Chapter 10 - JavaScript: Functions

Outline

10.2 Program Modules in JavaScript

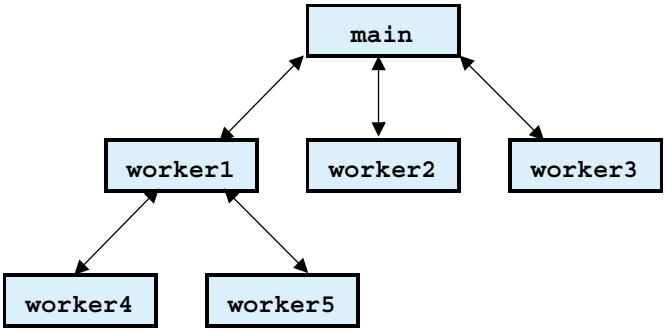
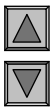


Fig. 10.1 Hierarchical boss-function/worker-function relationship.



Outline

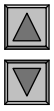
SquareInt.html

```
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 numbers from 1 to 10</h1>" );
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
22           // The following square
23           // only when the function
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>
```

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

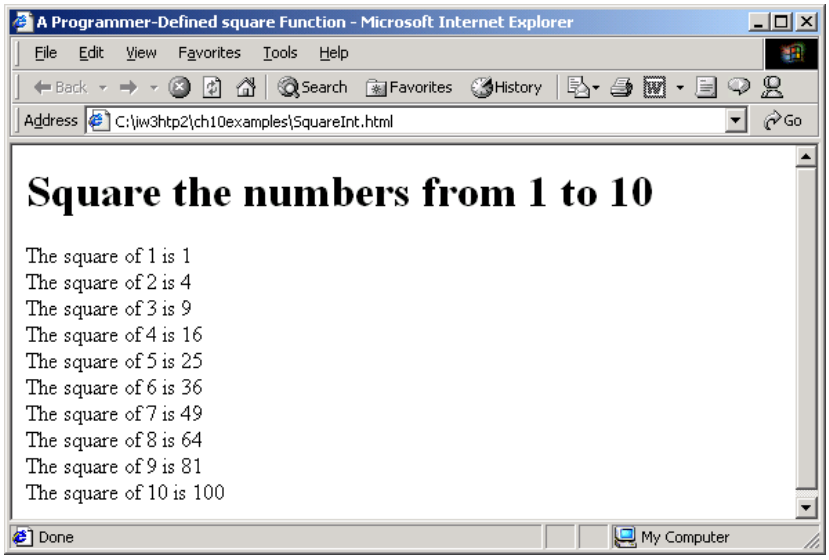
Variable **y** gets the value of variable **x**.

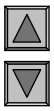
The **return** statement passes the value of **y * y** back to the calling function.



Outline

Program Output





Outline

Maximum.html

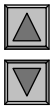
```
1   <?xml version = "1.0"?>
2   <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5   <!-- Fig. 10.3: maximum.html -->
6   <!-- Maximum function        -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>Finding the Maximum of
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 thir
20
21           var value1 = parseFloat( inpu
22           var value2 = parseFloat( input2 );
23           var value3 = parseFloat( input3 );
24
25           var maxValue = maximum( v
26
27           document.writeln( "First
28              "<br />Second number: " + value2 +
29              "<br />Third number: " + value
30              "<br />Maximum is: " + maxVal
31
32           // maximum method definition (called from line 25)
33           function maximum( x, y, z )
34           {
35              return Math.max( x, Math.max( y, z ) );
```

Prompt for the user to input three integers.

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

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

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

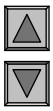


Outline

Maximum.html

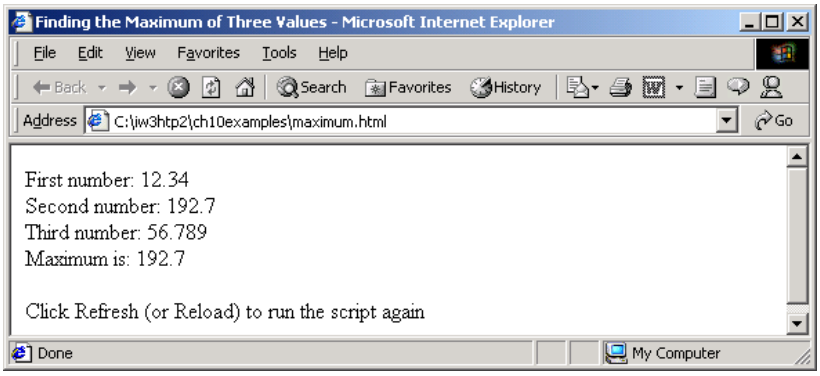
```
36           }
37           // -->
38        </script>
39
40     </head>
41     <body>
42        <p>Click Refresh (or Reload) to run the script again</p>
43     </body>
44  </html>
```

