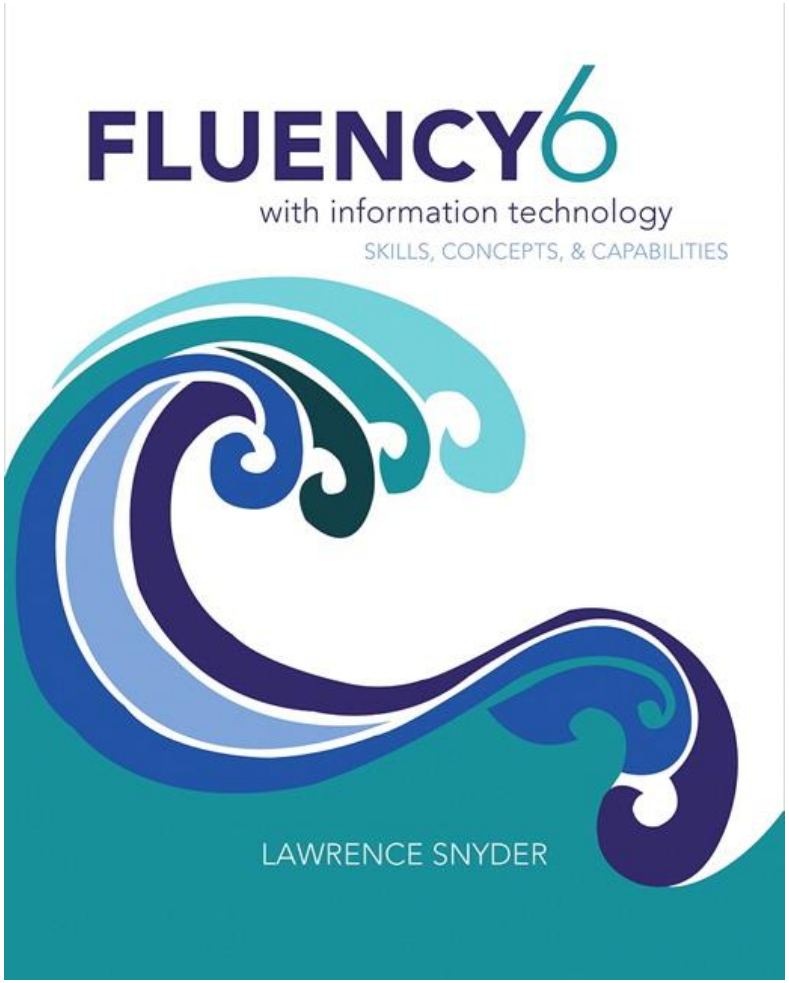


Chapter 18

A JavaScript Program

Learning Objectives

- Use the Bean Counter application as a model to do the following:
  - Write input elements
  - Create a button table
  - Write an event handler in JavaScript
  - Produce a UI similar to that of the Bean Counter
- Trace the execution of the Bean Counter, saying what output is produced by a given input
- Explain event-based programming in JavaScript and the use of event handlers

Preliminaries

- HTML files are made of ASCII text
- Avoid word processor formatting since it confuses browsers
- We'll use the basic text editor of Chapter 4 for our JavaScript development
  - File format must be text or txt
  - File extension must be html
  - Operating system knows an html file will be processed by a browser

Creating your JavaScript

- Open your starterPage.html in a text editor
- Save it with the file named, bean.html
- To include JavaScript in an HTML file, enclose the JavaScript text in:

  <script>

  …

  </script> tags

Creating your JavaScript

- Place the espresso pricing program from Chapter 17 in the <script>...</script> section
- Add the statement alert(price); to the bottom of your Javascript
- Time to test your program?
  - Save it
  - Find the file on your computer and open it
  - JavaScript will run with HTML

Bean Counter v.0

```
<!doctype html>
<html>
 <head>
  <meta charset="UTF-8">
  <title>Bean Counter</title>
 </head>
 <body>
  <h2> Confirming that bean.html works </h2>
  <script>
   var drink = "latte";
   var ounce = 12;
   var shots = 2;
   var taxRate = 0.088;
   var price;
   if (drink == "espresso")
     price = 1.40;
   if (drink == "latte" || drink == "cappuccino") {
     if (ounce == 8)
       price = 1.95;
     if (ounce == 12)
       price = 2.35;
     if (ounce == 16)
       price = 2.75;
   }
   if (drink == "Americano")
     price = 1.20 + 0.30*(ounce/8);
   price = price + (shots - 1)*.50;
   price = price + price*taxRate;
   alert(price);
  </script>
 </body>
</html>
```

(a)

(b)

Figure 18.1 Version 0 of the Bean Counter program without the user interface, and with fixed inputs: (a) the HTML and JavaScript code, and (b) running in the Firefox browser.

