Microsoft

Begin to Code with JavaScript

Rob Miles

Begin to Code with JavaScript

Rob Miles

HTML comments

Note that these comments only work in the `<script>` part of the program. You can add comments to the HTML, as we saw in Chapter 2, but you use a different character sequence to mark the start and end of the comments:

```
<!-- Rob's Pizza Calculator Page Version 1.0 -->
```

The start of the comment is marked by the sequence `<!--` and the end of the comment by the sequence `-->`. As with JavaScript comments, the text between the two sequences is ignored by the browser.

Global and local variables

At the start of this section, we wanted to make a totalizer program that can be used to add up a bunch of numbers. We can now create this. Let's assume that we are creating a solution for a customer who really does want to totalize some numbers. You have sat down with her and agreed on the following design for the application.

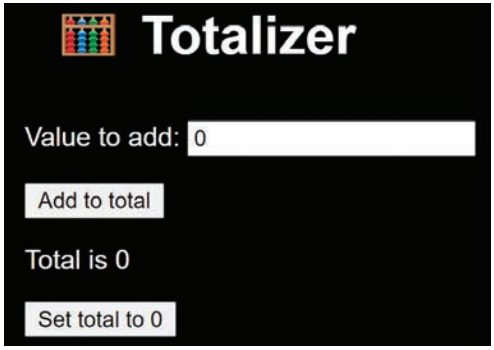

Figure 4-10 Totalizer

Your customer would like a stylish black background containing the Abacus emoji. We can create this by using a style sheet that sets the color scheme for the application and finding the symbol number for the abacus (`&#x1f9ee;`).

She wants to be able to type in a value and press the **Add to total** button to add the value to the total. She also wants a button she can use to set the total back to zero when she has finished adding one set of values. You agree on the design shown in **Figure 4-10** above.

PROGRAMMER'S POINT

Getting a good specification is vital

The sample page shown in **Figure 4-10** is a good start for the specification of the Totalizer application. It is very important that you get a solid specification for any work that you perform, even (or perhaps especially) if you are working for someone you know. The nice thing about the screenshot of the application is that it sets out exactly what the solution should look like. However, there are some questions I'd want answered, too.

I would like to know if there is any upper limit to the amount to be added to the total. I'd also like to know if the Totalizer should accept negative numbers to be subtracted from the total or whether the total should always be increased. The answers to these questions tell me whether the Totalizer should detect and reject invalid input values. The customer might be assuming that negative values should not be added (or might never have thought about this issue). Either way, as the builder of the solution, you need to know how it should work. Otherwise, you might end up having conversations with your customer that include phrases such as, "It isn't supposed to do that...".

Global variables

The Totalizer program is interesting because it is the first program we have written that needs to "remember" something between function calls. Until now, every program that we have created takes data from the input elements, does something to it, and then displays the result. Any variables that we have created to store data in a function during data processing are discarded as soon as the process has finished. For example, consider the temperature conversion program, which takes a temperature entered in Fahrenheit and converts it to centigrade.

```
function doTempConvert() {

  var fahrenheitElement = document.getElementById("fahrenheitText");
  var fahrenheitText = fahrenheitElement.value;
  var fahrenheit = Number(fahrenheitText);

  var centigrade = (fahrenheit-32)/1.8;

  var resultElement = document.getElementById("resultParagraph");
  var resultString = centigrade.toFixed(1) + " degrees Centigrade";
  resultElement.innerText = resultString;
}
```

Get a reference to the input element

Get the text from the input element

Convert the text into a numeric value

Convert the value into Centigrade

Get a reference to the output element

Build the result string

Put the result string on the output element

All the variables in this function will be destroyed once the function has finished. They are described as *local* because they are local to the function. This is how the JavaScript manages variables created using `var`. Most of the time, this is exactly what you want. We don't want the program to use any values left over from a previous use of the function. However, the Totalizer program needs to retain the total value for use in successive calls of the function that adds values to it. In other words; we want to write some code like this:

```
var total=0; // Global variable to hold the total

/* This function reads the value from the valueText element
   and adds it to the global total value */
function doAddToTotal() {

  var valueElement = document.getElementById("valueText");
  var valueText = valueElement.value;
  var value = Number(valueText);

  total = total + value; // update the global total value

  // Display the updated total
  var resultElement = document.getElementById("resultParagraph");
  var resultString = "Total is " + total;
  resultElement.innerText = resultString;
}
```

The variable `total` is special. It exists outside any JavaScript functions. We call it a *global* variable because it can be used by any function in my application. I've added a comment above the declaration of the `total` variable. This is because I like my global variables to stand out in the code.

