Hello World

A practical introduction to computer programming

Your first computer program

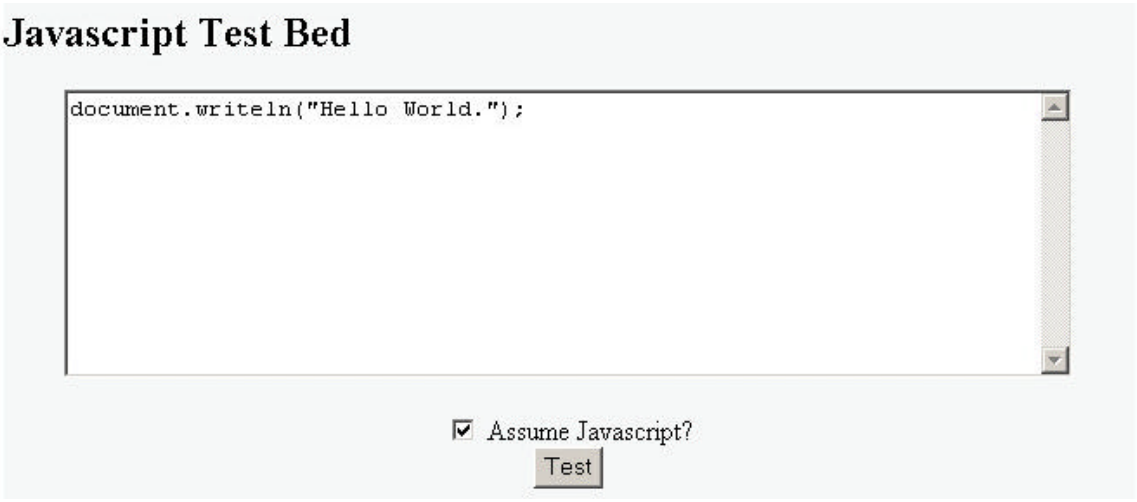
As the title suggests, this piece aims to introduce you to computer programming through practice, not theory. Accordingly, let's get started.

Step 1: Load our Javascript Editor

To start programming in Javascript, load the web page http://www.leeholmes.com/pers/projects/testbed.html.

Step 2: Write a simple program

Next, erase "Enter some code" and type the following into the text field:



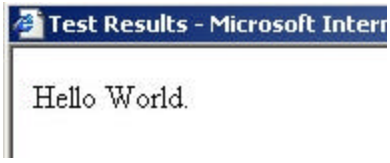
Make sure that you've checked the box, "Assume Javascript?" Also, note that *everything* matters in this sentence: punctuation, brackets, quotes, letters, and semicolons. However, spaces only matter when they're inside of words or quotes:

- "document.write ln(…)" doesn't run.
- "document.writeln(" H e llo W orld ");" looks odd.
- "document . writeln ("Hello World") ;" runs and looks fine.

Step 3: Run your program

To run your program, click the "Test" button.

Step 4: View the results



You should see the words, "Hello World." appear in a new web page.

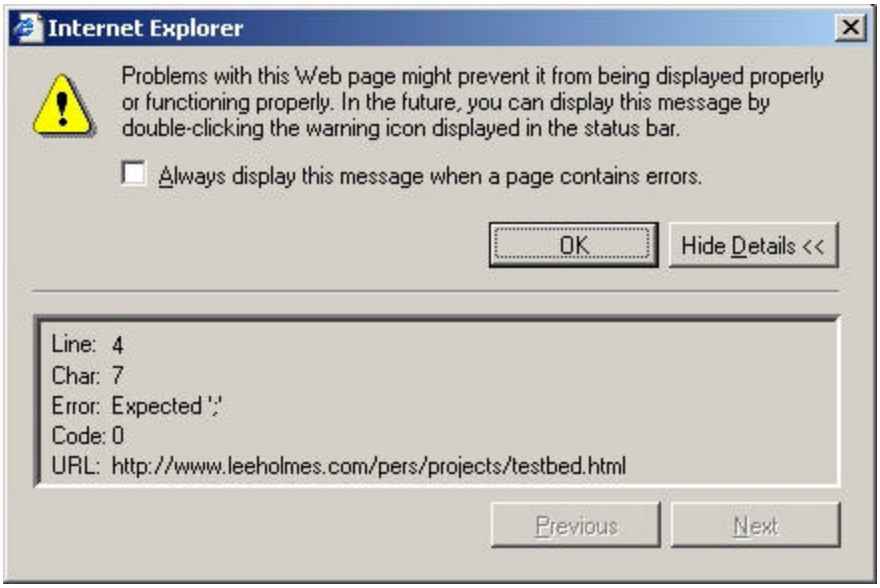
Step 5: Introduce an error

To create an error, close the "Test Results" window and type the following into the same text field:

Step 6: View the results

Click the "Test" button again to see the results. You should see a new web page appear with an error. For example, Internet Explorer complains:

If you double-click the error, you should see a more descriptive error message:



Because of the way the "Javascript Testbed" works, subtract 3 from the line number that your browser complains about.

