9

JavaScript: Functions

Outline

9.2 Program Modules in JavaScript

- JavaScript programs are written by combining new functions that the programmer writes with "prepackaged" functions and objects available in JavaScript
- The term method implies that a function belongs to a particular object
- We refer to functions that belong to a particular JavaScript object as methods; all others are referred to as functions.
- 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.

9.2 Program Modules in JavaScript (Cont.)

- **You can define programmer-defined functions that perform specific tasks and use them at many points in a script**
  - **The actual statements defining the function are written only once and are hidden from other functions**
- **Functions are invoked by writing the name of the function, followed by a left parenthesis, followed by a comma-separated list of zero or more arguments, followed by a right parenthesis**
- **Methods are 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 constants, variables or expressions**

9.4 Function Definitions

- `return` statement
  - passes information from inside a function back to the point in the program where it was called
- A function must be called explicitly for the code in its body to execute
- The format of a function definition is

  function *function-name*(*parameter-list*)

  {

  declarations and statements

  }

9.4 Function Definitions (Cont.)

- **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**

Fig. 9.2 | Programmer-defined function **square** (Part 1 of 2).

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.2: SquareInt.html -->
<!-- Programmer-defined function square. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>A Programmer-Defined square Function</title>
      <script type = "text/javascript">
         <!--
         document.writeln( "<h1>Square the numbers from 1 to 10</h1>" );

         // square the numbers from 1 to 10
         for ( var x = 1; x <= 10; x++ )
            document.writeln( "The square of " + x + " is " +
               square( x ) + "<br />" );

         // The following square function definition is executed
         // only when the function is explicitly called.

         // square function definition
         function square( y )
         {
            return y * y;
         } // end function square
         // -->
      </script>
   </head><body></body>
</html>
```

Calls function `square` with `x` as an argument, which will return the value to be inserted here

Begin function `square`

Names the parameter `y`

Returns the value of `y * y` (the argument squared) to the caller

End function `square`

Fig. 9.3 | Programmer-defined `maximum` function (Part 1 of 3).

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.3: maximum.html -->
<!-- Programmer-Defined maximum function. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Finding the Maximum of Three Values</title>
      <script type = "text/javascript">
         <!--
         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 );

```

Creates integer values from user input

Fig. 9.3 | Programmer-defined **maximum** function (Part 2 of 3).

```
20          var maxValue = maximum( value1, value2, value3 );
21
22                 nt.writeln( "First number: " + value1 +
23                 r />Second number: " + value2 +
24                 r />Third number: " + value3 +
25                 r />Maximum is: " + maxValue );
26
27                 imum function definition (called from line 20)
28                 on maximum( x, y, z )
29          {
30             return Math.max( x, Math.max( y, z ) );
31          } // end function maximum
32          // -->
33       </script>
34    </head>
35    <body>
36       <p>Click Refresh (or Reload) to run the script again</p>
37    </body>
38 </html>
```

