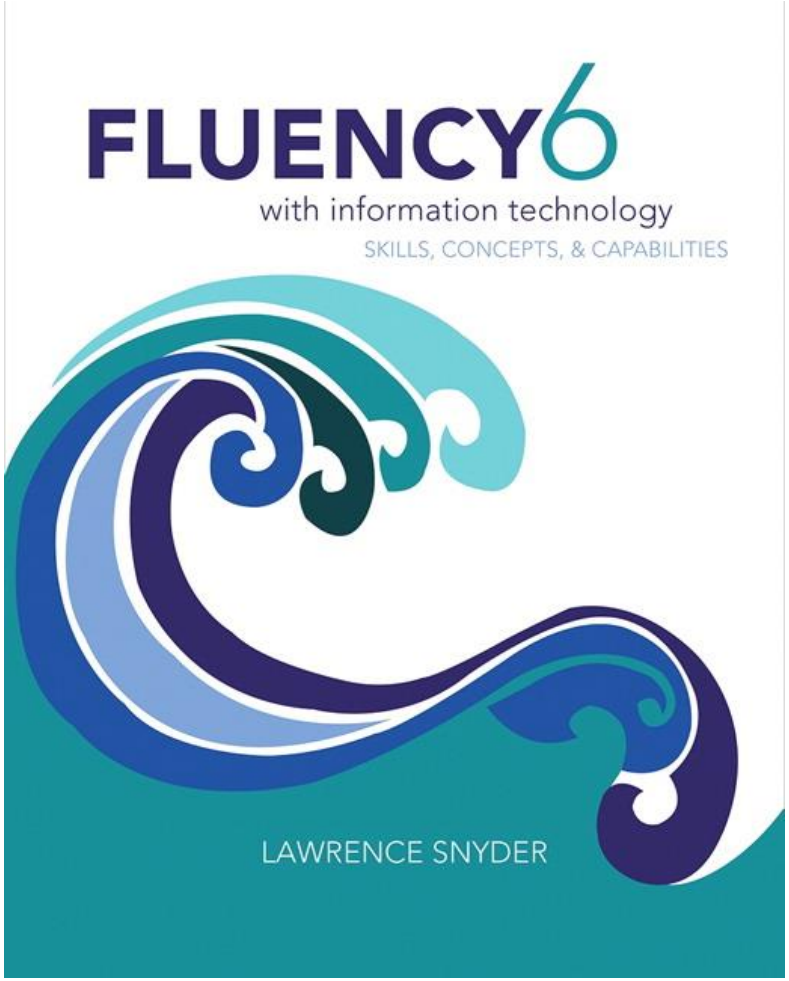


Chapter 19

Programming Functions

Table of Contents

Learning Objectives

- Apply JavaScript rules for functions, declarations, return values, function calls, scope of reference, and local/global variable reference
- Apply your knowledge of functions in the context of publicly available software
- Design Web applications for mobile use
- Write JavaScript functions with the proper structure
- Build a UI that contains functions
- Explain what a computer-generated random number is

Standard Form of Function

- Functions are packages for algorithms
- PLs have a standard form for writing function declarations
- JavaScript requires a standard form for writing function declarations

```
function <name>( <parameter list> ) {
        <statement list>
}
```

- All of the punctuation is important
  - Parentheses always follow a function name
  - Curly braces {…} always come in pairs

Converting Some Temperatures

- First, let's write the Celsius to Fahrenheit conversion function in JavaScript using Scratchpad (Tools > Web Developer > Scratchpad)
- In the example, the name is convertC2F, the only parameter in the list is tempInC, and the only statement is a return statement

```
1 function convertC2F ( tempInC ) {
2     return 9/5 * tempInC +32;
3 }
4
5 convertC2F(0);
6
7 /*
8 32
9 */
```

Figure 19.1 Running the convertC2F( ) function in Scratchpad: the function is in lines 1–3, its call, that is, its application (on the input value 0), in line 5, and its result in line 8 in a JavaScript comment.

function <name> (<parameter list>) { < statement list> }

Picking a Name

- Function name begin with a letter, use any mix of letters, numbers, and underscores (_)
- Avoid reserved words
- Try to pick a meaning name that describe what the function does

Parameters

- The <parameter list> is simply a list of variables for the inputs separated by commas
- The parameters carry the input values to the function that the function will compute on
- The parameters don't have to be declared

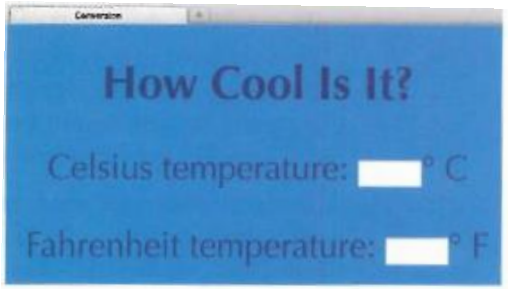
```
function convertC2F (tempInC) {
        return 9/5*tempInC + 32;
}
```

Function Definition

- The function definition is the algorithm written in a programming language
- A function definition follows the language's general rules for program statements
- JavaScript uses the statement return <expression> to get what the result is
- How do you get an answer from the function? It must be called!
- Calling a function!
  - Ask the computer to run or execute the statements of the function to produce the answers
  - Simply write the function's name and put the input values (called arguments) in parentheses
  - Then, the computer follows the definition of the function and returns the answer
- Functions are declared only once
- Functions are typically called many times

```
function convertC2F ( tempInC ) {
    return 9/5 * tempInC +32;
}

convertC2F(0);

/*
32
*/
```