What did we learn?

This simple program illustrates several important points of computer programming.

1) Javascript lines should end with a semicolon (;).
2) Whatever we write between *document.writeln(*" and *");* gets printed to the web page. Just so you know, "writeln" is short for "write line."
3) We can double-click our browser's error message to examine errors in our Javascript.

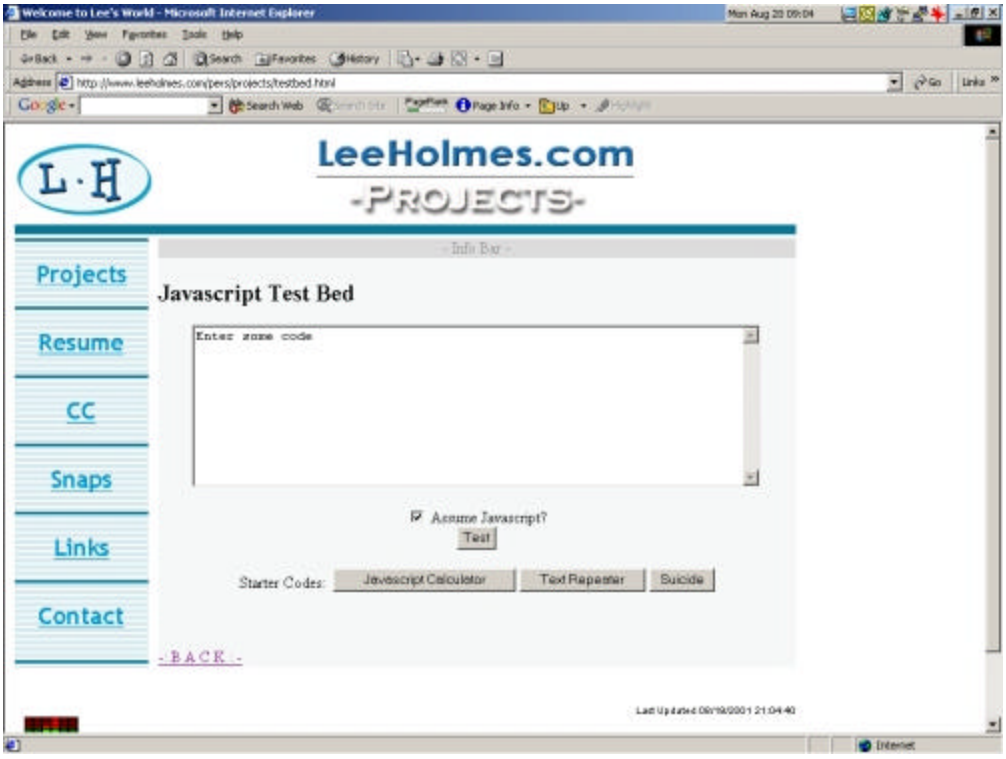
Exercise 1

Write a Javascript program to print your name to the web page. When you're done, go to the end of this document for the answer.

Dynamic Programming

You might have noticed that our last program wasn't very flexible. If you wanted to say "Hello, Web" you'd have to change the code. Instead, try this.

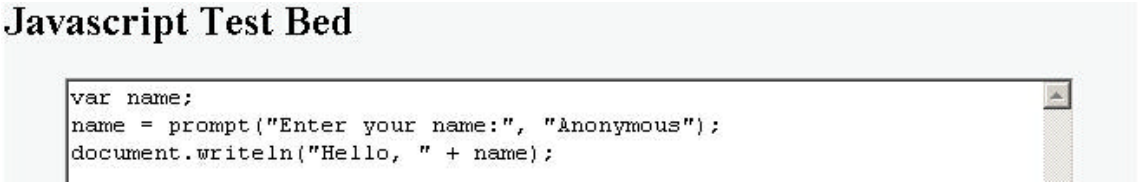
Step 1: Load our Javascript Editor



Load (or re-load) the web page http://www.leeholmes.com/pers/projects/testbed.html.