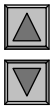
Program Output



Outline

Program Output





Outline

RandomInt.html

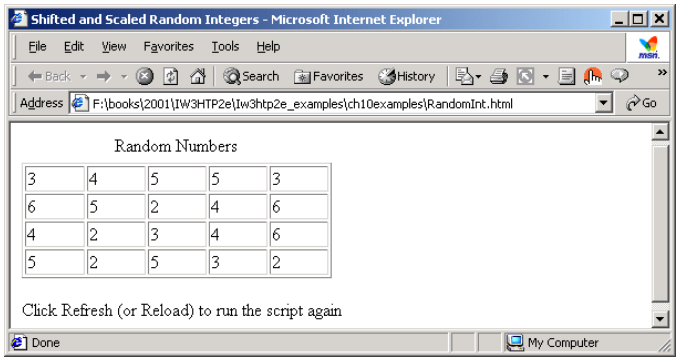
```
1    <?xml version = "1.0"?>
2    <!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
3       "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
4
5    <!-- Fig. 10.4: RandomInt.html       -->
6    <!-- Demonstrating the Random method -->
7
8    <html xmlns = "http://www.w3.org/1999/xhtml">
9       <head>
10         <title>Shifted and Scaled Random Integers</title>
11
12         <script type = "tex
13            <!--
14            var value;
15
16            document.writeln(
17               "<table border = \"1\"
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               if ( i % 5 == 0 && i != 20 )
28                  document.writeln( "</tr><tr>" );
29            }
30
31            document.writeln( "</tr></table>" );
32            // -->
33         </script>
34
35      </head>
```

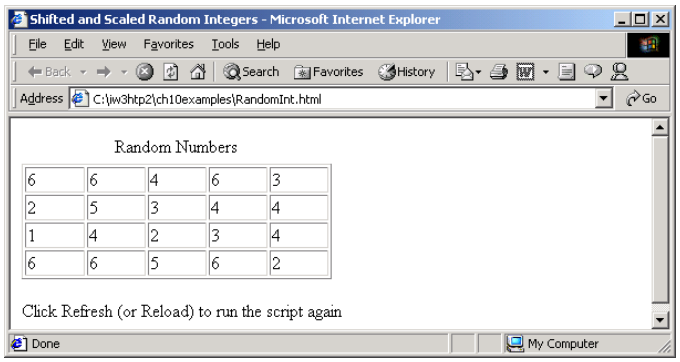
The for loop creates 4 rows with 5 cells of a table.

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

ated with a random number generated by method **random**.

```
36      <body>
37         <p>Click Refresh (or Reload) to run the script again</p>
38      </body>
39   </html>
```





Outline

RandomInt.html

Program Output

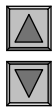
Outline

RollDie.html

```
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</
11
12         <script type = "text/javascript">
13            <!--
14            var frequency1 = 0, frequency2 = 0,
15                frequency3 = 0, frequency4 = 0,
16                frequency5 = 0, frequency6 = 0, face;
17
18            // summarize results
19            for ( var roll = 1; roll <= 60
20               face = Math.floor( 1 + Math
21
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:
```

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

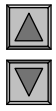
When the controlling expression, **face**, matches a **case** label, the respective **frequency** variable is incremented.



