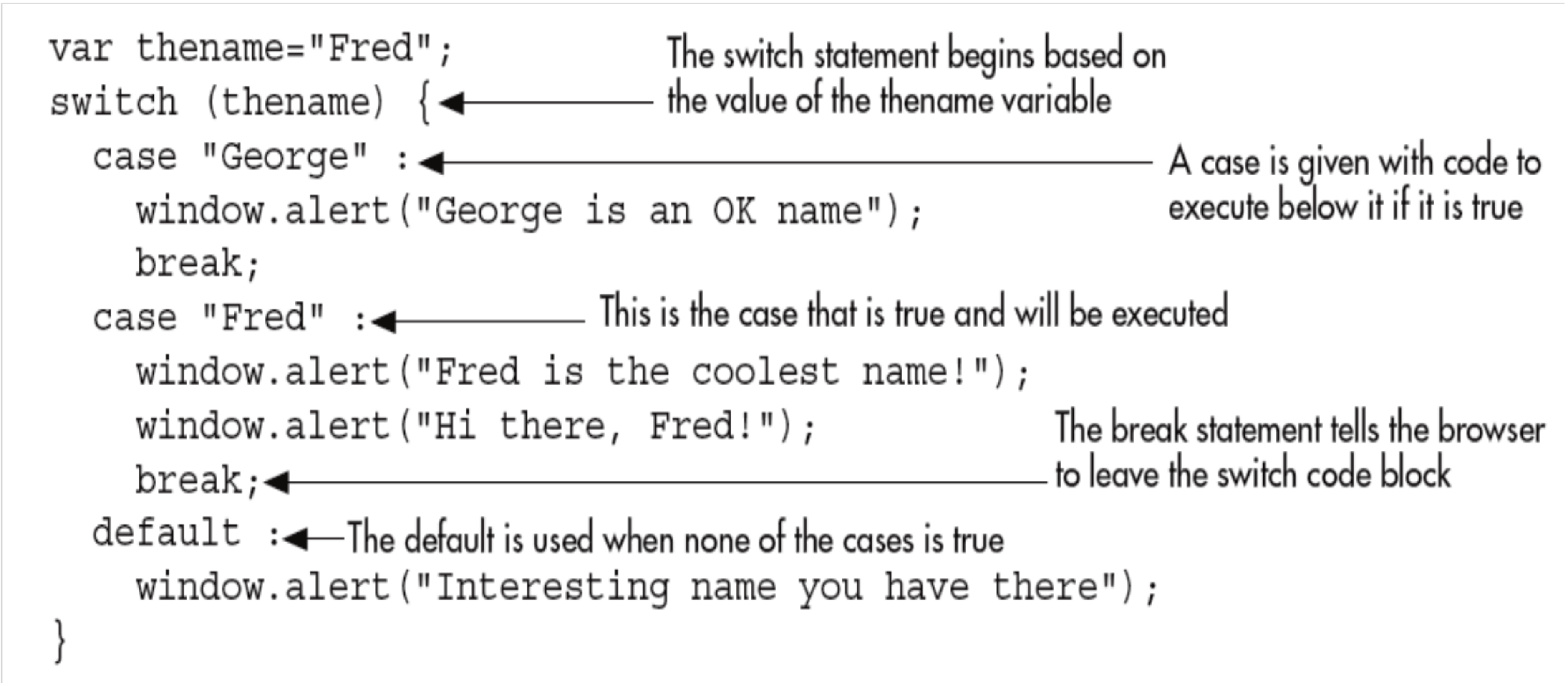
This if/else block is nested within a nested if block

Using the switch Statement

- The switch statement allows us to take a single variable value and execute a different block of code based on the value of the variable.
- If we wish to check for a number of different values, this can be an easier method than the use of a set of nested if/else statements.

```
switch (varname)
```

- We replace *varname* with the name of the variable we are testing. We could also replace it with some other sort of expression, such as the addition of two variables or some similar calculation, and have it evaluate.



```
var thename="Fred";
switch (thename) {
  case "George" :
    window.alert("George is an OK name");
    break;
  case "Fred" :
    window.alert("Fred is the coolest name!");
    window.alert("Hi there, Fred!");
    break;
  default :
    window.alert("Interesting name you have there");
}
```

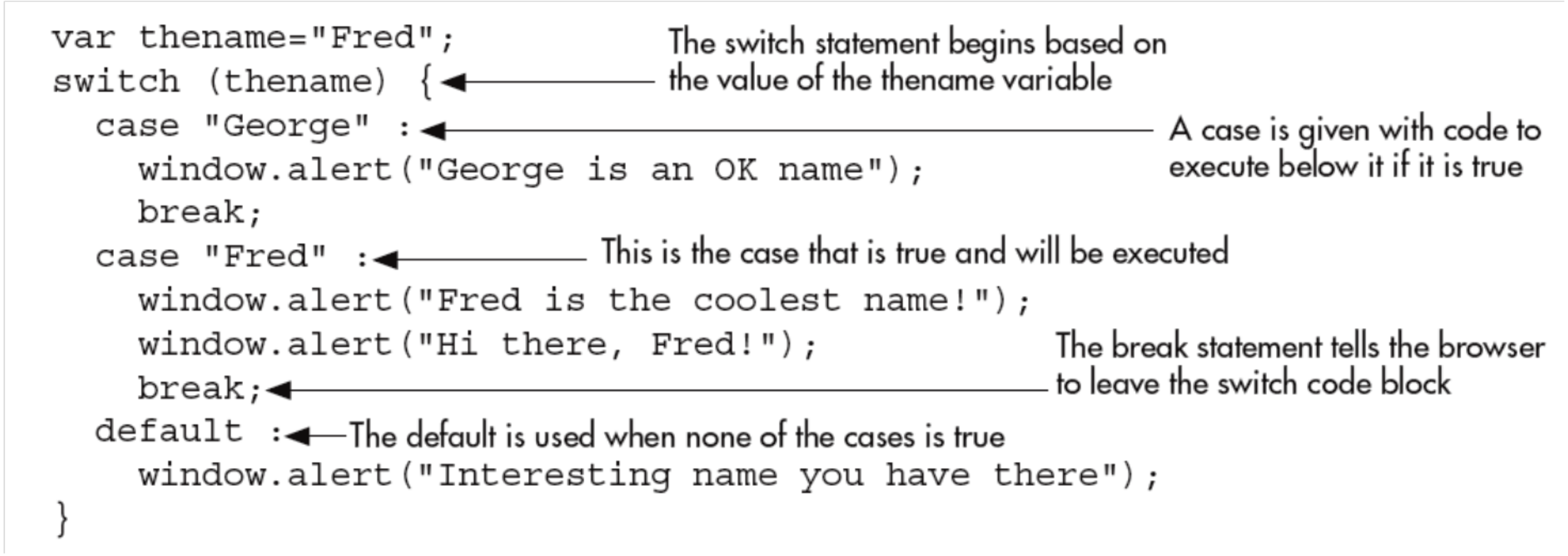
The switch statement begins based on the value of the thename variable