What We're Aiming for …

Figure 18.2 The Web interface for the Bean Counter program as it will appear when it is complete.

Review of HTML Basics

- Want to review before making changes? Check out Chapter 4!
- Change the title to

  <title>The Bean Counter</title>
- Add a global <style> section in the head
- Replace “Hello, World” with an <h1> heading that reads “the bean counter”
- Add an <hr />

Review of HTML Basics

- Next, add the paragraph below:

  figuring the price of espresso drinks
  so baristas can have time to chat.
- Use a
 to split the paragraph above over two lines
- Use saddlebrown as the background color, darkorange as the font color, and helvetica as the font family

Review of HTML Basics

- Make the heading appear in white
- Style the horizontal line to be shorter than the full window

What we have so far

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>The Bean Counter</title>
    <style type="text/css">
          body {background-color : saddlebrown; color : darkorange;
                font-family : helvetica; text-align : center}
          hr    {width:50%; color: darkorange}
          h1    {color : white;}
    </style>
  </head>
  <body>
    <h1> the bean counter</h1>
    <hr/>
    <p><b>figuring the price of espresso drinks<br />
          so baristas can have time to cha:</b></p>
  </body>
</html>
```

Figure 18.3 The Bean Counter interface to this point, and the HTML that produced it.

Interacting with a GUI

- Input facilities like buttons and checkboxes are known as elements of HTML forms
- They assist with activities like ordering products or answering survey questions
- When a form is complete, it is sent to the computer for processing
- Input elements, while not part of a survey, are still needed, so *form* tags must be used

Forms

- The form tags surround all input elements
- The action attribute of form is required for input
  - The only form attribute we're worried about is name.

