# Chapter 18: Interactive Web Pages with JavaScript

JavaScript supports communication and interaction with the reader of HTML pages. It allows you to make calculations and actions that are tailored to the individual responses that a person makes. In this chapter, you will learn how to use four new methods of communication, how to measure time intervals, and how to use loops to make otherwise difficult calculations. An illustration is developed that uses these techniques to calculate the learning curve predicted by a replacement model of learning.

## A. Alert, Prompt, and Confirm

The next examples illustrate three principles of JavaScript. First, JavaScript can be intermixed in an HTML page, not just in a separate section of code. Second, JavaScript can react to Events, such as loading or unloading a Web page, clicking on a button, moving the mouse over a link, or moving by tab or mouse click from one point of focus to another (e.g., moving from one text input box to another). Third, JavaScript provides new ways to communicate with the reader in addition to printing results in the Web page.

Consider the following example (*Ch18_ex1.htm*).

```
<HTML><HEAD><TITLE>Illustration of Forms and Event Handler</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Push Me" OnClick="alert('Hey! Cut it
out!')">
</FORM>
</BODY></HTML>
```

This example includes a form with a button that says, "Push Me." The *Event* that must be *handled* in this case is clicking of the button. When the reader clicks, `OnClick="statements"` causes the statements in quotes to be executed. In this example, pushing the button causes the following JavaScript statement to be executed:

```
alert("Hey! Cut it out!")
```

The `alert` statement produces a JavaScript alert box, in which is printed whatever is inside the parentheses. In this case, the parentheses contain a literal message. Note that this example uses quotes within quotes; in these cases, use the usual quotes for the outer pair, and use single quotes within. When using quotes, pay special attention to words like "don't" that contain an apostrophe, which may cause errors that will be hard for you to figure out. To include an apostrophe, use `\',` as illustrated in *Ch18_ex1b.htm.*

The `confirm` statement allows you to ask a question that has a yes (`true`) or no (`false`) answer, so that you can take action depending on the answer. The following example (*Ch18_ex2.htm*) illustrates its use:

```
<HTML><HEAD><TITLE>Illustration of Confirm</TITLE>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Click Here to Begin the Experiment"
OnClick="if (confirm('As you have read, you must be over 18 and are free
to quit at any time. Are you over 18 and wish to continue?'))alert('You
may continue'); else alert('Sorry.')">
</FORM>
</BODY></HTML>
```

This example shows that a number of JavaScript statements can be placed within the quotes. When there are multiple statements on the same line, they should be separated by semicolons. The `confirm` command returns `true` or `false`, depending on the response of the reader. In this case, the program merely presents an alert to inform the reader. However, one could also send the reader to different files depending on the reader's response, by writing different links to the page with the techniques of the previous chapter.

You can also use the `confirm` command in a regular link, as follows:

```
<A HREF="examples.htm" onClick="if(confirm('Are you sure you want to go
back to the list of examples?'))return true; else return false">Return
to list of examples</A>
```

The `prompt` command is like an input text box, and is illustrated with the following example (*Ch18_ex3.htm*):

```
<HTML><HEAD><TITLE>Illustration of Prompt</TITLE>
</HEAD><BODY>
<FORM>
<INPUT TYPE="button" VALUE="Click Here" OnClick="answer=prompt('what is
your name?','(type here)');alert('Hi! ' + answer)">
</FORM>
</BODY></HEAD>
```

This example also illustrates how the answer can be used as a variable and then used in the `alert`. Note that there are two fields in the statement `y = prompt(x, y)` command: `x` is the message printed in the box, and `y` is the value (or text) that the reader types in the box as an answer to the prompt.

