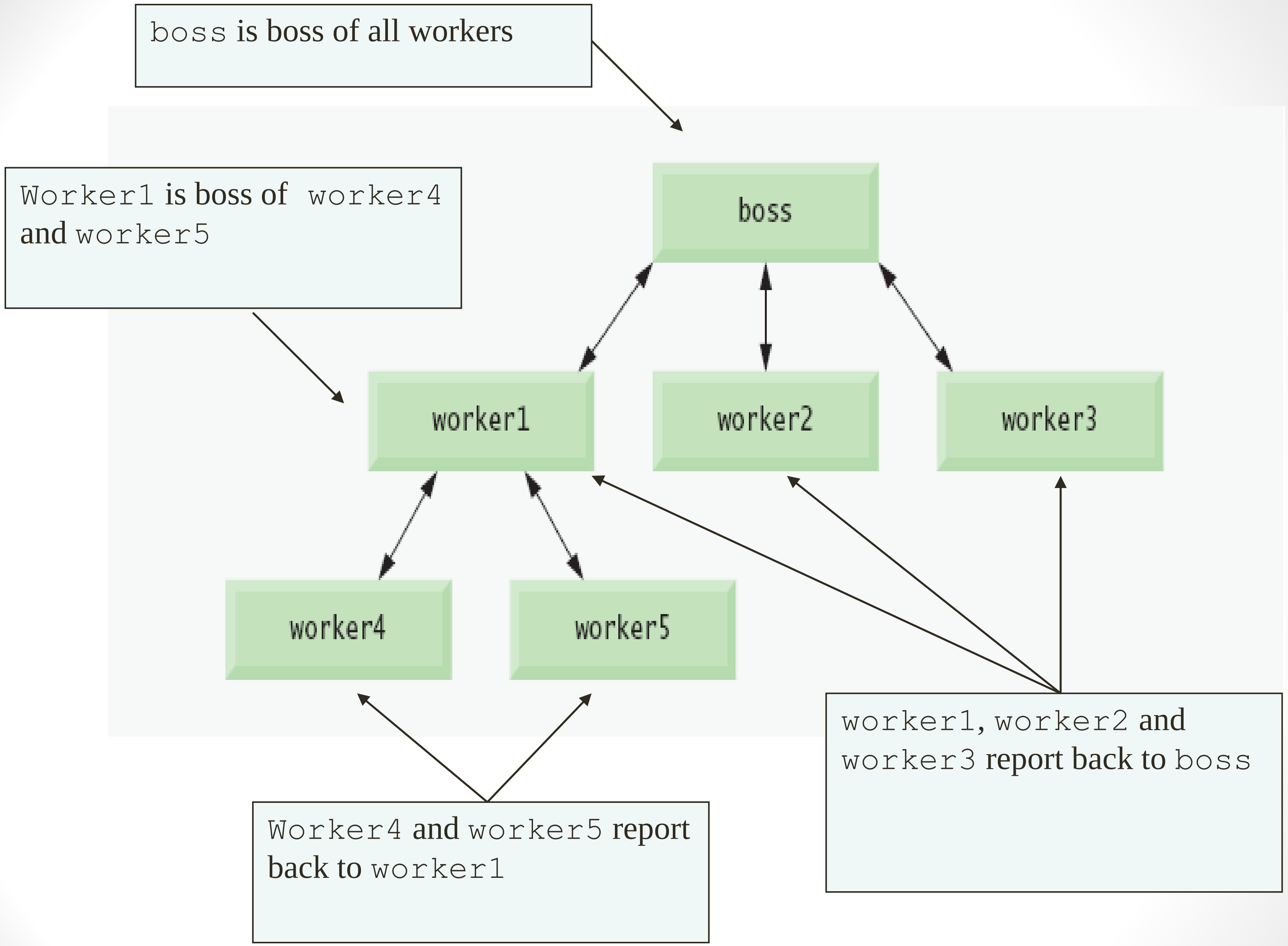
Introduction

- Divide and conquer: To develop and maintain a large program construct it from small, simple pieces
- Programs are written by combining new functions that the programmer writes with "prepackaged" functions and objects available in JavaScript
- Method: implies that a function belongs to a particular object
- JavaScript provides several objects that have a rich collection of methods for performing common mathematical calculations, string manipulations, date and time manipulations, and manipulations of collections of data called arrays
- Whenever possible, use existing JavaScript objects, methods and functions instead of writing new ones
- You can define programmer-defined functions that perform specific tasks and use them at many points in a script
- Functions: invoked by writing the name of the function, followed by a comma-separated list of zero or more arguments
- Methods: called in the same way as functions, but require the name of the object to which the method belongs and a dot preceding the method name
- Function (and method) arguments may be constant, variables or expressions


boss is boss of all workers
Worker1 is boss of worker4 and worker5
boss
worker1
worker2
worker3
worker4
worker5
worker1, worker2 and worker3 report back to boss
Worker4 and worker5 report back to worker1

Programmer-Defined Functions

- Variables declared in function definitions are local variables
- When a function is called, the arguments in the call are assigned to the corresponding parameters in the function definition
- Code that is packaged as a function can be executed from several locations in a program by calling the function explicitly
- Each function should perform a single, well-defined task, and the name of the function should express that task effectively promotes software reusability
- return statement passes information from inside a function back to the point in the program where it was called

Programmer-Defined Functions (2)

- Three ways to return control to the point at which a function was invoked
  - Reaching the function-ending right brace
  - Executing the statement **return;**
  - Executing the statement "**return *expression*;**" to return the value of *expression* to the caller
- When a return statement executes, control returns immediately to the point at which the function was invoked
- The general format of a function definition is

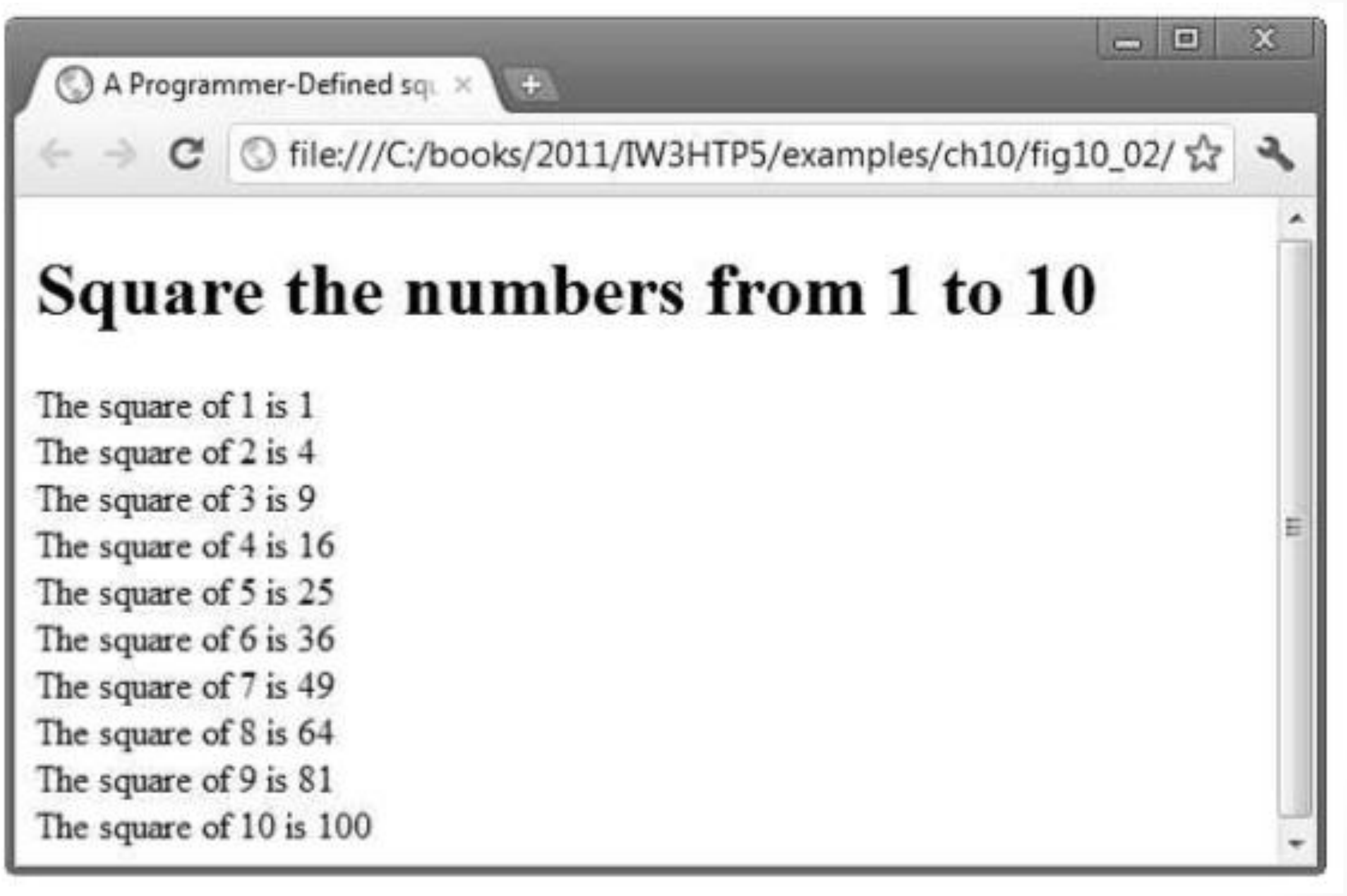
function function-name (parameter-list separated by commas)

{

  declarations and statements, i.e. the function body

}

```
 1  <!DOCTYPE html>
 2
 3  <!-- Fig. 9.2: SquareInt.html -->
 4  <!-- Programmer-defined function square. -->
 5  <html>
 6     <head>
 7        <meta charset = "utf-8">
 8        <title>A Programmer-Defined square Function</title>
 9        <style type = "text/css">
10           p { margin: 0; }
11        </style>
12        <script>
13
14           document.writeln( "<h1>Square the numbers from 1 to 10</h1>" );
15
16           // square the numbers from 1 to 10
17           for ( var x = 1; x <= 10; ++x )
18              document.writeln( "<p>The square of " + x + " is " +
19                 square( x ) + "</p>" );
20
21           // The following square function definition's body is executed
22           // only when the function is called explicitly as in line 19
23           function square( y )
24           {
25              return y * y;
26           } // end function square
27
28        </script>
29     </head><body></body> <!-- empty body element -->
30  </html>
```