A case is given with code to execute below it if it is true

This is the case that is true and will be executed

The break statement tells the browser to leave the switch code block

The default is used when none of the cases is true



```
var thename="Fred";
switch (thename) {
  case "George" :
    window.alert("George is an OK name");
    break;
  case "Fred"  :
    window.alert("Fred is the coolest name!");
    window.alert("Hi there, Fred!");
    break;
  default  :
    window.alert("Interesting name you have there");
}
```

The switch statement begins based on the value of the thename variable
A case is given with code to execute below it if it is true
This is the case that is true and will be executed
The break statement tells the browser to leave the switch code block
The default is used when none of the cases is true

- First, this example declares and assigns a variable named thename; it is given a value of Fred. Next, the switch statement begins, using the variable thename as the basis for comparison.Then, the block is opened with a curly bracket, followed by the first case statement. Written like this, it is saying, “If thename is equal to George then execute the commands after the colon at the end of this line.” If thename were equal to George, you would get an alert.

- The break statement, tells the browser to exit the code block and move on to the next line of code after the block.

- thename is not equal to George, so this case is skipped; however, the next comparison returns true because thename is equal to Fred in the script. Thus, the set of statements in this case block will be executed.

- Finally, there is keyword default. This is used in the event that none of the case statements returns true. If this happens, the default section of code will be executed. Notice that we don’t need the break statement after the default section of code, because it is at the end of the switch block anyway, so the browser will exit the block afterward, eliminating the need for the break statement.

Using the Conditional Operator

- The conditional operator allows us to shorten the amount of code required.
- It allows us to place a condition before the question mark (?) and place a possible value on each side of the colon (:)

```
varname = (conditional) ? value1 : value2;
```

```
var mynum=1;
var mymessage;
if (mynum==1) {
  mymessage="You win!";
}
else {
  mymessage="Sorry! Try again!";
}
```

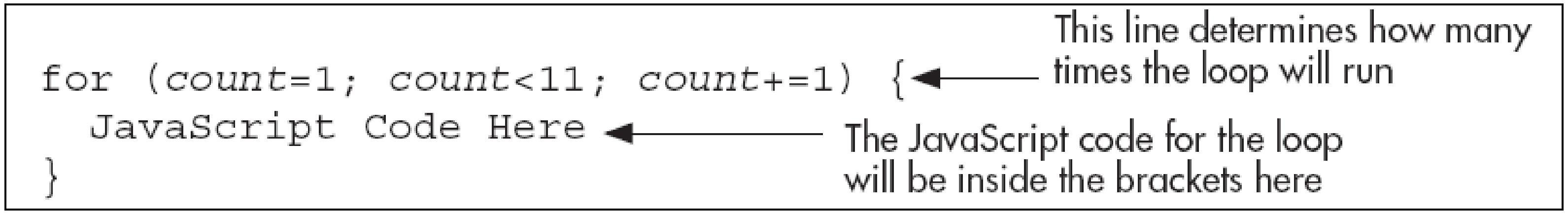
```
var mynum=1;
var mymessage;
mymessage = (mynum==1) ? "You win!" : "Sorry! Try Again!";
```

Loop

- A loop is a block of code that allows us to repeat a section of code a certain number of times, perhaps changing certain variable values each time the code is executed.
- Loops are useful because they allow us to repeat lines of code without retyping them or using cut and paste in your text editor. This not only saves the time and trouble of repeatedly typing the same lines of code, but also avoids typing errors in the repeated lines.
- The loop structures covered in this session are:
  - *for*,
  - *while*, and
  - *do while* loops.

for

- The first line of a for loop would look similar to the following line:



```
for (count=1; count<11; count+=1) {
  JavaScript Code Here
}
```

This line determines how many times the loop will run
The JavaScript code for the loop will be inside the brackets here

- The first thing is the for keyword. This is followed by a set of parentheses with three statements inside. These three statements tell the loop how many times it should repeat by giving it special information.

- The first statement (*var count*=1) creates a variable named *count* and assigns it an initial value of 1. This initial value can be any number. This number is used as a starting point for the number of times the loop will repeat.

- The next statement (*count*<11) tells the loop when to stop running. The loop will stop running based on this conditional statement. The condition here is to stop only when the variable *count* is no longer less than 11. This means that if we add 1 to the value of *count* each time through the loop, the loop's last run-through will be when *count* is equal to 10. When 1 is added to 10, it becomes 11; and that doesn't pass the conditional test, so the loop stops running

- The last statement in the set (*count* +=1) determines the rate at which the variable is changed and whether it gets larger or smaller each time. In the preceding code, we add 1 to the variable each time we go back through the loop.

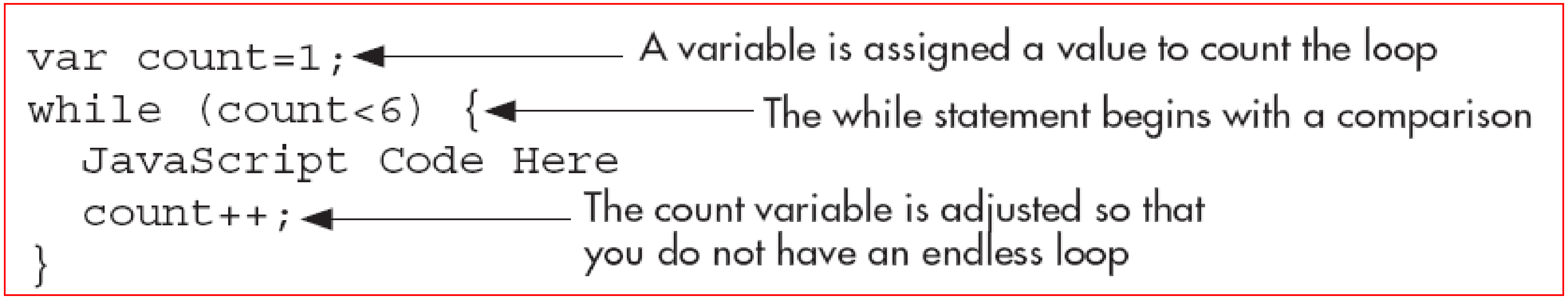
- To finish the structure, we insert the curly brackets to enclose the code that we wish to use within the loop.

while

- A while loop just looks at a short comparison and repeats until the comparison is no longer true.

```
while (count<11)
```

- The while statement does not create a variable the way a for statement can. When using a while loop, we must remember to declare the variable we wish to use and assign it a value before we insert it into the while loop.