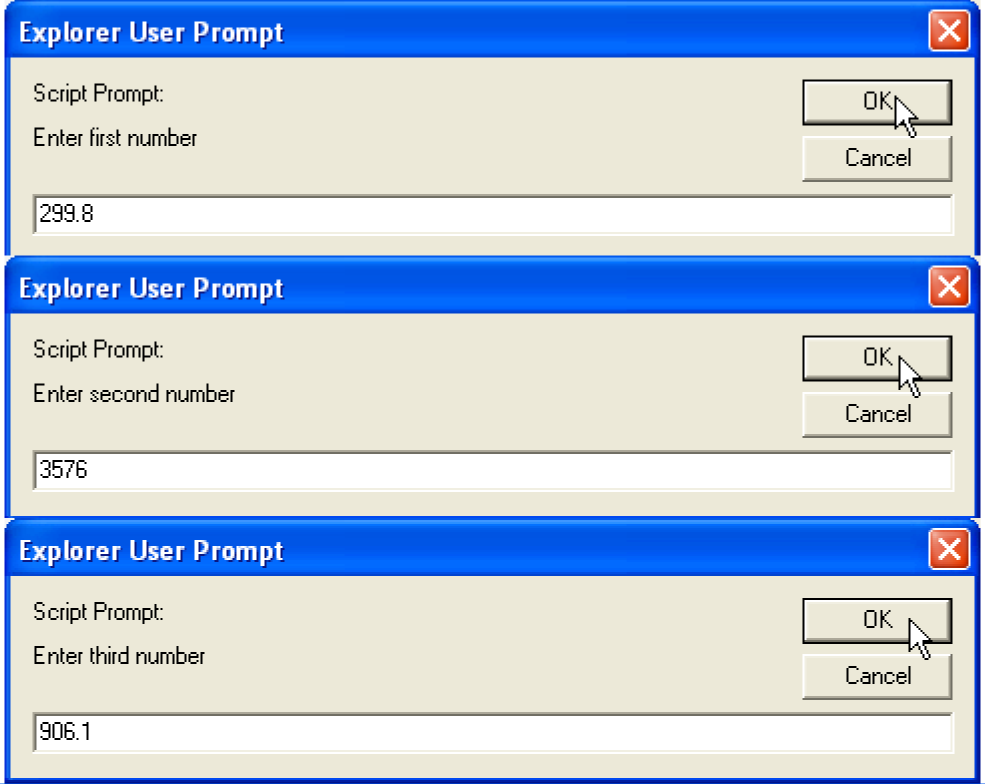
Variable `maxValue` stores the return value of the call to `maximum`

Calls function `maximum` with arguments `value1`, `value2` and `value3`

Begin function `maximum` with local variables `x`, `y` and `z`

Calls the `Math` object's method `max` to compare the first variable with the maximum of the other two

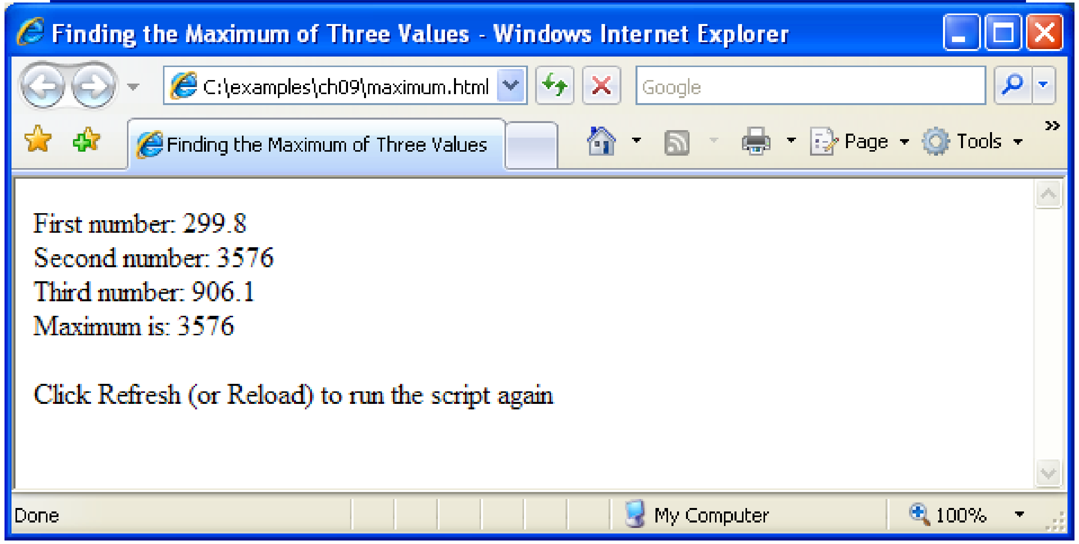
End function `maximum`



Script Prompt:
Enter first number
299.8

Script Prompt:
Enter second number
3576

Script Prompt:
Enter third number
906.1


Finding the Maximum of Three Values - Windows Internet Explorer
C:\examples\ch09\maximum.html
Google
Finding the Maximum of Three Values
Page
Tools
Done
My Computer
100%