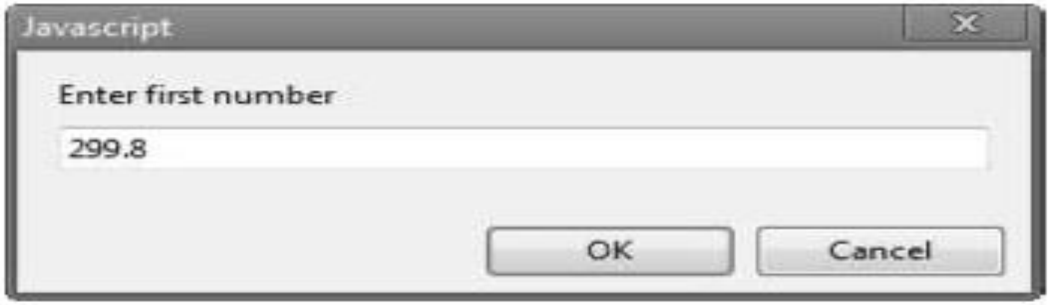


Square the numbers from 1 to 10

```
The square of 1 is 1
The square of 2 is 4
The square of 3 is 9
The square of 4 is 16
The square of 5 is 25
The square of 6 is 36
The square of 7 is 49
The square of 8 is 64
The square of 9 is 81
The square of 10 is 100
```

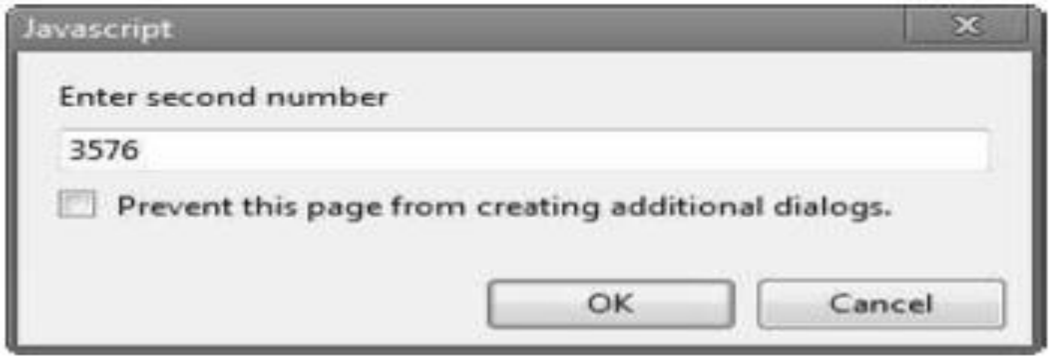
```
<!DOCTYPE html>

<!-- Fig. 9.3: maximum.html -->
<!-- Programmer-Defined maximum function. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Maximum of Three Values</title>
      <style type = "text/css">
         p { margin: 0; }
      </style>
      <script>

         var input1 = window.prompt( "Enter first number", "0" );
         var input2 = window.prompt( "Enter second number", "0" );
         var input3 = window.prompt( "Enter third number", "0" );

         var value1 = parseFloat( input1 );
         var value2 = parseFloat( input2 );
         var value3 = parseFloat( input3 );
```

```
21
22          var maxValue = maximum( value1, value2, value3 );
23
24          document.writeln( "<p>First number: " + value1 + "</p>" +
25             "<p>Second number: " + value2 + "</p>" +
26             "<p>Third number: " + value3 + "</p>" +
27             "<p>Maximum is: " + maxValue + "</p>" );
28
29          // maximum function definition (called from line 22)
30          function maximum( x, y, z )
31          {
32             return Math.max( x, Math.max( y, z ) );
33          } // end function maximum
34
35       </script>
36    </head><body></body>
37 </html>
```


Javascript

Enter first number

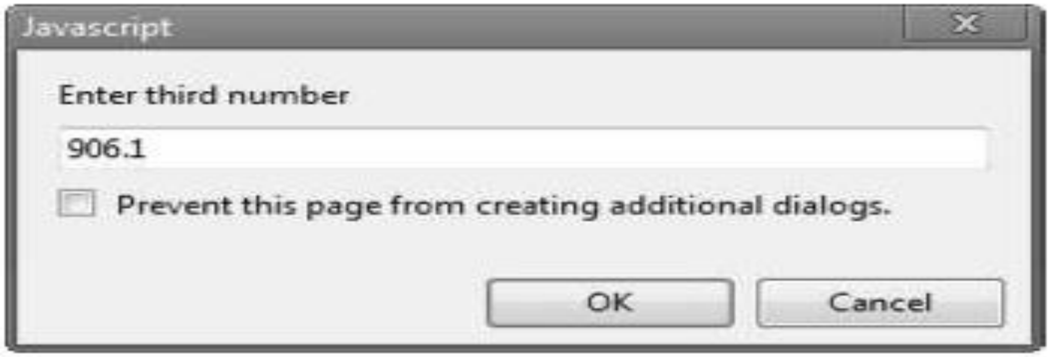
299.8


Javascript

Enter second number

3576

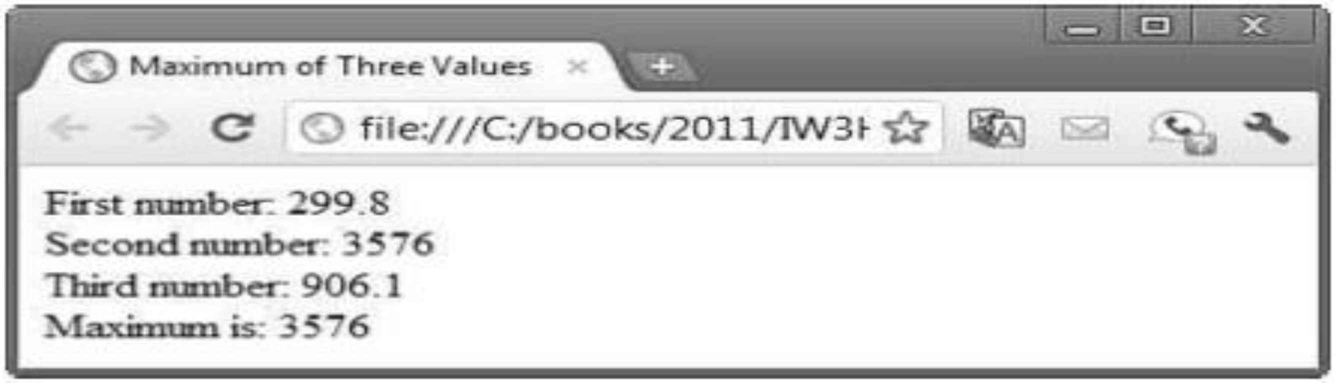
Prevent this page from creating additional dialogs.


Javascript

Enter third number

906.1

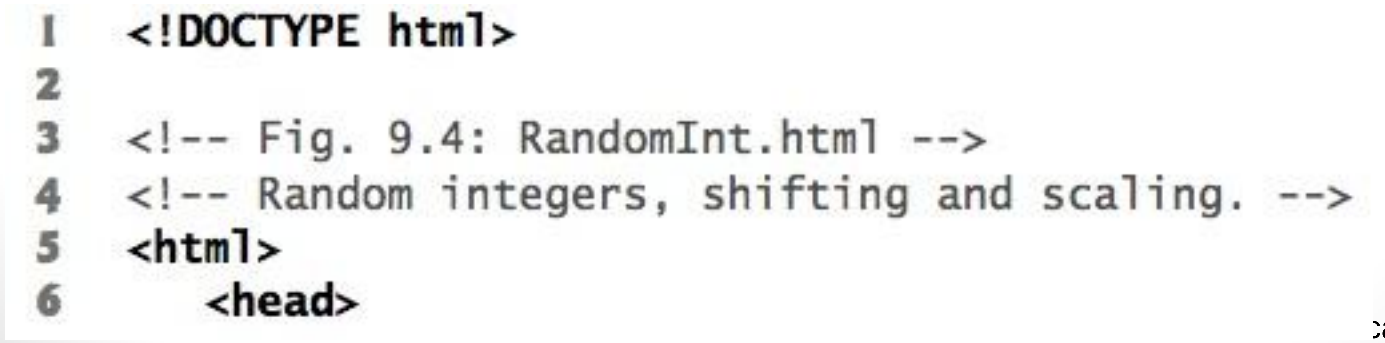
Prevent this page from creating additional dialogs.



```
First number: 299.8
Second number: 3576
Third number: 906.1
Maximum is: 3576
```

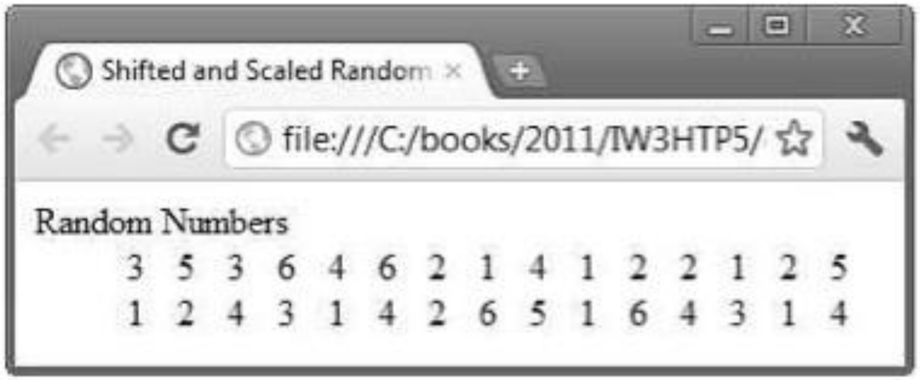
Random Number Generation

- Method `random` generates a floating-point value from 0.0 up to, but not including, 1.0
- Random integers in a certain range can be generated by scaling and shifting the values returned by `random`, then using `Math.floor` to convert them to integers
  - The scaling factor determines the size of the range (i.e. a scaling factor of 4 means four possible integers)
  - The shift number is added to the result to determine where the range begins (i.e. shifting the numbers by 3 would give numbers between 3 and 7)
- Method `Math.floor` rounds its argument down to the closest integer