A fourth way to communicate with the reader is to use the window status line. That line shows along the bottom of the browser's window. Consider the following:

```
<BODY OnLoad="window.status='Move your mouse over the links and look here'">
```

When the page loads, the status bar will show the message in `status=message`. Consider the following example:

```
<P><A HREF="examples.htm#eighteen"
onMouseOver="window.status='Click here to return to examples for Chapter
18'; return true"
```

```
onMouseOut="window.status='OK Try it again'">
Return to list of examples</A>
```

This example also illustrates the `onMouseOver` and `onMouseOut` events (the mouse pointer is over the link, or it leaves the link). Try the example on your browser and move the mouse over each link and away from each link. If you missed it, you need to reload *Ch18_ex4.htm* to see the status after `OnLoad`. This example behaves slightly differently on different browsers. You should realize that messages placed in the status line are less likely to be noticed by the reader of the page than an alert.

A fifth way to communicate is to open a new window (and create a new page "on the fly"), which will be described in the next chapter.

## B. Date and Time Information

JavaScript provides the *Date* object, which can be used to put the date and time in Web pages and experiments. It can also be used to control the timing of an experiment, or to measure response times in an experiment. The following example (*Ch18_ex5.htm*) illustrates the basic technique used to measure time intervals:

```
<HTML><HEAD><TITLE>Illustration of Date Object</TITLE>
</HEAD><BODY>
<FORM>
<INPUT TYPE="button" VALUE="Click Here to Begin" onClick="startTime=new
Date()">
<INPUT TYPE="button" VALUE="Click Here to Stop" onClick="finishTime=new
Date();seconds=(finishTime.getTime()-
startTime.getTime())/1000;alert('Your time interval = '+seconds+'
seconds')">
</FORM>
</BODY></HEAD>
```

The JavaScript statement, `startTime = new Date()`, creates a new instance of the Date Object. When the reader presses the first button, `startTime` contains a record of that time (and date). When the reader presses the second button, `finishTime = new Date()` creates another new instance of the Date Object, which stamps the time that the second button was pushed. The Date Object (like the Math Object mentioned in Chapter 17) has a number of methods (functions) associated with it that allow you to express the details of date and time. The `getTime()` method used in this example calculates and returns the number of milliseconds (.001 sec) since Jan 1, 1970. By taking the difference between two such times (in milliseconds), and dividing by 1000 (to convert to seconds), this program computes time intervals in seconds. Each interval is presented to the viewer with the `alert` command.

Some of the other Date Object methods are `getDate(), getDay(), getHours(), getMinutes(),getMonth(), getSeconds(), getYear(),` which get the date of the month, day of the week, hour of the day, minutes after the hour, month of the year, seconds of the minute, and year, respectively. One can also set the time and date by `set` methods (replace `get` by `set`) such as `setDate(n)`, where `n` = an integer from

1 to 31, which sets the date of the month; `setHours(h)`, where `h` = hours (0 to 23), which sets the hour of the day, etc.

In the next chapter, you will learn how to measure how much time was spent on a test or experiment and send the time measure back via the CGI script to your experiment's data file.

**C. Loops and Learning Models**

If you have done any computer programming before, you know that one of the most powerful techniques available is the use of loops to repeat computations. A loop causes a section of code to be executed repeatedly a specified number of times, or more generally, until a condition is satisfied.

Consider the following example (*Ch18_ex6.htm*), which calculates the amount owed after 15 years if a person borrowed $10,000 at 8% interest per year and made no payments until the last year:

```
var rate = .08       // set the interest rate (8%)
var debt =10000      // set the initial amount owed
for (i=1;i<=15;i++)
      {   debt = debt + debt*rate
          document.writeln("year = "+i+" Debt = "+debt+"\n")
       }
```

The variable, `i`, is the loop counter, or loop "index." The first expression in the parentheses after `for` represents the starting value of the index, which is the start of the first year. The second expression (`i<=15`) is the terminal condition; the loop will continue as long as the condition is true and will exit when this condition is false. The third expression (`i++`) indicates that the counter will be increased by 1 in each loop cycle (`i++` is the same as `i = i + 1`). Note that braces are used to designate all of the statements in the loop. In this case there are two statements in each cycle of the loop.