Forms and Functions



- Let's create a Web page for testing our Java Script
- Use forms to test the script
- Recall the following from Chapter 18:
  - Forms must be enclosed in <form> tags
  - Text boxes are specified by an <input type="text" . . . /> tag
  - Text boxes have a id, size, and onchange attributes
  - To refer to the value or contents of a text box with id= "tb", we write tb.value
  - The main event handler of interest is onchange

```
<form id="cool">
  <p> Celsius temperature:
    <input type="text" id="textTempC" size="4"
      onchange="textTempF.value=Math.round(
      convertC2F(textTempC.value))"/>&#176;  C</p>
  <p> Fahrenheit temperature:
    <input type="text" id="textTempF" size="4"
      onchange="textTempC.value=Math.round(
      convertF2C(textTempF.value))"/>&#176;  F</p>
</form>
```

```
<!doctype html> <html>
 <head> <meta charset="UTF-8"/><title>Conversion</title>
 <style> body {background-color : dodgerblue; font-family : optima; color: midnightblue; text-align : center}
            p       {font-size : x-large}
 </style>
</head>
 <body>   <h1>How Cool Is It? </h1>
          <script>  function convertC2F (tempInC) {   return 9/5*tempInC + 32;  }
                    function convertF2C (tempInF) {    return 5/9*(tempInF - 32);  }      </script>
   <form id="cool">
    <p> Celsius temperature:       <input  type="text“  id="textTempC" size="4"     onchange =
            "textTempF.value = Math.round( convertC2F(textTempC.value) )"/> &#176;  C </p>
    <p> Fahrenheit temperature:  <input  type="text“  id="textTempF" size="4"    onchange =
            "textTempC.value = Math.round( convertF2C(textTempF.value) )"/> &#176;  F</p>
    </form>
 </body>
</html>
```

```
<!doctype html>
<html>
  <head> <meta charset="UTF-8"/><title>Conversion</title>
    <style>
      body {background-color : dodgerblue; font-family : optima;
              color: midnightblue; text-align : center}
      p      {font-size : x-large}
    </style>
  </head>
  <body>
    <h1>How Cool Is It? </h1>
    <script>
      function convertC2F (tempInC) {
        return 9/5*tempInC + 32;
      }
      function convertF2C (tempInF) {
        return 5/9*(tempInF - 32);
      }
    </script>
    <form id="cool">
      <p> Celsius temperature:
        <input type="text" id="textTempC" size="4"
         onchange="textTempF.value=Math.round(
         convertC2F(textTempC.value))"/>&#176;  C</p>
      <p> Fahrenheit temperature:
        <input type="text" id="textTempF" size="4"
         onchange="textTempC.value=Math.round(
         convertF2C(textTempF.value))"/>&#176;  F</p>
    </form>
  </body>
</html>
```

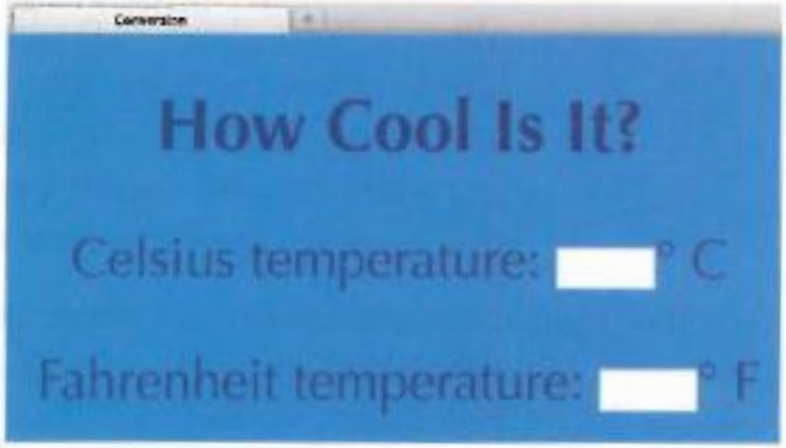


Figure 19.2 The HTML/JavaScript source for the temperature conversion page.

Writing Functions, Using Functions

- Flipping Electronic Coins
  - A coin flip is an unpredictable event whose 2 outcomes are “equally probable”
  - A computer could generate a random number between 0 and 1, and round to the nearest whole number
    - 0 could represent tails
    - 1 could represent heads
  - About half the time the outcome would be tails and the rest of the time it would be heads
  - Given a program and its input, isn’t the outcome is perfectly predictable?
  - They are not random in any way: Computers generate pseudo-random numbers
  - Most PLs have its own mathematics library (a collection of functions)
  - One of functions in the math library is Random Number Generator

Pseudo-random numbers

- An algorithm produces a sequence of numbers that passes the statistical tests for randomness (so, pseudorandom numbers are believable)
  - A sequence of pseudo-random numbers between 0 and 1 has the property that about half are closer to 0 and the others are closer to 1
  - In JavaScript the random number generator is called Math.random( )
  - Math.round( ) rounded the random number to the nearest whole number
- Returning 1 or 0 behaves like a coin flip
- When coinFlip( ) is called, it returns with equal probability a 0 or a 1
- An obvious improvement would be to return “Heads” and “Tails” rather than numbers

```
<script>
  function coinFlip( ) {
    return Math.round(Math.random());
  }
  function flipOut( ) {
    if (coinFlip( )==0)
      return 'Tails';
    else
      return 'Heads';
  }
</script>
```

Math.random( )

- Built-in function Math.random( ) produces a result in the interval [0,1)
  - Any number (except 1) is possible within those limits (and the limits of the computer)
  - The end point is not possible
- Multiply Math.random( ) by 2 and the interval over which the random numbers spread to [0,2)
- Generally, N * Math.random( ) expands to the interval [0,n)

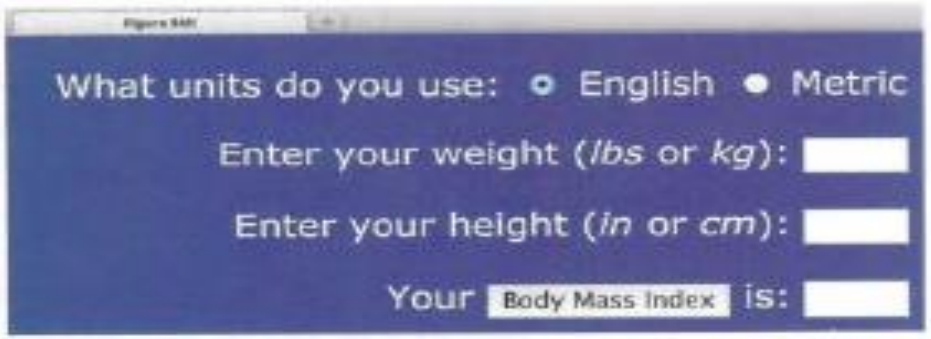
- The returning numbers are whole numbers with a decimal fraction
- If we throw away the decimal fraction using a built-in function Math.round(), we get only whole numbers
- Math.round(Math.random( )) gives you 0 or 1