```
<form name="unique_name">
… input elements …
</form>
```

Events and Event Handlers

- When the GUI inputs are used they cause an *event* to occur
- Buttons have a “click event” (as in you *click* the mouse to select the button)
- An event is an indication from the computer (operating system) that *something* just happened (mouse click)

Events and Event Handlers

- The JavaScript program needs to respond to that click:
  - It needs to do the operation that corresponds to the button command
- When JavaScript “finds out” about the event, it runs a piece of program called the event handler
- An *event handler* is the program that responds to the click

Three Input Elements

1. **Text Box**
   - <input type="text" id="*ref_name*" *value="displayed_text" size="n“* onchange="event_handler " />

   - *identifier* is the unique name of the element
   - *value* is the text to be placed in the box
   - *onchange* gives the event handler’s JavaScript instructions

Three Input Elements

2. Radio Button

- <input type="radio" name="*identifier*" onclick="*event_handler*" />*label text*
- *identifier* is the name of the element
- *label text* is shown beside the button
- *onclick* gives the event handler:
  - When the radio button is clicked, the center darkens to indicate that it is set
  - The event handler's JavaScript instructions are performed

Three Input Elements

3. Button

- <input type="button" value="*label*" *onclick="event_handler"* />
- *value* gives the text to be printed on the button
- *onclick* gives the event handler's JavaScript instructions
  - When the button is clicked, JavaScript's event handler is executed

Creating the Graphical User Interface

- All that remains is to:
  - create a table
  - fill in the entries

- Make sure to place the table between the form tags to ensure that the browser understands the inputs

Creating the GUI

- The table is a four-row, four-column table with two empty cells
- Buttons appear in all of the occupied cells but one
- Table is mostly a table of buttons
- We will build it in steps

Creating the GUI

- The procedure for building the table:
  1. **Create a button table**
  2. **Delete two buttons**
  3. **Insert text box**
  4. **Label the buttons**
  5. **Primp the interface**

1. Create a Button Table

<td>

<button form=”esp” onclick=’’>b</button>

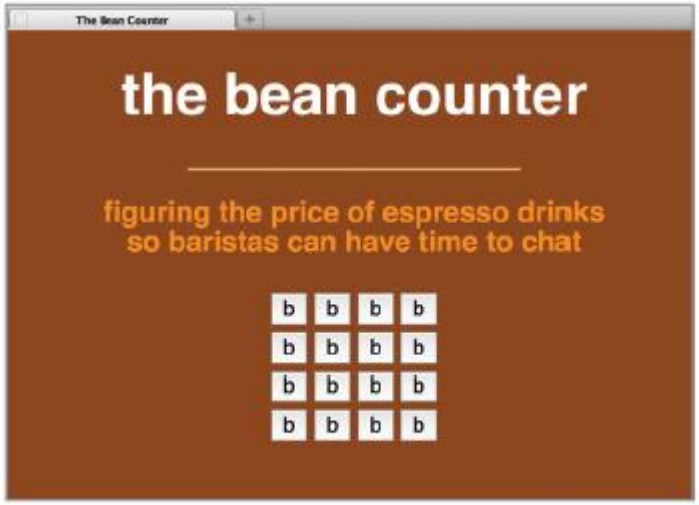
</td>

- "b" is a placeholder for the button label
- ' ' is a placeholder for the JavaScript text of the event handler

1. Create a Button Table

table {margin-left : auto; margin-right : auto; text-align : center }

- using CSS:
  - The table is centered by specifying that the left and right margins should be automatically positioned
  - text-align:center must also be included if we want IE users to see the table centered



(a)

Figure 18.4 Intermediate stages in the construction of the Bean Counter interface: (a) after Step 1, (b) after Step 3, (c) after Step 4, and (d) final form.

2. Delete Two Buttons

- In row 2, cell 4, and row 4, cell 2, remove the <button. . . >. . .</button> because these cells must be empty
- Cells can be empty, but they still need to be surrounded with <td> and </td> tags

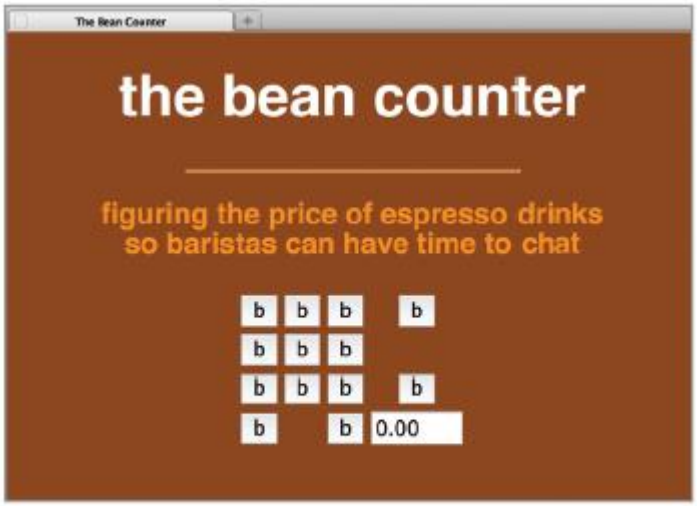
3. Insert Text Box

- Replace text box in the lower right corner with