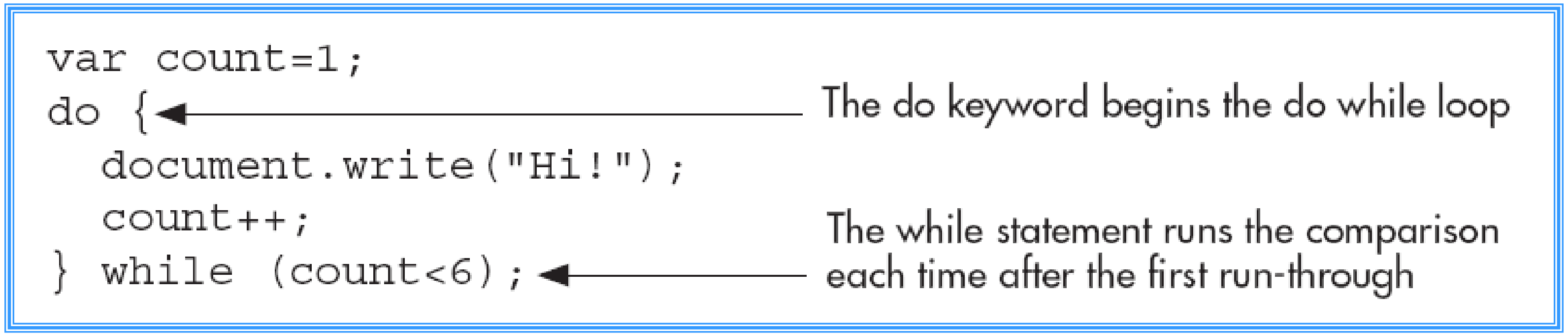
```
var count=1;
while (count<6) {
  JavaScript Code Here
  count++;
}
```

A variable is assigned a value to count the loop
The while statement begins with a comparison
The count variable is adjusted so that you do not have an endless loop

- First, 1 is assigned to the variable count before the loop begins. This is important to do so that the loop will run the way we expect it to run.
- This loop is set up to repeat five times, given the initial value of the variable and the increase in the value of the variable by 1 each time through (count++)
- In a while loop, we must also remember to change the value of the variable we use so that we do not get stuck in a permanent loop. If the previous sample loop had not included the count++ code, the loop would have repeated indefinitely.
- So, the main things we must remember with a while loop are to give the variable an initial value before the loop and to adjust the value of the variable within the loop itself.

do while

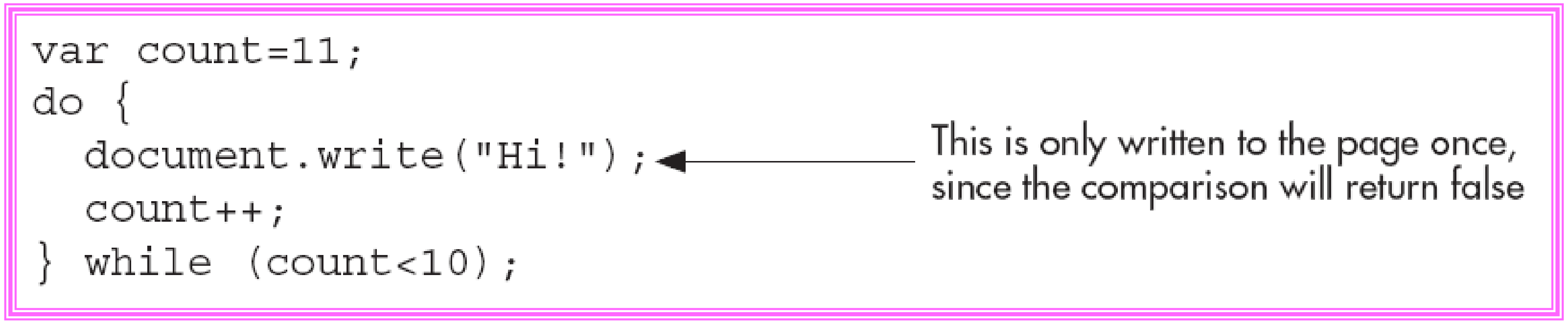
- It is special because the code within the loop is performed at least once, even if the comparison used would return false the first time.
- A comparison that returns false in other loops on the first attempt would cause them never to be executed. In the case of a do while loop, the loop is executed once, and then the comparison is used each time afterward to determine whether or not it should repeat.



```
var count=1;
do {
  document.write("Hi!");
  count++;
} while (count<6);
```

The do keyword begins the do while loop

The while statement runs the comparison each time after the first run-through



```
var count=11;
do {
  document.write("Hi!");
  count++;
} while (count<10);
```

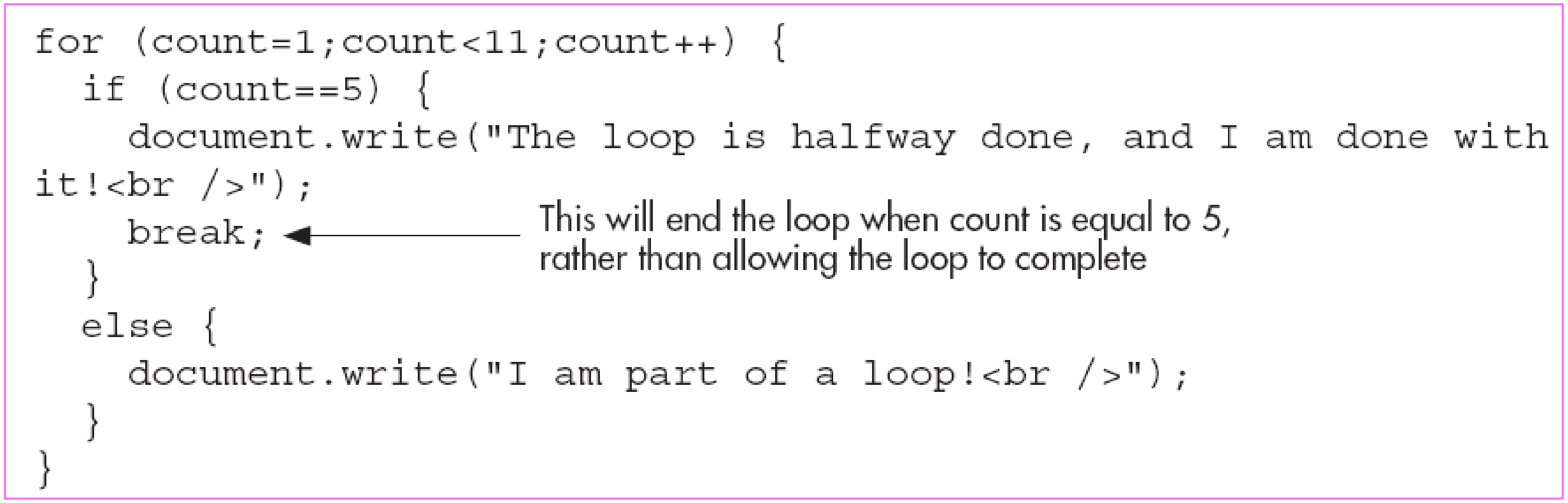
This is only written to the page once, since the comparison will return false

