Functions

Outline

Objectives

- In this tutorial, you will learn:
  - To understand how to construct programs modularly from small pieces called functions.
  - To be able to create new functions.
  - To understand the mechanisms used to pass information between functions.
  - To introduce simulation techniques that use random-number generation.
  - To understand how the visibility of identifiers is limited to specific regions of programs.

10.1 Introduction

- Software design
  - Break software up into modules
    - Easier to maintain and debug
  - Divide and conquer

10.2 Program Modules in JavaScript

- Modules in JavaScript
  - Functions
  - Methods
    - Belong to an object
  - JavaScript includes many useful pre-defined methods
    - Combine with programmer-defined methods to make a program

10.2 Program Modules in JavaScript

- Functions
  - Started by function call
  - Receive necessary information via arguments (parameters)
  - Boss-Worker relationship
    - Calling function
    - Called function
    - Return value when finished
    - Can have many tiers

10.2 Program Modules in JavaScript

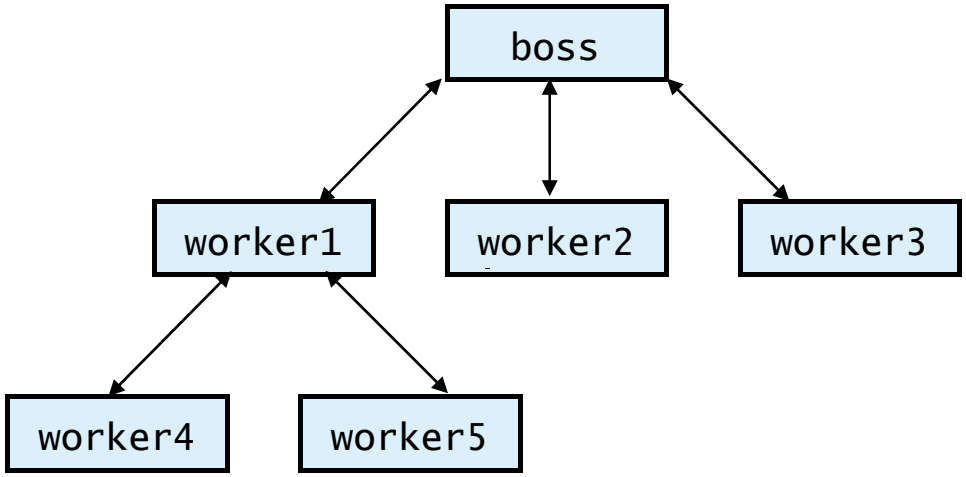


Fig. 10.1 Hierarchical boss-function/worker-function relationship.

10.2 Program Modules in JavaScript

- Function calls
  - Name
  - Left parenthesis
  - Arguments separated by commas
    - Constants, variables or expressions
  - Right parenthesis
  - Examples:

```
total += parseFloat( inputValue );

total += parseFloat( s1 + s2 );
```

10.3 Programmer-Defined Functions

- Defining functions
  - All variables declared in function are called local
    - Do not exist outside current function
  - Parameters
    - Also local variables
  - Promotes reusability
    - Keep short
    - Name clearly

10.4 Function Definitions

- Format of a function definition

```
function function-name( parameter-list )
{
    declarations and statements
}
```

  - Function name any valid identifier
  - Parameter list names of variables that will receive arguments
    - Must have same number as function call
    - May be empty
  - Declarations and statements
    - Function body (“block” of code)

10.4 Function Definitions

- Returning control
  - `return` statement
  - Can return either nothing, or a value

    `return` *expression*`;`
  - No return statement same as `return;`
  - Not returning a value when expected is an error

10.4 Function Definitions

- Writing a function to square two numbers
  - `for` loop from 1 to 10
  - Pass each number as argument to `square`
  - `return` value of argument multiplied by itself
  - Display result

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.2: SquareInt.html -->
6  <!-- Square function           -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>A Programmer-Defined square Function</title>
11
12       <script type = "text/javascript">
13          <!--
14          document.writeln(
15             "<h1>Square the numbe
16
17          // square the numbers from 1 to 10
18          for ( var x = 1; x <= 10; ++x )
19             document.writeln( "The square of " + x + " is " +
20                square( x ) + "<br />" );
21
```

Calling function `square` and passing it the value of `x`.

```
22          // The following square function's body is executed
23          // only when the functio
24
25          // square function definition
26          function square( y )
27          {
28             return y * y;
29          }
30          // -->
31       </script>
32
33    </head><body></body>
34 </html>
```

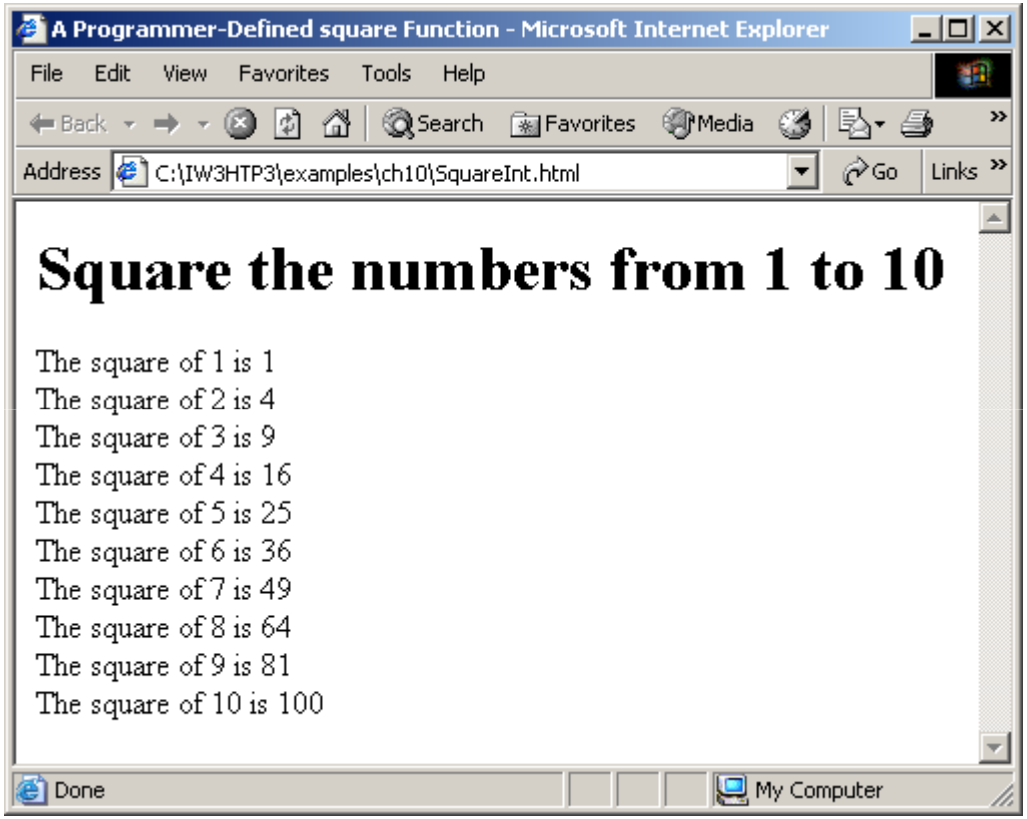
Variable y gets the value of variable x.

The `return` statement passes the value of `y * y` back to the calling function.

(2 of 2)

10.4 Function Definitions

Fig. 10.2 Using programmer-defined function `square`.



10.4 Function Definitions

- Finding the maximum of 3 numbers
  - Prompt for 3 inputs
  - Convert to numbers
  - Pass to maximum
  - `Math.max`

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.3: maximum.html -->
6  <!-- Maximum function        -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Finding the Maximum of Three Values</title>
11
12       <script type = "text/javascript">
13          <!--
14          var input1 =
15             window.prompt( "Enter first number", "0" );
16          var input2 =
17             window.prompt( "Enter second number", "0" );
18          var input3 =
19             window.prompt( "Enter third number", "0" );
20
21          var value1 = parseFloat( input1 );
22          var value2 = parseFloat( input2 );
23          var value3 = parseFloat( input3 );
```