Step 2: Write a simple program

Again, erase "Enter some code" and type the following into the text field:



Make sure that you've checked the box, "Assume Javascript?"

Step 3: Run your program

To run your program, click the "Test" button.

Step 4: View the results

Your program should ask for your name (with "Anonymous" as the default,)

then greet you on a new web page.

What did we learn?

This last program introduced a few more bits of complexity.

1) We can get information from a user with the Javascript "prompt" command. Whatever we put in the first set of quotes gets displayed in the prompt box. Whatever we put in the second set of quotes gets put into the prompt automatically (but the user can still change it.)
2) We can store information (what a user typed in a prompt, for example). For this, we write "*whereToStore = whatToStore;*" *whereToStore* (ie: name) is where to store the information. *whatToStore* is the thing (ie: prompt…) that creates the information.
3) We must first tell Javascript about this "storage place" (actually called a *variable*) with the line, "var *storageplace;*" Guess what "var" stands for?

4) Javascript can combine pieces of information (ie: “Hello, “ and a variable) into a single piece of information (ie: “Hello, Lee”). To do this, separate them with a “+” sign.
5) Information in quotes (ie: “Hello”, “Anonymous”) gets used as-is, while information outside of quotes (ie: name, 2+2) gets simplified. Here are some examples to illustrate:
   - “Hello “ + “World” gives “Hello World”
   - “Hello” + “World” gives “HelloWorld”
   - “Hello “ + name gives “Hello Lee” if the ‘name’ variable holds the information ‘Lee’
   - “1 + 2 is “ + (1+2) gives “1 + 2 is 3”
   - “1 + “ + myNumber + “ is “ + (1+myNumber) gives “1 + 3 is 4” if the ‘myNumber’ variable holds the information ‘3’
   - “1 + myNumber is “ + (1+myNumber) gives “1 + myNumber is 4” if the ‘myNumber’ variable holds the information ‘3’. Notice how putting ‘myNumber’ inside of the quotes stops Javascript from simplifying it!

Exercise 2

Write a program to:

1) Ask the user their favourite meal
2) Ask the user their favourite day
3) Use their responses to write a sentence like “I like hamburgers on sunday, too!”

When you’re done, go to the end of this document for the answer.

Making Decisions

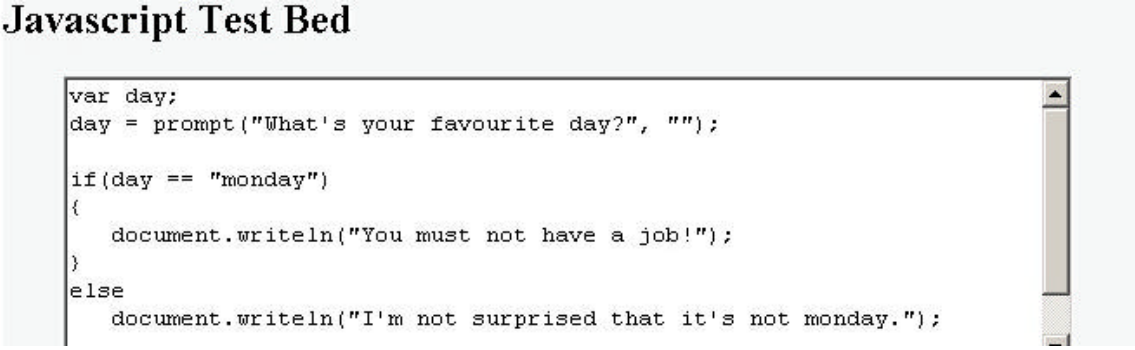
Even though your last program was friendly, you might not always agree with the user.

Step 1: Load our Javascript Editor

You’ve done this twice already! For detailed instructions, look back at a previous concept.

Step 2: Write a simple program

Erase “Enter some code” and type the following into the text field:



Again, make sure that you've checked the box, "Assume Javascript?"

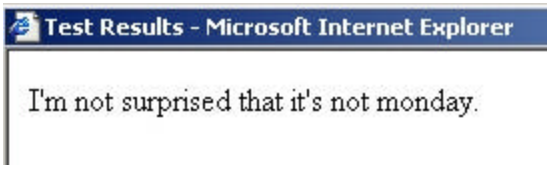
Step 3: Run your program

To run your program, click the "Test" button. If it doesn't seem to be working, note that Javascript considers "monday" and "Monday" to be different words!