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

Click Refresh (or Reload) to run the script again
```

Fig. 9.3 | Programmer-defined `maximum` function (Part 3 of 3).

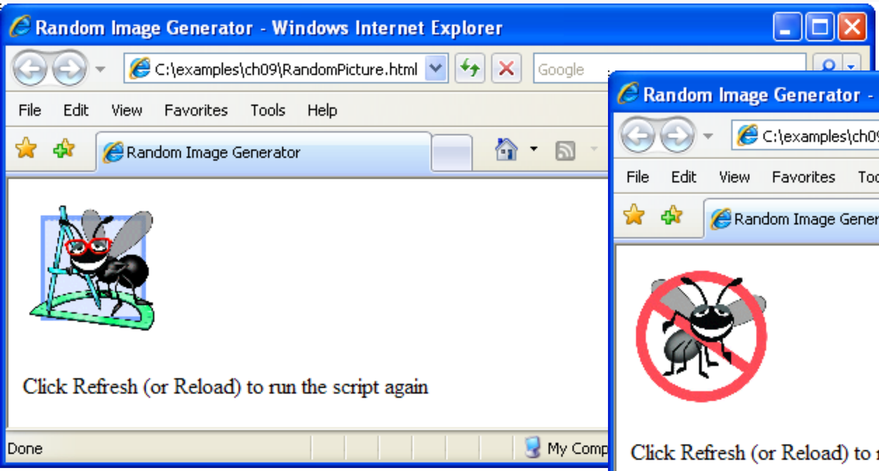
9.7 Example: Random Image Generator

- **We can use random number generation to randomly select from a number of images in order to display a random image each time a page loads**

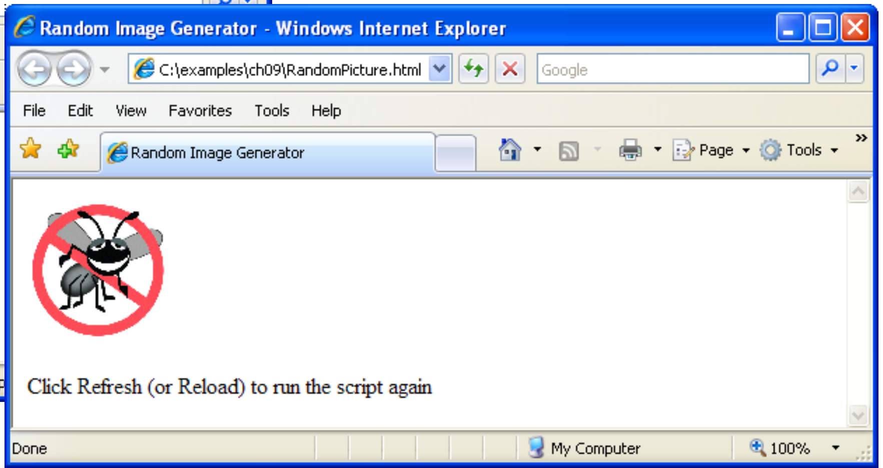
Fig. 9.7 | Random image generation using **`Math.random`.**

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.7: RandomPicture.html -->
<!-- Random image generation using Math.random. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Random Image Generator</title>
      <script type = "text/javascript">
         <!--
         document.write ( "<img src = \"" +
            Math.floor( 1 + Math.random() * 7 ) +  ".gif\" />" );
         // -->
      </script>
   </head>
   <body>
      <p>Click Refresh (or Reload) to run the script again</p>
   </body>
</html>
```

Creates an `src` attribute by concatenating a random integer from 1 to 7 with "`.gif\`" to reference one of the images `1.gif`, `2.gif`, `3.gif`, `4.gif`, `5.gif`, `6.gif` or `7.gif`


Random Image Generator - Windows Internet Explorer
C:\examples\ch09\RandomPicture.html
Google
File Edit View Favorites Tools Help
Random Image Generator
Done
My Comp

Click Refresh (or Reload) to run the script again


Random Image Generator - Windows Internet Explorer
C:\examples\ch09\RandomPicture.html
Google
File Edit View Favorites Tools Help
Random Image Generator
Page Tools
Done
My Computer
100%