Prompt for the user to input three integers.

```
24
25          var maxValue = maximum( value1, value2, value3 );
26
27          document.writeln( "First number: "
28             "<br />Second number: " + value
29             "<br />Third number: " + value3 +
30             "<br />Maximum is: " + maxValue
31
32          // maximum method definition (called from line 25)
33          function maximum( x, y, z )
34          {
35             return Math.max( x, Math.max
36          }
37          // -->
38       </script>
39
40    </head>
41    <body>
42       <p>Click Refresh (or Reload) to run the script again</p>
43    </body>
44 </html>
```

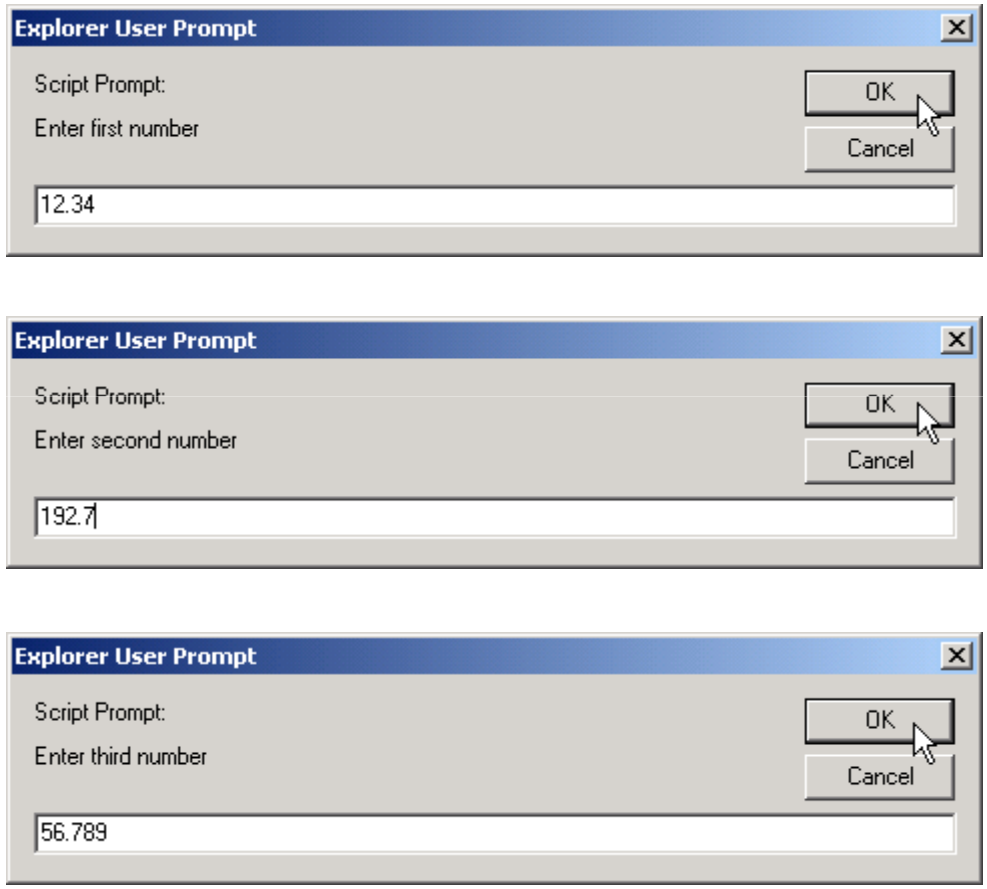
Call function `maximum` and pass it the value of variables `value1`, `value2` and `value3`.

Method `max` returns the larger of the two integers passed to it.

Variables `x`, `y` and `z` get the value of variables `value1`, `value2` and `value3`, respectively.

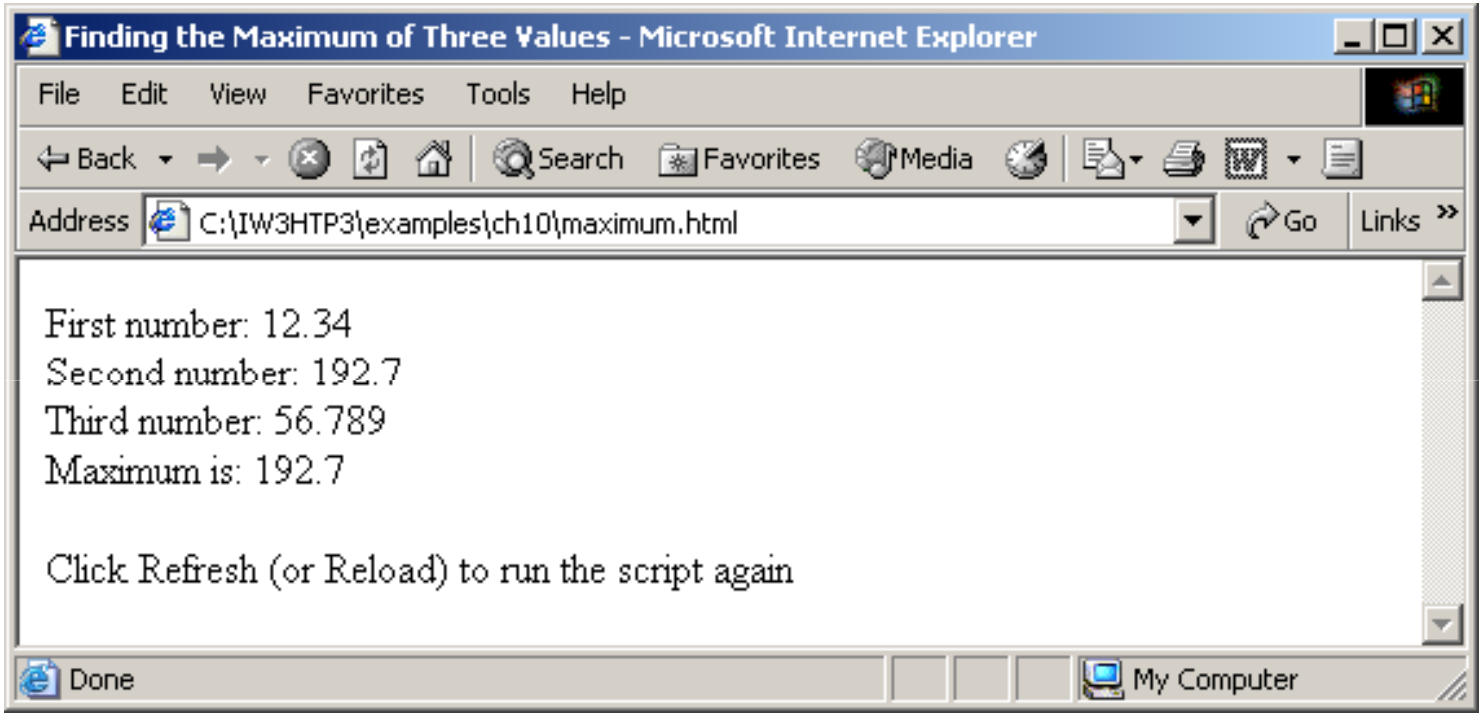
10.4 Function Definitions

Fig. 10.3 Programmer-defined `maximum` function (1 of 2).