Using break and continue

- The break and continue statements allow us to stop what a loop is currently doing, but work in different ways.

break

- The break statement stops the loop at that point and completely exits the loop, moving on to the next JavaScript statement after the loop.



```
for (count=1;count<11;count++) {
  if (count==5) {
    document.write("The loop is halfway done, and I am done with
it!<br />");
    break;
  }
  else {
    document.write("I am part of a loop!<br />");
  }
}
```

This will end the loop when count is equal to 5, rather than allowing the loop to complete

- This loop will go through normally until count is equal to 5. When this happens, the break statement is used to end the loop entirely. Thus, rather than going through the loop ten times, the loop will only be executed five times.

continue

- The continue statement will stop the loop from executing any statements after it during the current trip through the loop.
- However, it will go back to the beginning of the loop and pick up where it left off, rather than exiting the loop entirely.

```
for (count=1;count<11;count++) {
  if (count==5) {
       continue;
  }
    document.write(count+". I am part of a loop!<br />");
}
```

- This time, nothing is written to the page when count is equal to 5. Instead, the loop is told to go back to the beginning and continue from there. The result is that the “I am part of a loop!” message will be written to the page only nine times (since nothing happens when count is equal to 5). The loop is allowed to continue where it left off, rather than being left completely.

Object Based Programming

- JavaScript is an Object Based Programming language, and allows you to define your own objects and make your own variable types.
- An object is a way of modeling something real, even though the object is an abstract entity.
- Properties are the values associated with an object.
- Methods are the actions that can be performed on objects.

Why Objects Are Useful

- Objects are useful because they give a way to organize things within a script. Rather than having a bunch of similar variables that are out there on their own, we can group them together under an object.
- In JavaScript, we access object properties through the use of the dot operator, which is just a dot or period.
- For instance, if we wanted the value of the seats property of the car, we could access it with the following line:

```
var chtype= car.seats;
```

Creating Objects

- Naming conventions
- The structure of an object and
- Adding methods in objects

Naming

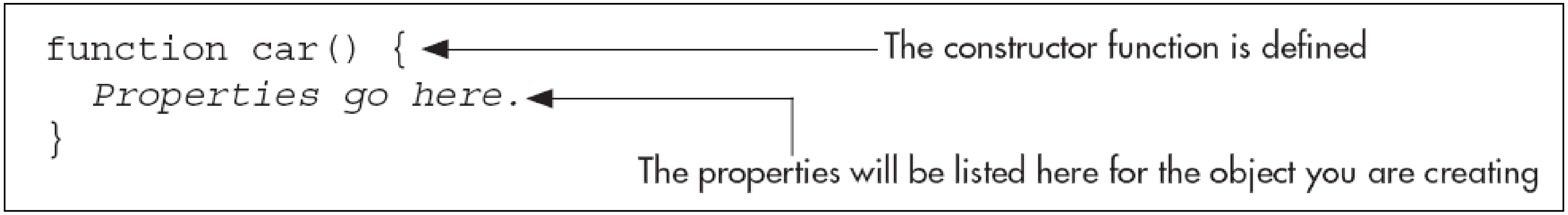
- As with variables and functions, there are certain rules we have to follow when naming objects in JavaScript. They are essentially the same rules we follow for naming variables and functions

Object Structure

- There are two ways to create objects in JavaScript:
  - by using a constructor function or
  - by using an object initializer

Constructor Functions

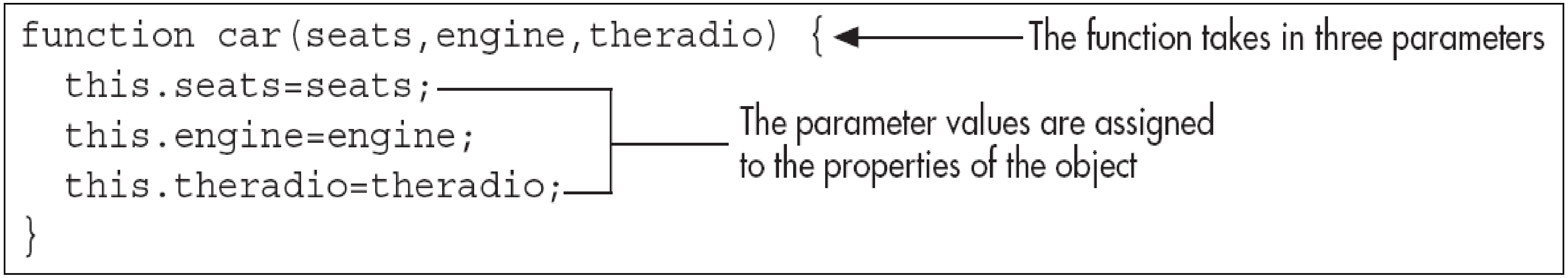
- A constructor function allows us to build an object using the same basic syntax as a regular function. The only difference is the code we place inside of the function and how we access its contents.
- For example, to create car object, we would create a constructor function named car() and then add properties within the function. The following example shows an outline of the car() function:



```
function car() {
  Properties go here.
}
```

The constructor function is defined
The properties will be listed here for the object you are creating

- To complete the preceding function, we need to add properties to the function.
- We create an object named car with the properties of seats, engine, and theradio. The following code shows how this is done:



```
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
}
```

The function takes in three parameters
The parameter values are assigned to the properties of the object

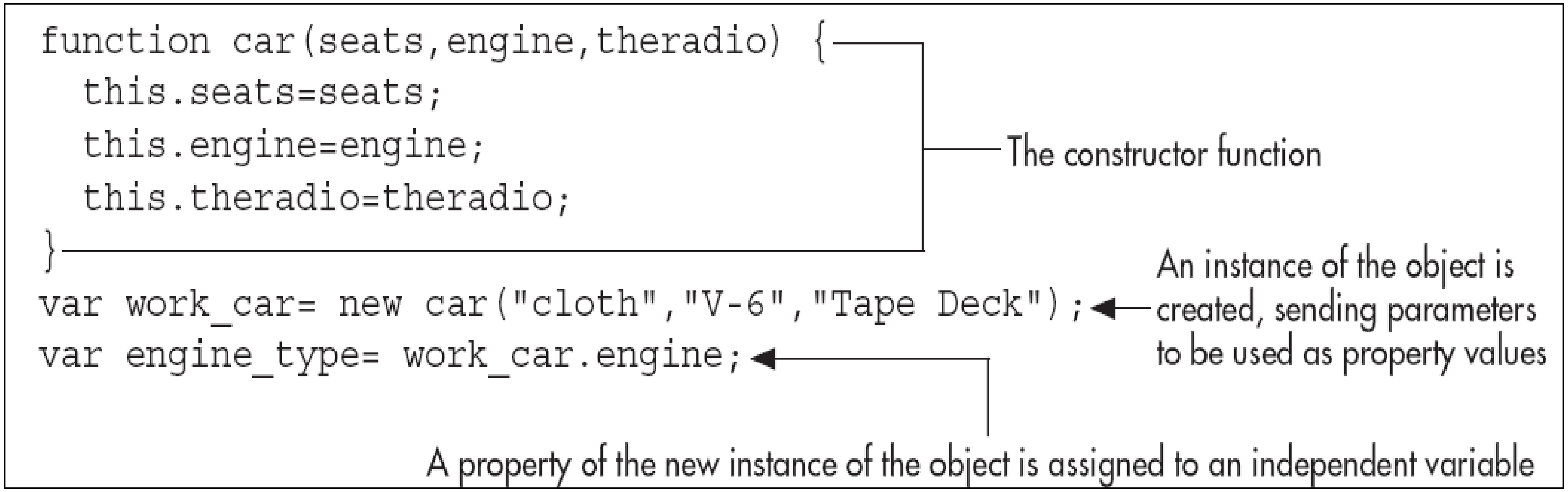
  - In this code, on the first line, the function takes three parameters, which is the number of properties we want the car object to have.
  - The next thing is that the values of the parameters are assigned to the properties we want the car object to have
  - There is a new keyword named ***this***. The keyword this in JavaScript is used to represent the current object being used.