The first one calculates the new debt, which is the old debt plus the interest, where the interest is the debt times the interest rate. The second statement is to print out the year and the debt. The statement, `document.writeln(text)` will write to the document and add a line return. The results show that a $10,000 loan at 8% will cost a total of $31,721.69 after 15 years. Now, try the same loan at 13% (i.e., change .08 in the program to .13).

Computer programs can help you understand very large numbers such as the size of a nation's National Debt. When the amount of money that a government spends exceeds the money it receives through taxes and other sources of revenue, the government borrows money to make up the deficit. The total amount of money owed by the government is called the national debt. The government pays interest on the money it borrows. Suppose a country has a population of 250 million citizens and a national debt of $5 trillion dollars. If the government decided to assign its debt equally to all citizens, the debt per person would be about $20,000. A family of four would owe $80,000. Suppose instead that the government borrowed at 8%, ran no further deficits (aside from the interest), and then assigned the debt to the same sized population. How much would that debt be per citizen in 15 years? Suppose interest rates rose to 13% instead of 8%? You can modify the above program to make the calculations. The modifications for 8% are in *Ch18_ex7.htm.*

The computer technique of loops is extremely powerful. In general the `for` loop expression is as follows:

```
For (initial expression; final condition; update expression) {
statements}
```

The initial expression specifies the starting conditions of the loop. The loop will continue to iterate (executing all of the statements in braces) as long as the final condition is true. After each iteration, the expression will be updated by the update expression. When the condition becomes false, the loop will end, and the program will continue to the next statements after the loop. (Each of the statements should be on a separate line; multiple statements may be on the same line, if they are separated by semicolons.)

Loops provide a convenient way to calculate the predictions of mathematical learning models. A paper by Estes (1950; 1994) has been judged one of the most important contributions to psychology in the last 100 years (Bower, 1994), because it led to great advances in understanding how people learn. In each trial of a learning experiment, the learner gives a response to a stimulus situation, which is either correct or incorrect. A reinforcement is then presented, which enables the learner to improve his or her performance. As the learner has more and more trials, the performance (probability of a correct response) increases. The proportion correct can be plotted as a function of the number of reinforcing trials presented. This graph is known as the learning curve.

Do people learn by accumulating more and more information, or do they learn by replacing old ideas with new ones? Estes proposed stimulus sampling theory, a theory in which one can formalize these intuitive ideas to test predictions of specific models. For a brief introduction to simple versions these models, see the Web page "Marbles and Memory" included on your CD. When the reinforcements are always correct, and the learners capable of eventually learning the task perfectly, the replacement model implies the following learning curve:

$$P(n) = P(n-1) + \Theta[1 - P(n-1)] \tag{18.1}$$

where $P(n)$ is the probability of being correct on trial $n$, $P(n - 1)$ is the probability of being correct on the previous trial, and $\Theta$ is the learning rate parameter. This model implies that the improvement on any trial is a constant proportion of the difference between performance on that trial and asymptotic (in this case, perfect) performance.

Bower (1961) conducted a paired-associates learning task in which data were remarkably well fit by the replacement model of Equation 18.1. Subjects were presented with 10 stimuli such as BX, and asked if each is a "1" or a "2." Since there are two possible answers, and the letter combinations were randomly paired with responses, the learners are right only 50% of the time on the first trial. However, after each test trial, they are given the correct answer, so eventually they learn the associations, such as whether BX is a "1" or a "2." Bower found that on each trial, Stanford undergraduates learn about one third of the items they have not yet learned on that trial. After about 10 trials, they are nearly perfect on all ten pairs. Bower's data also allowed one to test the theory that people learn each item in an all-or-none fashion against the theory that people learn gradually. The data were consistent with predictions of the all-or-none model.

The example, *Ch18_ex8.htm*, shows how to calculate the learning curve for Bower's (1961) experiment. The program is a straightforward application of a loop, as described above. In each iteration, the probability correct on the next trial is calculated from the previous probability of being correct and the learning rate parameter. The probability of being correct on the first trial (before any reinforcements) is also called the guessing rate in this model.