10.4 Function Definitions

Fig. 10.3 Programmer-defined `maximum` function (2 of 2).



10.5 Random-Number Generation

- Random-number generation introduces element of chance
  - `Math.random`

    ```
    var randomValue = Math.random();
    ```
  - Floating point value between 0 and 1
  - Adjust range by scaling and shifting
  - Math.floor
    - Always round down

    ```
    Math.floor( 1 + Math.random() * 6 )
    ```

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.4: RandomInt.html       -->
6  <!-- Demonstrating the Random method -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Shifted and Scaled Random Integers</title>
11
12       <script type = "text/javascript">
13          <!--
14          var value;
15
16          document.writeln(
17             "<table border = \"1\" width = \"50%\">" );
18          document.writeln(
19             "<caption>Random Numbers</caption><tr>" );
20
```

```
21          for ( var i = 1; i <= 20; i++ ) {
22             value = Math.floor( 1 + Math.random() * 6 );
23             document.writeln( "<td>" + value + "</td>" );
24
25             // write end and start <tr> tags when
26             // i is a multiple of 5 and not 20
27             & i != 20 )
28             ln( "</tr><tr>" );
29
30
31          document.writeln( "</tr></table>" );
32          // -->
33       </script>
34
35    </head>
36    <body>
37       <p>Click Refresh (or Reload) to run the script again</p>
38    </body>
39 </html>
```

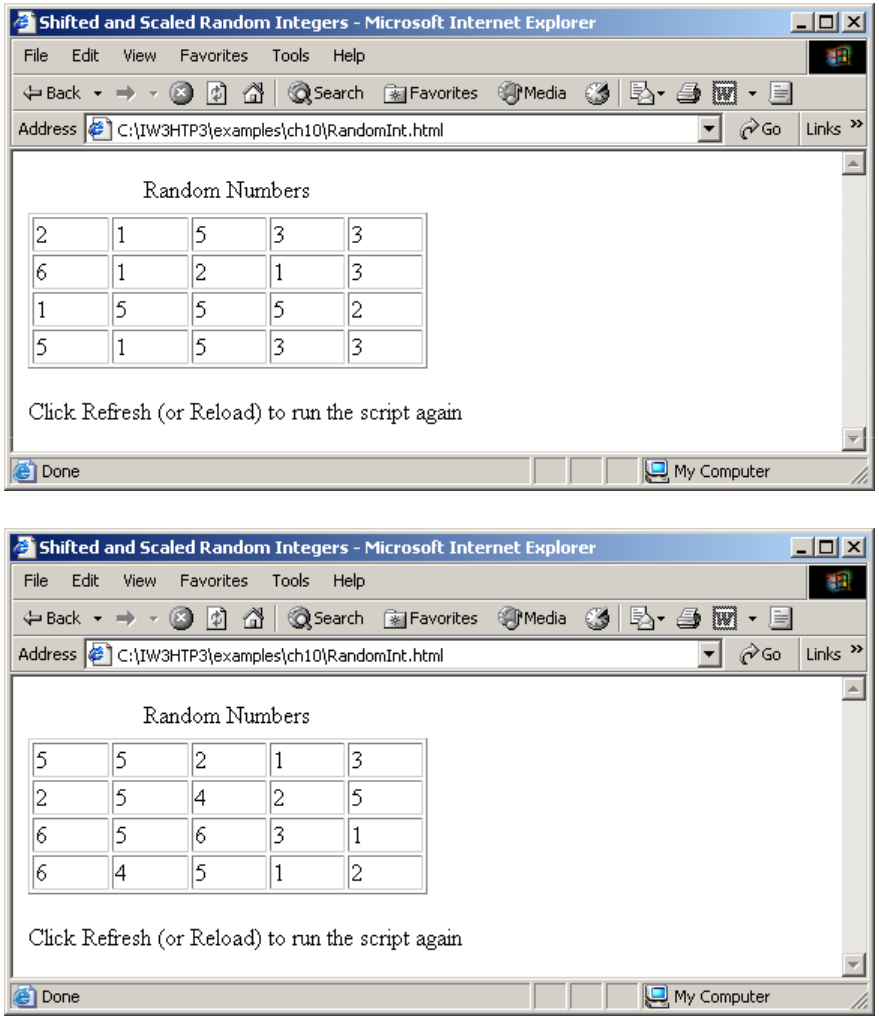
Method `floor` rounds the number generated by method `random` down.

Each cell is populated with a random number generated by method `random`.

The `for` loop creates 20 table cells (4 rows x 5 columns).

10.5 Random-Number Generation

Fig. 10.4 • Random integers, shifting and scaling.



```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.5: RollDie.html -->
6  <!-- Rolling a Six-Sided Die -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Roll a Six-Sided Die 6000 Times</title>
11
12       <script type = "text/javascript">
13          <!--
14          var frequency1 = 0, frequency2 =
15              frequency3 = 0, frequency4 = 0,
16              frequency5 = 0, frequency6 = 0, face;
17
18          // summarize results
19          for ( var roll = 1; roll <= 6000; ++roll ) {
20             face = Math.floor( 1 + Math.random() * 6 );
21
```

This expression uses method `random` to generate a random number between 1 and 6.