- Once the object's properties set with the constructor function, we need to create an *instance* of the object in order to use it, because a constructor function creates only the structure of an object, not a usable instance of an object.
- To create an instance of an object, we use another JavaScript keyword: new
- The use of the new keyword to create an instance of car object is shown in the following code:

```
var work_car= new car("cloth","V-6","Tape Deck");
```

  - We are creating a new variable named work_car. This variable will be a new instance of the car object due to the value we are assigning to it.
  - The work_car variable is assigned the result of the car constructor function, with a twist. In front of the call to the car function is the *new* keyword, which makes sure we are creating a new instance of the constructor function object.
  - The car function is called with values sent as parameters. These are the values we want to use for this instance of the car object. Given the order, we are saying that we want the seats to be cloth, the engine to be V-6, and the radio to be Tape Deck.

Putting the Pieces Together

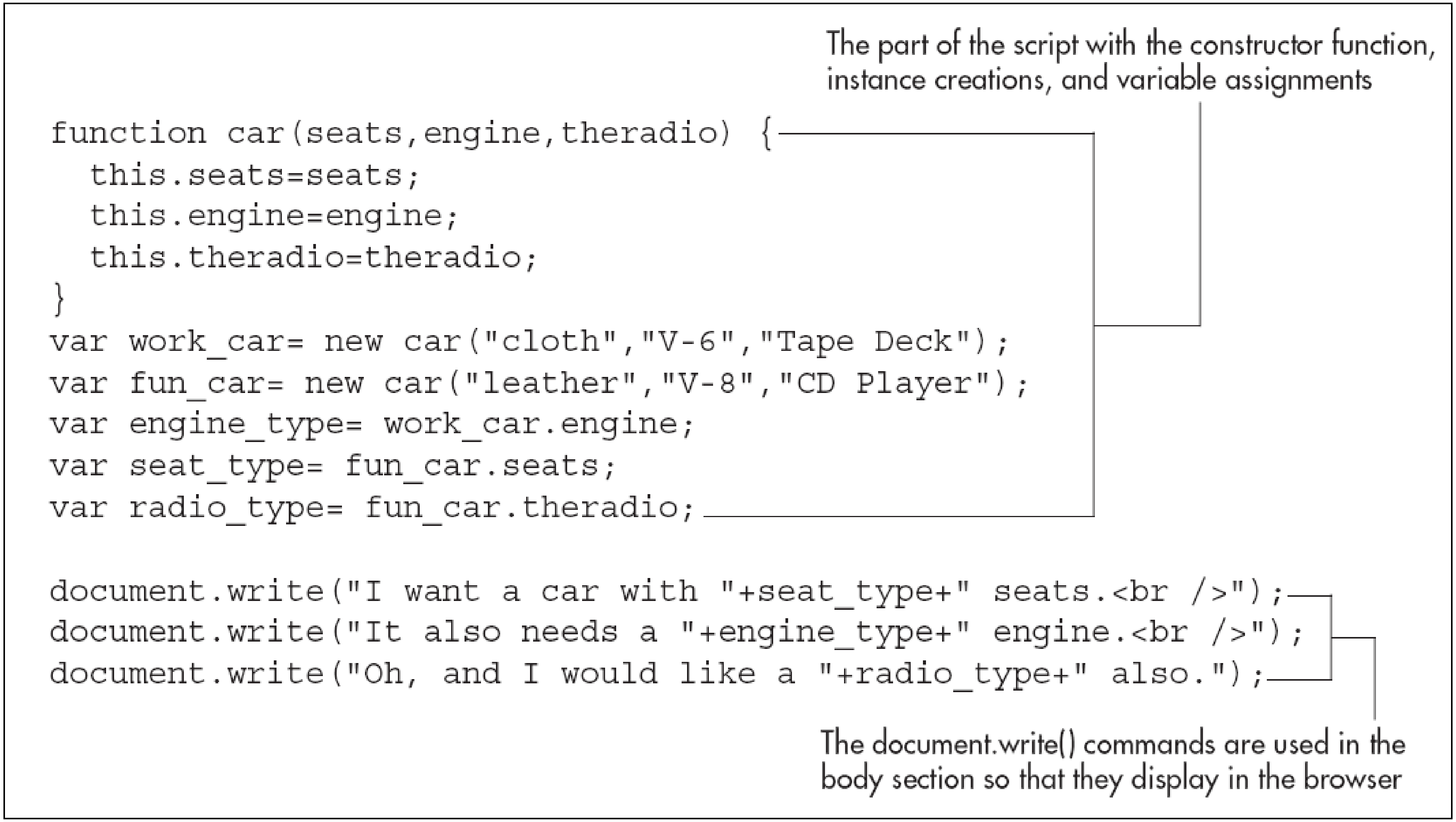


```
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
}
var work_car= new car("cloth","V-6","Tape Deck");
var engine_type= work_car.engine;
```

The constructor function

An instance of the object is created, sending parameters to be used as property values