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8"/><title>eCoin Flipping</title>
    <script>
      function coinFlip( ) {
        return Math.round(Math.random());
      }
      function flipOut( ) {
        if (coinFlip( )==0)
          return 'Tails';
        else
          return 'Heads';
      }
    </script>
    <style>
      body {background-color : #ccffcc; color : green;
            font-family : verdana; text-align : center}
    </style>
  </head>
  <body>
    <form id="flipper">
      <h2>Heads or Tails? </h2>
        <input type="button" value="Flip" onclick='ans.value=flipOut( );'/>
        <input  type="text" id="ans" size="5" onchange=" "/>
    </form>
  </body>
</html>
```

Figure 19.3 The JavaScript and image for the eCoin Flipping page.

Body Mass Index Computation [1/2]

- Uses input tag with radio buttons to select the English or Metric units
  - Recall that radio buttons are specified with <input . . . /> tags and must be placed within <form> tags
- The following are additional features of radio buttons:
  - All related radio buttons share the same name
    - if when clicking one the other should click off, then they must have the same name
  - Radio buttons can be preset by writing checked='checked‘
- onclick event handlers must also be written for the radio buttons

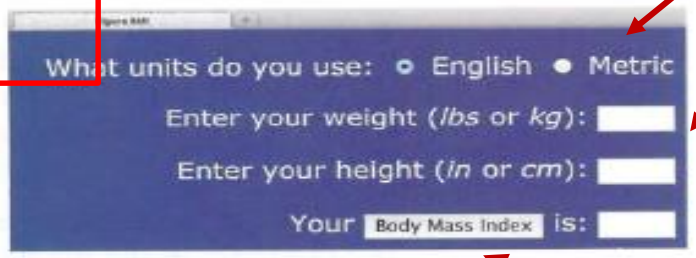


Body Mass Index Computation [2/2]

- What should happen when the user clicks the radio button?
  - Remember the type of units chosen… English(Lb, Inch) or Metric(Kg, Cm)?
  - When the Metric button is clicked, we want scale = "M"; as the response to the click-event

```
<script>
  var scale='E';
  function bmiM( weightKg, heightCm ) {
    var heightM = heightCm / 100;
    return weightKg / (heightM * heightM);
  }
  function bmiE( weightLbs, heightIn ) {
    return 703 * weightLbs / (heightIn * heightIn);
  }
  function BMI( units, weight, height) {
    if (units == "E")
      return bmiE( weight, height);  // lbs
    else
      return bmiM( weight, height)  // kgs
  }
</script>
```

```
<!doctype html>
<html>
  <head><meta charset="UTF-8"/><title>Figure BMI</title>
    <script>
      var scale='E';
      function bmiM( weightKg, heightCm ) {
        var heightM = heightCm / 100;
        return weightKg / (heightM * heightM);
      }
      function bmiE( weightLbs, heightIn ) {
        return 703 * weightLbs / (heightIn * heightIn);
      }
      function BMI( units, weight, height) {
        if (units == "E")
          return bmiE( weight, height);  // lbs
        else
          return bmiM( weight, height)  // kgs
      }
    </script>
    <style>
      body {background-color : indigo;
      color : white; font-family : verdana}
      p   {text-align : right}
    </style>
  </head>
  <body>
    <form name="mass">
      <p> What units do you use:
        <input type="radio" name="unit" onclick='scale="E"'
          checked/> English
        <input type="radio" name="unit" onclick='scale="M"'/>
          Metric</p>
      <p>Enter your weight (<i>lbs</i> or <i>kg</i>):
        <input type="text" id="wgt" size="4"/></p>
      <p> Enter your height (<i>in</i> or <i>cm</i>):
        <input type="text" id="hgt" size="4"/> </p>
      <p> Your
        <input type="button" value="Body Mass Index" id="figure"
          onclick="ans.value= BMI( scale, wgt.value, hgt.value)"/> is:
        <input type="text" id="ans" size="4"/></p>
    </form>
  </body>
</html>
```



radio button

text box

button

Figure 19.4 The image and source for the Figure BMI page.

Customizing Pages: eCoin Flipping [1/2]

- JavaScript code for Creating Page Content
  - A browser begins to create a page by reading through the HTML file
  - When the browser comes across the script tag <script>, the browser removes the script tag, then it does whatever the JavaScript tells it to do
  - A javascript built-in function document.write() inserts the text of its arguments into the Web page

document.write()
의 괄호안에 문장이
있으면 문장을 쓰고
HTML Tag가 있으면
HTML을 수행한다

Source File As Submitted

```
<body><p> The browser reads the
  HTML before it creates the page.
  When it comes to a script tag, it
  processes it immediately. If it
  has document.write( ) calls, the
  browser writes the argument</p>
  <script>
    document.write("into the file");
  </script>
  <p>at the point of the script .</p>
</body>
```

Text Used To Build Page

```
<body><p> The browser reads the
  HTML before it creates the page.
  When it comes to a script tag, it
  processes it immediately. If it
  has document.write( ) calls, the
  browser writes the argument</p>

  into the file

  <p>at the point of the script .</p>
</body>
```

Figure 19.5 An HTML source file containing a JavaScript document.write( ), and the HTML text used by the browser to create the page.

보조자료

document.write

- JavaScript로 문서 중간에 필요한 내용을 표시하려고 할때 사용하는built-in function
- 특별히 문서중간에 function의 결과치를 표현할때에 필요함

- document.write(“thank you”) ➔ thank you 가 화면에 찍힘
- document.write(1003) ➔ 1003 이 화면에 찍힘
- document.write(1 + 2) ➔ 3 // 1+2 가 수행된 결과가 화면에 찍힘
- document.write(F2C(10)) ➔ F2C(10)이 수행된 결과가 화면에 찍힘
- document.write(“aa” + “bb”) ➔ aabb 가 화면에 찍힘
- document.write(‘’) ➔

 HTML문장이 수행된 결과가 화면에 찍힘

Customizing Pages : eCoin Flipping [2/2]

- Customizing the Coin Flip
  - Lets use document.write() to display on-the-fly the proper heads or tails image
  - Locate 2 images to use
  - Change the flipOut() function from giving Heads or Tails output to instead giving a text string with the name of the image file we want to display

```
<body>
    <h2>Heads or Tails? </h2>
    <script>document.write('<img src=" ' +
        flipOut( ) + ' " alt="coin" width="150"/>');
    </script>
</body>
```

```
<script>
  function coinFlip( ) {
    return Math.round(Math.random());
  }
  function flipOut( ) {
    if (coinFlip( )==0)
      return "us1tails.jpg";
    else
      return "us1heads.jpg";
  }
</script>
```

 Or

 를 수행한다.....

