Chapter 19: JavaScript and Forms

In Chapter 18, you learned how to get information from a form to JavaScript. In this chapter, you will learn how to send information from JavaScript back to the form. This trick will be used in this chapter to make calculators, to check if the participant in an experiment completed all of the items, to score a test (and give individualized feedback), and to compute the time it took the participant to finish the task. You will learn a technique for passing these values from one form to another, and how to send these computed values to the data file, to be saved on the server. You will also learn how to create a new Web page "on the fly", which can be useful in many ways, but is illustrated here for the purpose of giving people feedback on a personality test. This chapter will also discuss how gullible people can be when they get information the supposedly describes them.

A. Calculators

Besides the learning curve calculators constructed in Chapter 18, you have seen the decision calculator in Chapter 8 and the Bayes Theorem calculator in Chapter 16. In this section, you will learn the principle behind those calculators. The programs, *surveyWiz* and *factorWiz*, are technically calculators that utilize this same principle. The first example is a very simple calculator that makes it easy to see the programming trick. This trick allows one to pass information in both directions between JavaScript and forms.

The example,*Ch19_ex1.htm*, illustrates the basic idea behind a calculator.

```
<HTML><HEAD><TITLE>Simple Multiplying Calculator</TITLE>
</HEAD><BODY>
<FORM NAME="MyForm">
<INPUT TYPE=text NAME=x1 SIZE=8 MAXLENGTH=9>
<FONT FACE="Arial">    X </FONT>
<INPUT TYPE=text NAME=x2 SIZE=8 MAXLENGTH=9>
   =
<INPUT TYPE=text NAME=ans SIZE=8 MAXLENGTH=9>
<INPUT TYPE="button" VALUE="Compute" OnClick="computeProd()">
</FORM>
 <SCRIPT LANGUAGE="JavaScript">
<!-- this comment hides the JavaScript from older browsers
var A = 0       // x1 = first number
var B = 0       // x2 = second number
var C = 0      // ans = product of the numbers

//         This function computes the product of x1 and x2
function computeProd() {
  with (document.forms){
     A = 1.0*(MyForm.x1.value)
     B = 1.0*(MyForm.x2.value)
     C = A*B
     MyForm.ans.value = C
 } return}
//  this hides the end of the HTML comment tag from Javascript -->
</SCRIPT>
<P><P><A HREF="examples.htm#ninetine">Return to list of examples</A>
</BODY></HTML>
```

This calculator will not calculate anything except the product of two numbers, but it does illustrate an important new idea. Note the expression, `with (document.forms) {statements}`. This expression makes clear (to different browsers) that we are referring to forms. The statement, `MyForm.ans.value = C`, is used to send the computed result back to the form. Forms are part of the document, `MyForm` is the name of the form; `x1`, `x2`, and `ans` are the names of the variables within the form; and we plan to use the *values* of those elements. Theoretically, one can also refer to the value of `x1` as follows: `document.MyForm.x1.value`; however, this way of identifying information does not

always work properly with all browsers. I advise you to use the `with (document.forms){statements}` technique as in this example to make your JavaScripts most likely to work correctly on different browsers. The technique usually saves typing as well.

Within the function, `computeProd()`, the variables `MyForm.x1.value` and `MyForm.x2.value` are multiplied by 1.0 to help different browsers interpret the results as numbers. The statement `MyForm.ans.value = C` causes the answer to be printed to the form when it returns from the function. Thus, forms provide a two-way street for getting information from the reader, and sending information back to the reader.

The calculator in *Ch19_ex1.htm* looks fairly primitive compared to the Bayesian calculator in Chapter 16. You can make the calculator look better by putting it inside a table with a large border, and adding a bit of color to the page. That has been done in *Ch19_ex2.htm*, which has only these cosmetic changes.

A calculator that only multiplies is not particularly useful. However, the calculator in *Ch19_ex3.htm* can be used to add, subtract, multiply, divide, and use a full list of scientific functions. To use the functions, the user needs to know about the properties and methods available in the JavaScript Math Object.