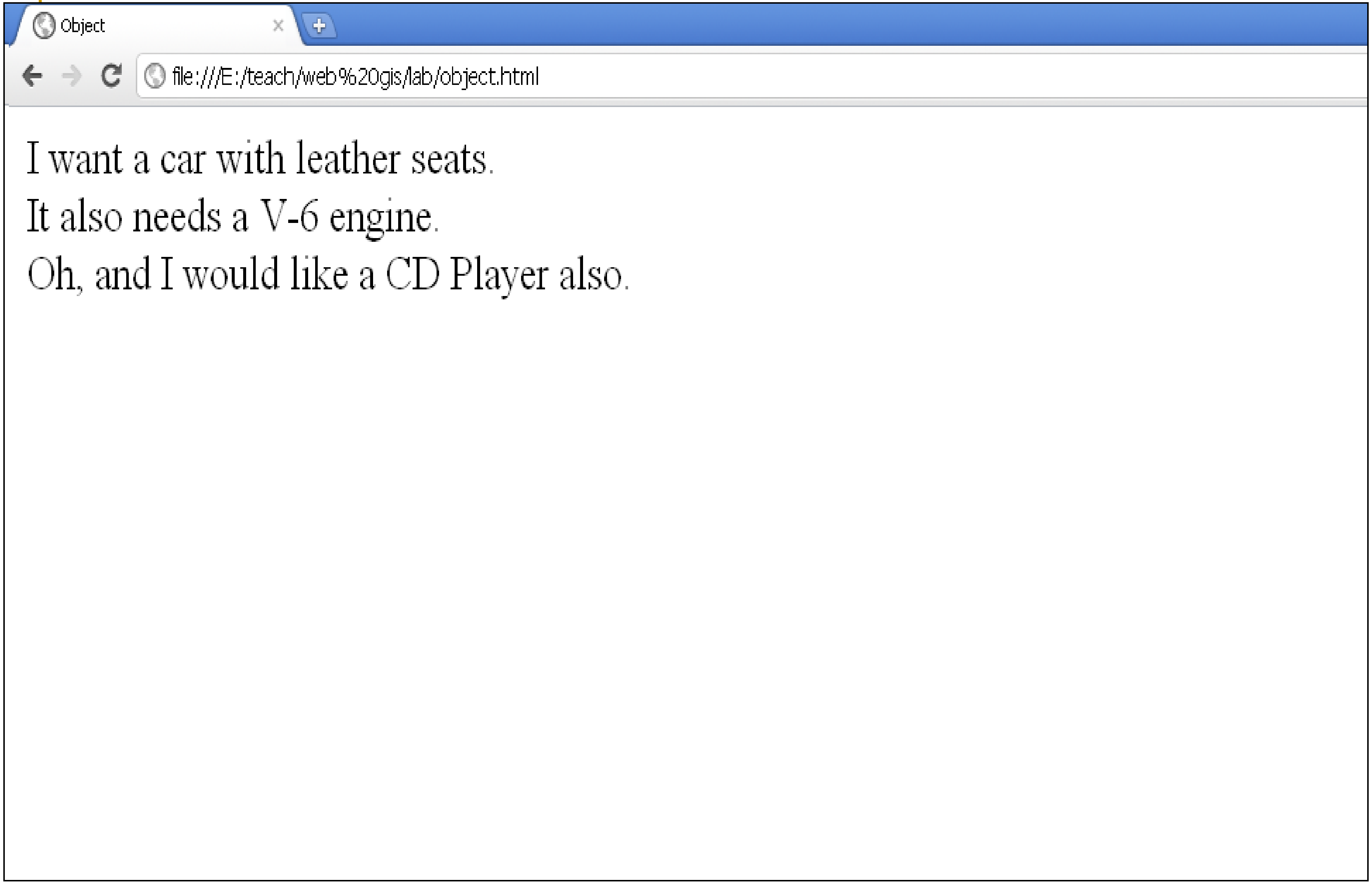
A property of the new instance of the object is assigned to an independent variable



The part of the script with the constructor function, instance creations, and variable assignments

```
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
}
var work_car= new car("cloth","V-6","Tape Deck");
var fun_car= new car("leather","V-8","CD Player");
var engine_type= work_car.engine;
var seat_type= fun_car.seats;
var radio_type= fun_car.theradio;

document.write("I want a car with "+seat_type+" seats.<br />");
document.write("It also needs a "+engine_type+" engine.<br />");
document.write("Oh, and I would like a "+radio_type+" also.");
```

The document.write() commands are used in the body section so that they display in the browser



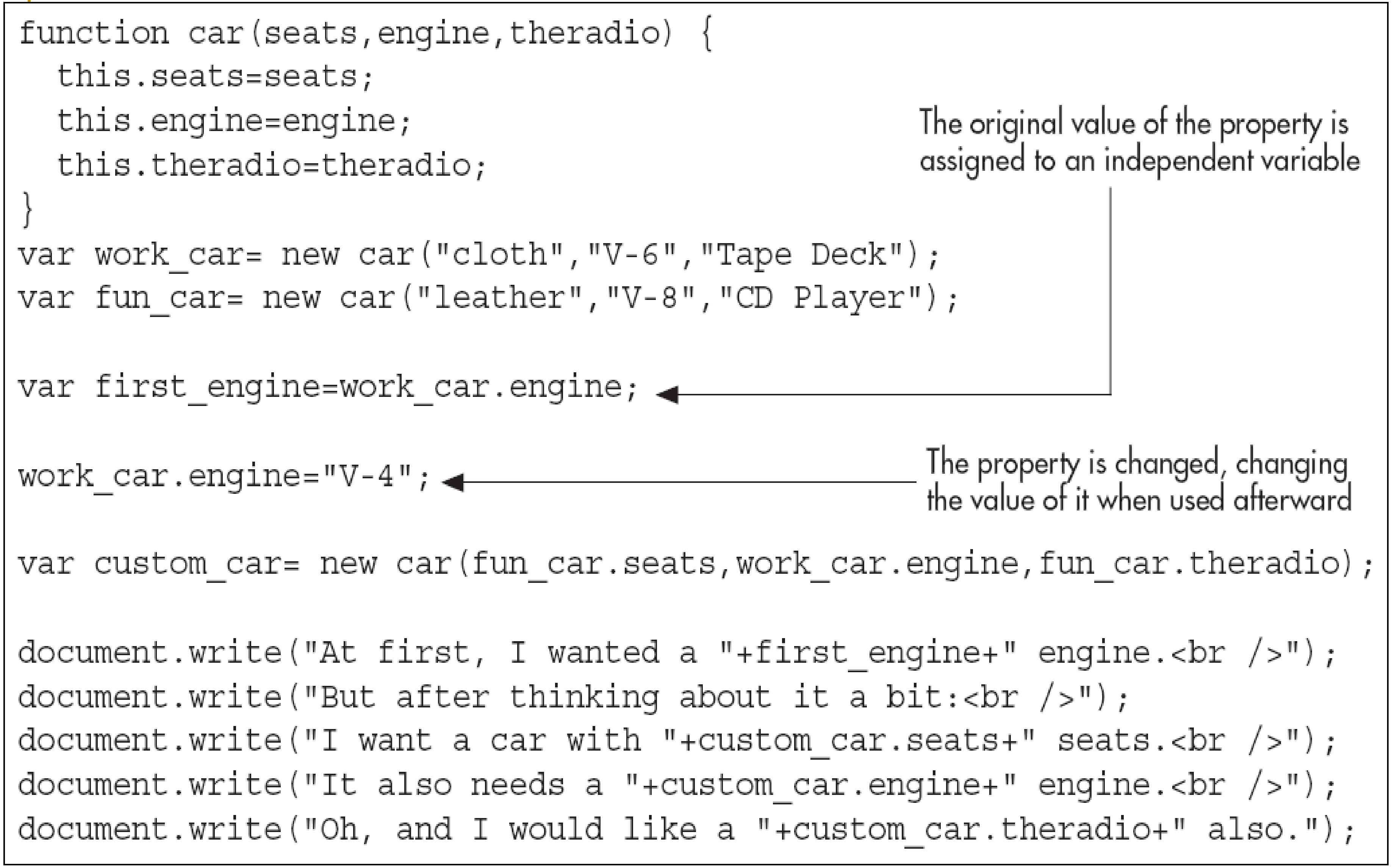
I want a car with leather seats.
It also needs a V-6 engine.
Oh, and I would like a CD Player also.