```
<!doctype html>
<html>
  <head>
   <meta charset="UTF-8"/><title>eCoin Flipping</title>
   <script>
     function coinFlip( ) {
       return Math.round(Math.random());
     }
     function flipOut( ) {
       if (coinFlip( )==0)
          return "us1tails.jpg";
       else
          return "us1heads.jpg";
     }
   </script>
    <style>
      body {background-color : black; color : goldenrod;
            font-family : verdana; text-align : center}
    </style>
  </head>
  <body>
      <h2>Heads or Tails? </h2>
      <script>document.write('<img src="' +
         flipOut( ) + '" alt="coin" width="150"/>');
      </script>
  </body>
</html>
```

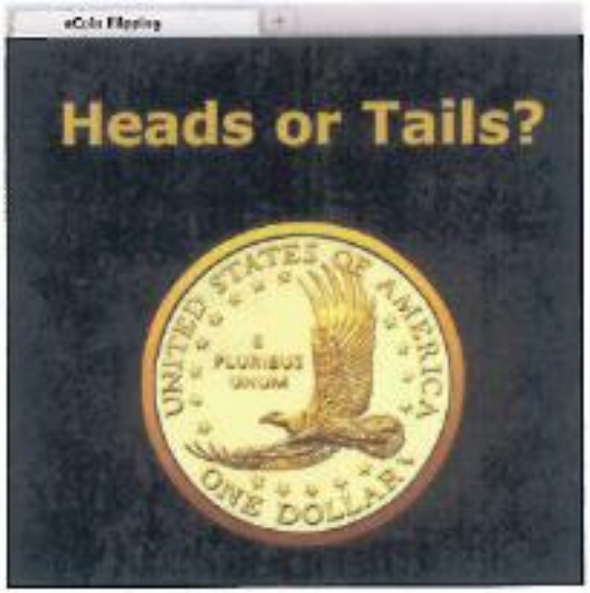


Figure 19.6 A revised eCoin flipping program, which flips the coin on loading using document.write( ).

Customizing Pages: Temperature Conversion

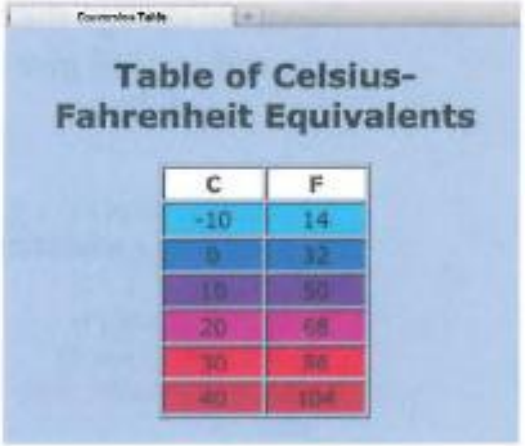
- Table of Equivalents
  - Suppose we want a table of temperature conversions for a Web Page with a column for Celsius and a column for Fahrenheit
  - Use document.write() to create the table on-the-fly

```
document.write('<tr>  <td>-10</td> <td>' convertC2F(-10)  '</td> <tr>')
```

➔ `<tr>  <td>-10</td> <td> convertC2F(-10)  </td> <tr>`

- Automatically Created Rows
  - When the browser encounters script tags it does what the JavaScript tells it to do and calls document.write()
  - The browser must construct the function's arguments using concatenation
  - When the browser builds the page, the table is formed from our created on-the-fly rows that use our conversion function

```
<script>
 function convertC2F ( tempInC ) {
   return (9 / 5) * tempInC + 32;
 }
</script>
 <style>
  body {background-color : lightsteelblue;
         color : black; font-family:verdana;
         text-align : center}
  table {margin-left : auto;
         margin-right : auto; }
  th    {min-width : 70px;
         background-color : white}
</style>
</head>
<body><h2> Table of Celsius-<br/>Fahrenheit Equivalents</h2>
```




```
 <script>
  document.write("<table border='1'>");
  document.write("<tr><th> C </th><th> F </th></tr>");
  document.write('<tr style="background-color : #00ccff">');
  document.write('    <td>-10</td><td>' + convertC2F(-10) + '</td></tr>');
  document.write('<tr style="background-color : #0088ff">');
  document.write('    <td> 0 </td><td>' + convertC2F(0)  + '</td></tr>');
  document.write('<tr style="background-color : #8800cc">');
  document.write('    <td> 10</td><td>' + convertC2F(10) + '</td></tr>');
  document.write('<tr style="background-color : #cc0088">');
  document.write('    <td> 20</td><td>' + convertC2F(20) + '</td></tr>');
  document.write('<tr style="background-color : #ff0033">');
  document.write('    <td> 30</td><td>' + convertC2F(30) + '</td></tr>');
  document.write('<tr style="background-color : #cc0033">');
  document.write('    <td> 40</td><td>' + convertC2F(40) + '</td></tr>');
  document.write("</table>");
</script>
```

Figure 19.7 Source text and image for the Conversion Table computation.

Not Working!!

```html
<!doctype html>
<html>
   <head>
      <meta charset="UTF-8"/>
      <title>Conversion Table</title>
       <script>
           function convertC2F ( tempInC ) {
              return (9 / 5) * tempInC + 32;}
        </script>
        <style>
        body {background-color : lightsteelblue;
                  color : black; font-family:verdana;
                  text-align : center}
        table {margin-left : auto; margin-right : auto; }
        th      {min-width : 70px;  background-color : white}
      </style>
   </head>
   <body><h2> Table of Celsius-<br/>Fahrenheit Equivalents</h2>
           <table border='1'>
           <tr><th> C </th><th> F </th></tr>
           <tr style="background-color : #00ccff">
           <td>-10</td> <td> <script> convertC2F(-10) </script> </td> </tr>
           <tr style="background-color : #0088ff">
           <td> 0 </td> <td> <script> convertC2F(0) </script> </td></tr>
           <tr style="background-color : #8800cc">
           <td> 10</td> <td> <script> convertC2F(10) </script> </td></tr>
           <tr style="background-color : #cc0088">
           <td> 20</td> <td> <script> convertC2F(20) </script> </td></tr>
           <tr style="background-color : #ff0033">
           <td> 30</td> <td> <script> convertC2F(30) </script> </td></tr>
           <tr style="background-color : #cc0033">
           <td> 40</td> <td> <script> convertC2F(40) </script> </td></tr>
           </table>
   </body>
```

파일(F) 편집(E) 보기(V) 설정(S) ?

Working!!

```html
<html>
  <head>
    <meta charset="UTF-8"/>
    <title>Conversion Table</title>
     <script>
        function convertC2F ( tempInC ) {
           return (9 / 5) * tempInC + 32;}
      </script>
      <style>
      body {background-color : lightsteelblue;
                color : black; font-family:verdana;
                text-align : center}
      table {margin-left : auto; margin-right : auto; }
      th     {min-width : 70px;  background-color : white}
    </style>
  </head>
  <body><h2> Table of Celsius-<br/>Fahrenheit Equivalents</h2>
        <table border='1'>
        <tr><th> C </th><th> F </th></tr>
        <tr style="background-color : #00ccff">
        <td>-10</td> <td> <script> document.write(convertC2F(-10))</script> </td> </tr>
        <tr style="background-color : #0088ff">
        <td> 0 </td> <td> <script> document.write(convertC2F(0)) </script> </td></tr>
        <tr style="background-color : #8800cc">
        <td> 10</td> <td> <script> document.write(convertC2F(10)) </script> </td></tr>
        <tr style="background-color : #cc0088">
        <td> 20</td> <td> <script> document.write(convertC2F(20)) </script> </td></tr>
        <tr style="background-color : #ff0033">
        <td> 30</td> <td> <script> document.write(convertC2F(30)) </script> </td></tr>
        <tr style="background-color : #cc0033">
        <td> 40</td> <td> <script> document.write(convertC2F(40)) </script> </td></tr>
        </table>
  </body>
</html>
```

Making a Web-Based Phone App

- We will write and then load our app using a browser on our smartphone or tablet
- Our app will work just fine on a laptop or desktop, but will be designed for something smaller
- Design for Mobility
  - The Bean Counter app would be hard to use on a phone display because of the small buttons and drop-down menu, so we exclude the bean counter
  - The touch metaphor benefits from larger blocks and a more “open” organization
  - We will use a two-dimensional grid (table) of blocks that the user will tap (touch)

Figure 19.8 Navigation Web page for executing user-created functions from this book, organized for use on a mobile device.