## D. Functions in JavaScript

Suppose one were to repeat Bower's experiment with people who learn more slowly than Stanford students? This would require a change in the value of the learning rate parameter. Suppose one did an experiment where there were three possible answers (e.g., "1", "2", or "3")? In such a case, people would be right on the first trial one third of the time instead of half the time. To allow the user to input different values of the guessing rate and learning rate, one could use the prompt commands, as illustrated in *Ch18_ex9.htm*. Another way to do it is to use INPUT tags within forms, which as you learned in Chapter 5, is a good way to get information from the reader. That technique is used in *Ch18_ex10.htm*.

These examples illustrate two new aspects of JavaScript. Both use JavaScript functions to calculate the curve. Both examples also show that a Web page can be changed dynamically, based on the input provided by the user. This is especially clear in *Ch18_ex10.htm*, where the window's title and background color are also changed inside the function.

Functions are defined as in the following form:

```
function myFunction(parameters) {statements; return}
```

where there may be many statements in the brackets. Functions (and methods) have parentheses that may contain a list of parameters, separated by commas. The name of the function in the above example is `myFunction()`. Functions can be called by other functions or they can be called by an event handler. In *Ch18_ex9.htm*, a button is pressed that calls the function [`onClick="calcCurve()"`]. The function, `calcCurve()`, in turn contains prompts to ask the reader for the learning rate and guessing rate (try .33 and .50; then try .33 and .33). Note that the prompt command receives a text input. This text input must be changed to numerical information, which can be done with the `eval()`

function. The `eval()` function is a built-in, global function of JavaScript that is very useful. In the next section, you will learn that `eval()` can do much more than just convert text to numbers. The reader can enter an expression, such as `2+2*3/7,` and `eval` will correctly evaluate it.

In *Ch18_ex10.htm*, the transfer of parameters from a form to a function is illustrated. Note the following section of this example:

```
<FORM NAME="test">
Guessing Rate (G):    <INPUT TYPE=text NAME="guess" SIZE=5 value=".5">
Learning Rate (<FONT FACE="symbol">Q</FONT>):
<INPUT TYPE=text NAME="theta" SIZE=5 value=".335">
Compute the Curve:<INPUT TYPE=button VALUE="compute"
OnClick="calCurv(test.theta.value,test.guess.value)">
</FORM>
```

This section of the code shows that when the button is clicked, the function `calCurv()` is executed with parameters `test.theta.value` and `test.guess.value`. Note the construction of these variables. They are placed within a form, which is the active object. The form name is `test`, the variable names are `theta` and `guess`, and the "`value`" contains the text that was typed in the box by the viewer. Because these responses are supposed to be numbers (but JavaScript considers them text), when they arrive in the function they are processed by `eval`, to make sure that JavaScript recognizes them as numbers. Try entering 2/7 in one of the text boxes (literally). The `eval` function will correctly evaluate it.

The techniques of `alert`, `confirm`, and `prompt` have the advantage that they command the reader's attention, and the reader must respond to them before anything else can happen. In *Ch18_ex9.htm*, for example, the reader must respond to the prompt for theta before he or she can enter the guessing rate. On the other hand, it may be restrictive to the reader to be forced to decide one issue before seeing the next question. The use of Input text boxes, as in *Ch18_ex10.htm*, allows the reader to change the parameters in either order and to view them both while deciding how to change them.

**E. Probability Learning**

In the Bower learning experiment, the learner always receives the correct answer on each trial, so the undergraduates eventually perform perfectly. What happens if the feedback is probabilistic? Many situations in life are not perfectly predictable. For example, Store A might usually have lower average prices than B, but on a given day a certain item might be cheaper at store B than A.