Property Values

- We can change the value of an object property by assigning it a new value, just like a variable.
- For example, if we wanted to change the value of the work_car engine property from the previous examples, we could just assign it a new value

```
work_car.engine= "V-4";
```

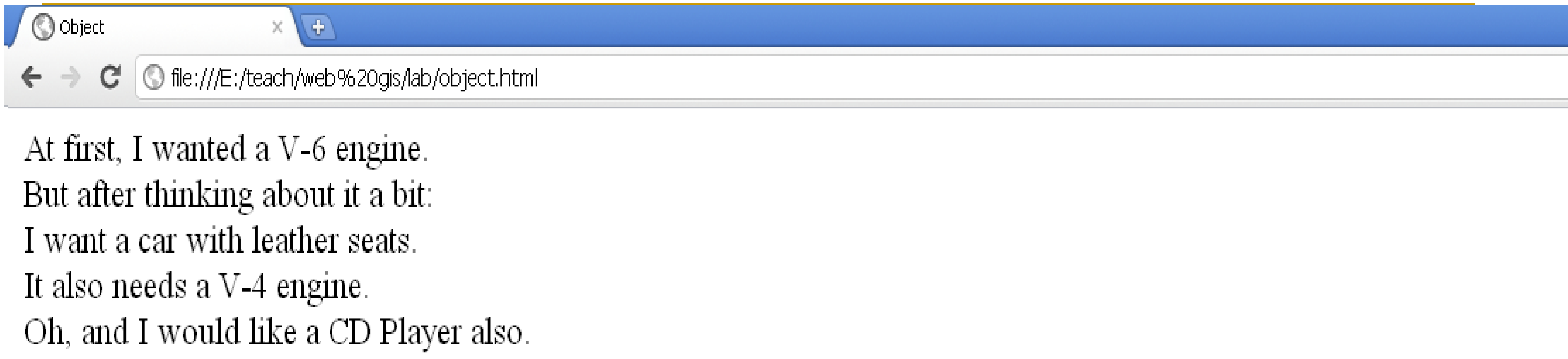
  - It is important to note that the preceding assignment will change the value of the work_car.engine property for any calls made to it after the change.
  - Anything we do with its value before the change would not be affected.



```
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
}
var work_car= new car("cloth","V-6","Tape Deck");
var fun_car= new car("leather","V-8","CD Player");

var first_engine=work_car.engine;

work_car.engine="V-4";

var custom_car= new car(fun_car.seats,work_car.engine,fun_car.theradio);

document.write("At first, I wanted a "+first_engine+" engine.<br />");
document.write("But after thinking about it a bit:<br />");
document.write("I want a car with "+custom_car.seats+" seats.<br />");
document.write("It also needs a "+custom_car.engine+" engine.<br />");
document.write("Oh, and I would like a "+custom_car.theradio+" also.");
```

The original value of the property is assigned to an independent variable

The property is changed, changing the value of it when used afterward



```
At first, I wanted a V-6 engine.
But after thinking about it a bit:
I want a car with leather seats.
It also needs a V-4 engine.
Oh, and I would like a CD Player also.
```

Adding Methods

- A method is a function call that is part of an object.
- The function called can perform various tasks that we might want to execute with the properties of the object.
- Consider the car object that we created with a constructor function. We might want to have functions for the car to stop, accelerate, or turn. We might also decide to make a calculation of some sort, like a payment. If we wanted to add a function for that object that would calculate the monthly payments on the various types (instances) of cars that we sent to it, we could create a function like the following:

Function to be used as a method is defined

The payment variable is created and given an initial value

```
function get_payment() {
  var the_payment=250;
  if(this.seats == "leather") {
    the_payment+=100;
  }
  else {
    the_payment+=50;
  }

 if(this.engine == "V-8") {
   the_payment+=150;
 }
 else {
   the_payment+=75;
 }
 if(this.theradio == "CD Player") {
   the_payment+=35;
 }
 else {
  the_payment+=10;
 }
 return the_payment;
}
```

An if/else statement decides how much to add to the payment variable based on the value of the seats property of an object

An if/else statement decides how much to add to the payment variable based on the value of the engine property of an object

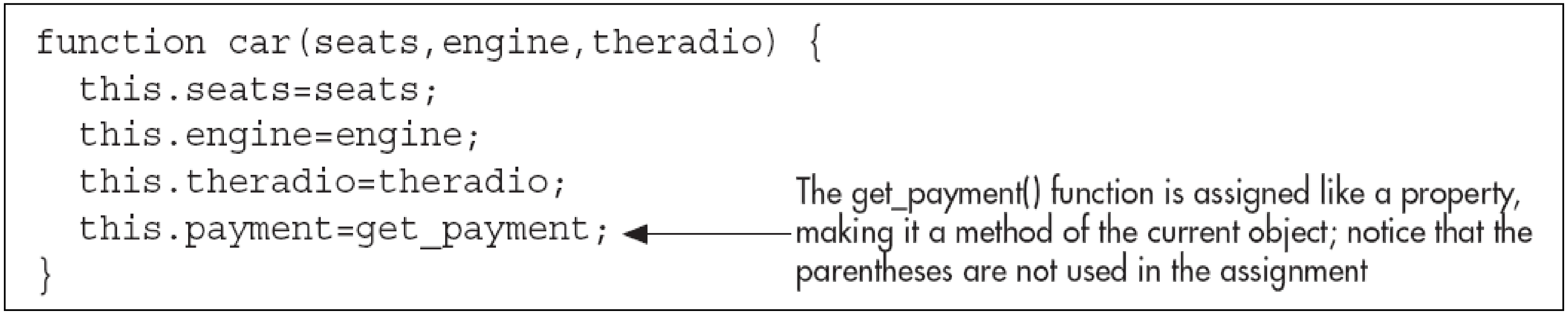
An if/else statement decides how much to add to the payment variable based on the value of the theradio property of an object

The value of the_payment is returned

The previous function is really long. How it can be shortened?