```html
<!doctype html>
<html>
 <head>
  <meta charset="UTF-8"><title>myApps</title>
  <style>
    body   {background-color:black; color:cyan;font-family:helvetica}
    h2     {color:white;text-align:center;}
    table  {margin-left:auto;margin-right:auto;}
    td     {background-color:orangered; color:white;min-width:100px;
              text-align:center; padding:20px;}
    td.alta {background-color:deeppink;}
    td.altb {background-color:fuchsia;}
    a      {text-decoration:none;color:white;}
  </style>
 </head>
 <body>
  <h2>myApps</h2>
  <table border="0">
   <tr><td><a href="bmi.html"> bmi </a></td>
        <td><a href="temperature.html"> C° ≈ F°</a></td></tr>
   <tr><td class="alta"><a href="counter.html"> counter </a></td>
        <td class="alta"><a href="rps.html"> RPS</a></td></tr>
   <tr><td class="altb"><a href="flipOut.html">coin flip</a></td>
        <td class="altb"><a href="itsMagic.html"> magic 8</a></td></tr>
  </table>
   <script type="text/javascript">
    var today = new Date( );   // Get today's date
    var myBdate = new Date( );  // Get a date object to modify
    var difference;  // Declare a temporary variable
    myBdate.setFullYear(1995);  // Set my birth year to 1995
    myBdate.setMonth(6);   // Set my birth mo to July (mos start at 0)
    myBdate.setDate(4);  // Set my birth day to 4th
    myBdate.setHours(12);  // Set my hour of birth to noon
    myBdate.setMinutes(0);// Set my minute of birth to o'clock
    myBdate.setSeconds(0);    // Set my second of birth on the hour
    difference = today.getTime( ) - myBdate.getTime( );
    difference = Math.floor(difference/1000);
    document.write(" <p style='text-align:center'> my age: " + difference +
      " seconds </p>");
   </script>
 </body>
</html>
```

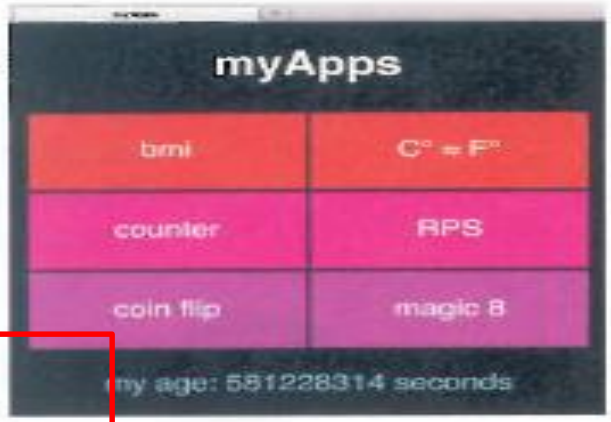


Figure 19.9 The HTML for the navigation Web page. (To include the "age" text at the bottom, see the companion Fluency Byte).

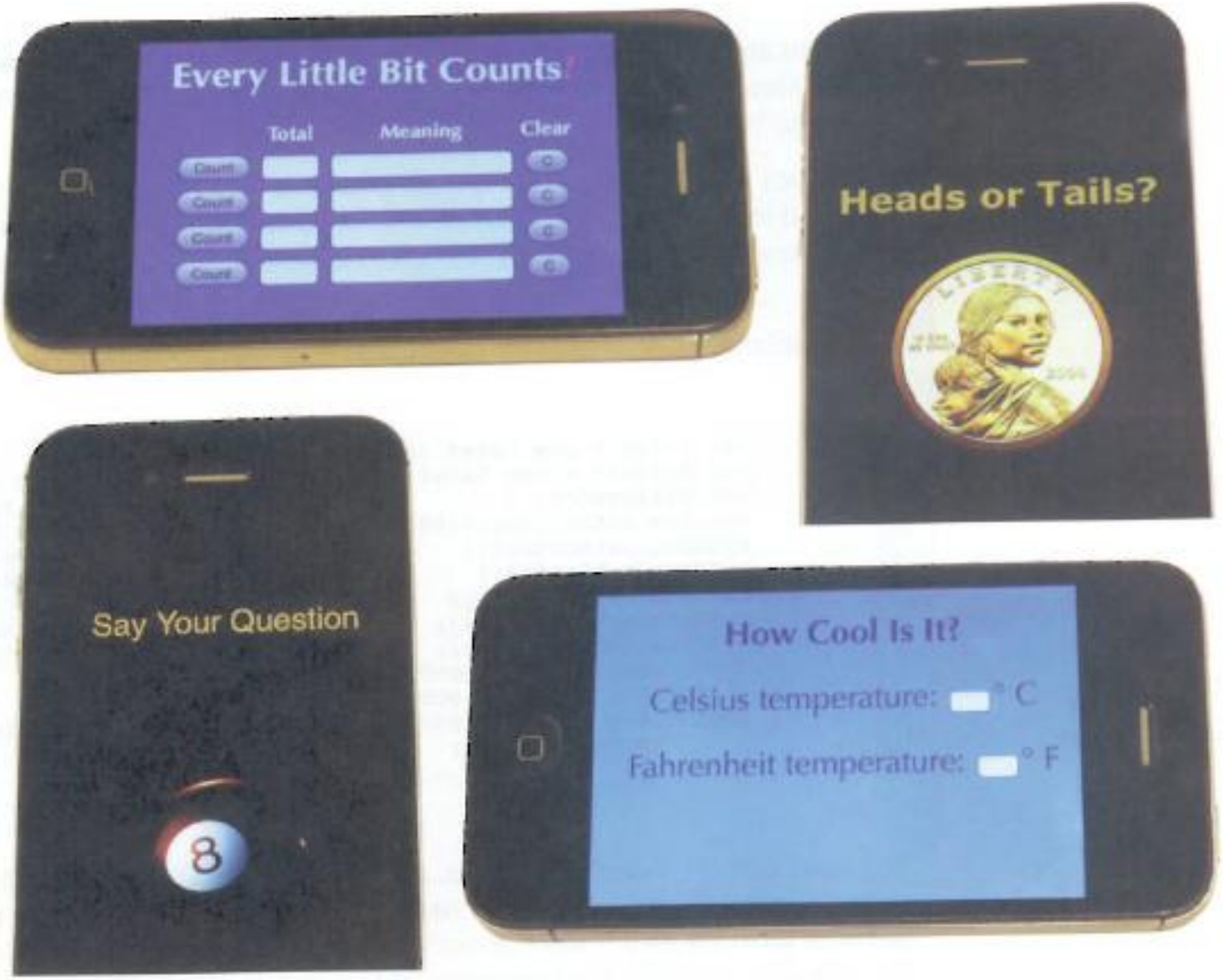


Figure 19.10 Examples of pages connecting to functions written in this chapter (the Magic Decider app is written in Chapter 20).

Every Little bit Counts

- We will create a Counter Assistant page for keeping track of counts
  - Clicking the Count button increments the Total field, the Meaning field can be filled in with any text, and the C button clears the fields
  - We write a function row() to create a row of the table, placing the entire HTML text in the function
- Requires us to use a sequence of document.write() function calls
- It relies on 4 global variables to keep track of the counts
- It sets up the structure of the table and then calls a row() function, which constructs the rows and their input controls
- The row() function has a single parameter that is the number of the row being specified

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8"/><title>Counter Assistant</title>
     <style type="text/css">
       body {background-color : blueviolet; color : white; font-family : optima;
              text-align : center}
       table {margin-left : auto; margin-right : auto}
     </style>
    <script>
      var count1=0, count2=0, count3=0, count4=0;
      function row(num) {
        document.write( '<tr><td><input type="button" value="Count" ' );
        document.write( ' onclick="count'+num+'=count'+num+'+1;' );
        document.write('arch'+num+'.value=count'+num+'"/></td>' );
        document.write('<td><input type="text" size="5" id="arch'+num+'"/></td>' );
        document.write('<td><input type="text" size="20" id="what'+num+'"/></td>' );
        document.write('<td><input type="button" value="C" ' );
        document.write( ' onclick="arch'+num+'.value='+'" ';" );
        document.write("what"+num+".value=' ';" );
        document.write('count'+num+'=0"/></td></tr>' );
      }
    </script>
  </head>
  <body>
    <h2>Every Little Bit Counts<i style="color : hotpink">!</i></h2>
    <form>
      <table>
        <tr><th> </th><th> Total </th><th> Meaning </th><th>Clear</th></tr>
        <script>
          row(1); row(2); row(3); row(4);
        </script>
      </table>
    </form>
  </body>
</html>
```