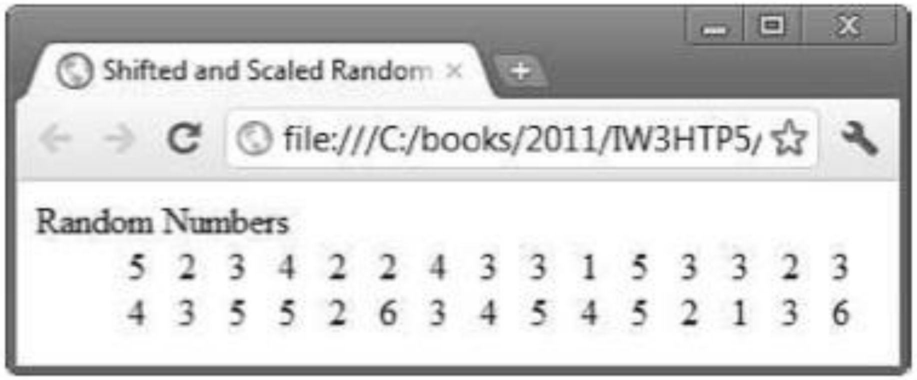


```
<!DOCTYPE html>

<!-- Fig. 9.4: RandomInt.html -->
<!-- Random integers, shifting and scaling. -->
<html>
   <head>
```

```
        <meta charset = "utf-8">
        <title>Shifted and Scaled Random Integers</title>
        <style type = "text/css">
           p, ol { margin: 0; }
           li    { display: inline; margin-right: 10px; }
        </style>
        <script>

           var value;

           document.writeln( "<p>Random Numbers</p><ol>" );

           for ( var i = 1; i <= 30; ++i )
           {
              value = Math.floor( 1 + Math.random() * 6 );
              document.writeln( "<li>" + value + "</li>" );
           } // end for

           document.writeln( "</ol>" );

        </script>
     </head><body></body>
  </html>
```



```
Random Numbers
    3 5 3 6 4 6 2 1 4 1 2 2 1 2 5
    1 2 4 3 1 4 2 6 5 1 6 4 3 1 4
```



```
Random Numbers
    5 2 3 4 2 2 4 3 3 1 5 3 3 2 3
    4 3 5 5 2 6 3 4 5 4 5 2 1 3 6
```

Display Random Images Example

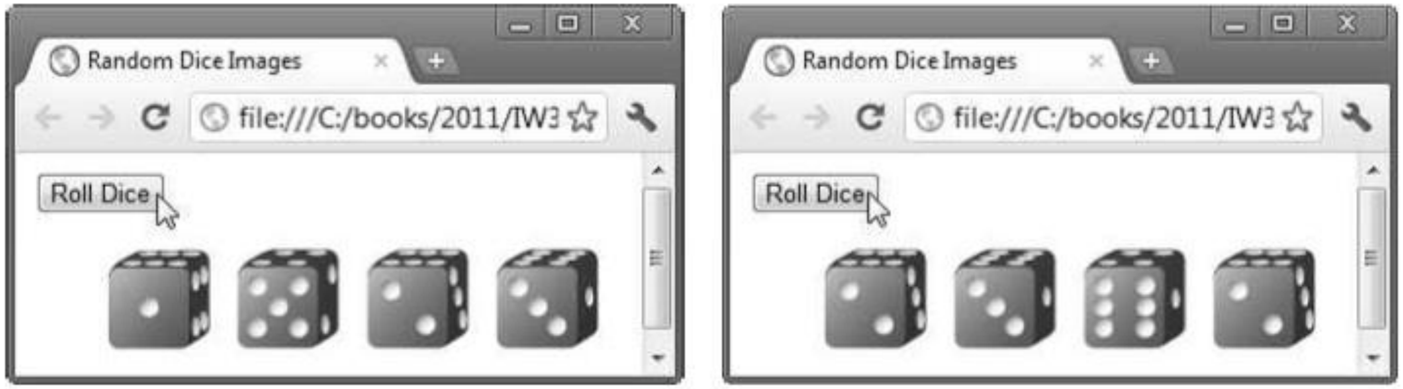
```
<!DOCTYPE html>

<!-- Fig. 9.5: RollDice.html -->
<!-- Random dice image generation using Math.random. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Random Dice Images</title>
      <style type = "text/css">
         li { display: inline; margin-right: 10px; }
         ul { margin: 0; }
      </style>
      <script>
         // variables used to interact with the i mg elements
         var die1Image;
         var die2Image;
         var die3Image;
         var die4Image;

         // register button listener and get the img elements
         function start()
         {
            var button = document.getElementById( "rollButton" );
            button.addEventListener( "click", rollDice, false  );
            die1Image = document.getElementById( "die1" );
            die2Image = document.getElementById( "die2" );
            die3Image = document.getElementById( "die3" );
            die4Image = document.getElementById( "die4" );
         } // end function rollDice

```

Display Random Images Example (2)

```
            // roll the dice
            function rollDice()
            {
               setImage( die1Image );
               setImage( die2Image );
               setImage( die3Image );
               setImage( die4Image );
            } // end function rollDice

            // set image source for a die
            function setImage( dieImg )
            {
               var dieValue = Math.floor( 1 + Math.random() * 6 );
               dieImg.setAttribute( "src", "die" + dieValue + ".png" );
               dieImg.setAttribute( "alt",
                  "die image with " + dieValue + " spot(s)" );
            } // end function setImage

            window.addEventListener( "load", start, false );
         </script>
      </head>
      <body>
         <form action = "#">
            <input id = "rollButton" type = "button" value = "Roll Dice">
         </form>
         <ol>
            <li><img id = "die1" src = "blank.png" alt = "die 1 image"></li>
            <li><img id = "die2" src = "blank.png" alt = "die 2 image"></li>
            <li><img id = "die3" src = "blank.png" alt = "die 3 image"></li>
```

Display Random Images Example (3)

```
                <li><img id = "die4" src = "blank.png" alt = "die 4 image"></li>
            </ol>
        </body>
    </html>
```



About events

- Typically basic interactions is achieved by the means of a dialog box, either an alert or a prompt,
- For more sophisticated interactions the use of GUIs and GUI even handling is recommended,
- Need to introduce a GUI type of input, e.g. **a button**, and associate it with a particular event through the **addEventListener** method creating an **event handler,**
- Method **addEventListener** is available for every DOM node and takes 3 arguments:
    - The name of the event for which the handler is registered,
    - The function to be called to handle the event,
    - Typically the value **false** except some very particular cases.
- The **window's load** event triggers function **start** to register the Roll Dice button's **click** event handler,
- The **getElementById method** finds the element with the matching id attribute and returns a JavaScript object representing the element.

Rolling Dice Frequencies (1)

```
<!DOCTYPE html>

<!-- Fig. 9.6: RollDice.html -->
<!-- Rolling 12 dice and displaying frequencies. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Die Rolling Frequencies</title>
      <style type = "text/css">
         img            { margin-right: 10px; }
        table        { width: 200px;
                       border-collapse: collapse;
                       background-color: lightblue; }
        table, td, th { border: 1px solid black;
                       padding: 4px;
                       margin-top: 20px; }
        th           { text-align: left;
                       color: white;
                       background-color: darkblue; }
      </style>
      <script>
         var frequency1 = 0;
         var frequency2 = 0;
         var frequency3 = 0;
         var frequency4 = 0;
         var frequency5 = 0;
         var frequency6 = 0;
         var totalDice = 0;

         // register button event handler
         function start()
         {
            var button = document.getElementById( "rollButton" );
            button.addEventListener( "click", rollDice, false );
         } // end function start

```

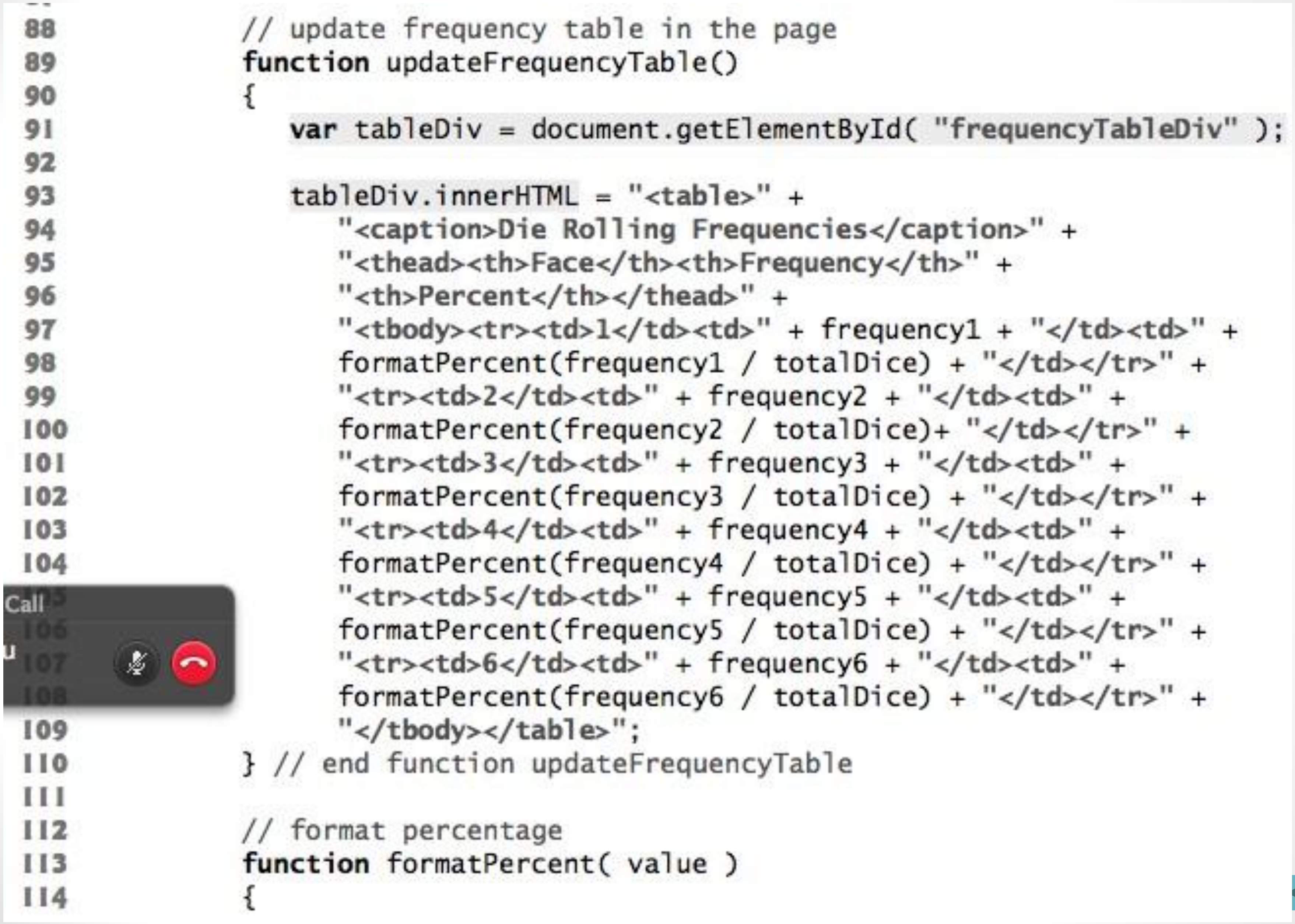
Rolling Dice Frequencies (2)

```
      // register button event handler
      function start()
      {
         var button = document.getElementById( "rollButton" );
         button.addEventListener( "click", rollDice, false );
      } // end function start

      // roll the dice
      function rollDice()
      {
         var face;  // face rolled

         // loop to roll die 12 times
         for ( var i = 1; i <= 12; ++i )
         {
            face = Math.floor( 1 + Math.random() * 6 );
            tallyRolls( face ); // increment a frequency counter
            setImage( i, face ); // display appropriate die image
            ++totalDice; // increment total
         } // end die rolling loop

         updateFrequencyTable();
      } // end function rollDice

      // increment appropriate frequency counter
      function tallyRolls( face )
      {
         switch ( face )
         {
            case 1:
               ++frequency1;
               break;
```