In Chapter 17, you learned that the statement, `x = Math.pow(2,3),` would compute 2 to the 3rd power ($2^3 = 8$), and set x = 8. Using `with`, as above, you can also write this expression as follows:

```
with (Math) {x = pow(2,3)}
```

This trick (`with (Math)`) is combined with the `eval()` function, which can evaluate a JavaScript expression, to produce the brain of this simple, but powerful calculator.

```
<HTML><HEAD><TITLE>Evaluation Calculator</TITLE></HEAD>
```

```
<BODY BGCOLOR='#ddccee'>
<H3>Calculator that Evaluates an Expression</H3>
<FORM NAME="MyForm">
<TABLE BORDER=8 CELLPADING=0 CELLSPACING=0 BGCOLOR="eeddff">
<TR><TD><INPUT TYPE=text NAME=x1 SIZE=50 MAXLENGTH=80></TD>
    <TD> = </TD>
    <TD><INPUT TYPE=text NAME=ans SIZE=12 MAXLENGTH=14></TD>
    </TR>
<TR><TD> Type an expression; e.g., 2*(3-1)</TD>
    <TD>  </TD>
    <TD><INPUT TYPE="button" VALUE="Compute" OnClick="computeEval()"></TD>
    </TR>
</TABLE> </FORM>
 <SCRIPT LANGUAGE="JavaScript">
<!-- this comment hides the JavaScript from older browsers
var A = 0       // A is the evaluation of x1, the expression
var C = 0      // ans = evaluated expression
function computeEval() {    //  This function evaluates an expression
  with (document.forms){
  with (Math) {
     A = eval((MyForm.x1.value))}
     MyForm.ans.value = A
 } return}
//  this JavaScript comma hides the end of the HTML comment tag  -->
</SCRIPT></BODY></HTML>
```

Try out the evaluation calculator with expressions such as `2*(3-2)+4/5` or `log(pow(3,4))`. The area of a 10 inch diameter pizza would be `PI*pow(10/2,2)`. In the Web page on disk, there is a list of Math methods and properties, to make it easier to use them with the calculator. These properties and methods are also presented in Tables 19.1 and 19.2. Remember that JavaScript is case sensitive: `pi` instead of `PI` or `POW` instead of `pow` will not work! Try it. Insert Tables 19.1 and 19.2 about here.

It is also interesting to enter random numbers in the calculator. Here is a case where the calculator gives different answers each time you push the button. Try the following: `floor(10*random()) + 1`
It should give random integers from 1 to 10, different each time you click *compute*. Remove the `floor` function to get random numbers from 1 to (almost) 11. Because this calculator will calculate any JavaScript expression, it can be useful when programming.

B. Checking for Missing Data

Sometimes a person might inadvertently skip an item. For some experiments, it may be important to check for omitted items before scoring a test or recording the data. Example *Ch19_ex4.htm* shows how to check for missing items and also how to pass data from JavaScript to a form that is compatible with protocols used in this book. The example has three parts. First, there is a form (called *test*) that consists of two test items (it could be more, but only two are used to keep this example short). Second, there is a form that is set up to send data to a file by the generic script. The second form, called *DataForm* in this example, will receive the data from the first form and send them (along with the experiment name, date, and time) to the script, *generic2* (*generic2* writes to