Figure 19.12 The makeTable( ) and row( ) functions, which generate the Counter Assistant application.

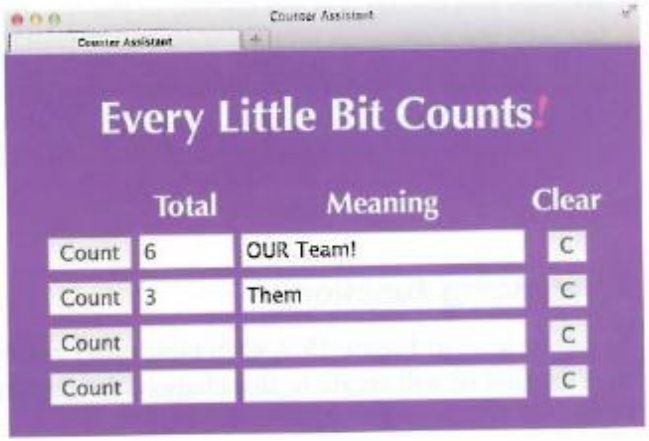


Figure 19.11 The Counter Assistant page to track four items.

2 Reasons to Write Functions

- Most functions are general
  - They are written for a specific application
  - We hope that we will have a chance to use them again
  - They are building blocks for future programs
- Some functions are not building blocks
  - They must run within a document with a form, and that form must have within it input controls with specific names
- Managing complexity is the other reason to write functions
- The 2 reasons for packaging algorithms into functions:
  - SW Reuse: the building blocks of future programming
  - Complexity management: keeps our sanity while solving problems

Social Functions regarding SW functions

- “Boldly go” after other people’s software that you may not fully understand!
  - There is a strong tradition in computing to share code
  - From the Open Source movement, to all browsers displaying the Page Source
  - Suppose you found a page using the <canvas> tag from HTML5 in Figure19.13

    <canvas id="canvas" width="150" height="150“ ></canvas>
  - When the window is loaded, the onload event handler is acticated

    <body onload = “draw()”>

- With almost no understanding of <canvas>

  you’ve already tried it out!

참고자료

HTML Canvas Tag for Graphics

```
<canvas id="myCanvas" width="200" height="100"></canvas>
```

Draw onto the canvas with JavaScript

Canvas는 비워져있는 그림그리는 틀: 그림은 JavaScript code로 그린다

- All drawing on the canvas must be done inside a JavaScript:

```
<script>
  var c = document.getElementById("myCanvas");
  var ctx = c.getContext("2d");
  ctx.fillStyle="#FF0000";
  ctx.fillRect(0,0,150,75);
</script>
```

그림을 그리는데 필요한 기능을 가진 getContext() method를 부름

  - The fillStyle property can be a CSS color, a gradient, or a pattern.
  - The fillRect(x,y,width,height) method draws a rectangle filled with the current fill style.

참고자료

Built-In Properties and Methods of Canvas Tag

- ctx.fillStyle = "rgb(200,0,0)"; // fillStyle은 built-in property, rgb는 색상표

  ctx.fillRect(10,10,55,50); // built-in method

- ctx.fillStyle = " rgba(0,0,200,0.5) " ; //opacity (투명도) 를 조절이 포함된 rgb

- ctx.moveTo(100,100); // line drawing

  ctx.lineTo(200,200);

- ctx,moveTo(75,25); // Bezier Curve for Smooth Line

  ctx.quadraticCurveTo(25,100,50,100); // (75, 25) --- (50,100) --- (25,100)

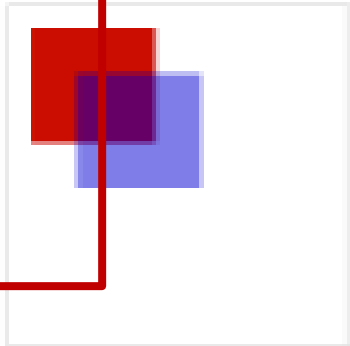
A skeleton template

```
<!DOCTYPE html>
<html>
  <head>
    <title>Canvas tutorial</title>
    <script type="text/javascript">
      function draw(){
        var canvas =document.getElementById('tutorial');
        if (canvas.getContext){
          var ctx = canvas.getContext('2d');
        }
      }
    </script>
    <style type="text/css">
      canvas { border: 1px solid black; }
    </style>
  </head>
  <body onload="draw();">
    <canvas id="tutorial" width="150" height="150"></canvas>
  </body>
</html>
```