Outline

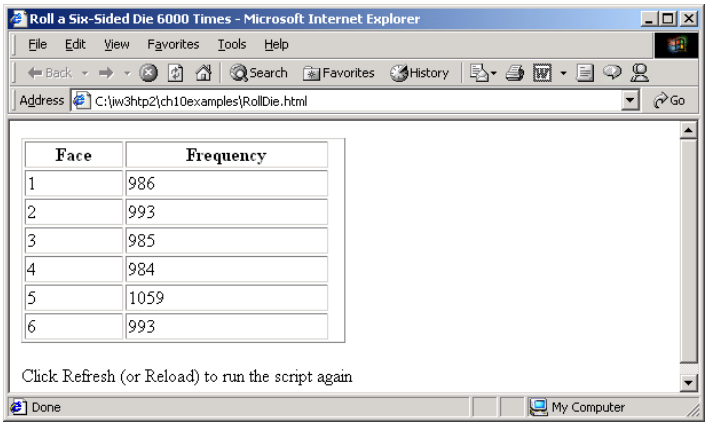
```
36                    ++frequency5;
37                    break;
38                 case 6:
39                    ++frequency6;
40                    break;
41              }
42           }
43
44           document.writeln( "<table border = \"1\"" +
45              "width = \"50%\">" );
46           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>
```

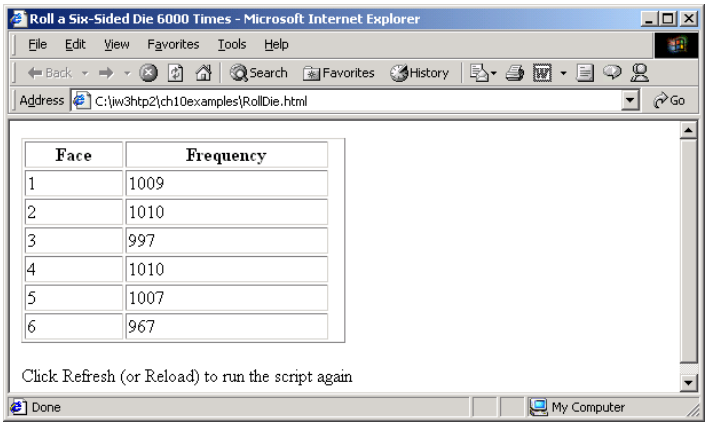
The results of the dice being rolled 600 times are displayed in a table.

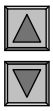


Outline

Program Output







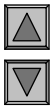
Outline

Craps.html

```
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,
20                myPoint = 0, // poi
21                gameStatus = CONTINU
22
23            // process one roll of the dice
24            function play()
25            {
26               if ( firstRoll ) {        //
27                  sumOfDice = rollDice();
28
29                  switch ( sumOfDice ) {
30                     case 7: case 11:
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
```

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

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



Outline

Craps.html

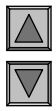
```
36                      gameStatus = LOST;
37                      // clear point field
38                      document.craps.point.value = "";
39                      break;
40                   default:
41                      gameStatus = CONTINUE_R
42                      myPoint = sumOfDice;
43                      document.craps.point.va
44                      firstRoll = false;
45                }
46             }
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 again.";
63                else
64                   window.status = "Player l
65                      "Click Roll Dice to pl
66
67                firstRoll = true;
68             }
69          }
70
```

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

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

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

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



Outline

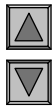
Craps.html

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

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 vlaues of the dice are placed in their respective form fields.