data2.csv). Third, there is a function, which is called by a button, that checks for blanks and sends the data from the *test* form to *DataForm*.

```
<FORM NAME="test">
<P>1. If x = 2, what is 2x - 2?
       <INPUT TYPE=TEXT NAME="Answer1" SIZE=8 MAXLENGTH=8><BR>
<P>2. What is the next number in this series: 1, 1, 2, 3, 5, 8, 13, ?
       <INPUT TYPE=TEXT NAME="Answer2" SIZE=8 MAXLENGTH=8>
<P><INPUT TYPE="button" VALUE="Check Test" OnClick="checkTest(2)">
</FORM>
<P><FORM NAME="DataForm" METHOD=Post
ACTION="http://psych.fullerton.edu/cgi-win/polyform.exe/generic2">
  <INPUT TYPE="hidden" NAME="00Exp" VALUE="Check_blanks_test">
  <INPUT TYPE="hidden" NAME="01date" VALUE=pfDate>
  <INPUT TYPE="hidden" NAME="02time" VALUE="pfTime">
  <INPUT TYPE="hidden" NAME="03Ans1">
  <INPUT TYPE="hidden" NAME="04Ans2">
  <P>When you finish the test and checking, push the button below:
  <INPUT TYPE="submit" VALUE="Finished">
</FORM>
 <SCRIPT LANGUAGE="JavaScript">
function checkTest(n) {       // this function checks for blanks
  n=1.0*n
  with (document.forms){
  var errors="#"
  var flag="You can now press the Finished button. "
  var message2="Please complete the following items. Thank you.\n "
        for(var i=1;i<=n; i++){
              j=i-1
              if (test.elements[j].value =="" ){errors=errors+i+", "}
              k=i+2 // there are 3 hidden items before the data.
              DataForm.elements[k].value=test.elements[j].value
         }
    if(errors !="#") {alert(message2+errors)}
    else {alert(flag)}
 }return}
//  hide the end of the HTML comment tag from Javascript -->
</SCRIPT>
```

The example illustrates another useful trick for referring to data obtained in forms. In the first form, *test*, the responses to the first two items are stored in `document.test.Answer1.value` and `document.test.Answer2.value`. Because form

elements are also stored in an array called `elements[],` we can also refer to these two answers as `document.test.elements[0].value` and `document.test.elements[1].value`. (Arrays in JavaScript use 0 for the first entry, 1 for the second, 2 for the third, etc. So, the first item is `elements[0]`. Incidentally, forms are also stored in an array, as are images, which are numbered in the same way, starting at zero. This means that forms that have not been named can still be addressed).

Similarly, the form, "DataForm," designed to be compatible with the CGI scripts used in this book for organizing the data file, contains 5 variables with names such as `00Exp`, `01Date`, `02Time`, `03Ans1`, and `04Ans2`. Each variable name is preceded by a two-digit number that is used by CGI script (*generic2*) to organize the data. However, JavaScript requires that each variable name must start with a letter. Therefore, it is helpful to be able to refer to these variables as `document.DataForm.elements[0].value` to `document.DataForm.elements[4].value`.

Load *Ch19_ex4.htm* from the list of examples. Leave everything blank and click the button marked "Check test." Clicking this button calls the function checkTest(2), which checks whether `(with (document.forms)) test.elements[j] ==""`. This checks for items left blank (i.e., ""). Recall that the item numbers are 1 and 2, but the element number is one less (0, and 1); for that reason, `j = i - 1`. If `element[j]` is left blank, then trial `i` is added to the list of errors (note also that a comma and a space are added to the list, to make the list easier to read). The next statement copies the answers to the other form (`k = i+2` allows for the three hidden variables in *DataForm*, which are elements 0, 1, and 2).

To check if you understand the program, try adding one or two additional items to the questionnaire. You will add your new items to the form *test* and their counterparts to form *DataForm*. You also have to change the number 2 to the new number of test items in the function call, `OnClick="checkTest(2)"`.

C. Scoring and Timing a Test with Feedback

To score a test, one can simply total up how many items have answers that agree with the answer key. To find out how much time a person spent on a page, one can record the time from the `onLoad` event until the person clicks a button to score the test. A method for accomplishing these two tasks is illustrated in *Ch19_ex5.htm*.

The example of *Ch19_ex5.htm* is very similar to *Ch19_ex4.htm*. Instead of checking for blanks, each response is checked against the answer key. If one were scoring an IQ test, personality test, or calculating a diagnostic index for a person, the method would be essentially the same. In this example, the variable `T` holds the total number of items correct. Note that for each correct answer, the statement `T=T+1` is executed, which increments the total by one. The technique for measuring a time delay is essentially the same as used in Chapter 18, except that the first time is set when the page loads (`<BODY OnLoad="timeLoaded=new Date()">`). The technique for giving the feedback in this case is through an alert. However, this could also have been done easily with an `INPUT TEXT` box.