```
<form input=”esp”>
  <input type=”text” id=”disp” value=”0.00”
      size=”5” onchange=’ ’ />
</form>
```

3. Insert Text Box

- The input box will display the resulting price, hence the name disp
- The box is “5” characters wide since no drink inputs will result in a price of more than four digits plus the decimal point
- The box is inside a form because HTML requires it.



(b)

Figure 18.4 Intermediate stages in the construction of the Bean Counter interface: (a) after Step 1, (b) after Step 3, (c) after Step 4, and (d) final form.

4. Label the Buttons

- Next, go through each of the table cells and change the value attribute of each button from “b” to its proper button label
- First column is the number of shots
- Second column is the drink sizes
- Third column is the type of drinks (espresso, latte, cappuccino, Americano)
- The two items in the last column are the controls: (Clear, Total)

(c)

Figure 18.4 Intermediate stages in the construction of the Bean Counter interface: (a) after Step 1, (b) after Step 3, (c) after Step 4, and (d) final form.

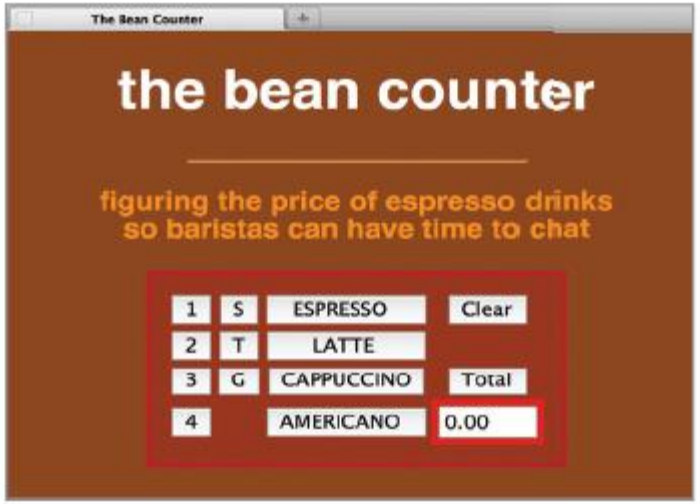
5. Primp the Interface

- The buttons would look better if they were all the same width
- Because the button is as wide as its value text, simply add spaces before and after the drink name to make the button wider and to balance the position of the text
- Use   (non-breaking space) since HTML discards plain spaces

5. Primp the Interface

- Give the table a background color
  - Remember that colors can be numeric
  - Make #993300 the background color
- Add a border to the table styling:
  - Make it a medium solid line
  - Color should be firebrick
- Add 8 pixels of padding to the buttons
- Add a medium red border to the price text box

What We're Aiming for …



(d)

Figure 18.4 Intermediate stages in the construction of the Bean Counter interface: (a) after Step 1, (b) after Step 3, (c) after Step 4, and (d) final form.

Event-Based Programming

- The Bean Counter program should behave like a calculator
  - Each time a button is clicked, something should happen
  - This is a user-caused event
- Programming the Bean Counter application amounts to defining in JavaScript the actions that should be performed when each button is clicked

The onclick Event Handler

```
<td>
<button form=”esp” onclick=’ ’>Total</button>
</td>
```

- An onclick event handler is used because we want the computer to do the action we programmed ***when the Total button is clicked***

The onclick Event Handler

```
<td>
<button form=”esp” onclick=’ ’>Total</button>
</td>
```


- To use JavaScript to calculate the price, we insert the price computation code *inside* the quotes for the *onclick* attribute
  - –But lose the alert; we’ll replace it later
- We now have an *onclick event handler*!

```
<td><button form="esp"
    onclick='
       var price = -10;
       var taxRate = 0.087;
       if (drink == "espresso")
          price = 1.40;
       if (drink == "latte" || drink == "cappuccino") {
       if (ounce == 8)
          price = 1.95;
       if (ounce == 12)
          price = 2.35;
       if (ounce == 16)
          price = 2.75;
       }
       if (drink == "Americano")
          price = 1.20 + .30 * (ounce/8);
       price = price + (shots - 1) * .50;
       price = price + price * taxRate;
       /* One more assignment statement needed here */
    '>  Total  </button></td>
```

Figure 18.5 The **Total** button tag with the price computation inserted as the event handler. (Notice that the three temporary declarations of Figure 18.1 have been removed, as has the temporary **alert( )** command at the end.)

What Happens in a Click Event?

- When the Total button is clicked:
  - The browser looks for the *onclick* event handler in the Total button input tag
  - The browser finds the JavaScript instructions to perform the button action
  - The browser runs those instructions, which implements the action, and then waits for the next event

Shots Button

```
<td>
<button form=”esp” onclick=’shots = 1’> 1
</button>
</td>
```

- For the shots buttons
  - The number of shots the customer requests is identified by which shot button is selected
  - The 2 button assigns shots the value 2, etc.

Size and Drink Buttons

```
<td>
<button form=”esp” onclick=’ounce = 8’> S
</button>
</td>
<td><button form=”esp”
onclick=’drink=”espresso”’>   
ESPRESSO    </button></td>
```

- The drink is assigned as "espresso", **not "ESPRESSO"**, as written on the button to match the comparison in the code

Clear Button and Initializations

- Clicking the Clear button resets all of the variables (drink, ounce, and shots) to their initial values
- But, there are no initial values yet!
- Sometimes in programming, you need to go back and add information, it's not always perfect the first time through!

Clear Button and Initializations

```
<script>
        var shots = 1;
        var drink = "none";
        var ounce = 0;
</script>
```

- Declarations should be placed at the beginning of the program just after the <body> tag
- Declarations must be enclosed in <script> tags

Clear Button and Initializations

- Initial value for shots is 1 (every espresso drink has at least one shot)
- Initial values for drink and ounce are chosen to be illegal values, so that an error message, indicating that an input is missing can be generated
- The Clear button should make these same assignments setting everything back to their initial values

Clear Button and Initializations