Click Refresh (or Reload) to run the script again

9.8 Scope Rules

- **Each identifier in a program has a scope**
- **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 are accessible in any part of a script and are said to have global scope**
  - **Thus every function in the script can potentially use the variables**

9.8 Scope Rules (Cont.)

- **Identifiers declared inside a function have function (or local) scope and can be used only in that function**
- **Function scope begins with the opening left brace ({) of the function in which the identifier is declared and ends at the terminating right brace (}) of the function**
- **Local variables of a function and function parameters have function scope**
- **If a local variable in a function has the same name as a global variable, the global variable is "hidden" from the body of the function.**

9.8 Scope Rules (Cont.)

- `onload` **property of the** `body` **element calls an event handler when the** `<body>` **of the XHTML document is completely loaded into the browser window**

Fig. 9.8 | Scoping example (Part 1 of 3).

```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.8: scoping.html -->
<!-- Scoping example. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>A Scoping Example</title>
      <script type = "text/javascript">
         <!--
         var x = 1; // global variable

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

            document.writeln( "local x in start is " + x );

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

            document.writeln(
               "<p>local x in start is " + x + "</p>" );
         } // end function start

```

Global variable declaration

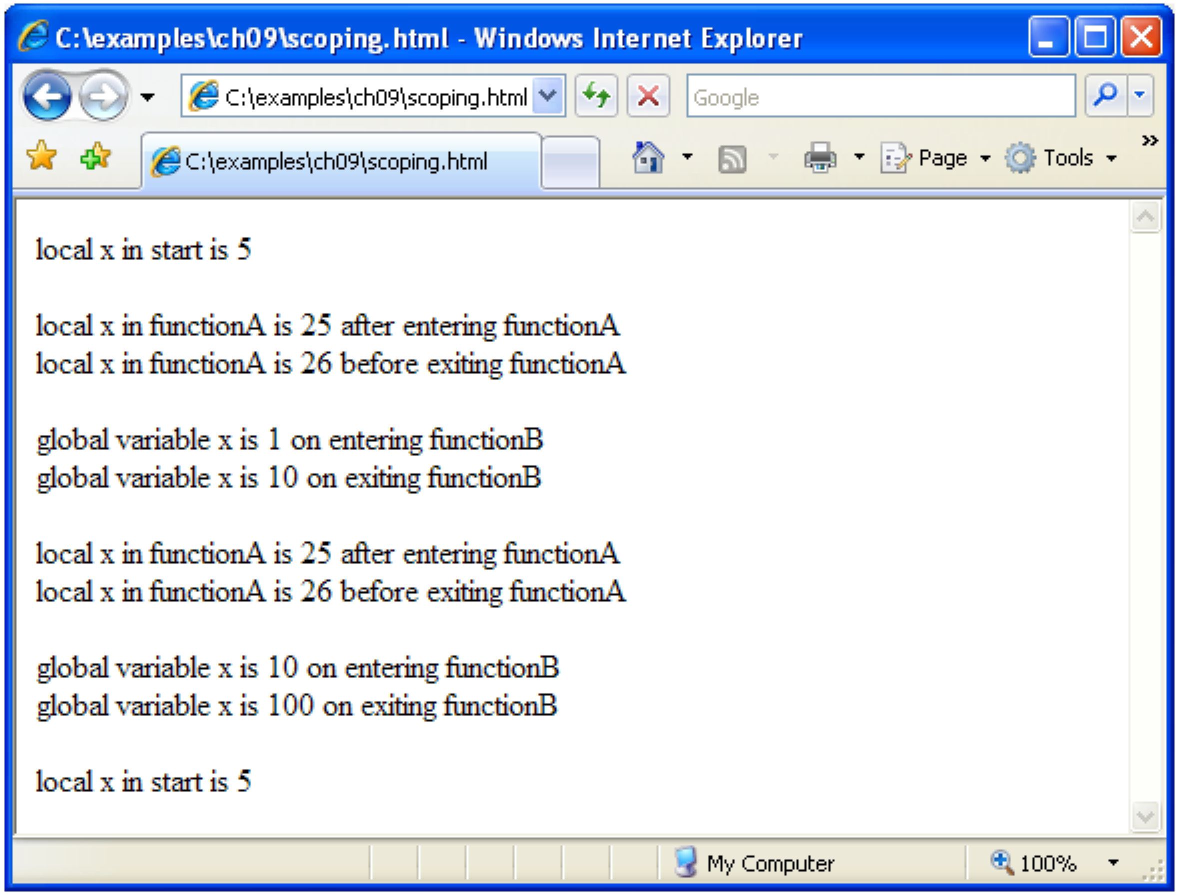
Local variable in function `start`

Fig. 9.8 | Scoping example (Part 2 of 3).