D. The Barnum Effect

If you have not already done so, now would be a good time to take the personality test that was constructed with the help of *surveyWiz.htm* in Chapter 10. A link is provided to this test from the examples of Chapter 19.

When you took the test, did the description provided by the program seem an accurate description of your personality? Many people seem to think so. Is that evidence of validation of the test? In that experiment everyone got the same description, which seems opposite the idea of what we usually mean by personality. A personality test is supposed to describe what it is that is different and distinctive about a person, not what it is that most people think is a profound insight.

People often seem gullible when someone tells them about themselves. That is why psychics and spiritualists can make money. Do you know the difference between a psychoanalyst and a psychic? The psychoanalyst starts out by asking the client, "What is your problem?" The psychic has a harder task, as the psychic must figure out what the person's problem is, reveal it to them and predict what will happen next. Magicians describe the technique psychics use to look at a person and give them a false sense of knowledge about themselves as the art of "cold reading." Magicians publish books on how to fool people with phony psychic readings, but magicians are honest in admitting that they use tricks to create illusions for entertainment. Psychics use the same tricks to fool people and relieve them of their money at the same time. For that reason, magicians consider spiritualists and psychics to be dishonest, unethical magicians. Harry Houdini and The Amazing Randi are two magicians who exposed frauds who misused magician’s tricks.

This demonstration has become known as the *Barnum effect*, named after the circus showman P.T. Barnum, who said, "There's a sucker born every minute."

The Barnum personality test gave everybody the same feedback! If people say that this description is an accurate description of their individual personality, they are experiencing the illusion of validity produced by the Barnum effect. The median rating of accuracy was 7

on a 9-point scale. 106 out of 243 people tested said the description was either *very accurate* or *very very accurate*. But everyone read the same description.

Not only is the general public gullible, but so too are personnel managers, who use psychological tests to hire and promote employees (Stagner, 1958). Stagner gave a personality test to personnel managers, and then provided them with descriptions similar to those given when you push the *Score Test* button in the personality test. The personnel managers were impressed by the accuracy of the test, based on their feeling that the test was so accurate in its description of them.

The bottom line is that self-validation is not real validation. A valid personality test must do more than just give a person the feeling (illusion) of being understood. As noted in Chapter 10, a valid personality test can be used to predict individual differences in behavior.

E. Creating a New Web page

In addition to the survey created by surveyWiz in Chapter 10, the Barnum demonstration uses the following section of JavaScript to create a new page to display the feedback:

```
To score your test and see a description of your personality, press the
Score Test button below.<BR>
<INPUT type="button" value="Score Test" onClick="feedback()">
<SCRIPT Language="JavaScript">
function feedback() {
MyWindow=window.open("feedback.htm","W2","toolbar=no,menubar=no,scrollbars=
no,width=370,height=280")
MyWindow.focus()
}
</SCRIPT>
<P>We are interested in your feedback to help us make more accurate
diagnoses of personality. How accurate is the description of your
personality (provided when you push the Score Test button)?
```

The statement, `MyWindow=window.open(x, y, z)` creates and opens a new window. This new window could contain information that could be written to it, but in the above example, it contains the HTML in the file, *feedback.htm*. The variable x holds the name of a file, if any, to load to the new window; y holds the name that could be used for a hyperlink; and z holds a list of the new window's characteristics. `MyWindow.focus()` causes the window to be put in the foreground. (Otherwise, it might end up under the browser's window, especially after a click in the browser's window). Now load *Ch19_ex6.htm*, listed below:

```
<HTML><HEAD><TITLE>Example of new window</TITLE></HEAD> <BODY
BGCOLOR="lightblue">
Here's an example with a new window.
<FORM NAME="MyForm">
<P>What is your favorite color? <BR>
(You can try names, e.g., purple, or hexadecimal, e.g., 4499aa)<BR>
<INPUT TYPE="text" SIZE =25 NAME=color2>
<P>What is your name?<BR>
<INPUT TYPE="text" SIZE =25 NAME=name2>
<P><INPUT TYPE="button" VALUE=" Open new window"
onClick="NW(MyForm.color2.value,MyForm.name2.value)">
</FORM>
<SCRIPT>
function NW(color,name) {
  var newPage="<HTML><HEAD><TITLE>new window for "+name+"</TITLE></HEAD>"
  newPage+="<BODY BGCOLOR="+color+">Here is the new window you wanted,
"+name+". "
  newPage+="<FORM><INPUT TYPE=button Value='close' onClick='self.close()'>"
  newPage+="</FORM></BODY></HTML>"
  MyWindow=window.open("","W2","width=350,height=350")
  MyWindow.document.write(newPage)
}
</SCRIPT>
</BODY></HTML>
```