The conditional operator is used in place of the if/else statements to make the code shorter

```
function get_payment() {
  var the_payment=250;
  the_payment += (this.seats == "leather") ? 100 : 50;
  the_payment += (this.engine == "V-8") ? 150 : 75;
  the_payment += (this.theradio == "CD Player") ? 35 : 10;
  return the_payment;
}
```



```
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
  this.payment=get_payment;
}
```

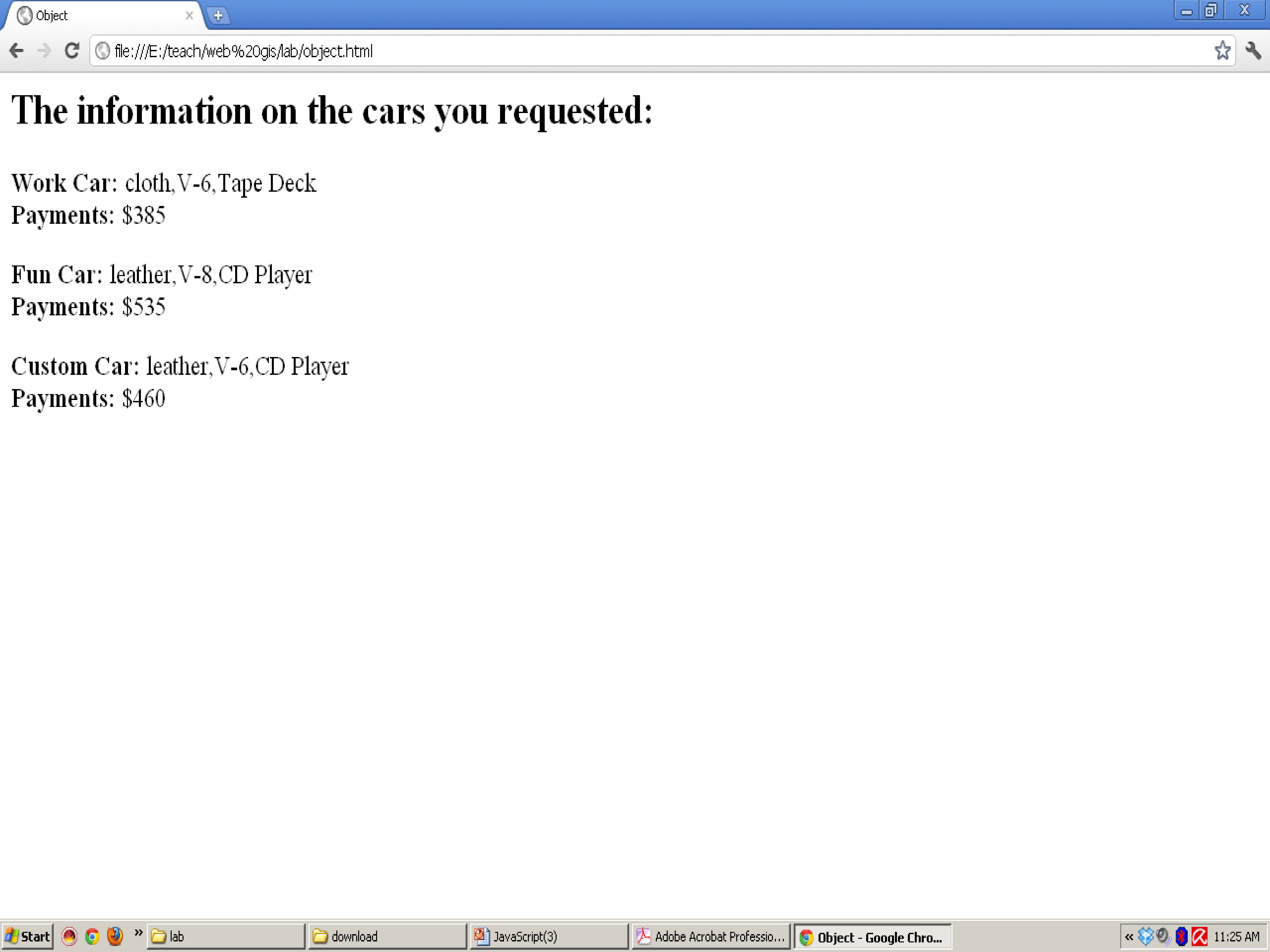
The get_payment() function is assigned like a property, making it a method of the current object; notice that the parentheses are not used in the assignment

- Notice that this example defines a method named payment that calls the get_payment() function from the constructor function.
- Also notice that when the function is called here, the parentheses are not used on the end of the function call. This is how outside function becomes a method of the object (by assigning it the function rather than the result of the function).

```
function get_payment() {
  var the_payment=250;
  the_payment += (this.seats == "leather") ? 100 : 50;
  the_payment += (this.engine == "V-8") ? 150 : 75;
  the_payment += (this.theradio == "CD Player") ? 35 : 10;
  return the_payment;
}
function car(seats,engine,theradio) {
  this.seats=seats;
  this.engine=engine;
  this.theradio=theradio;
  this.payment=get_payment;
}

var work_car= new car("cloth","V-6","Tape Deck");
var fun_car= new car("leather","V-8","CD Player");
var custom_car= new car(fun_car.seats,work_car.engine,fun_car.theradio);

var work_car_payment= work_car.payment();
var fun_car_payment= fun_car.payment();
var custom_car_payment= custom_car.payment();

document.write("<h2>The information on the cars you requested:</h2>");
document.write("<strong>Work Car: </strong>");
document.write(work_car.seats+","+work_car.engine+","+work_car.theradio);
document.write("<br />");
document.write("<strong>Payments:</strong> $"+work_car_payment);
document.write("<p>");
document.write("<strong >Fun Car: </strong>");
document.write(fun_car.seats+","+fun_car.engine+","+fun_car.theradio);
document.write("<br />");
document.write("<strong>Payments:</strong> $"+fun_car_payment);
document.write("</p>");
document.write("<p>");
document.write("<strong>Custom Car: </strong>");
document.write(custom_car.seats+","+custom_car.engine+",");
document.write(custom_car.theradio);
document.write("<br />");
document.write("<strong>Payments:</strong> $"+custom_car_payment);
document.write("</p>");
```

The returned value of the method function for each of three instances of the object is assigned to three independent variables

Various object properties and variables are used in the document.write() statements to create a listing of the cars, their features, and the payment amounts


Object
file:///E:/teach/web%20gis/lab/object.html
Start
lab
download
JavaScript(3)
Adobe Acrobat Professio...
Object - Google Chro...
11:25 AM

The information on the cars you requested:

Work Car: cloth,V-6,Tape Deck
Payments: $385

Fun Car: leather,V-8,CD Player
Payments: $535

Custom Car: leather,V-6,CD Player
Payments: $460