```
<td><button
form=”esp” onclick=’
        shots = 1;
        drink = "none";
    ounce = 0;
        disp.value =
“0.00”

’>Clear</button></td>
```

- The disp.value assignment places 0.00 in the price window

Referencing Data Across Controls

- The output window was defined with
  <input type=”text” id=”disp” … />
- The id is a global name
- The .value names an attribute of the window, its displayed value
- So assigning disp.value changes what is displayed

Changing the Window

- When the assignment changes the value back to 0.00, the browser displays the assigned value and the 0.00 is treated as an *output*
- *input* as *output?*
  - The window is seen from both the user's and the computer's point of view. If one side gets information (input) from it, the other must have put (output) the information

Displaying the Total

- The Total event handler must show the price – the output
- This is handled in the same way that the Clear button event handler clears the price window—by assigning to the value attribute of price
- The final line of the Total event handler is replaced by

```
disp.value = price;
```

Critiquing the Bean Counter

- Every design must be critiqued to ensure that it meets the requirements:
  - Did it solve the problem ?
  - Can it be improved?
- Experiment with the Bean Counter application to see how well it works.
- Does the design fulfill the barista's needs?

Numbers Versus Money

- The final price is shown as a decimal number with several places, not as currency with only two decimal places
- Change last line of total button handler to:

```
disp.value = Math.round(price*100)/100;
```

Numbers Versus Money

- The result is then rounded by using the built-in JavaScript function Math.round( ) to eliminate any digits to the right of the decimal point that is less than a penny
- Finally, that result is divided by 100 again to convert back to a “dollars amount”
- This avoids too many places (no 10.198372) but may have too few (10.2)

Numbers Versus Money

- This computation is a standard way to remove unwanted digits…it's not perfect, but it gets us to the correct amount
- We can make sure the number of digits is right like this:

```
disp.value =
    (Math.round(price*100)/100).toFixed(2);
```

Organization

- The organization of the buttons is generally consistent with how the application will be used
- Espresso drinks are typically named with syntax of “how many shots?”, “what size?”, and “what kind of drink?”
  “double tall latte”

Feedback

- The form does not give a barista any feedback about the current settings of the variables
- There should always be feedback for every operation.
- Adding a window above each column of buttons that gives the current setting might be helpful


the bean counter
figuring the price of espresso drinks
so baristas can have time to chat
2
T
Latte
1
S
ESPRESSO
Clear
2
T
LATTE
3
G
CAPPUCCINO
Total
4
AMERICANO
3.10

Who's Up?

- A further improvement would be to make a record of who is serving
- Add a row at the top of the table
- Add a drop-down containing the server names

Final Bean Counter

Bean Counter Recap

- We created a user interface using HTML
- We added event handlers to make the interface operate
- The events are handled by code written in Javascript
- This is event-driven programming.
  - Programs come in small pieces
  - Other program styles are more monolithic

Program and Test

- The programming process was (and is) incremental
  - Begin by producing a minimal 19-line HTML program and test it
  - Add to it and test it
  - Improve one feature at a time and test as you go

Program and Test

- Write and solve one event handler at a time
  - Are there similarities among the various events?
  - Can those similarities help in developing the other event handlers?
- Finally, critique the result

Program and Test

- This strategy of breaking the task into tiny pieces and testing the program after each small milestone had two advantages:
  - At no point did we have to solve a complex task!
  - Continual testing meant that we immediately knew where any errors were located…the newly added code!

Assess the Program Design

- When the initial design is completed, critiqued the result
- This is not a critique of the programming, but a critique of how well our solution solved the problem: did it fulfill the barista's needs?
- This is an important part of any design effort, but especially so for software

Summary

- Used HTML to set up a context in which event handlers perform the actual work
  - The setup involved placing buttons and other input elements on a Web page, so a user could enter data and receive results
  - This is the input /output part of the application and it is principally written in HTML

Summary

- Wrote JavaScript code for the event handlers
  - This is the processing part of the application. We used the event-based programming style and the basic instructions of Chapter 17
  - This style will be used throughout the rest of the book
  - HTML will simply be the input / output part of a program written in JavaScript