This example shows that you can use one Web page to create another Web page "on the fly." The new window can contain characteristics (e.g., the title of the page, its color, and text) that depend on the user's input. You can also use this example to check color values, try typing in *teal*, *cyan*, or *salmon*, for example. This example does not use `Window.focus()`. Experiment with adding or leaving out this statement to the example. What happens if you do not close the new window before clicking in the main window?

F. Summary

In this chapter you learned how to use HTML forms as two-way communication devices. In calculators, forms are used to accept values from the user and to display results computed in

JavaScript routines to the user. When forms are used to collect data, the data can be checked to make sure the participant answered all items. They can also be checked for correctness, a score can be computed, and feedback can be given by any of the methods described in Chapter 18, or by opening a new Web page to give the computed feedback. This chapter also showed how to collect data in one form and use JavaScript to pass the values to a hidden form that can return data to the server. The properties and methods of the Math Object are summarized in Tables 19.1 and 19.2.

Table 19.1 Mathematical Constants Available in JavaScript (Math Properties).

E	(Base of natural logs, about 2.718)
LN10	(Natural log of 10, about 2.302)
LN2	(Natural log of 2, about .693)
PI	(pi = ratio of circumference of circle to diameter, about 3.142)
SQRT1_2	(square root of 1/2, about .707)
SQRT2	(square root of 2, about 1.414)

Note: These can be used in the following ways:

```
Y=  Math.PI
with (Math) {Y = PI}
```

Table 19.2 Mathematical Functions Available in JavaScript (Math Methods)

abs(x)	(absolute value of x)
acos(x)	(arc cosine of x, in radians)
asin(x)	(arc sine of x, in radians)
atan(x)	(arc tangent of x, in radians)
ceil(x)	(next greater integer than x)
cos(x)	(cosine of x, x in radians)
exp(x)	(exponential function of x)
floor(x)	(next smaller integer)
log(x)	(natural logarithm of x)
max(x,y)	(maximum of x, y)
min(x,y)	(minimum of x, y)
pow(x,y)	(x raised to the y power)
random()	(pseudo-random number, uniform between 0 and 1)
round(x)	(rounds x to the nearest integer)
sin(x)	(sine of x, where x in radians)
sqrt(x)	(square root of x)
tan(x)	(tangent of x, where x is in radians)

```
y = Math.log(2)
```

```
with (Math) {y=log(2)}
```

G. Exercises

1. Convert the multiplying calculator so that it takes x and raises it to the y power.

2. Now create a calculator that adds, subtracts, multiplies, divides, and finds powers, depending on which button is pressed. (Create five buttons and five functions to make the computations).

3. In *Ch19_ex3.htm*, try entering `3 == 2`. Can you predict the answer?

4. Add four items to *Ch19_ex4.htm*, so that missing answers will be detected.

5. Add 2 radio button items to *Ch19-ex4.htm*. Check if they have been answered. Hint: Be sure to code "" for the extra (non-response) button.

6. Change *Ch19_ex5.htm* so that it computes the time until the first item is completed, instead of time until the score button is pressed. Hint: What event should you use to trigger the timing?

7. Look at the probability learning experiment from Chapter 18. You should now understand the main components of the JavaScript that runs that experiment.

8. Add `window.focus()` to *Ch19_ex6.htm*. Try clicking in the main window and then click the button.

9. Project idea: Write a JavaScript routine to compute simple reaction times. The page will contain a form that says “Ready”, “Set”, and “Go”. Participant will click a button as fast as possible, to record the time.