Rolling Dice Frequencies (3)

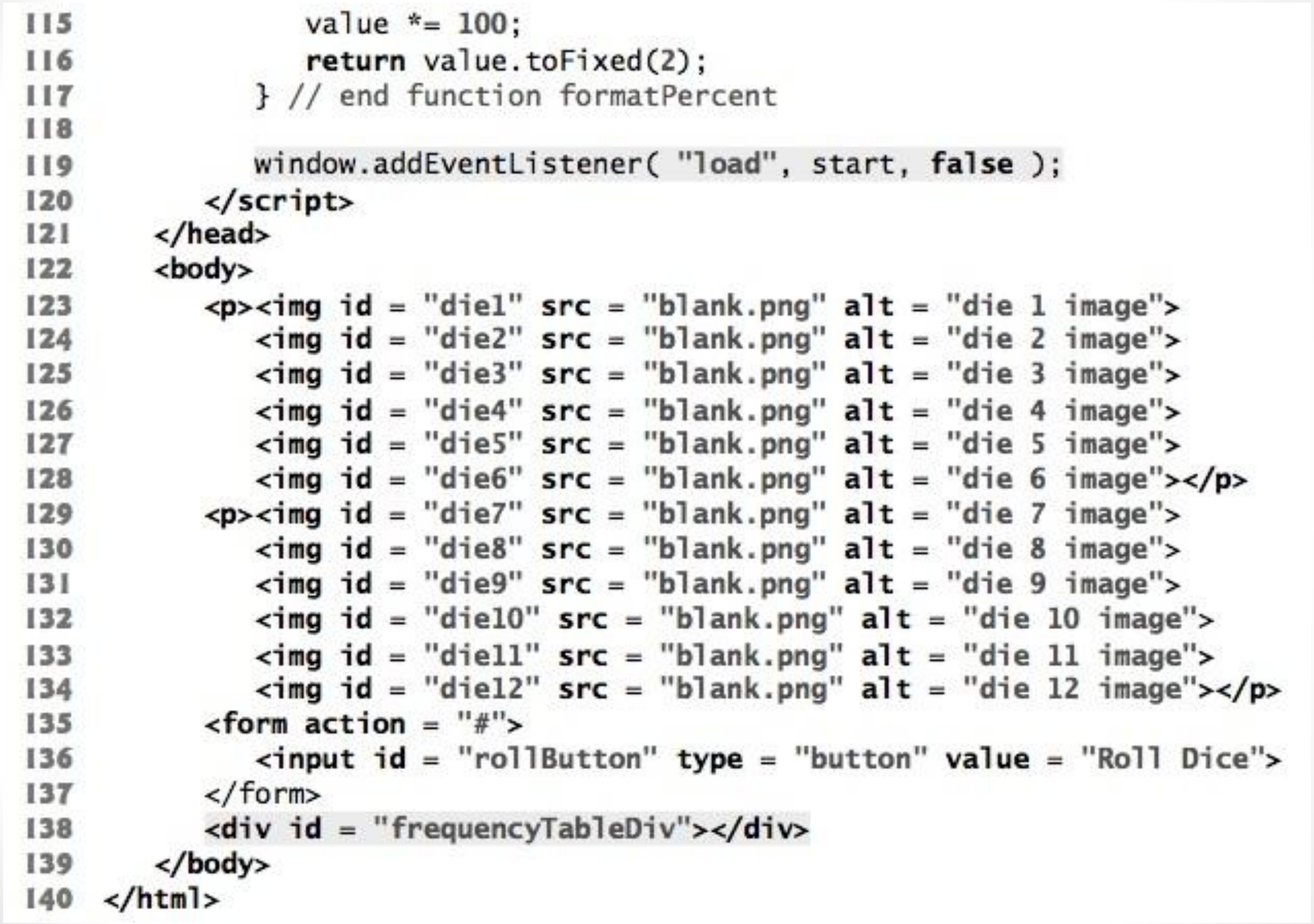
```
      case 2:
         ++frequency2;
         break;
      case 3:
         ++frequency3;
         break;
      case 4:
         ++frequency4;
         break;
      case 5:
         ++frequency5;
         break;
      case 6:
         ++frequency6;
         break;
   } // end switch
} // end function tallyRolls

// set image source for a die
function setImage( dieNumber, face )
{
   var dieImg = document.getElementById( "die" + dieNumber );
   dieImg.setAttribute( "src", "die" + face + ".png" );
   dieImg.setAttribute( "alt", "die with " + face + " spot(s)" );
} // end function setImage

```

Rolling Dice Frequencies (4)



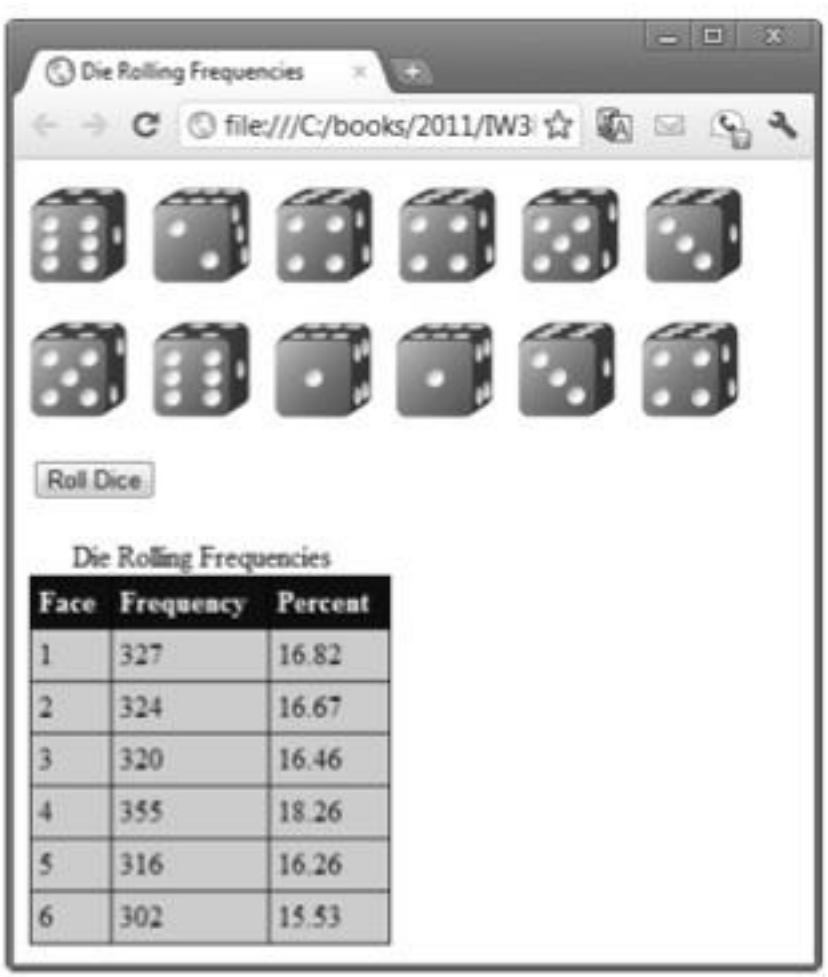
```
// update frequency table in the page
function updateFrequencyTable()
{
   var tableDiv = document.getElementById( "frequencyTableDiv" );

   tableDiv.innerHTML = "<table>" +
      "<caption>Die Rolling Frequencies</caption>" +
      "<thead><th>Face</th><th>Frequency</th>" +
      "<th>Percent</th></thead>" +
      "<tbody><tr><td>1</td><td>" + frequency1 + "</td><td>" +
      formatPercent(frequency1 / totalDice) + "</td></tr>" +
      "<tr><td>2</td><td>" + frequency2 + "</td><td>" +
      formatPercent(frequency2 / totalDice)+ "</td></tr>" +
      "<tr><td>3</td><td>" + frequency3 + "</td><td>" +
      formatPercent(frequency3 / totalDice) + "</td></tr>" +
      "<tr><td>4</td><td>" + frequency4 + "</td><td>" +
      formatPercent(frequency4 / totalDice) + "</td></tr>" +
      "<tr><td>5</td><td>" + frequency5 + "</td><td>" +
      formatPercent(frequency5 / totalDice) + "</td></tr>" +
      "<tr><td>6</td><td>" + frequency6 + "</td><td>" +
      formatPercent(frequency6 / totalDice) + "</td></tr>" +
      "</tbody></table>";
} // end function updateFrequencyTable

// format percentage
function formatPercent( value )
{
```