```
22            switch ( face ) {
23               case 1:
24                  ++frequency1;
25                  break;
26               case 2:
27                  ++frequency2;
28                  break;
29               case 3:
30                  ++frequency3;
31                  break;
32               case 4:
33                  ++frequency4;
34                  break;
35               case 5:
36                  ++frequency5;
37                  break;
38               case 6:
39                  ++frequency6;
40                  break;
41            }
42         }
43
```

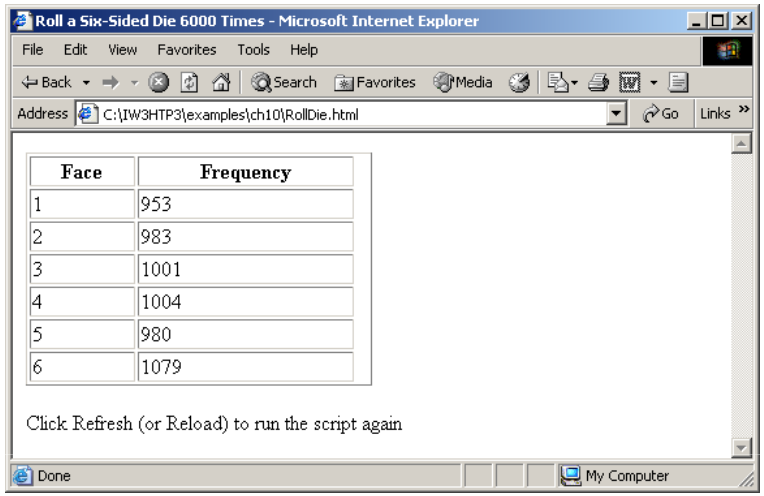
When the controlling expression, `face`, matches a `case` label, the respective `frequency` variable is incremented.

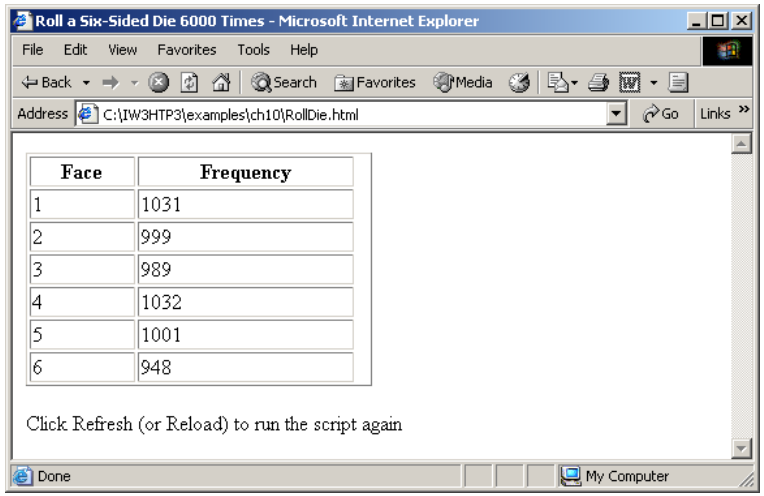
```
44          document.writeln( "<table border = \"1\"" +
45             "width = \"50%\">" );
46          document.writeln( "<thead><th>Face</th>" +
47             "<th>Frequency<th></thead>" );
48          document.writeln( "<tbody><tr><td>1</td><td>" +
49             frequency1 + "</td></tr>" );
50          document.writeln( "<tr><td>2</td><td>" + frequency2 +
51             "</td></tr>" );
52          document.writeln( "<tr><td>3</td><td>" + frequency3 +
53             "</td></tr>" );
54          document.writeln( "<tr><td>4</td><td>" + frequency4 +
55             "</td></tr>" );
56          document.writeln( "<tr><td>5</td><td>" + frequency5 +
57             "</td></tr>" );
58          document.writeln( "<tr><td>6</td><td>" + frequency6 +
59             "</td></tr></tbody></table>" );
60          // -->
61       </script>
62
63    </head>
64    <body>
65       <p>Click Refresh (or Reload) to run the script again</p>
66    </body>
67 </html>
```

The results of rolling the die 600 times are displayed in a table.

10.5 Random-Number Generation

Fig. 10.5 • Rolling a six-sided die 6000 times.





10.6 Example: Game of Chance

- Craps
  - Click **Roll Dice**
  - Text fields show rolls, sum and point
  - Status bar displays results

10.6 Example: Game of Chance

- Uses XHTML forms
  - Gather multiple inputs at once
  - Empty `action` attribute
  - `name` attribute allows scripts to interact with form
- Event handling and event-driven programming
  - Assign a function to an event
  - `onclick`
- Constants
  - Variable that cannot be modified
  - Part of many languages, not supported in JavaScript
    - Name "constant" variables with all capital letters
  - Make values easier to remember/change

10.6 Example: Game of Chance

- Changing properties
  - Access with dot (`.`) notation
  - `value` property of text fields
  - `status` property of `window`

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
4
5  <!-- Fig. 10.6: Craps.html -->
6  <!-- Craps Program         -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Program that Simulates the Game of Craps</title>
11
12       <script type = "text/javascript">
13          <!--
14          // variables used to test the state of the game
15          var WON = 0, LOST = 1, CONTINUE_ROLLING = 2;
16
17          // other variables used in program
18          var firstRoll = true,            // true if first roll
19              sumOfDice = 0,               // sum of the dice
20              myPoint = 0, // point if no win/loss on first roll
21              gameStatus = CONTINUE_ROLLING; // game not over yet
22
```

```
23          // process one roll of the dice
24          function play()
25          {
26             if ( firstRoll ) {          // f
27                sumOfDice = rollDice();
28
29                switch ( sumOfDice ) {
30                   case 7: case 11:       // win on fi
31                      gameStatus = WON;
32                      // clear point field
33                      document.craps.point.value = "";
34                      break;
35                   case 2: case 3: case 12: // lose on first roll
36                      gameStatus = LOST;
37                      // clear point field
38                      document.craps.point.value =
39                      break;
40                   default:                 // re
41                      gameStatus = CONTINUE_ROLLING;
42                      myPoint = sumOfDice;
43                      document.craps.point.value = myPoint;
44                      firstRoll = false;
45                }
46             }
```

If the value of `firstRoll` is `true`, then function `rollDice` is called.

If function `rollDice` returns a value of `7` or `11`, the player wins and the `break` statement causes program control proceeds to the first line after the `switch` structure.

If function `rollDice` returns a `2`, `3` or `12`, the player loses and the `break` statement causes control to proceed to first line after the switch structure.

```
47            else {
48               sumOfDice = rollDice();
49
50               if ( sumOfDice == myPoint ) // win by making point
51                  gameStatus = WON;
52               else
53                  if ( sumOfDice == 7 )    //
54                     gameStatus = LOST;
55            }
56
57            if ( gameStatus == CONTINUE_ROLLING )
58               window.status = "Roll again";
59            else {
60               if ( gameStatus == WON )
61                  window.status = "Player wins.
62                     "Click Roll Dice to play ag
63               else
64                  window.status = "Player loses. " +
65                     "Click Roll Dice to play again.";
66
67               firstRoll = true;
68            }
69         }
70
```

If the value of `firstRoll` is `false`, function `rollDice` is called to see if the point has been reached.

If the values returned by function `rollDice` equals 7, the player loses.

If the value returned by function `rollDice` equals the value of variable `myPoint`, the player wins because the point has been reached.

`window` method `status` displays a message in the status bar of the browser.

```
71          // roll the dice
72          function rollDice()
73          {
74             var die1, die2, workSum;
75
76             die1 = Math.floor( 1 +
77             die2 = Math.floor( 1 + Math.random() * 6 );
78             workSum = die1 + die2;
79
80             document.craps.firstDie.value = d
81             document.craps.secondDie.value =
82             document.craps.sum.value = workSum;
83
84             return workSum;
85          }
86          // -->
87       </script>
88
89    </head>
```

Function `rollDice` is called to simulate the rolling of two dice on the craps table.