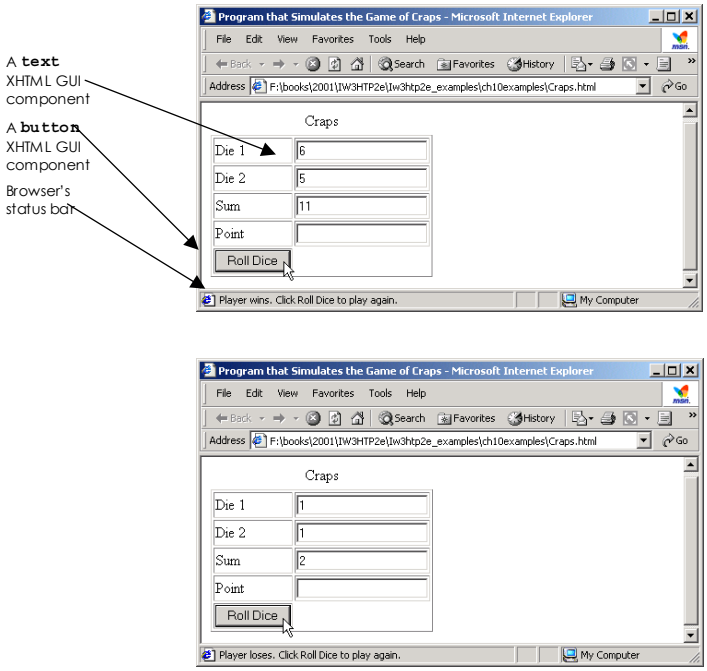


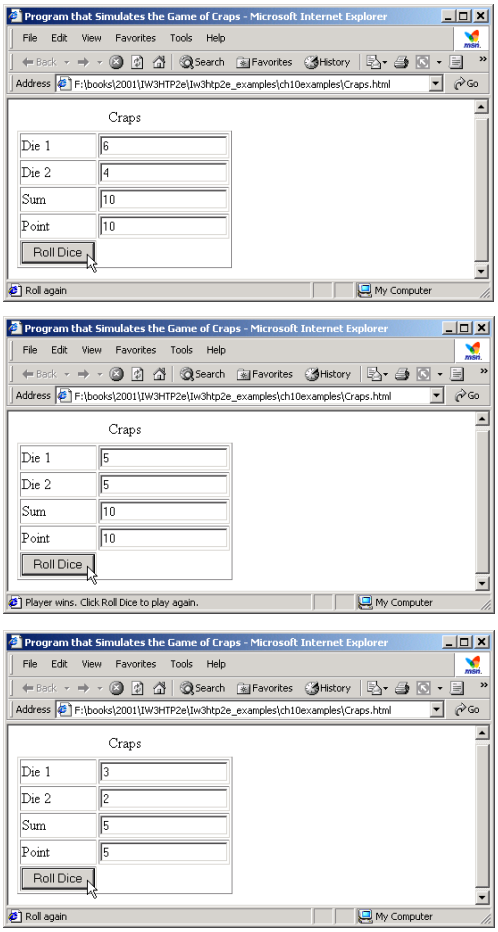
Outline

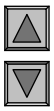
Craps.html

```
106         <tr><td><input type = "button" value = "Roll Dice"
107             onclick = "play()" /></td></tr>
108         </table>
109      </form>
110   </body>
111 </html>
```

Program Output

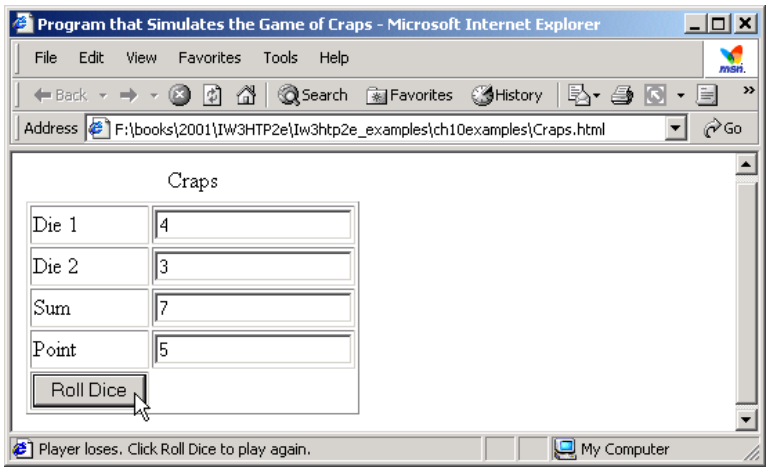


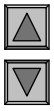




Outline

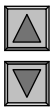
Program Output





Outline

Program Output



Outline

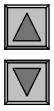
Scoping.html

```
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. 10.7: scoping.html    -->
6   <!-- Local and Global Variables -->
7
8   <html xmlns = "http://www.w3.org/1999/xhtml">
9      <head>
10        <title>A Scoping Example</tit
11
12        <script type = "text/javascript">
13           <!--
14           var x = 1;       // global var
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(); // functi
23              functionB(); // functionB uses global variable x
24              functionA(); // functionA reinitializes local x
25              functionB(); // global variable x retains its value
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
```

