Functions

Bok, Jong Soon
javaexpert@nate.com
www.javaexpert.co.kr

Function

- Is a block of code that executes only when you tell it to execute.
- Can be placed both in the <head> and in the <body> section of a document.
- There are three approaches to creating functions in JavaScript:
  - Declarative/static
  - Dynamic/anonymous
  - Literal

Declarative Functions

- The most common type format of function.

```
function functionname(param1, param2, ..., paramn){
        function statements
}
```

- Is parsed once, when the page is loaded, and the parsed result is used each time the function is called.
- Is simple to read and understand.
- Has limited negative consequences, such as memory leaks.
- Is also more familiar to developers.

Declarative Functions (Cont.)

- A function definition (also called a function declaration) consists of the `function` keyword, followed by
  - The name of the function.
  - A list of arguments to the function, enclosed in parentheses and separated by commas.
  - The JavaScript statements that define the function, enclosed in curly braces, `{ }`.

Declarative Functions (Cont.)

```
<head>
 <meta http-equiv="Content-Type" content="text/html;charset=utf-8">

  <script type="text/javascript">
     function myFunction() {
        alert("Hello World!");
     }
  </script>
</head>
<body>
  <button onclick="myFunction()">Try it</button>
</body>
```

Function Naming Conventions and Size

- Functions perform actions.
- Typical type of function's name is a *verb*.
- Have the advantage of being easy to read and somewhat self-documenting.
- If have performing more than one task, may want to consider splitting the function into smaller units.
- Should be to keep functions small.
- Specifies to a task.

Function Arguments and Returns

- When call a function, can pass along some values to it, these values are called *arguments* or *parameters*.
- These arguments can be used inside the function.
- Can send as many arguments as you like, separated by commas (`,`).

```
function myFunction(var1,var2) {
      some code
}
myFunction(argument1,argument2);
```

Function Arguments and Returns (Cont.)

```
<html>
 <head>
  <meta http-equiv="Content-Type" content="text/html;charset=utf-8">

    <script type="text/javascript">
        function myFunction(name,job) {
            alert("Welcome " + name + ", the " + job);
        }
    </script>
 </head>
 <body>
    <button onclick="myFunction('Harry Potter','Wizard')">Try it</button>
 </body>
</html>
```

Function Arguments and Returns (Cont.)

- Sometimes you want your function to return a value back to where the call was made.
- This is possible by using the **`return`** statement.
- When using the **`return`** statement, the function will stop executing, and return the specified value.

```
<script type="text/javascript">

   function square(number){
      return number * number;
   }

   document.write(square(4));   //print 16

</script>
```

Function Arguments and Returns (Cont.)

- Variables based on primitives, such as a string, boolean, or number, are passed to a function ***by value***.
- Means that if you change the actual argument in the function, the change is ***not*** reflected in the calling program.
- Objects passed to a function, on the other hand, are passed ***by reference***.
- Changes in the function to the object are reflected in the calling program.

Function Arguments and Returns (Cont.)

```
<script type="text/javascript">
   function alterArgs(strLiteral, aryObject) {
      // overwrite original string
      strLiteral = "Override";
      aryObject[aryObject.length] = "three";
   }

   var str = "Original Literal";
   var ary = new Array("one","two");
   alterArgs(str,ary);
   document.writeln("string literal is " + str + "<br /> ");
   document.writeln("Array object is " + ary);
</script>
```

Passing Arguments by Value versus Reference

- Call by reference.

```
<script type="text/javascript">
   function myFunc(theObject) {
      theObject.make = "Toyota";
   }

   var mycar = {make: "Honda", model: "Accord", year: 1998}, x, y;

   x = mycar.make;     // x gets the value "Honda"

   myFunc(mycar);
   y = mycar.make;
   document.write(y);   //print Toyota
</script>
```

Passing Arguments by Value versus Reference (Cont.)

- Call by value.

```
<script type="text/javascript">
   function myFunc(theObject) {
      theObject = {make: "Ford", model: "Focus", year: 2006};
   }

   var mycar = {make: "Honda", model: "Accord", year: 1998}, x, y;
   x = mycar.make;     // x gets the value "Honda"
   myFunc(mycar);
   y = mycar.make;     // y still gets the value "Honda"
   document.write(x + "<br />" + y);   //print both Honda
</script>
```

Passing Arguments by Value versus Reference (Cont.)