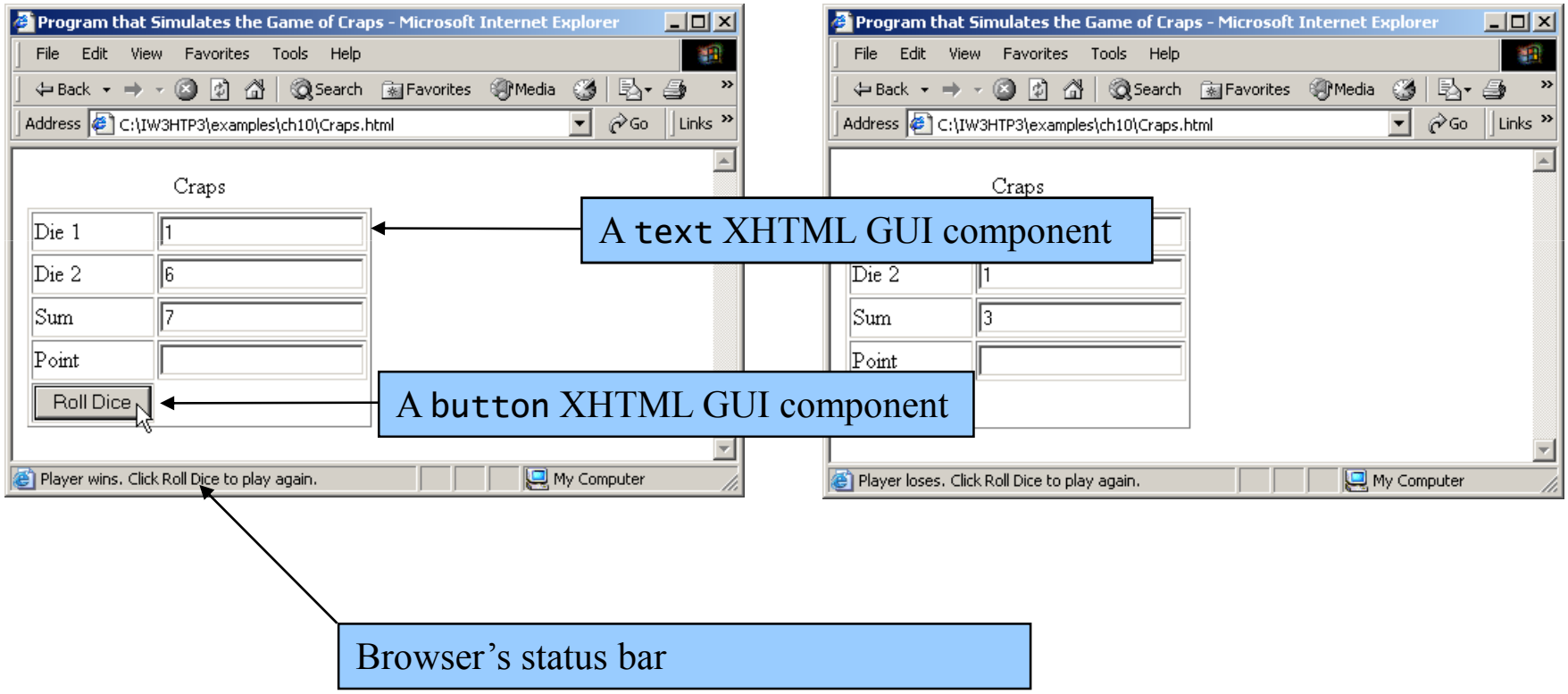
Methods `random` and `floor` are used to generate the values for the two dice.

Referencing the names of form elements in the XHTML document, the values of the dice are placed in their respective form fields.

```
90     <body>
91        <form name = "craps" action = "">
92           <table border = "1">
93           <caption>Craps</caption>
94           <tr><td>Die 1</td>
95               <td><input name = "firstDie" type = "text" />
96               </td></tr>
97           <tr><td>Die 2</td>
98               <td><input name = "secondDie" type = "text" />
99               </td></tr>
100          <tr><td>Sum</td>
101              <td><input name = "sum" type = "text" />
102              </td></tr>
103          <tr><td>Point</td>
104              <td><input name = "point" type = "text" />
105              </td></tr>
106          <tr><td><input type = "button" value = "Roll Dice"
107              onclick = "play()" /></td></tr>
108          </table>
109       </form>
110    </body>
111 </html>
```

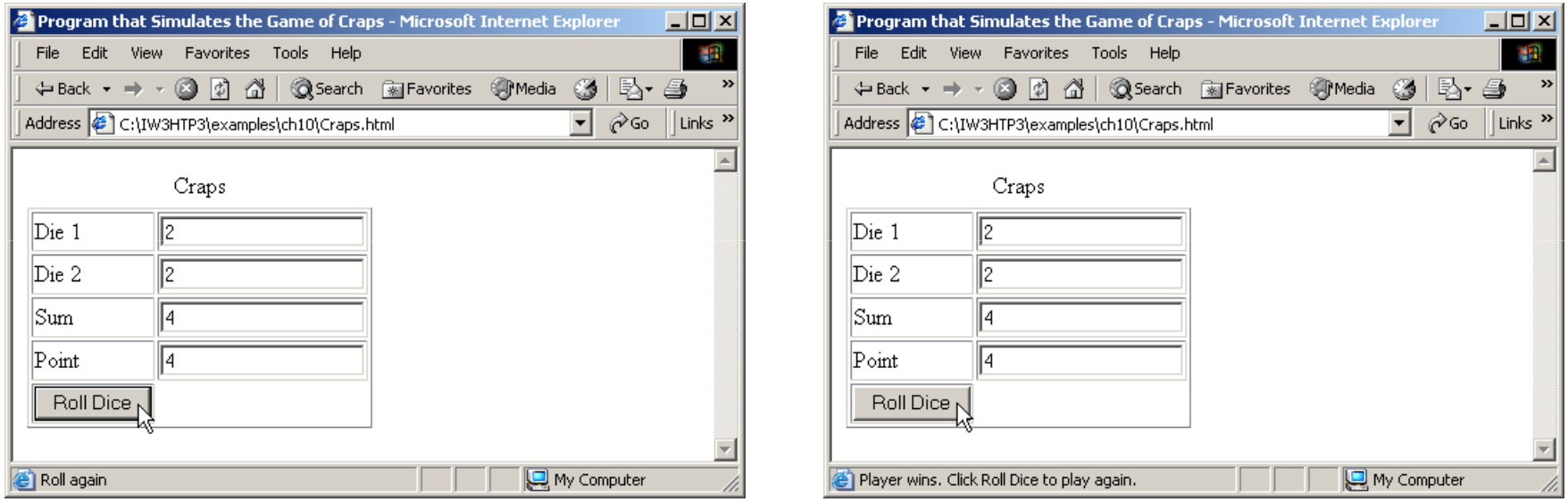
10.6 Example: Game of Chance

Fig. 10.6 Craps game simulation.