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

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

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



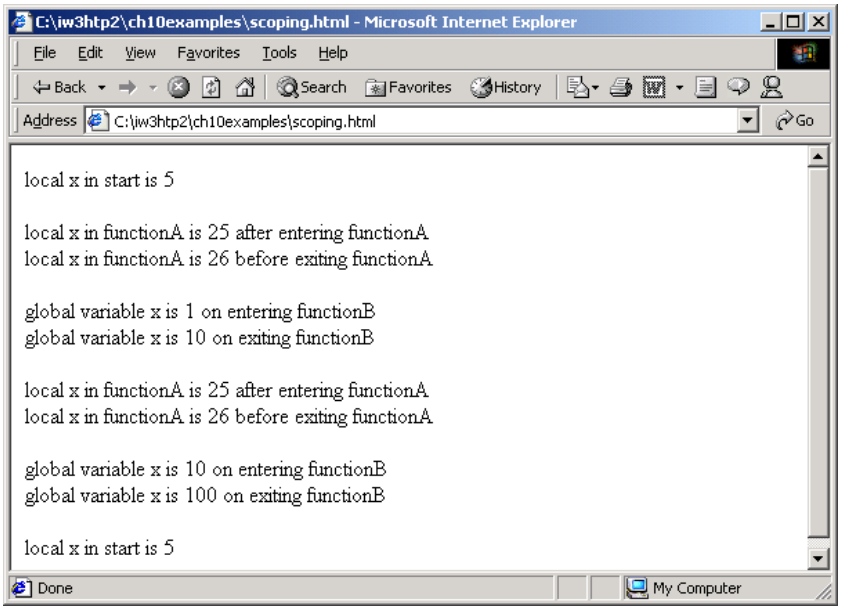
Outline

Scoping.html

```
35
36              document.writeln( "<p>local x in functionA is " +
37                                x + " after entering functionA" );
38              ++x;
39              document.writeln( "<br />local x in functionA is " +
40                 x + " before exiting functionA" + "</p>" );
41           }
42
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>
```

The value of **x** is incremented.

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



Program Output

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 C, ASCII Character Set) are encoded in a hexadecimal format (see the Number Systems appendix) 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, the function 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. 10.8 **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 the "Number Systems" appendix 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. 10.8 `JavaScript` global functions.

10.10 Recursion

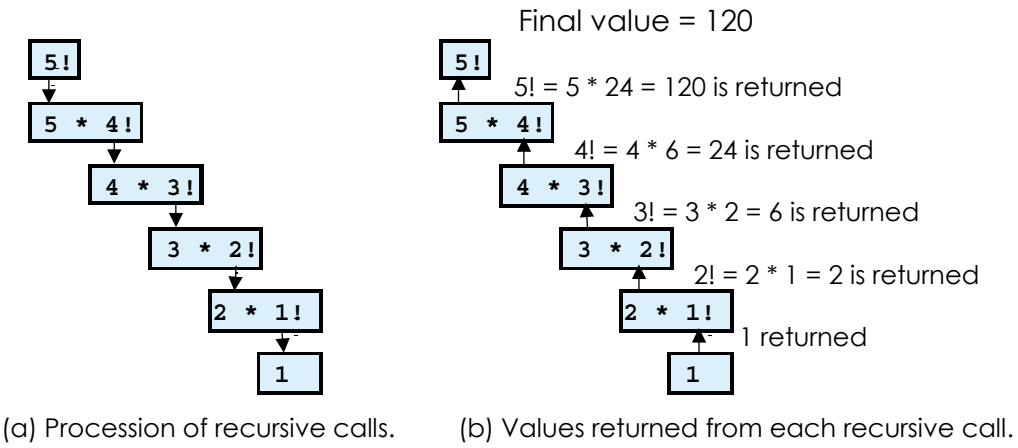
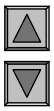


Fig. 10.9 Recursive evaluation of 5!.



Outline

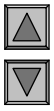
FactorialTest.html

```
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   <!-- Fig. 10.10: FactorialTest.html -->
5   <!-- Recursive factorial example    -->
6
7   <html xmlns = "http://www.w3.org/1999/xhtml">
8      <head>
9         <title>Recursive Factorial Function
10
11        <script language = "javascript">
12           document.writeln( "<h1>Factorials
13           document.writeln(
14              "<table border = '1' width = '100%'>" );
15
16           for ( var i = 0; i <= 10; i++ )
17              document.writeln( "<tr><td>" + i + "!</td><td>" +
18                                factorial( i )
19
20           document.writeln( "</table>" );
21
22           // Recursive definition of function factorial
23           function factorial( number )
24           {
25              if ( number <= 1 )  // base
26                 return 1;
27              else
28                 return number * factorial( number - 1 );
29           }
30        </script>
31     </head><body></body>
32  </html>
```