그림을 그리는데 필요한 기능을 가진 getContext() method가 browser에서 지원이 되는지를 확인

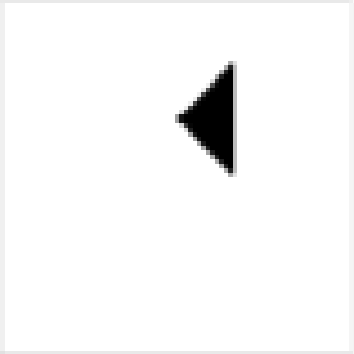
A simple example

```
<html>
  <head>
    <script type="application/javascript">
      function draw() {
        var canvas =document.getElementById("canvas");
        if (canvas.getContext) {
          var ctx = canvas.getContext("2d");

          ctx.fillStyle = "rgb(200,0,0)";
          ctx.fillRect (10, 10, 55, 50);

          ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
          ctx.fillRect (30, 30, 55, 50);
        }
      }
    </script>
  </head>
  <body onload="draw();">
    <canvas id="canvas" width="150" height="150"></canvas>
  </body>
</html>
```



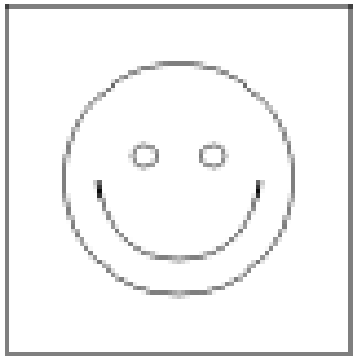
A simple example

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    ctx.beginPath();
    ctx.moveTo(75,50);
    ctx.lineTo(100,75);
    ctx.lineTo(100,25);
    ctx.fill();
  }
}
```



A simple example

```
function draw() {
  var canvas =document.getElementById('canvas');
  if (canvas.getContext){
    var ctx = canvas.getContext('2d');

    ctx.beginPath();
    ctx.arc(75,75,50,0,Math.PI*2,true); // Outer circle
    ctx.moveTo(110,75);
    ctx.arc(75,75,35,0,Math.PI,false);   // Mouth (clockwise)
    ctx.moveTo(65,65);
    ctx.arc(60,65,5,0,Math.PI*2,true);  // Left eye
    ctx.moveTo(95,65);
    ctx.arc(90,65,5,0,Math.PI*2,true);  // Right eye
    ctx.stroke();
  }
}
```



draw2box.html

```
<html>
 <head><title>Sample Canvas Code</title>
 <script type="application/javascript">
  function draw() {
   var canvas = document.getElementById("canvas");
   if (canvas.getContext) {
    var ctx = canvas.getContext("2d");

    ctx.fillStyle = "rgb(200,0,0)";
    ctx.fillRect (10, 10, 55, 50);

    ctx.fillStyle = "rgba(0, 0, 200, 0.5)";
    ctx.fillRect (30, 30, 55, 50);
   }
  }
 </script>
 <style>canvas {border-style:solid;border-color:lightgray}</style>
 </head>
 <body onload="draw();">
  <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Added to show canvas shape

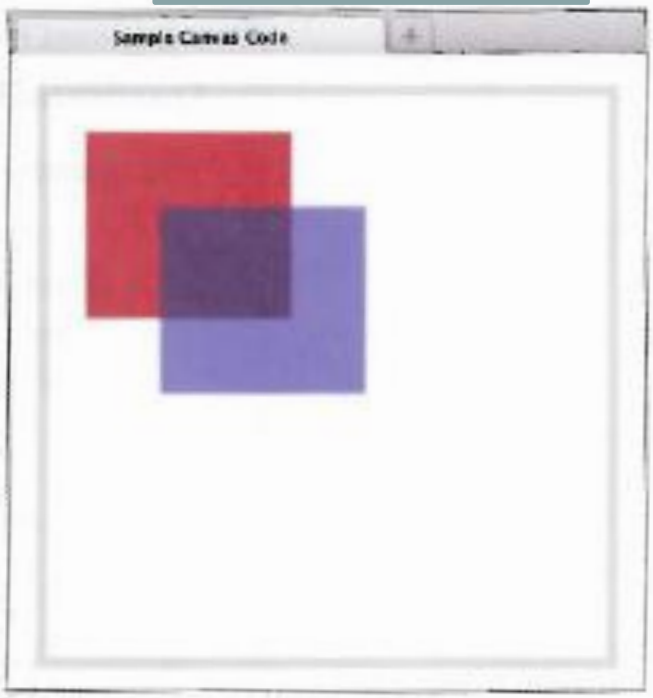


Figure 19.13 Example of the basic use of the <canvas> tag, (a) code, and (b) display.

Making a Word Ballon Comment [1/4]

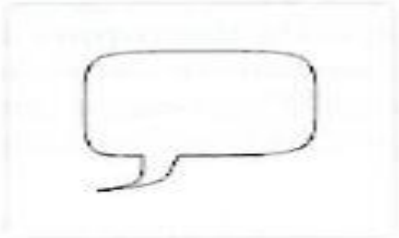
- Suppose you happens to know Bezier functions
- Now you want to create a Web page that has an image and a word balloon on it

Quadratic Bezier curves

This example uses multiple quadratic Bézier curves to render a speech balloon.

```
function draw() {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');

    // Quadratic curves example
    ctx.beginPath();
    ctx.moveTo(75,25);
    ctx.quadraticCurveTo(25,25,25,62.5);
    ctx.quadraticCurveTo(25,100,50,100);
    ctx.quadraticCurveTo(50,120,30,125);
    ctx.quadraticCurveTo(60,120,65,100);
    ctx.quadraticCurveTo(125,100,125,62.5);
    ctx.quadraticCurveTo(125,25,75,25);
    ctx.stroke();
  }
}
```

Screenshot



Live sample

Canvas class
-beginPath()
-moveTo
-quadraticCurveTo()
-stroke()
-…..
-Many many built-in functions

Figure 19.14 Mozilla tutorial on quadratic Bézier functions to make word balloons (from https://developer.mozilla.org/en-US/docs/Web/Guide/HTML/Canvas_tutorial/Drawing_shapes).