Rolling Dice Frequencies (5)



```
            value *= 100;
            return value.toFixed(2);
         } // end function formatPercent

         window.addEventListener( "load", start, false );
      </script>
   </head>
   <body>
      <p><img id = "die1" src = "blank.png" alt = "die 1 image">
         <img id = "die2" src = "blank.png" alt = "die 2 image">
         <img id = "die3" src = "blank.png" alt = "die 3 image">
         <img id = "die4" src = "blank.png" alt = "die 4 image">
         <img id = "die5" src = "blank.png" alt = "die 5 image">
         <img id = "die6" src = "blank.png" alt = "die 6 image"></p>
      <p><img id = "die7" src = "blank.png" alt = "die 7 image">
         <img id = "die8" src = "blank.png" alt = "die 8 image">
         <img id = "die9" src = "blank.png" alt = "die 9 image">
         <img id = "die10" src = "blank.png" alt = "die 10 image">
         <img id = "die11" src = "blank.png" alt = "die 11 image">
         <img id = "die12" src = "blank.png" alt = "die 12 image"></p>
      <form action = "#">
         <input id = "rollButton" type = "button" value = "Roll Dice">
      </form>
      <div id = "frequencyTableDiv"></div>
   </body>
</html>
```

Rolling Dice Frequencies (6)



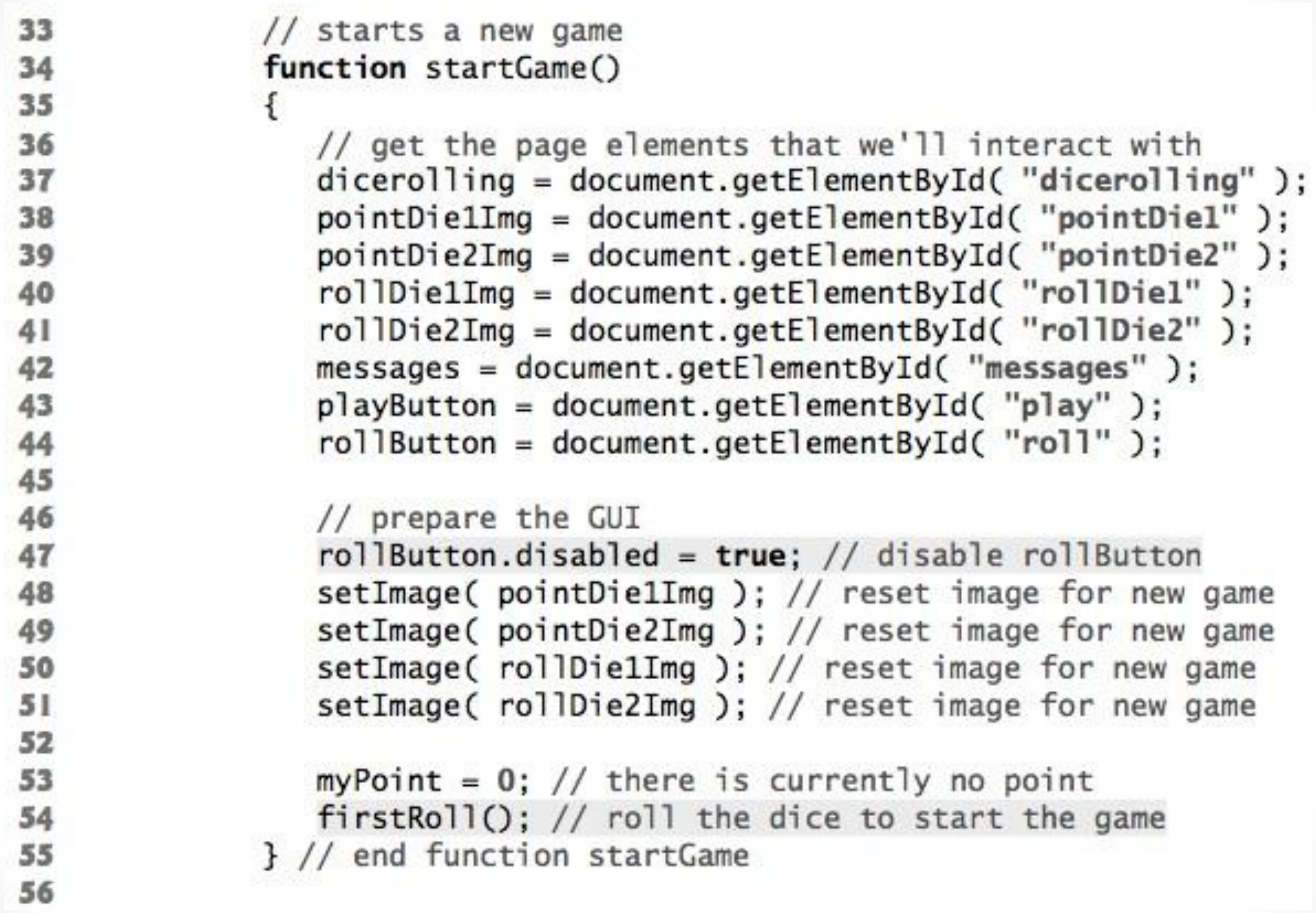
Die Rolling Frequencies

Face	Frequency	Percent
1	327	16.82
2	324	16.67
3	320	16.46
4	355	18.26
5	316	16.26
6	302	15.53

Example: Game of Chance

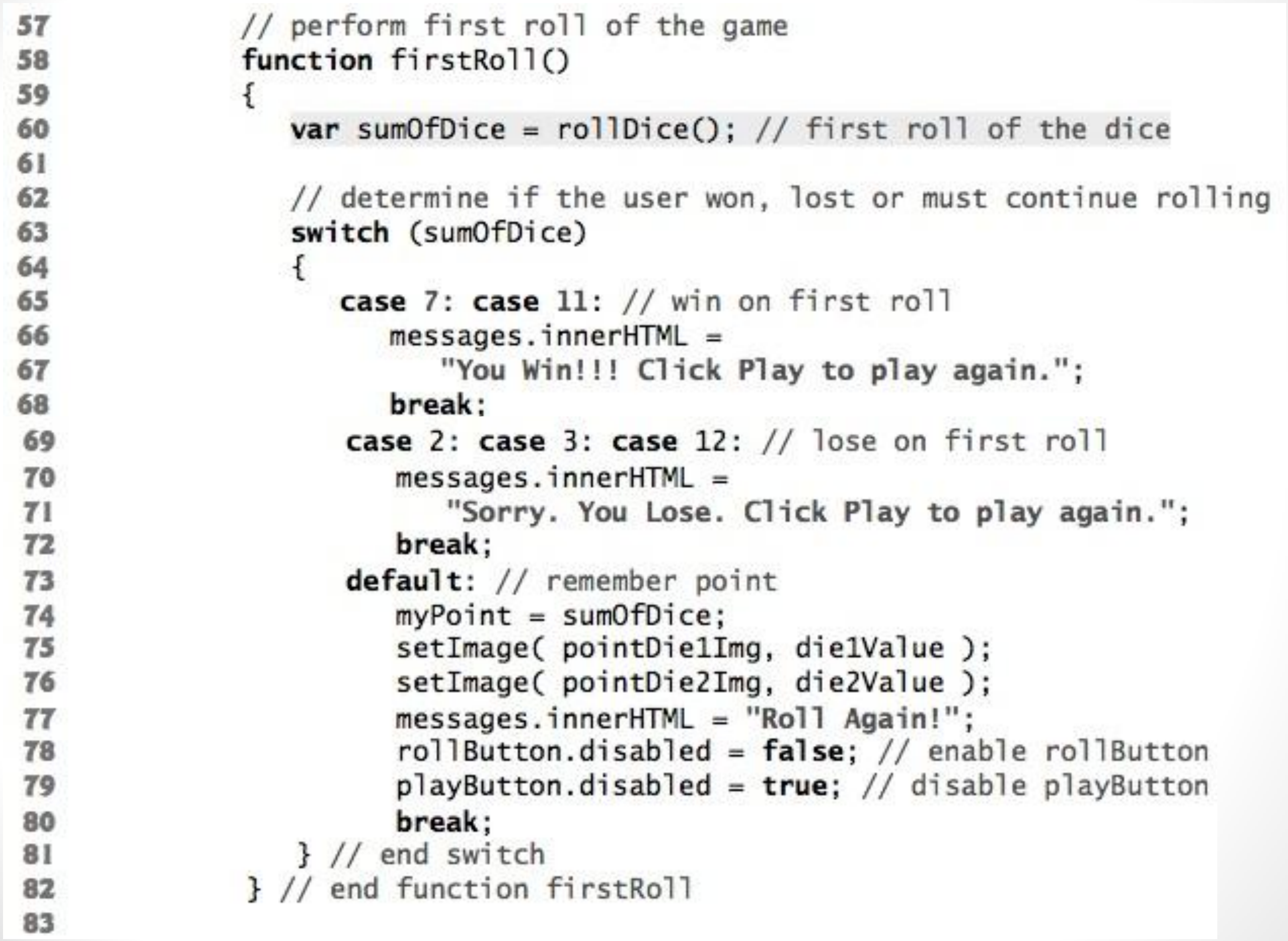
```
<!DOCTYPE html>

<!-- Fig. 9.7: Craps.html -->
<!-- Craps game simulation. -->
<html>
   <head>
      <meta charset = "utf-8">
      <title>Craps Game Simulation</title>
      <style type = "text/css">
         p.red  { color: red }
         img    { width: 54px; height: 54px; }
         div    { border: 5px ridge royalblue;
                  padding: 10px; width: 120px;
                  margin-bottom: 10px; }
         .point { margin: 0px; }
      </style>
      <script>
          // variables used to refer to page elements
          var pointDie1Img; // refers to first die point img
          var pointDie2Img; // refers to second die point img
          var rollDie1Img; // refers to first die roll img
          var rollDie2Img; // refers to second die roll img
          var messages; // refers to "messages" paragraph
          var playButton; // refers to Play button
          var rollButton; // refers to Roll button
          var dicerolling; // refers to audio clip for dice

          // other variables used in program
          var myPoint; // point if no win/loss on first roll
          var die1Value; // value of first die in current roll
          var die2Value; // value of second die in current roll