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

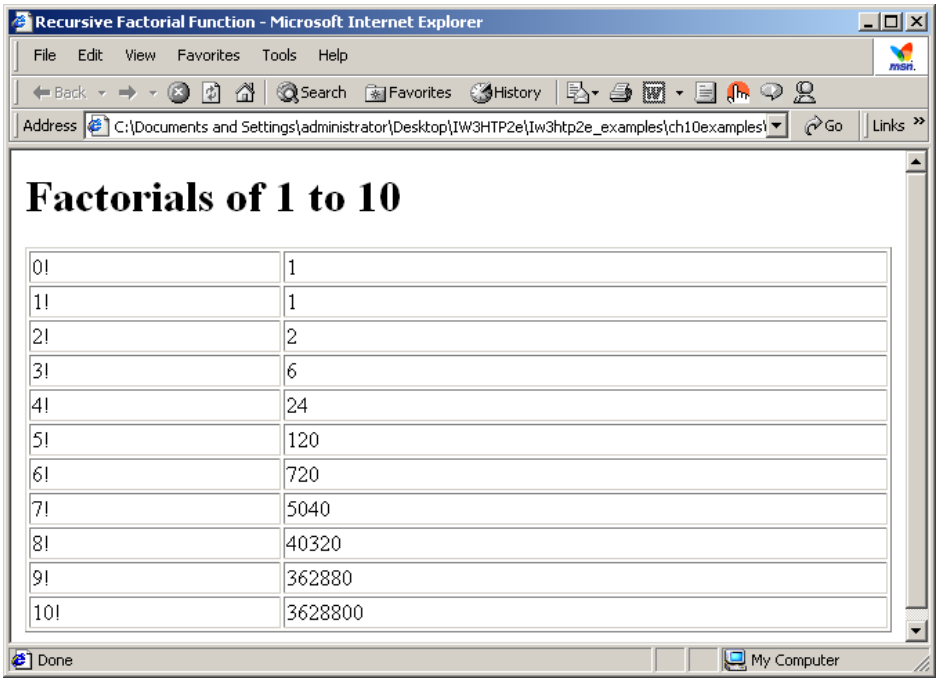
Variable **number** gets the value of variable **i**.

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



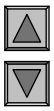
Outline

Program Output



Factorials of 1 to 10

0!	1
1!	1
2!	2
3!	6
4!	24
5!	120
6!	720
7!	5040
8!	40320
9!	362880
10!	3628800



Outline

FibonacciTest.html

```
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   <!-- Fig. 10.11: FibonacciTest.html -->
5   <!-- Recursive Fibonacci example    -->
6
7   <html xmlns = "http://www.w3.org/1999/xhtml">
8      <head>
9         <title>Recursive Fibonacci Function</title>
10
11        <script language = "javascript">
12
13           // Event handler for button XHTML component in myForm
14           function getFibonacciValue()
15           {
16              var value = parseInt(
17                 document.myForm.number.value );
18              window.status =
19                 "Calculating Fibonacci number for " + value;
20              document.myForm.result.value = fibonacci( value );
21              window.status = "Done calculating Fibonacci number";
22           }
23
24           // Recursive definition of
25           function fibonacci( n )
26           {
27              if ( n == 0 || n == 1 )  // base case
28                 return n;
29              else
30                 return fibonacci( n -
31           }
32        </script>
33     </head>
34
```

Convert from a string to an integer the value the user typed into the number text field.

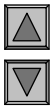
Display the number the user entered in the status bar.

the

The status bar displays a message that the call to function **fibonacci** is complete.

Test for base case (**n** equal to **1** or **0**).

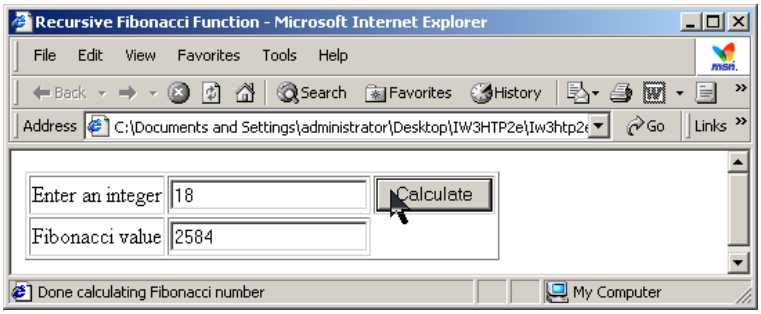
Two recursive calls are made if **n** is greater than **1**.