PROGRAMMER'S POINT

Global variables are a necessary evil

If you talk to some programmers, they might tell you that your programs should never use global variables. This is because a global variable represents a possible failure point that is out of your control. I can be sure that all the variables in my functions contain correct values. This is because each time a function runs, it makes clean new copies of every variable, but a global variable exists in outside my functions. I can't regard it as "clean" because I don't know what other functions might have been doing with it. Mistakes by other programmers could make my functions do the wrong thing, and that is bad. If another

function changes, the contents of total my function could display a result that is incorrect. However, in the case of the Totalizer program, a global variable is the simplest way I can make it work.

Programmers talk about functions having *side effects*. These are things that the function does that change the state of the system in which they are running. In the case of the Totalizer, the `doAddToTotal` function has a side effect that increases the value of `total` by the amount entered by the user. This is a side effect that is present by design. It is important to avoid unintended side effects.

CODE ANALYSIS

Global variables and side effects

It is important that you understand the difference between local and global variables, so here are some questions you might have considered.

Question: How can I tell if a variable is global?

Answer: A global variable is declared outside of any function. The `total` variable below is not part of any function, therefore it is global. The variable is also set to `0` when it is created.

```
var total=0;
```

In Chapter 3, in the "Create a ticking clock" section, we discovered a JavaScript function called `onload`, which runs when a web page is loaded. A program could initialize (but of course not declare) global variables in that function.

Question: Is the value of a global variable retained if I reload the web page?

Answer: No. When the page is reloaded, the JavaScript environment is reset and new global variables are created.

Question: Is a single global variable shared between multiple tabs of the same page being viewed in a single browser? In other words, if I opened several views of the Totalizer program, would the value of `total` be shared between them?

Answer: No. Each web page runs a separate JavaScript environment.

Question: Do any other functions in the Totalizer have side effects?

Answer: Yes. The function that is called to clear the total back to 0 will set the value of `total` back to 0.

```
/* This function clears the total value and updates the display
*/
function doZeroTotal(){
  total=0; // set the global total to 0

  // update the display
  var resultElement = document.getElementById("resultParagraph");
  resultElement.innerText = "Total is 0";
}
```

This function sets the value of `total` to 0 and then updates the display to reflect this.

Question: How could I create a Totalizer that stored the total value when the page was not being used?

Answer: The browser provides a feature called "local storage," which can be used to store values when a web page is not active. In Chapter 9, we will use this to create an address book. A totalizer program could use local storage to hold a total value that would persist when the totalizer page was not being used.

You can find my version of the Totalizer program in the **Ch04 Working with data\Ch04-08 Totalizer** folder. This includes the style sheet that sets up the requested color scheme.

MAKE SOMETHING HAPPEN

Make some party games

There is no better way to show off your programming skills than by using them to make some silly party games. At least, that's what I think. We can create a good-looking party game using our skills with CSS and JavaScript. The basis of many games is randomness. We know how to use JavaScript to create random numbers, so let's see if we can make some games using this.

"Nerves of Steel"

We can use our ability to make random numbers, coupled with the `setTimeout` function we used to make egg timers in Chapter 2, to create a "Nerves of Steel" party game. The game works like this:

1. One player presses the **Start Game** button.
2. The program displays **Players stand**.
3. The program then pauses for a random time between 5 and 20 seconds. While the program is paused, players can sit down. The players need to keep track of the last person to sit down.
4. When the time interval expires, the program displays "`Last to sit down wins`." Players still sanding are eliminated, and the winner is the last person to sit down.

This is a variant of the egg timer program from Chapter 2. Rather than set a timeout for a fixed duration (5 minutes), the program selects a random time for the duration. You can make the game properly skillful if the game displays the selected time at the start of the game. You can also improve it using sound effects to mark the start and end of the timeout session.