```

Example: Game of Chance (2)



```
      // starts a new game
      function startGame()
      {
         // get the page elements that we'll interact with
         dicerolling = document.getElementById( "dicerolling" );
         pointDie1Img = document.getElementById( "pointDie1" );
         pointDie2Img = document.getElementById( "pointDie2" );
         rollDie1Img = document.getElementById( "rollDie1" );
         rollDie2Img = document.getElementById( "rollDie2" );
         messages = document.getElementById( "messages" );
         playButton = document.getElementById( "play" );
         rollButton = document.getElementById( "roll" );

         // prepare the GUI
         rollButton.disabled = true; // disable rollButton
         setImage( pointDie1Img ); // reset image for new game
         setImage( pointDie2Img ); // reset image for new game
         setImage( rollDie1Img ); // reset image for new game
         setImage( rollDie2Img ); // reset image for new game

         myPoint = 0; // there is currently no point
         firstRoll(); // roll the dice to start the game
      } // end function startGame

```

Example: Game of Chance (3)

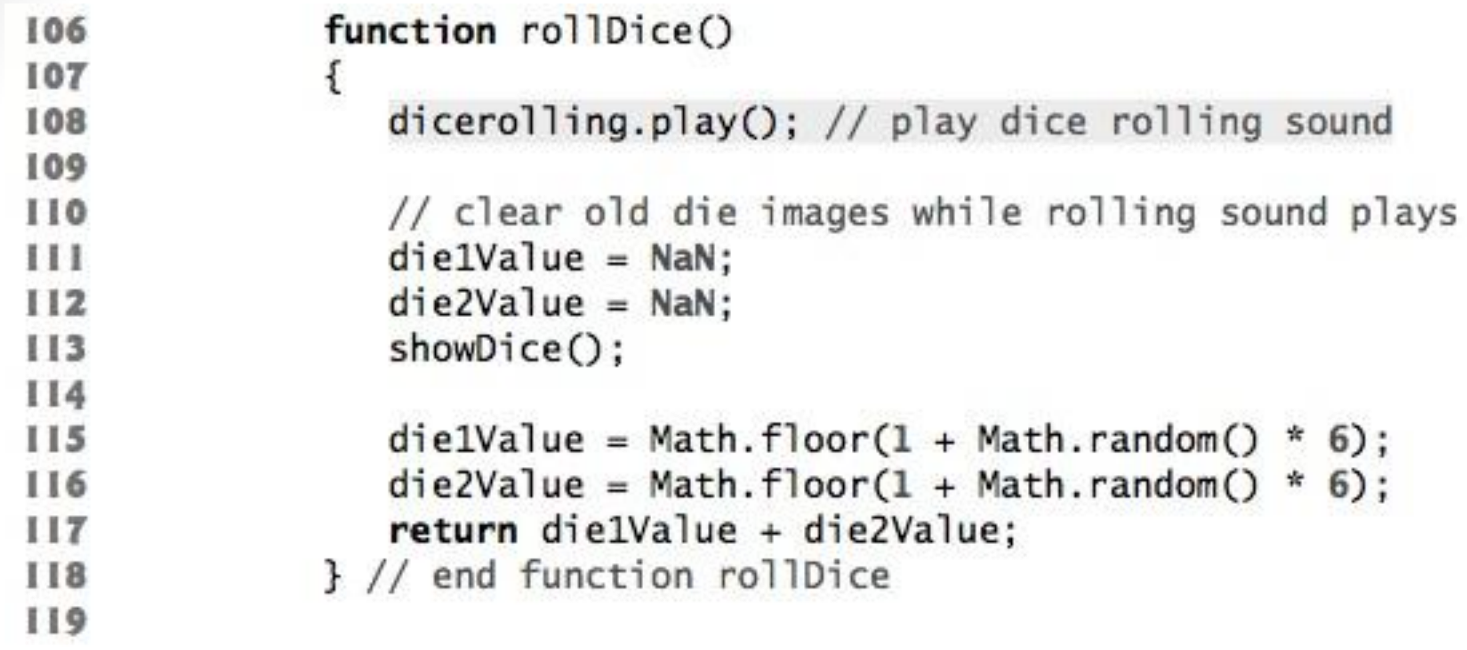


```
      // perform first roll of the game
      function firstRoll()
      {
         var sumOfDice = rollDice(); // first roll of the dice

         // determine if the user won, lost or must continue rolling
         switch (sumOfDice)
         {
            case 7: case 11: // win on first roll
               messages.innerHTML =
                  "You Win!!! Click Play to play again.";
               break;
            case 2: case 3: case 12: // lose on first roll
               messages.innerHTML =
                  "Sorry. You Lose. Click Play to play again.";
               break;
            default: // remember point
               myPoint = sumOfDice;
               setImage( pointDie1Img, die1Value );
               setImage( pointDie2Img, die2Value );
               messages.innerHTML = "Roll Again!";
               rollButton.disabled = false; // enable rollButton
               playButton.disabled = true; // disable playButton
               break;
         } // end switch
      } // end function firstRoll
```

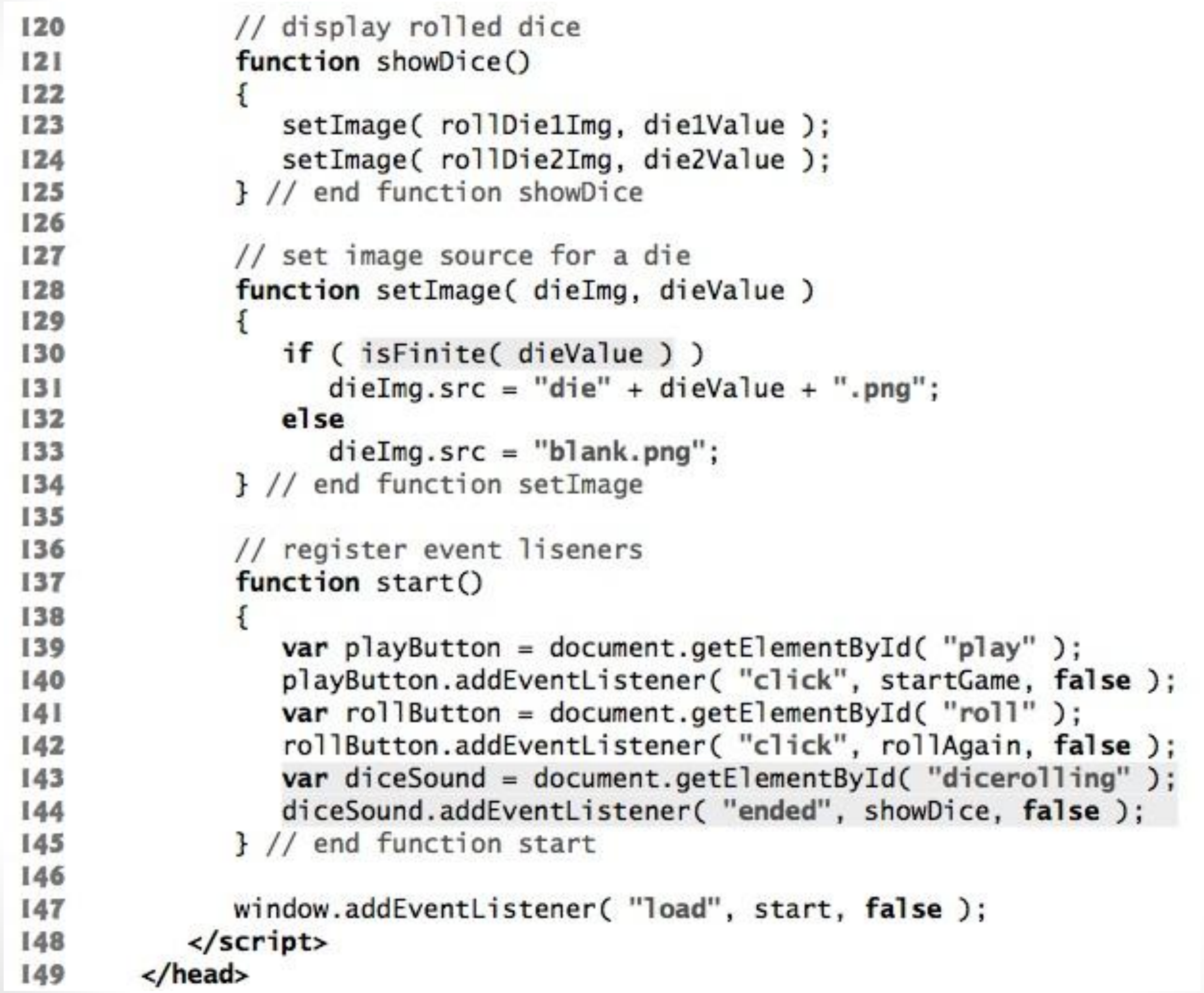
Example: Game of Chance (4)

```
// called for subsequent rolls of the dice
function rollAgain()
{
   var sumOfDice = rollDice(); // subsequent roll of the dice

   if (sumOfDice == myPoint)
   {
      messages.innerHTML =
         "You Win!!! Click Play to play again.";
      rollButton.disabled = true; // disable rollButton
      playButton.disabled = false; // enable playButton
   } // end if
   else if (sumOfDice == 7) // craps
   {
      messages.innerHTML =
         "Sorry. You Lose. Click Play to play again.";
      rollButton.disabled = true; // disable rollButton
      playButton.disabled = false; // enable playButton
   } // end else if
} // end function rollAgain

```

Example: Game of Chance (5)


```
function rollDice()
{
   dicerolling.play(); // play dice rolling sound

   // clear old die images while rolling sound plays
   die1Value = NaN;
   die2Value = NaN;
   showDice();

   die1Value = Math.floor(1 + Math.random() * 6);
   die2Value = Math.floor(1 + Math.random() * 6);
   return die1Value + die2Value;
} // end function rollDice
```

Example: Game of Chance (6)