Making a Word Ballon Comment [2/4]

- Add Text to a Balloon
  - if we add a <p> tag after the <canvas> text it writes the text below the balloon
  - We apply absolute position to the paragraph element, to move it to the correct position

wordballoon1.html

```
<!doctype html>
<html>
 <head><title>Word Balloons</title>
  <script type="application/javascript">
function draw() {
 var canvas = document.getElementById('canvas');
 if (canvas.getContext) {
  var ctx = canvas.getContext('2d');
  // Quadratric curves example
  ctx.beginPath();
  ctx.moveTo(75,25);
  ctx.quadraticCurveTo(25,25,25,62.5);
  ctx.quadraticCurveTo(25,100,50,100);
  ctx.quadraticCurveTo(50,120,30,125);
  ctx.quadraticCurveTo(60,120,65,100);
  ctx.quadraticCurveTo(125,100,125,62.5);
  ctx.quadraticCurveTo(125,25,75,25);
  ctx.stroke();
 }
}
 </script>
 </head>
 <body onload="draw();">
  <canvas id="canvas" width="150" height="150"></canvas>
 </body>
</html>
```

Figure 19.16 Initial word balloon program, which combines two Mozilla tutorial programs.

Figure 19.17 Trying to place text on a page: (a) placing a paragraph, (b) CSS's absolute positioning reference, and (c) the text placed in the balloon.

```
<p style="position:absolute;left:40px;top:45px">Text</p>
```

Making a Word Ballon Comment [3/4]

- Move the Balloon Around
  - To have a word balloon anywhere on the screen we need to expand the canvas
  - We will use window.innerWidth and window.innerHeight
  - Making this change did not actually move the balloon
  - We revise the draw() function to have x and y as parameters
  - We also use the same parameters for the text

```
document.write('<canvas id="canvas" width="' + window.innerWidth
  + '" height="' + window.innerHeight + '"></canvas>');
```

```
function draw(x, y) {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
    // Quadratric curves example
    ctx.beginPath();
    ctx.moveTo(75+x,25+y);
    ctx.quadraticCurveTo(25+x,25+y,25+x,62.5+y);
    ctx.quadraticCurveTo(25+x,100+y,50+x,100+y);
    ctx.quadraticCurveTo(50+x,120+y,30+x,125+y);
    ctx.quadraticCurveTo(60+x,120+y,65+x,100+y);
    ctx.quadraticCurveTo(125+x,100+y,125+x,62.5+y);
    ctx.quadraticCurveTo(125+x,25+y,75+x,25+y);
    ctx.stroke();
  }
}
```

```
function comment(x, y, remark) {
  document.write('<p style="position:absolute; left:' + (40+x)
    + 'px; top:' + (45+y) + 'px;">' + remark + '</p>');
}
```

Making a Word Ballon Comment [4/4]

- Place an Image
  - The easiest way to cover the window with an image is to make it the background

```
body {background-image:url('emperor.jpg');background-repeat:no-repeat; }
```

- Position the balloon
  - We estimate that the balloon should be about 700 pixels from the left side, and 10 pixels down from the top
  - We need to fill in the background of the word balloon with white

wordballoonfinal.html

```html
<!doctype html>
<html>
 <head>  <meta charset="UTF-8"/> <title>Word Balloon Page</title>
 <script>
function draw(x, y) {
  var canvas = document.getElementById('canvas');
  if (canvas.getContext) {
    var ctx = canvas.getContext('2d');
    // Quadratric curves example
    ctx.beginPath();
    ctx.moveTo(75+x,25+y);
    ctx.quadraticCurveTo(25+x,25+y,25+x,62.5+y);
    ctx.quadraticCurveTo(25+x,100+y,50+x,100+y);
    ctx.quadraticCurveTo(50+x,120+y,30+x,125+y);
    ctx.quadraticCurveTo(60+x,120+y,65+x,100+y);
    ctx.quadraticCurveTo(125+x,100+y,125+x,62.5+y);
    ctx.quadraticCurveTo(125+x,25+y,75+x,25+y);
    ctx.stroke();
    ctx.fillStyle = "rgb(255,255,255)";
    ctx.fill();
  }
}
function comment(x, y, remark) {
   document.write('<p style="position:absolute; left:' + (40+x)
      + 'px; top:' + (45+y) + 'px;">' + remark + '</p>');
}
</script>
 <style> body {background-image:url('emperor.jpg');background-repeat:no-repeat; } </style>
 </head>
 <body onload="draw(700,10);">
  <script>
     document.write('<canvas id="canvas" width="' + window.innerWidth
        + '" height="' + window.innerHeight + '"></canvas>');
     comment(700,10,"Let's Ditch.");
  </script>
</body>
```

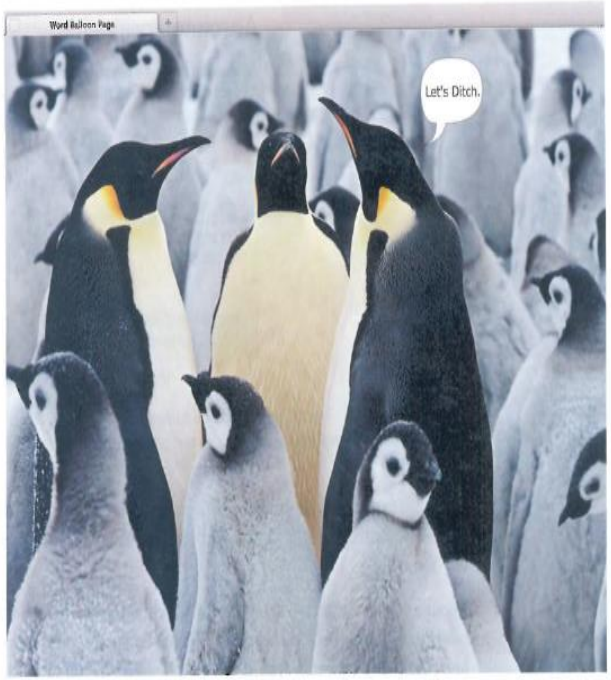


Figure 19.15 A planned page combining an image and a word balloon comment.

Summary [1/2]

- The 3 parts of a function (name, parameter list, and definition) are specified in a function declaration using a standard form
- One time declaration of a function specifies to the computer how the function works
  - To call functions, we give the function name and its input values, known as arguments
- Writing functions packages algorithms, but to get their benefit in JavaScript and HTML requires that we develop Web pages (UI) with which we give the inputs to the functions and get their answers displayed
- There are 3 different ways to display the results of a function in HTML:
  - using alert()
  - interacting with a page that has text boxes
  - using document.write() to include the results of a function while the page is being constructed

Summary [2/2]

- We put all of our knowledge about functions into a small Web Apps page; the source code is shown in Appendix F
  - It gave us the ability to apply functions directly using our smartphones
  - The next thing is to think up and implement some apps of personal interest
- Finally, we “boldly went” online to find tutorial SW examples that we could put to use simply by trying them out and noticing what happened
  - We quickly figured them out and put them to use