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

               document.writeln( "<p>local x in functionA is " +
                                 x + " after entering functionA" );
               ++x;
               document.writeln( "<br />local x in functionA is " +
                  x + " before exiting functionA" + "</p>" );
            } // end functionA

            function functionB()
            {
               document.writeln( "<p>global variable x is " + x +
                  " on entering functionB" );
               x *= 10;
               document.writeln( "<br />global variable x is " +
                  x + " on exiting functionB"  + "</p>" );
            } // end functionB
            // -->
         </script>
      </head>
      <body onload = "start()"></body>
   </html>
```

Local variable in function functionA, initialized each time functionA is called

Calls function start when the body of the document has loaded into the browser window


C:\examples\ch09\scoping.html - Windows Internet Explorer
C:\examples\ch09\scoping.html
Google
C:\examples\ch09\scoping.html
Page
Tools
My Computer
100%

```
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
```

Fig. 9.8 | Scoping example (Part 3 of 3).

9.9 JavaScript Global Functions

- **JavaScript provides seven global functions as part of a `Global` object**
- **This object contains**
  - **all the global variables in the script**
  - **all the user-defined functions in the script**
  - **all the built-in global functions listed in the following slide**
- **You do not need to use the `Global` object directly; JavaScript uses it for you**

9.10 Recursion

- **A recursive function calls itself, either directly, or indirectly through another function.**
- **A recursive function knows how to solve only the simplest case, or base case**
  - **If the function is called with a base case, it returns a result**
  - **If the function is called with a more complex problem, it divides the problem into two conceptual pieces—a piece that the function knows how to process (the base case) and a simpler or smaller version of the original problem.**
- **The function invokes (calls) a fresh copy of itself to go to work on the smaller problem; this invocation is referred to as a recursive call, or the recursion step.**