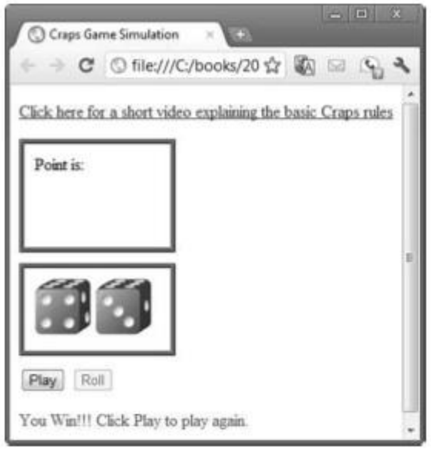


```
      // display rolled dice
      function showDice()
      {
         setImage( rollDie1Img, die1Value );
         setImage( rollDie2Img, die2Value );
      } // end function showDice

      // set image source for a die
      function setImage( dieImg, dieValue )
      {
         if ( isFinite( dieValue ) )
            dieImg.src = "die" + dieValue + ".png";
         else
            dieImg.src = "blank.png";
      } // end function setImage

      // register event liseners
      function start()
      {
         var playButton = document.getElementById( "play" );
         playButton.addEventListener( "click", startGame, false );
         var rollButton = document.getElementById( "roll" );
         rollButton.addEventListener( "click", rollAgain, false );
         var diceSound = document.getElementById( "dicerolling" );
         diceSound.addEventListener( "ended", showDice, false );
      } // end function start

      window.addEventListener( "load", start, false );
   </script>
</head>
```

Example: Game of Chance (7)

```
   <body>
      <audio id = "dicerolling" preload = "auto">
         <source src = "http://test.deitel.com/dicerolling.mp3"
            type = "audio/mpeg">
         <source src = "http://test.deitel.com/dicerolling.ogg"
            type = "audio/ogg">
         Browser does not support audio tag</audio>
      <p><a href = "CrapsRules.html">Click here for a short video
         explaining the basic Craps rules</a></p>
      <div id = "pointDiv">
         <p class = "point">Point is:</p>
         <img id = "pointDie1" src = "blank.png"
            alt = "Die 1 of Point Value">
         <img id = "pointDie2" src = "blank.png"
            alt = "Die 2 of Point Value">
      </div>
      <div class = "rollDiv">
         <img id = "rollDie1" src = "blank.png"
            alt = "Die 1 of Roll Value">
         <img id = "rollDie2" src = "blank.png"
            alt = "Die 2 of Roll Value">
      </div>
      <form action = "#">
         <input id = "play" type = "button" value = "Play">
         <input id = "roll" type = "button" value = "Roll">
      </form>
      <p id = "messages" class = "red">Click Play to start the game</p>
   </body>
</html>
```

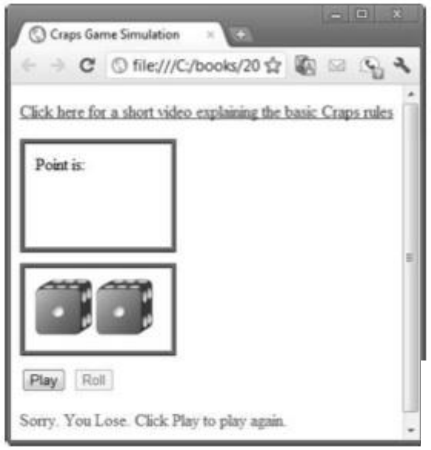
Example: Game of Chance (8)



Click here for a short video explaining the basic Craps rules

Point is:

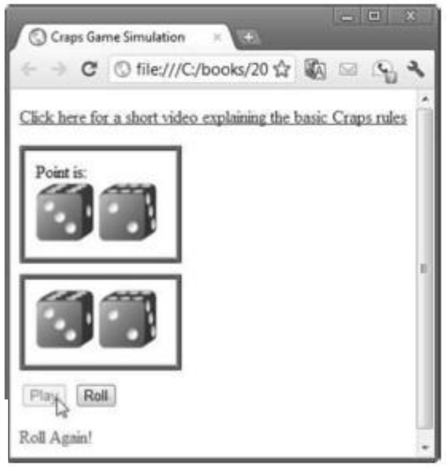
You Win!!! Click Play to play again.



Click here for a short video explaining the basic Craps rules

Point is:

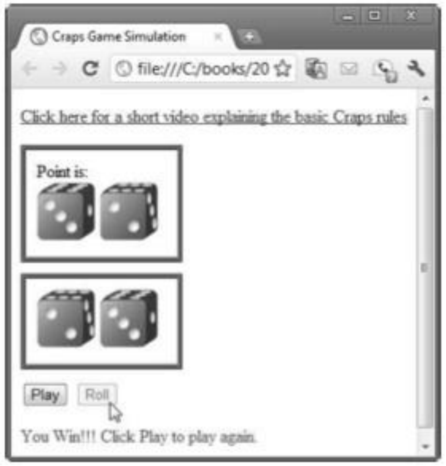
Sorry. You Lose. Click Play to play again.



Click here for a short video explaining the basic Craps rules

Point is:

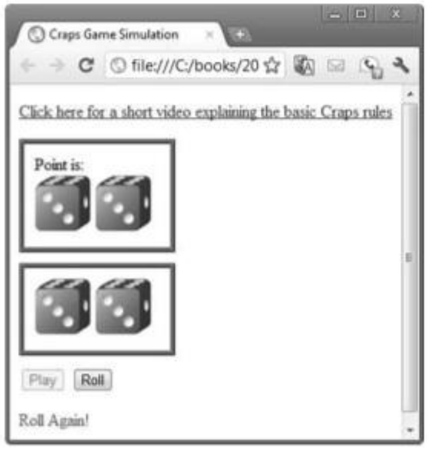
Roll Again!



Click here for a short video explaining the basic Craps rules

Point is:

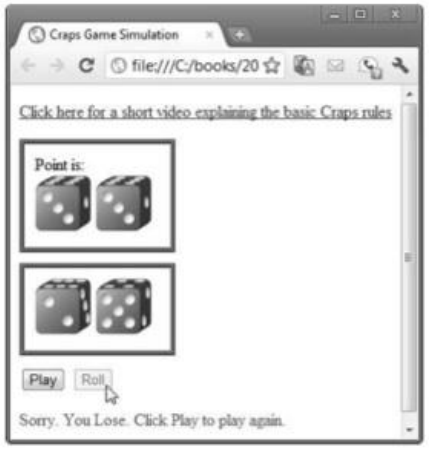
You Win!!! Click Play to play again.



Click here for a short video explaining the basic Craps rules

Point is:

Roll Again!



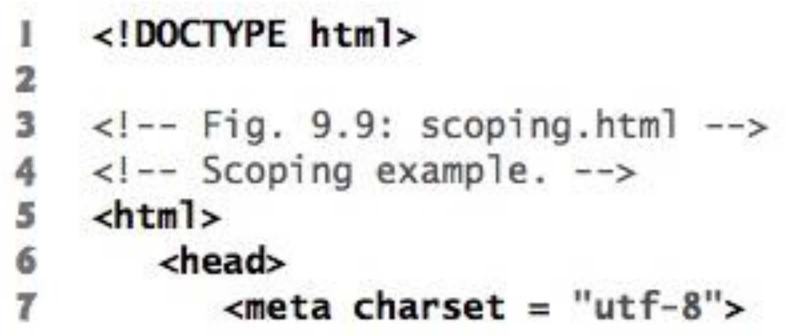
Click here for a short video explaining the basic Craps rules

Point is:

Sorry. You Lose. Click Play to play again.

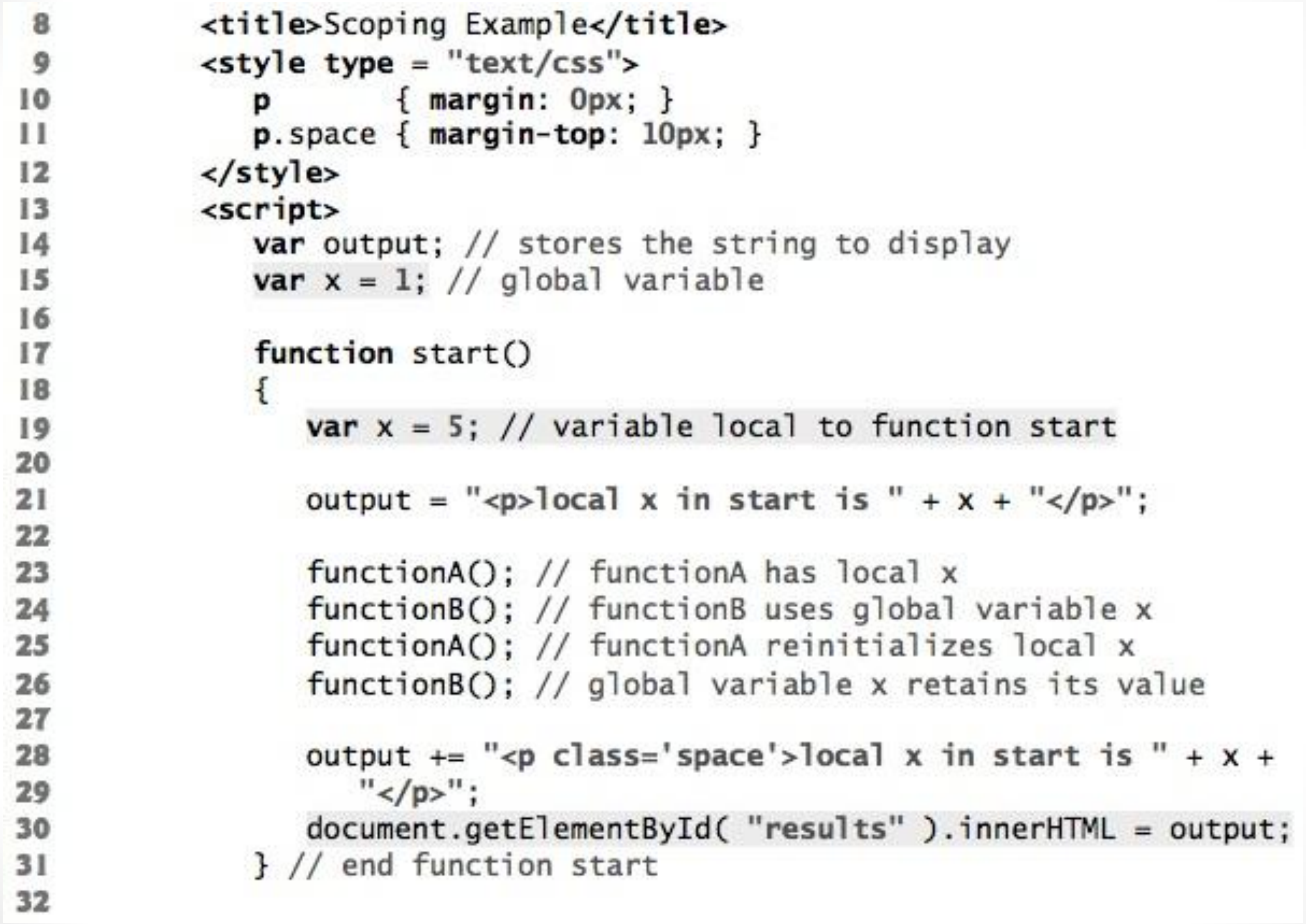
Scope rules

- The **scope** of an identifier for a variable or function is the portion of the program in which the identifier can be referenced,
- **Global variables or script-level variables:** declared in the head element and are accessible in any part of a script,
- **Function or local variables:** declared inside a function can be used only in that function,



```
<!DOCTYPE html>

<!-- Fig. 9.9: scoping.html -->
<!-- Scoping example. -->
<html>
   <head>
      <meta charset = "utf-8">
```