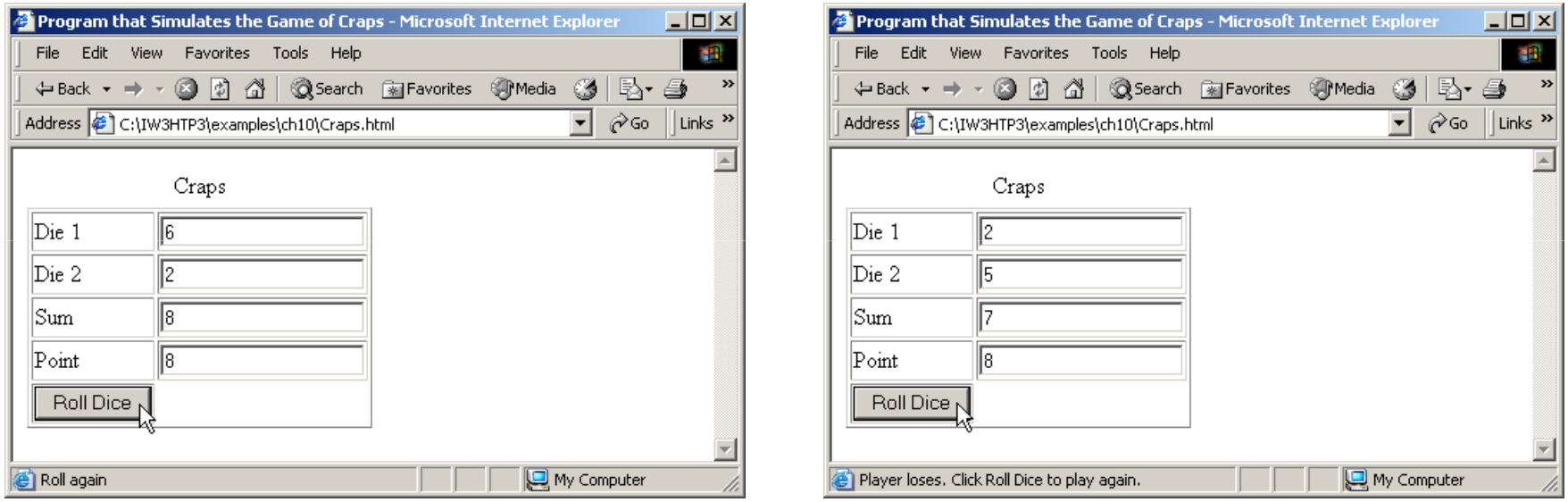
10.6 Example: Game of Chance

Fig. 10.6 Craps game simulation.



10.6 Example: Game of Chance

Fig. 10.6 Craps game simulation.



10.7 Another Example: Random Image Generator

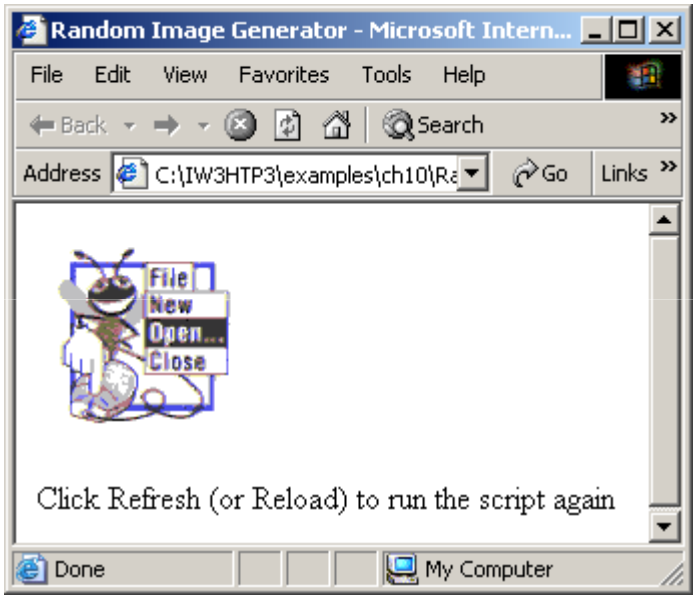
- Randomly selecting an image
  - Images have integer names (i.e., `1.gif`, `2.gif`, …, `7.gif`)
  - Generate random number in proper range
  - Update `src` property

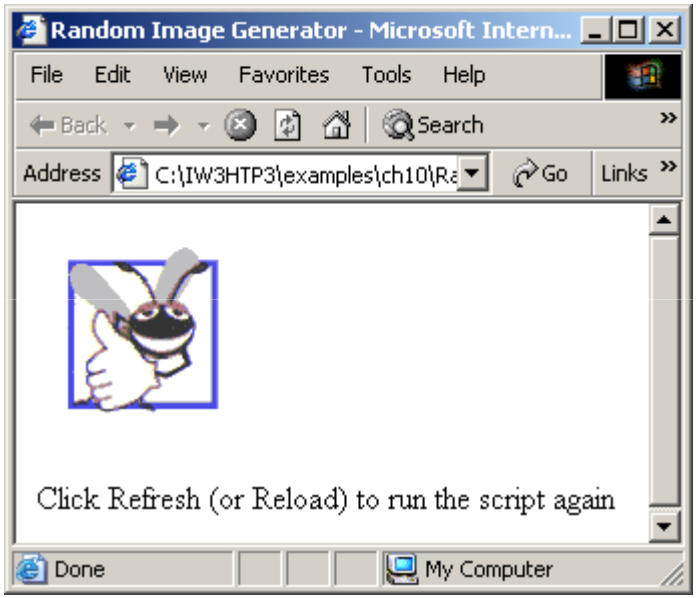
```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
3     "http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">
4
5  <!-- Fig. 10.7: RandomPicture.html     -->
6  <!-- Randomly displays one of 7 images -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Random Image Generator</tit
11
12       <script type = "text/javascript">
13          <!--
14          document.write ( "<img src = \"" +
15             Math.floor( 1 + Math.random() * 7 ) +
16             ".gif\" width = \"105\" height = \"100\" />" );
17          // -->
18       </script>
19
20    </head>
21
22    <body>
23       <p>Click Refresh (or Reload) to run the script again</p>
24    </body>
25 </html>
```

Inserting a random number into the image's `src` property with `document.write` and `Math.random`

10.7 Another Example: Random Image Generator

Fig. 10.7 Random image generation using Math.random.





10.8 Scope Rules

- Scope
  - Portion of program where identifier can be referenced
  - Inside function is local or function scope
    - Identifiers exist only between opening and closing braces
    - Local variables hide global variables

10.8 Scope Rules

- Scope demonstration
  - Global variable `x` initialized to `1`
  - `start` has local variable `x` initialized to `5`
  - `functionA` has local variable `x` initialized to `25`
  - `functionB` has no local variable `x`
  - Observe output of each function

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.8: scoping.html    -->
6  <!-- Local and Global Variables -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>A Scoping Example</title>
11
12       <script type = "text/javascript">
13          <!--
14          var x = 1;       // global variable
15
16          function start()
17          {
18             var x = 5;  // variable local to function start
19
20             document.writeln( "local x in start is " + x );
21
22             functionA(); // functionA has local x
23             functionB(); // functionB uses global variable x
24             functionA(); // functionA reinitializes local x
25             functionB(); // global variable x retains its value
```

To begin the program, variable x is initialized to 1.

Function `start` changes the value of x to 5.