Step 4: View the results

Your program should ask for your favourite day,

then reply to you on a new web page.



Background

- Javascript considers certain things "true," while it considers others "false."
- "1 is 1" is a *true* statement. In Javascript, we write this as "1 == 1". That's with two equals signs. We use a single equals sign to put information into variables, but two equals signs to test things.
- "1 is 2" is a *false* statement.

- 'day == "monday"' (from the example) is true only if the user really typed "monday."

What did we learn?

This is our first program whose output we couldn't predict!

1) We can put a true or false statement inside the brackets of "if(…)"
2) If the statement is true, Javascript executes the line of code immediately beneath the if().
3) This single line of code can be many lines of code if you surround them with the curly braces, { and }.
4) You can use an "else" statement to deal with everything that your "if" statement does not. Rule number 3 applies to this one, as well.

A common trap

What if we wanted to respond differently to different days of the week? A first attempt might look like this:

Javascript Test Bed

```
var day;
day = prompt("What's your favourite day?", "");

if(day == "monday")
   document.writeln("You must not have a job!");
if(day == "friday")
   document.writeln("Party time!");
else
   document.writeln("You didn't type monday or friday.");
```

Notice that we've written a catch-all "else statement" as a default response. However, when we run the program and type "monday", we get:

When we type "friday", though, we get the proper response:

Obviously, we've done something wrong. It seems that the "else" only works on the "if…friday" part! We aren't the first to run into this difficulty.

We solve this problem with an "if-else ladder." If-else ladders guarantee that Javascript only executes ONE of the chunks of code. They look like this: if(…) else if (…) [many else ifs] else (…).

Javascript Test Bed

```
var day;
day = prompt("What's your favourite day?", "");

if(day == "monday")
   document.writeln("You must not have a job!");
else if(day == "friday")
   document.writeln("Party time!");
else
   document.writeln("You didn't type monday or friday.");
```

Verbally, we would say: "If they typed 'monday', then respond appropriately. Otherwise, if they typed 'friday', then respond appropriately. Otherwise, respond vaguely."

Exercise 3

Write a program to translate English days of the week to their French equivalent:

1) Ask for a day of the week
2) Convert that to the French equivalent. From Sunday to Saturday, the French names for days are: Dimanche, Lundi, Mardi, Mercredi, Jeudi, Vendredi, and Samedi.
3) Print the French version to the screen.

When you're done, go to the end of this document for the answer.

Repeating Yourself

Our last program was useful, but cumbersome to use a lot. How do we fix that?

Step 1: Load our Javascript Editor

You should be good at this by now…

Step 2: Write a simple program

Erase "Enter some code" and type the following into the text field:

Javascript Test Bed

```
for(var counter = 0; counter < 10; counter = counter + 1)
{
   var math = prompt("What do you want to know? (or 'quit')", "");

   // Exit if they type "quit"
   if(math == "quit") break;

   document.writeln(math + " is " + eval(math));
   document.writeln("<BR>");
}
```

Like always, make sure that you've checked the box, "Assume Javascript?"

Step 3: Run your program

To run your program, click the "Test" button.

Step 4: View the results

Your program should ask for some math expressions and calculate their answers.

It asks you 10 times, or until you type "quit": whichever comes first. If this example does not make it clear, the '*' symbol is Javascript's multiply command.

Background

"For loops" are complex, so I'll talk about them in 3 parts:

```
for(var counter = 0; counter < 10; counter = counter + 1)
```

- part A
- part B
- part C

1) Part A: Javascript runs this part of the for-loop *before* anything else. To deal with part A specifically, imagine that we wrote "var counter = 0;" on a line of its own. What would that do? It would create a variable (called "counter") and set it to zero. An important point is that we can use this variable anywhere in the following block of code. For example:

Javascript Test Bed

```
for(var counter = 0; counter < 10; counter = counter + 1)
{
   document.writeln("Counter is: " + counter);
   document.writeln("<BR>");
}
```