One implication derived from stimulus sampling theory was the prediction of probability matching. Probability learning experiments found evidence of probability matching, as predicted by the model. If a person is reinforced 60% of the time for one choice and 40% of the time for the other, people tend to respond 60% of the time to the choice that was reinforced 60% of the time. To experience a probability learning test, load the experiment, *ProbLearn.htm,* from the list of examples. When the experiment starts, your task is to click on button R1 or R2, attempting to predict whether R1 or R2 will be correct on the next trial. Try to predict whether the next reinforcement will be on the left or the right, by clicking on one of the two buttons, R1, or R2. At first, you have no idea what is coming next, so you will have to guess. Eventually, you will learn to do better than chance. See how well you can predict the next event.

After you have completed 100 trials, you will receive an alert with feedback on your performance. You can compare your percentage of choosing R2 with the percentage of reinforcements of R2. A finding in the literature on this paradigm found that people "match," even though that behavior is not optimal. For example, if R2 is correct 60% of the time, then R1 is correct 40% of the time. If a person chooses R2 60% of the time (assuming that person does not have ESP), that person will be correct on R2 36% of the time (.6*.6) and correct on R1 16% of the time (.4*.4). Therefore, the person is correct a total of only 52% of the time. By *always* choosing R2, the person would have been right 60% of the time. For more information on this topic, including references to the literature, see Bower (1994). Data for the Web version of the experiment are included on the CD.

## F. Summary

The new ideas of Chapter 18 are summarized in Table 18.1.

Table 18.1.

| **Statement** | **description** |
|---|---|
| `alert(`*`message`*`)` | Presents message in a box. |
| `confirm(`*`question`*`)` | Box with yes or no question is displayed. Returns `true` or `false`, depending on answer. |
| `prompt(`*`x, y`*`)` | Presents message *x*. Reader enters *y* in the textbox of the prompt message. |
| `onClick="`*`javascript statements`*`"` | Event handler. Executes the statements when the event occurs. |
| `window.status` | Contents of the status bar at the bottom of the browser window. |
| `for (i = 1; i <= n; i++)` `{statements}` | Loop that repeats the block of statements, incrementing *i* by one each time until *i* > *n*. |
| `eval(`*`expression`*`)` | Evaluates an expression such as 2*(3-2). |
| *`timeNow`*` = new Date()` `y = `*`timeNow`*`.getTime()` | Creating an instance of the Date() object. Methods of the Date Object include `getTime(),` which returns time since 1-1-70 in milliseconds. |
| `function `*`funName`*`(`*`parameters`*`)` `{`*`statements`*`}` | JavaScript function. When the function is called, statements in brackets are executed. |

## G. Exercises

1. Create the script to replace “?” below so that when the link is clicked, it sends an alert that says “o.k., you're an ambulance.” `<A HREF = "filename.htm" onClick = "?">Call me an ambulance!</A>`

2. Create a Web page that includes at least one example of alert, prompt, and confirm.

3. Create a JavaScript that records the time from when a page is loaded until the reader clicks a button. Use `<BODY OnLoad = "startTime= New Date()">` to record the time when the page loads. When the button is pressed, calculate the number of minutes. (Hint: find seconds and convert to minutes.)

4. Use loops to calculate the first 20 Fibonacci numbers. Each Fibonacci number is the sum of the two preceding numbers. $f_1 = 1$, $f_2 = 1$, $f_3 = f_1 + f_2 = 2$, $f_4 = f_2 + f_3 = 3$, …, $f_n = + f_{n-2.} + f_{n-1}$.

5. Use loops to calculate the learning curve according to the accumulation model that implies that $P(n) = \frac{G + \Theta(n-1)}{1 + \Theta(n-1)}$. Make one version with prompts for G and Θ; make another variation with forms to receive the parameters. (Hint: study Examples 9 and 10).

6. Project idea: Analyze the data included on CD to see if probability matching characterizes the data. Can you think of a manipulation that will produce more "optimal" behavior from people? If you can think of such a manipulation, randomly assign people to conditions, with one condition receiving the usual instructions and the other receiving the new instructions that you think will produce more optimal behavior in the probability learning experiment. Collect data and see if the new manipulation is effective.