```
26
27          document.writeln(
28             "<p>local x in start is " + x + "</p>" );
29       }
30
31       function functionA()
32       {
33          var x = 25;  // initialized each time
34                       // functionA is called
35
36          document.writeln( "<p>loca
37                            x + " after entering functionA" );
38          ++x;
39          document.writeln( "<br />local x in functionA is " +
40             x + " b                                        );
41       }
42
```

Function `functionA` changes the value of `x` to 25.

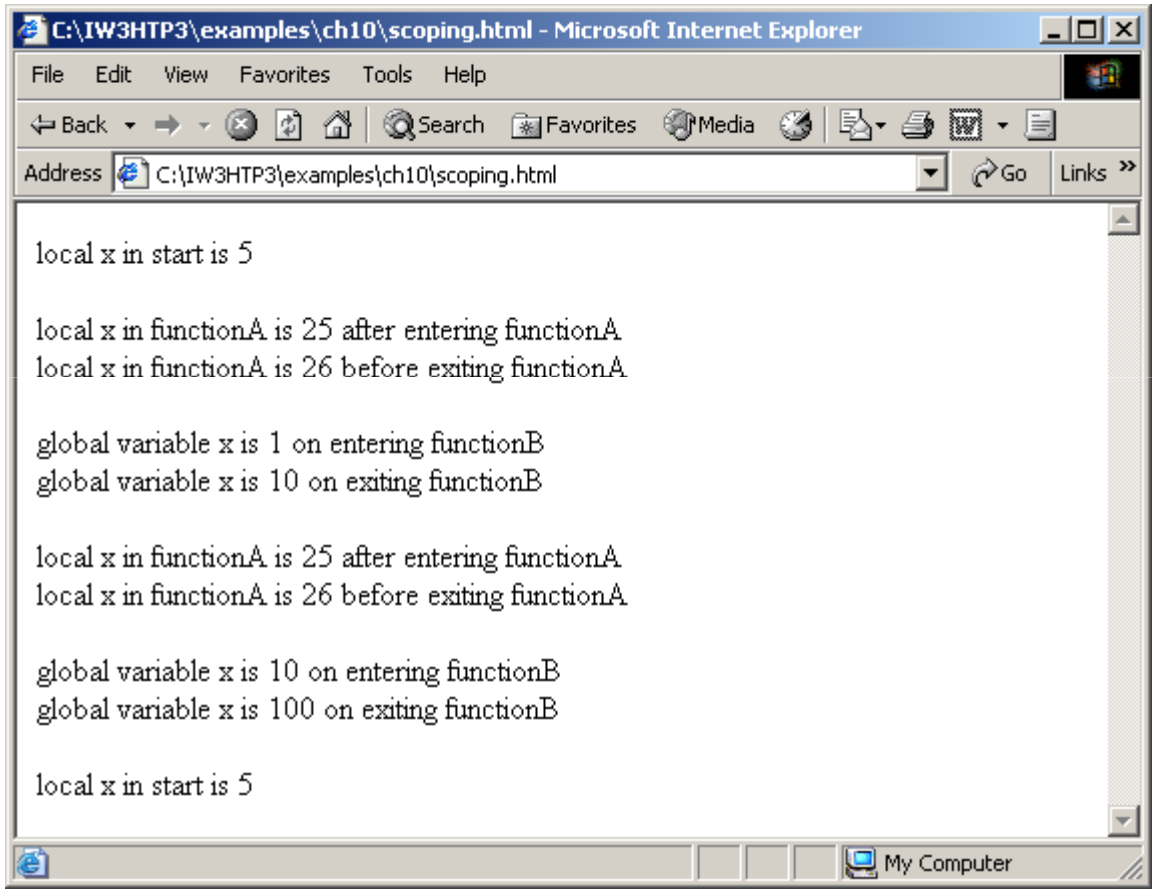
The value of `x` is incremented.

```
43          function functionB()
44          {
45             document.writeln( "<p>global variable x is " + x +
46                " on entering functionB" );
47             x *= 10;
48             document.writeln( "<br />global variable x is " +
49                x + " on exiting functionB" + "</p>" );
50          }
51          // -->
52       </script>
53
54    </head>
55    <body onload = "start()"></body>
56 </html>
```

Function `functionB` multiplies the value of `x` by 10.

10.8 Scope Rules

Fig. 10.8 Scoping example.



10.9 JavaScript Global Functions

- `Global` object
  - Always available
  - Provides 7 methods
  - Do not need to explicitly reference `Global` before method call
  - Also holds all global variables, user defined functions

10.9 JavaScript Global Functions

Global function	Description
`escape`	This function takes a string argument and returns a string in which all spaces, punctuation, accent characters and any other character that is not in the ASCII character set (see Appendix D, ASCII Character Set) are encoded in a hexadecimal format (see Appendix E, Number Systems) that can be represented on all platforms.
`eval`	This function takes a string argument representing JavaScript code to execute. The JavaScript interpreter evaluates the code and executes it when the `eval` function is called. This function allows JavaScript code to be stored as strings and executed dynamically.
`isFinite`	This function takes a numeric argument and returns `true` if the value of the argument is not `NaN`, `Number.POSITIVE_INFINITY` or `Number.NEGATIVE_INFINITY`; otherwise, the function returns `false`.
`isNaN`	This function 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.

Fig. 10.9 JavaScript global functions.

10.9 JavaScript Global Functions

Global function	Description
`parseFloat`	This function 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`	This function 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 (e.g., `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 indicates that the first argument string is in **octal** format and base 16 indicates that the first argument string is in **hexadecimal** format. See see Appendex E, Number Systems, for more information on binary, octal and hexadecimal numbers.
`unescape`	This function takes a string as its argument and returns a string in which all characters previously encoded with `escape` are decoded.

Fig. 10.9 JavaScript global functions.

10.10 Recursion

- Recursive functions
  - Call themselves
    - Recursion step or recursive call
    - Part of `return` statement
  - Must have base case
    - Simplest case of problem
    - Returns value rather than calling itself
  - Each recursive call simplifies input
    - When simplified to base case, functions return

10.10 Recursion

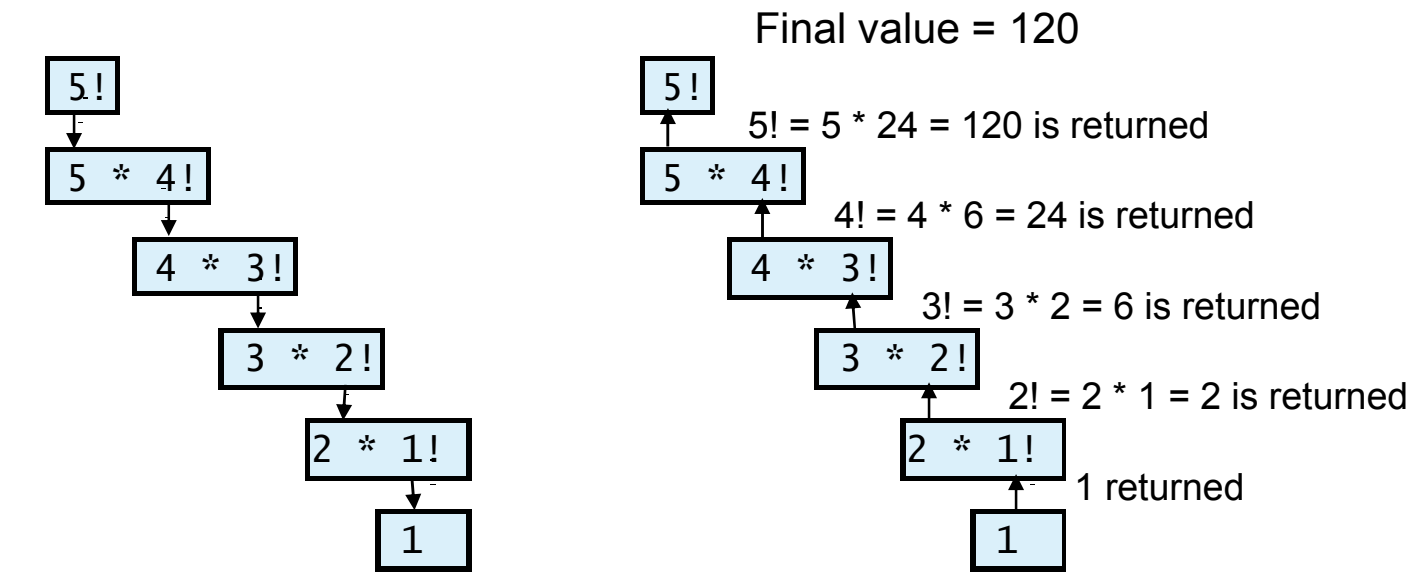
- Factorials
  - Product of calculation $n \cdot (n - 1) \cdot (n - 2) \cdot \ldots \cdot 1$
  - Iterative approach:

```
var factorial = 1;

for ( var counter = number; counter >= 1; --counter )
   factorial *= counter;
```

  - Note each factor is one less than previous factor
    - Stops at 1: base case
    - Perfect candidate for recursive solution

10.10 Recursion



(a) Procession of recursive calls. (b) Values returned from each recursive call.

Fig. 10.10 Recursive evaluation of 5!.

```
1  <?xml version = "1.0"?>
2  <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3     "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5  <!-- Fig. 10.11: FactorialTest.html -->
6  <!-- Recursive factorial example    -->
7
8  <html xmlns = "http://www.w3.org/1999/xhtml">
9     <head>
10       <title>Recursive Factorial Function</title>
11
12       <script language = "javascript">
13          document.writeln( "<h1>Factorials of
14          document.writeln(
15             "<table border = '1' width = '100%'>" );
16
17          for ( var i = 0; i <= 10; i++ )
18             document.writeln( "<tr><td>" + i + "!</td><td>" +
19                         factorial( i ) + "</td></tr>" );
20
21          document.writeln( "</table>" );
22
```

Calling function `factorial` and passing it the value of `i`.

```
23          // Recursive definition of function factorial
24          function factorial( number )
25          {
26             if ( number <= 1 )  // base c
27                return 1;
28             else
29                return number * factorial( number - 1 );
30          }
31       </script>
32    </head><body></body>
33 </html>
```

Variable `number` gets the value of variable `i`.

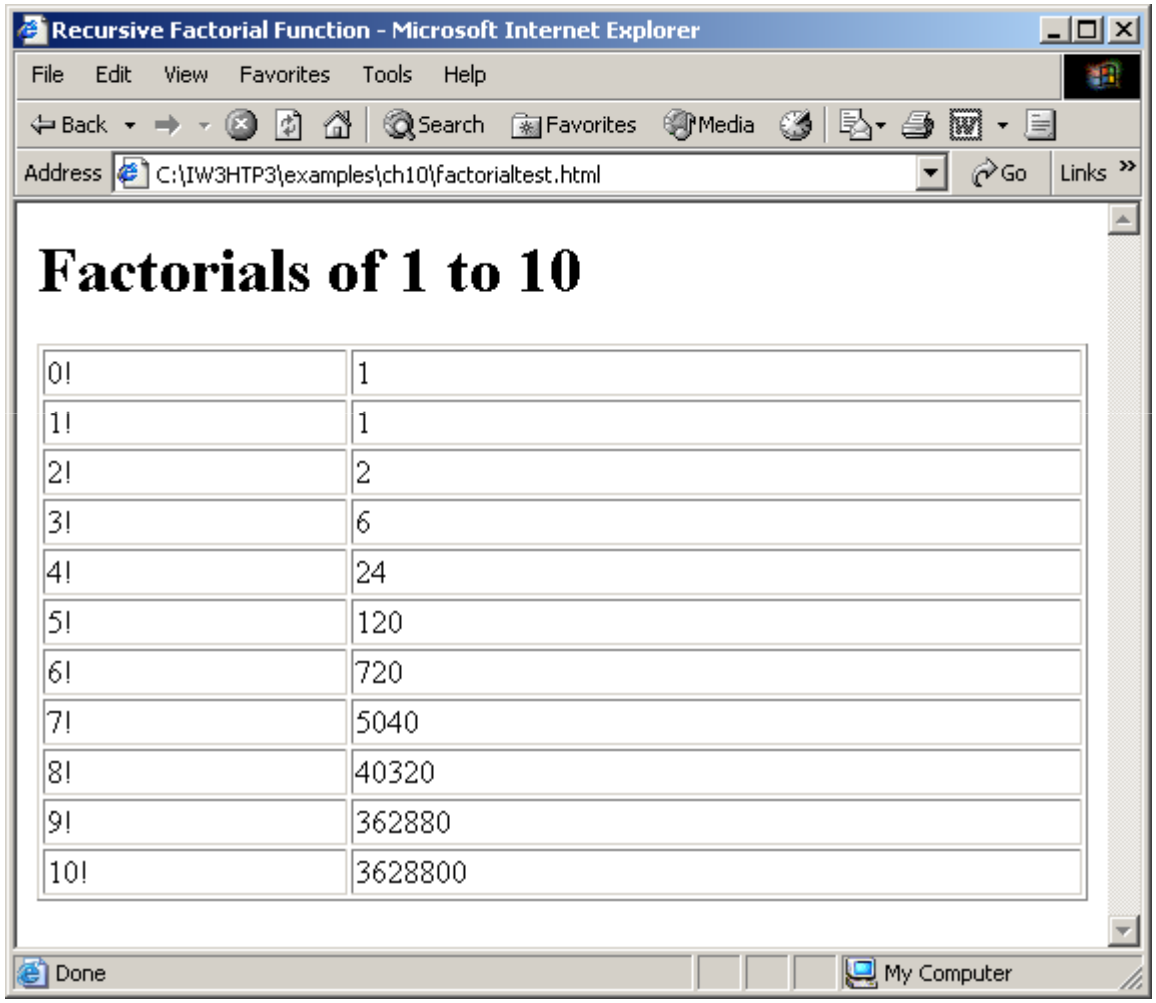
Call to function `factorial` and passing it `1` less than the current value of `number` .

factorial.html

(2 of 2)

10.10 Recursion

Fig. 10.11 Factorial calculation with a recursive function.



10.11 Recursion vs. Iteration

- Iteration
  - Explicitly uses repetition structures to achieve result
  - Terminates when loop-continuation condition fails
  - Often faster than recursion
- Recursion
  - Repeats through function calls
  - Terminates when base case reached
  - Slower due to function call overhead
    - Each call generates new copy of local variables
  - Easy to read and debug when modeling problem with naturally recursive solution

10.11 Recursion vs. Iteration

Chapter	Recursion examples and exercises
10	Factorial function Sum of two integers Multiply two integers Raising an integer to an integer power Visualizing recursion
12	Printing a string input at the keyboard backward
13	Navigating the object hierarchy in Dynamic HTML

Fig. 10.12 Recursion examples and exercises in the text.