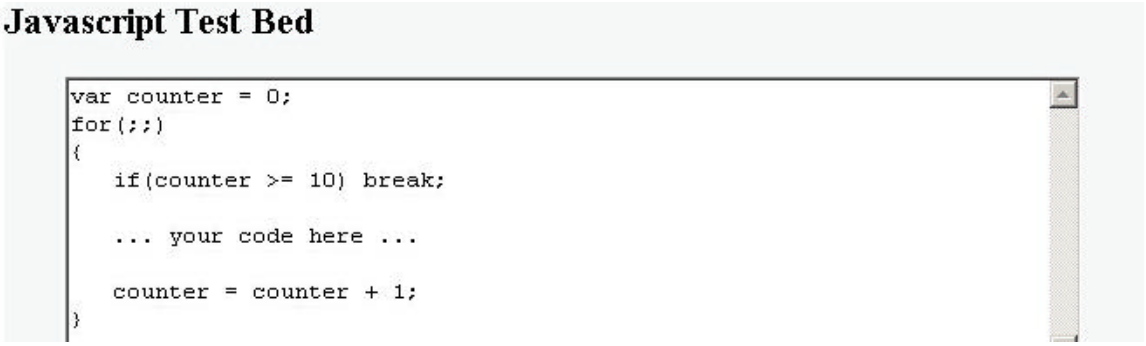
2) Part B: Javascript runs this part before each run of the loop. It's often called "the test." Do you remember when I said "Javascript considers certain things *true*, while it considers others *false*?" Well, if this test is *true* (ie: 0 < 10, 9 < 10, or counter < 10 when the variable 'counter' is really less than 10,) then Javascript executes the code between the curly braces. Otherwise, it stops running the for loop.
3) Part C: Javascript runs this part after each loop, but before "part B." In this case, we add 1 to our counter. Without this line, 'counter' would always be 0 – so counter would never go above 10 – so our loop would run forever!

The big picture

At a high level, we can see how Javascript understands our last for loop:

1) Create a variable called 'counter' and set it to 0.
2) Check if 'counter' is less than 10. Counter is 0, so the test is *true*.
3) Run the code between the curly braces
4) Add 1 to 'counter'. 'counter' is now 1.
5) Run steps 2 through 4 until 'counter' is 10. In that case, the statement '10 < 10' is *false* and we stop the for-loop. In the end, we run the for-loop 10 times: counter is 0, 1, 2, 3, 4, 5, 6, 7, 8, and 9.

An equivalent (but never used) form might look like this:



Here are some more example for loops:

for(var counter = 0; counter < 100; counter = counter + 1)
- counts from 0 to 99 in steps of 1.

for(var myCounter = 1; myCounter < 200; myCounter = myCounter + 1)
- counts from 1 to 199 in steps of 1.

for(var lineNumber = 1; lineNumber <= 10; lineNumber = lineNumber + 1)
- counts from 1 to 10 in steps of 1.

for(var x = 1; x <= 100; x = x + 2)
- counts from 1 to 100 in steps of 2.

What did we learn?

We've introduced a lot into this program so let's go over its concepts.

1) We can re-run our code (without having to re-type it) with a "for loop."
2) We can create a variable and assign it at the same time!
3) We can comment about our code if we start the line with "//".
4) We can break out of a for-loop with the "break" command.
5) We can use Javascript's "eval" command to evaluate expressions.
6) We can make Javascript write to a new line if we write "
" to the web page.

Exercise 4

Write a program to display the 7-times table (up to 12) on a web page. Let Javascript do the math, and (of course) do not code each case by hand! When you're done, go to the end of this document for the answer.

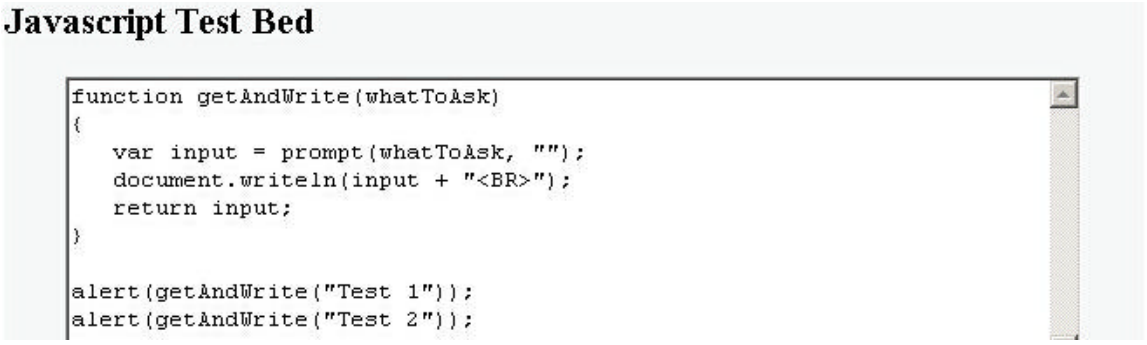
Delegating Responsibility

When you write programs, you'll often have to do the same thing several times – but not necessarily in a loop. Until now, you've had to simply re-type the code that does that thing. There is a better way.