- Call by value.
  - Number
  - String
  - Boolean
  - null
  - undefined
- Call By Reference
  - Object
  - Array
  - Function
  - Date
  - RegExp
  - Error

Passing Arguments by Value versus Reference (Cont.)

```
<script type="text/javascript">
    var myNum = 100;
    var myObj = {name : 'David', age : 12};
    function changeValues(num, obj){
        num = 0 ;
        obj.name = "Sujan";
    }
    changeValues(myNum, myObj);
    document.write(myNum + "<br />");    //print 100
    document.write(myObj.name + "<br />");   //print Sujan
</script>
```

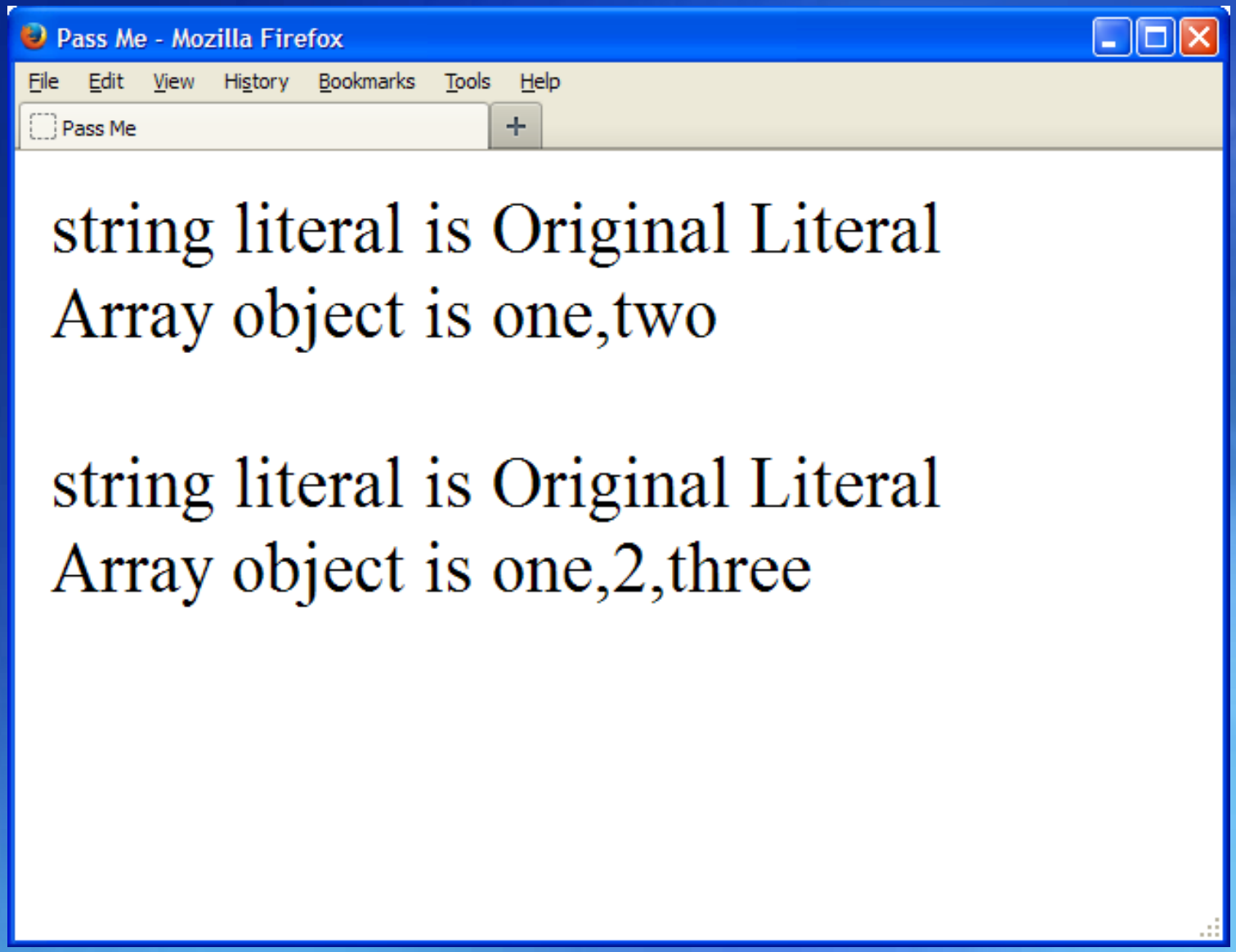
Lab1 : Pass By Value vs Pass By Reference

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - passme.html

Lab1 : passme.html

```
9   <script>
10      function alterArgs(strLiteral, aryObject) {
11
12          // overwrite original string
13          strLiteral = "Override";
14          aryObject[1] = "2";
15          aryObject[aryObject.length] = "three";
16      }
17
18      function testParams() {
19          var str = "Original Literal";
20          var ary = new Array("one","two");
21
22          document.writeln("string literal is " + str + "<br />");
23          document.writeln("Array object is " + ary + "<br /><br />");
24
25          alterArgs(str,ary);
26
27          document.writeln("string literal is " + str + "<br /> ");
28          document.writeln("Array object is " + ary);
29      }
30      testParams();
31  </script>
```

Lab1 : Result


Pass Me - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Pass Me
string literal is Original Literal
Array object is one,two
string literal is Original Literal
Array object is one,2,three

Variable Scope

- Global variable
- Local variable

```
<script type="text/javascript">
    var a = "apple";
    function myFunct(){
        var a = "book";
        document.write(a + "<br />");   // print book
    }
    myFunct();
    document.write(a + "<br />");   // print apple
</script>
```

```
<script type="text/javascript">
    var a = "apple";
    function myFunct(){
        a = "book";
        document.write(a + "<br />");   // print book
    }
    myFunct();
    document.write(a + "<br />");   // print book
</script>
```