9.10 Recursion (Cont.)

- **The recursion step executes while the original call to the function is still open (i.e., it has not finished executing)**
- **For recursion eventually to terminate, each time the function calls itself with a simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case**
  - **At that point, the function recognizes the base case, returns a result to the previous copy of the function, and a sequence of returns ensues up the line until the original function call eventually returns the final result to the caller**

Fig. 9.11 | Factorial calculation with a recursive function (Part 1 of 2).

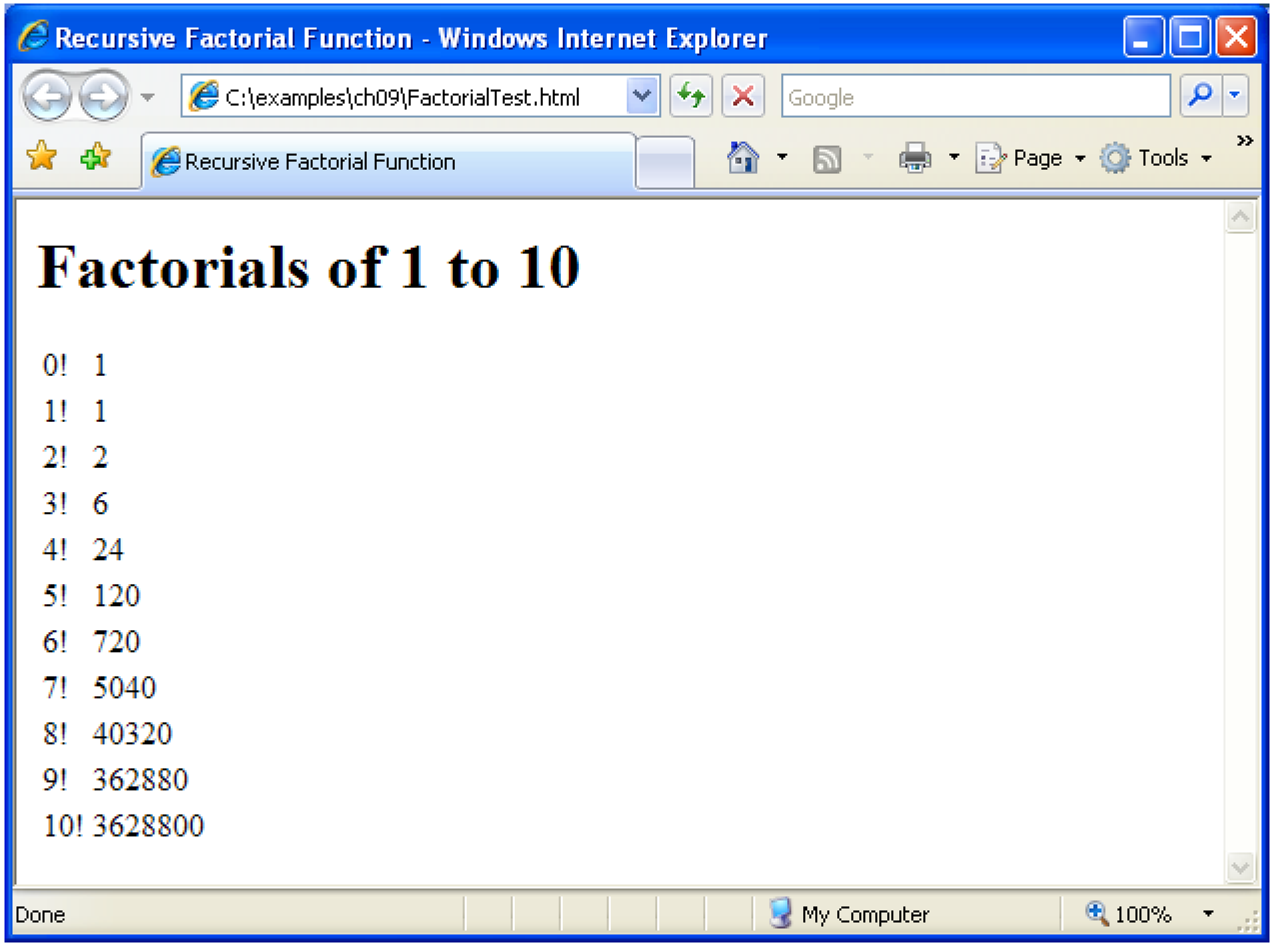
```
<?xml version = "1.0" encoding = "utf-8"?>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
   "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">

<!-- Fig. 9.11: FactorialTest.html -->
<!-- Factorial calculation with a recursive function. -->
<html xmlns = "http://www.w3.org/1999/xhtml">
   <head>
      <title>Recursive Factorial Function</title>
      <script type = "text/javascript">
      <!--
         document.writeln( "<h1>Factorials of 1 to 10</h1>" );
         document.writeln( "<table>" );

         for ( var i = 0; i <= 10; i++ )
            document.writeln( "<tr><td>" + i + "!</td><td>" +
               factorial( i ) + "</td></tr>" );

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

         // Recursive definition of function factorial
         function factorial( number )
         {
            if ( number <= 1 )   // base case
               return 1;
            else
               return number * factorial( number - 1 );
         } // end function factorial
         // -->
      </script>
   </head><body></body>
</html>
```

Calls function `factorial` with argument `i`

Base case

While the base case is not reached, return the number * (number – 1)!, which is `number * factorial ( number – 1 )`

`factorial` calls itself with a new argument and waits until this new value is returned before returning a value itself


Recursive Factorial Function - Windows Internet Explorer
C:\examples\ch09\FactorialTest.html
Google
Recursive Factorial Function
Page
Tools
Done
My Computer
100%

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
```

Fig. 9.11 | Factorial calculation with a recursive function (Part 2 of 2).