Step 1: Load our Javascript Editor

Step 2: Write a simple program

Erase "Enter some code" and type the following into the text field:



Like always, make sure that you've checked the box, "Assume Javascript?"

Step 3: Run your program

To run your program, click the "Test" button.

Step 4: View the results

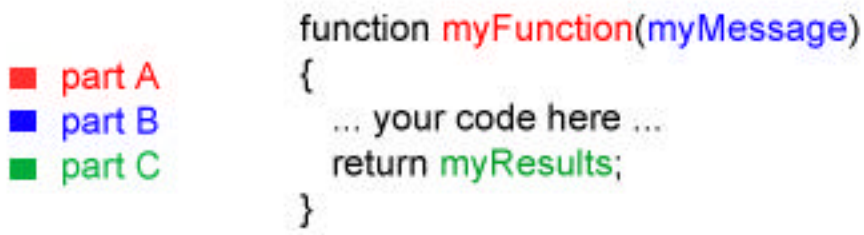
Your program should prompt you with "Test 1" or "Test 2",

then write your responses to the page – each on their own line. You should also see an alert box appear with your response in it.

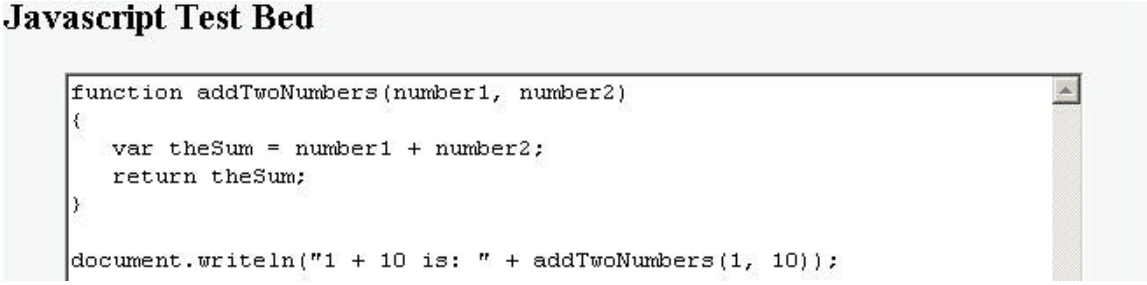
Background

Functions are complex, but easier than for loops. The power of functions is simple. You write them once (see below,) but you can use their "nickname" whenever you want – without having to re-write the whole function. We only wrote the "getAndWrite" function once (in the example above,) but we used it twice.

I'll talk about functions in 3 parts:



1) Part A: This is simply a nickname for your function so that you can use it in the rest of your program. You've used these nicknames several times already: "prompt()", "writeln()", and "eval()" to name a few. Luckily, the good folks who wrote Javascript wrote the guts of those functions for you.
2) Part B: This is a variable (or list of variables separated by commas) that let you give information to your function. For example, pretend that you want to write a function to add two numbers. You might want to use the function like "var mySum = addTwoNumbers(1, 10);". You would then write it:



Once we've passed this information in, we can use it like any other variable. We can even pass this information into other functions (like we've done with the prompt() in the getAndWrite() function above!) We call these input variables *parameters*.

3) Part C: When we write functions, Javascript lets us *return* information back to the person who used it. For example, our addTwoNumbers() function returns the sum of the two numbers. Luckily, you've used this concept already: the prompt() function returns whatever the user typed in the prompt box!

What did we learn?

We've introduced another very major concept into this program. Let's go over its concepts.

1) *Functions* let us write some code and call it many times – instead of writing the same code many times.
2) Functions have *names* to let us refer to them.
3) Functions can have *parameters* to let us give information to them.
4) Functions can have *return values* to give us the result of whatever they did.
5) Javascript has an *alert* function to pop up a browser "alert" box.

Exercise 5

Write a program to display the "12x12" times-table chart on a web page.