Scope rules (2)

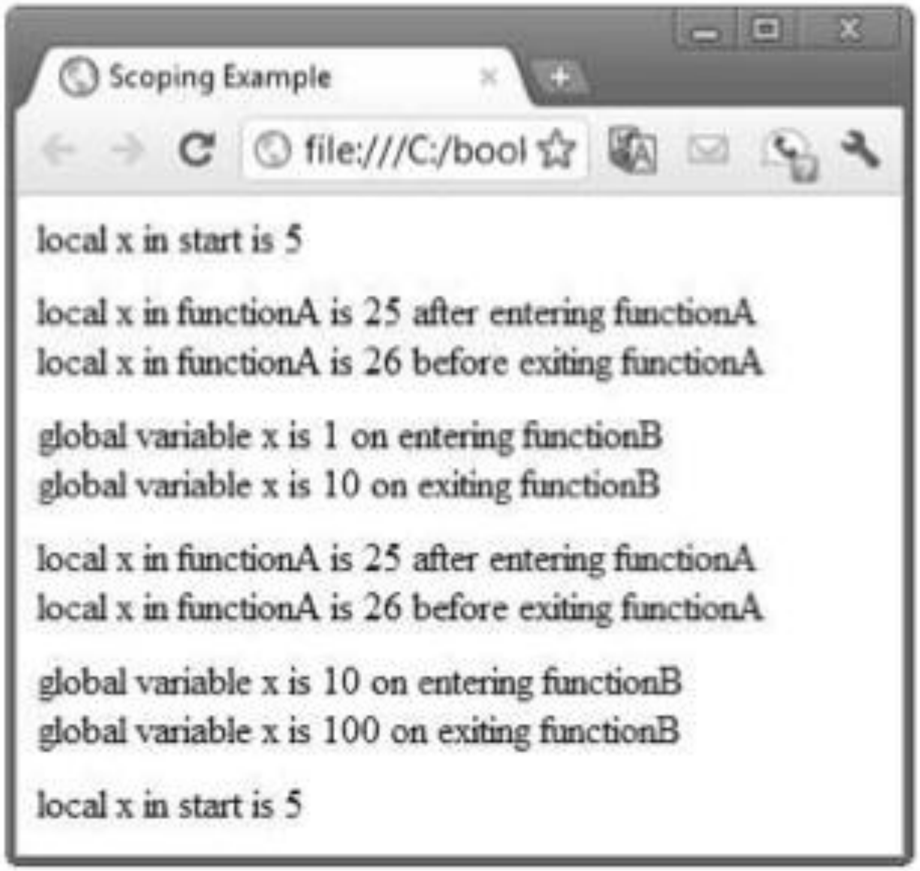


```
      <title>Scoping Example</title>
      <style type = "text/css">
         p       { margin: 0px; }
         p.space { margin-top: 10px; }
      </style>
      <script>
         var output; // stores the string to display
         var x = 1; // global variable

         function start()
         {
            var x = 5; // variable local to function start

            output = "<p>local x in start is " + x + "</p>";

            functionA(); // functionA has local x
            functionB(); // functionB uses global variable x
            functionA(); // functionA reinitializes local x
            functionB(); // global variable x retains its value

            output += "<p class='space'>local x in start is " + x +
               "</p>";
            document.getElementById( "results" ).innerHTML = output;
         } // end function start

```

Scope rules (3)

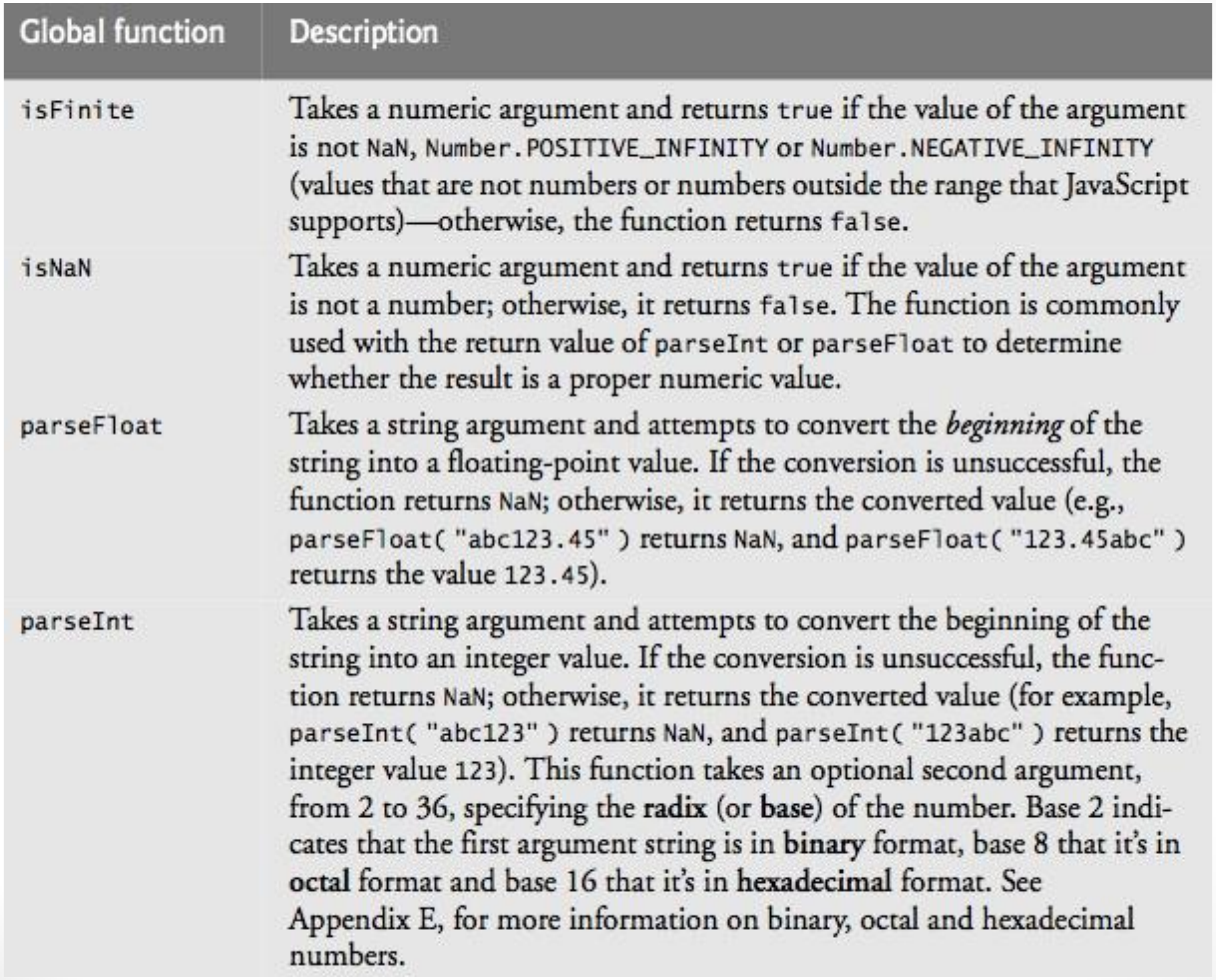
```
         function functionA()
         {
            var x = 25; // initialized each time functionA is called

            output += "<p class='space'>local x in functionA is " + x +
               " after entering functionA</p>";
            ++x;
            output += "<p>local x in functionA is " + x +
               " before exiting functionA</p>";
         } // end functionA

         function functionB()
         {
            output += "<p class='space'>global variable x is " + x +
               " on entering functionB";
            x *= 10;
            output += "<p>global variable x is " + x +
               " on exiting functionB</p>";
         } // end functionB

         window.addEventListener( "load", start, false );
      </script>
   </head>
   <body>
      <div id = "results"></div>
   </body>
</html>
```

Scope rules (4)



```
local x in start is 5

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 1 on entering functionB
global variable x is 10 on exiting functionB

local x in functionA is 25 after entering functionA
local x in functionA is 26 before exiting functionA

global variable x is 10 on entering functionB
global variable x is 100 on exiting functionB

local x in start is 5
```

JavaScript Global Functions



Global function	Description
`isFinite`	Takes a numeric argument and returns `true` if the value of the argument is not `NaN`, `Number.POSITIVE_INFINITY` or `Number.NEGATIVE_INFINITY` (values that are not numbers or numbers outside the range that JavaScript supports)—otherwise, the function returns `false`.
`isNaN`	Takes a numeric argument and returns `true` if the value of the argument is not a number; otherwise, it returns `false`. The function is commonly used with the return value of `parseInt` or `parseFloat` to determine whether the result is a proper numeric value.
`parseFloat`	Takes a string argument and attempts to convert the *beginning* of the string into a floating-point value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (e.g., `parseFloat( "abc123.45" )` returns `NaN`, and `parseFloat( "123.45abc" )` returns the value `123.45`).
`parseInt`	Takes a string argument and attempts to convert the beginning of the string into an integer value. If the conversion is unsuccessful, the function returns `NaN`; otherwise, it returns the converted value (for example, `parseInt( "abc123" )` returns `NaN`, and `parseInt( "123abc" )` returns the integer value `123`). This function takes an optional second argument, from 2 to 36, specifying the **radix** (or **base**) of the number. Base 2 indicates that the first argument string is in **binary** format, base 8 that it's in **octal** format and base 16 that it's in **hexadecimal** format. See Appendix E, for more information on binary, octal and hexadecimal numbers.