Outline

FibonacciTest.html

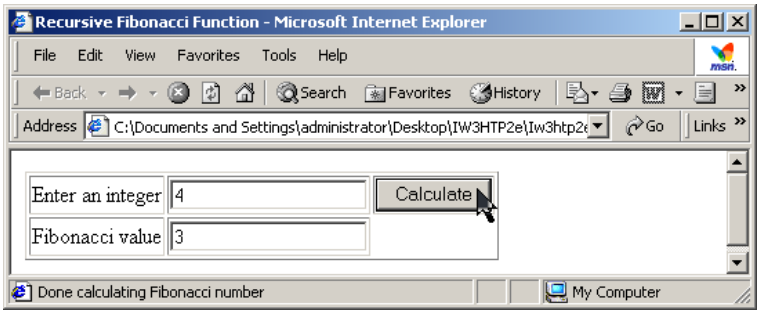
```
35     <body>
36        <form name = "myForm">
37           <table border = "1">
38              <tr><td>Enter an integer</td>
39              <td><input name = "number" type = "text"></td>
40              <td><input type = "button" value = "Calculate"
41                 onclick = "getFibonacciValue()"</tr>
42              <tr><td>Fibonacci value</td>
43              <td><input name = "result" type = "text"></td></tr>
44           </table>
45        </form></body>
46  </html>
```

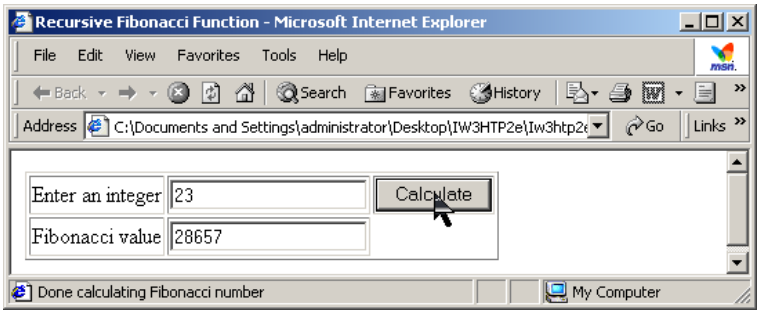


Program Output

Outline

Program Output





10.11 Example Using Recursion: Fibonacci Series

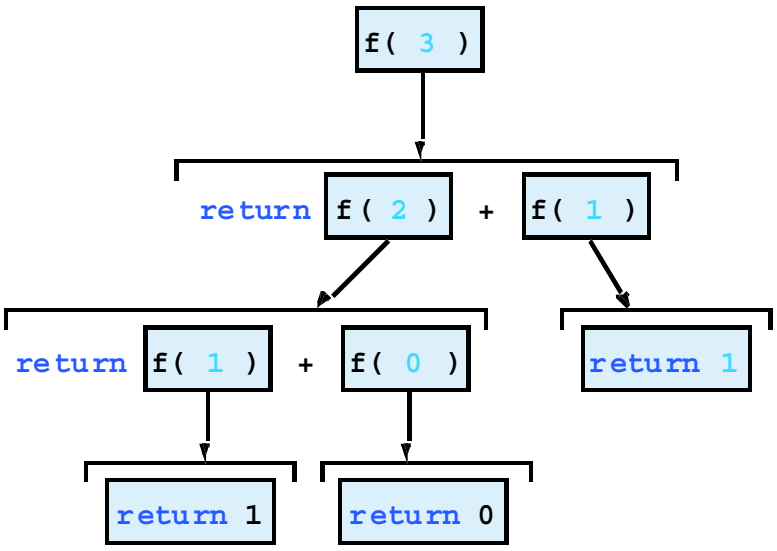


Fig. 10.12 Set of recursive calls to function fibonacci.

10.12 Recursion vs. Iteration

Chapter	Recursion examples and exercises
10	Factorial function Greatest common divisor Sum of two integers Multiply two integers Raising an integer to an integer power Towers of Hanoi Visualizing recursion
11	Sum the elements of an array Print an array Print an array backward Check if a string is a palindrome Minimum value in an array Selection sort Eight Queens Linear search Binary search Quicksort Maze traversal
12	Printing a string input at the keyboard backward

Fig. 10.13 Summary of recursion examples and exercises in the text.