Let Javascript do the math, and (of course) do not code each case by hand! To reduce repetition, you might consider writing a function to help you out. This function, for example, might write any *single* times table to the screen. All you need, then, is to call the function 12 times. When you're done, go to the end of this document for the answer.

But I want to learn more!

This wraps up our practical introduction to computer programming. However, you still have boundless learning opportunities if you want to explore further.

If Javascript interests you, search the internet for "javascript html programming". If you didn't know this already, you can add Javascript to your web pages with very little

difficulty! If you want to learn how to create a web page, search the internet for "html tutorial"!

Now that you have a solid foundation in Javascript, you might want to check out the official Netscape Javascript Reference at http://developer.netscape.com/docs/manuals/communicator/jsref/index.htm.

If programming (but not necessarily Javascript) interests you, look into one of the other programming languages out there. "*Batch programming*" and the "*Windows scripting host*" help you automate common computer tasks. The *Jscript* language in the Windows Scripting Host is a version of Javascript, so you should have little difficulty learning it. "*Unix shell scripting*" helps streamline the Unix or Linux prompt that you might see at work or your Internet Service Provider. The "*Java*" and "*C++*" languages help you create larger, more complex computer programs. To look at these languages, search the internet for "windows scripting tutorial," "java tutorial," or any other language that strikes your fancy.

I wish you good luck and perseverance!

Exercise Answers

Exercise 1

Remember that anything we put inside the quotes of *document.writeln("…");* gets printed to the web page. So:

Javascript Test Bed

```
document.writeln("Your Name Here");
```

Exercise 2

Three lessons help us here:

1) We get information with a prompt: prompt("...","…");
2) We store information with a variable: var …;
3) We join information with a "+".

Javascript Test Bed

```
var meal;
meal = prompt("What's your favourite meal?", "");

var day;
day = prompt("What's your favourite day?", "");

document.writeln("I like to eat " + meal + " on " + day + " too!");
```

Exercise 3

Remember that we can test what a user types in with an if (or if-else) statement.

To answer this exercise, simply use one with 7 sections!

Javascript Test Bed

```
var answer = prompt("Day to translate?", "");
var translatedDay;

if(answer == "sunday")
   translatedDay = "dimanche";
else if(answer == "monday")
   translatedDay = "lundi";
else if(answer == "tuesday")
   translatedDay = "mardi";
else if(answer == "wednesday")
```

```
   translatedDay = "mecredi";
else if(answer == "thursday")
   translatedDay = "jeudi";
else if(answer == "friday")
   translatedDay = "vendredi";
else if(answer == "saturday")
   translatedDay = "samedi";
else
   translatedDay = "unknown to me!";

document.writeln(answer + " translated to French is " +
translatedDay);
```

Of course, you need not use a variable to hold the response. You could always type the "document.writeln(…)" for each option.

Exercise 4

Remember that we can use a for-loop to repeat something many times.

This code uses a technique called "constants" to help you read (and change) the program. Constants are variables that you define, but do not change. We write constants in all uppercase. If you want to change the times-table or how high it goes, this code lets you change the constant -- not everywhere that you use it.

Javascript Test Bed

```
var TIMESTABLE = 7;
var UPPERLIMIT = 12;

document.writeln("My " + TIMESTABLE + " times table:<BR>");

// Show the timestable up to UPPERLIMIT
for(var x = 0; x <= UPPERLIMIT; x++)
   document.writeln((TIMESTABLE * x) + " ");
```

Exercise 5

Remember that we can use a function to wrap common code into a simple statement.

I've introduced a new type of comment, a "block comment," into this program. In this case, Javascript ignores everything between '/*' and '*/'. It's a good habit to comment your functions this descriptively. In the future, you'll be able to quickly understand the function without having to read the underlying code!

Javascript Test Bed

```
var UPPERLIMIT = 12

for(var timestable = 1; timestable <= 12; timestable = timestable + 1)
{
   writeTimesTable(timestable, UPPERLIMIT);
}

/*
   function writeTimesTable
   input: whichTimesTable -- Which table to run through
   input: howHigh -- how much of the times table to show
   returns: nothing
*/
function writeTimesTable(whichTimesTable, howHigh)
{
   document.writeln("My " + whichTimesTable + " times table:<BR>");
   for(var counter = 0; counter <= howHigh; counter = counter + 1)
      document.writeln((whichTimesTable * counter) + " ");
   document.writeln("<BR>");
}
```