Variable Scope (Cont.)

```
<script type="text/javascript">
    var a = "apple";
    function myFunct(){
        a = "book";
        document.write(a + "<br />");   // print book
    }
    function myFunct2(){
        var a = "tree";
        myFunct();
        document.write(a + "<br />");    // print tree
    }
    myFunct2();
    document.write(a + "<br />");   // print book
</script>
```

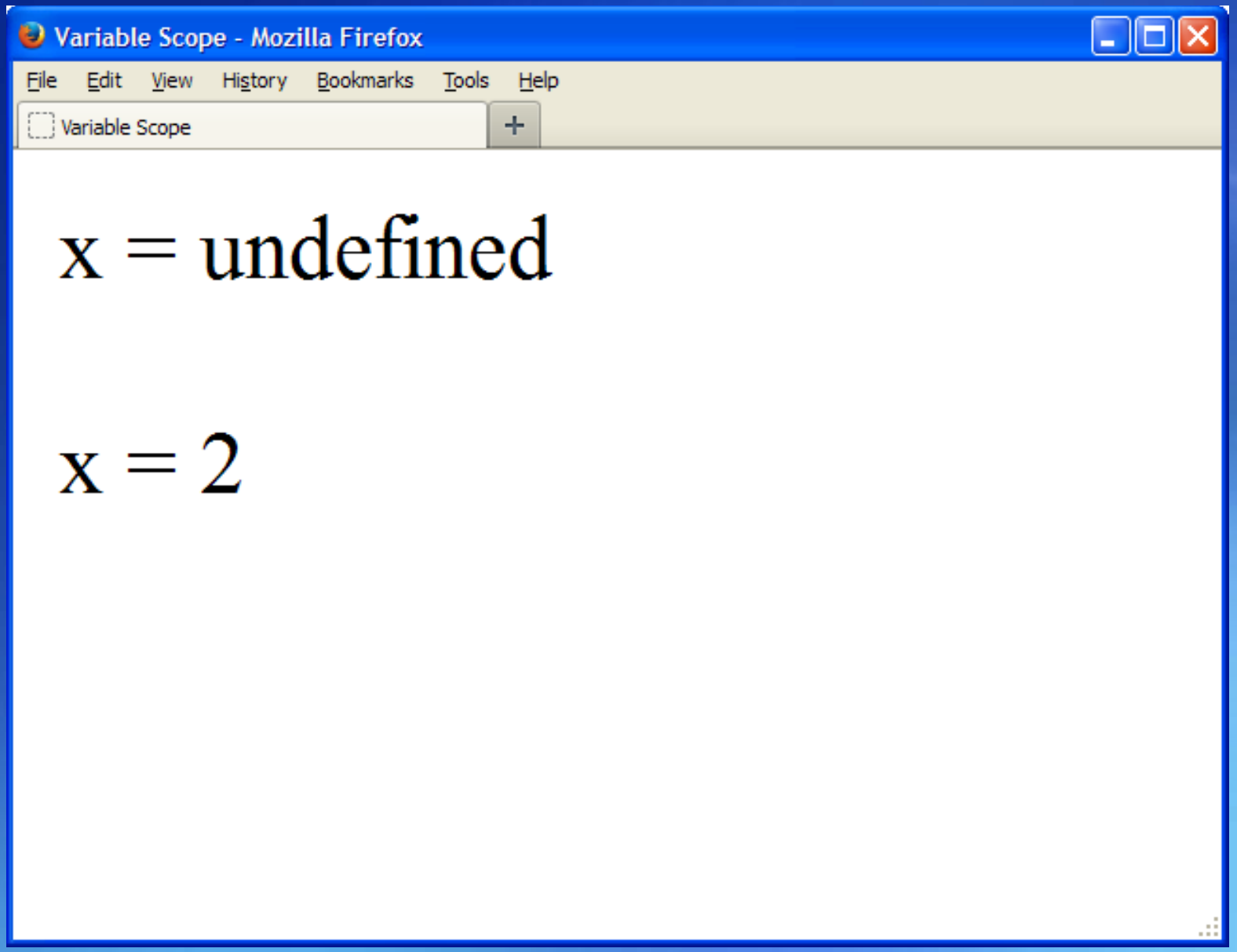
Lab2 : Variable Scope

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - variablescope.html

Lab2 : variablescope.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> Variable Scope </title>
5          <meta charset="utf-8">
6      </head>
7      <body>
8          <script>
9              var x = 1;
10             function f(){
11                 document.write("x = " + x + "<br><br>");
12                 var x = 2;
13                 document.write("x = " + x);
14             }
15             f();
16         </script>
17     </body>
18 </html>
```

Lab2 : Result


Variable Scope - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Variable Scope
x = undefined
x = 2

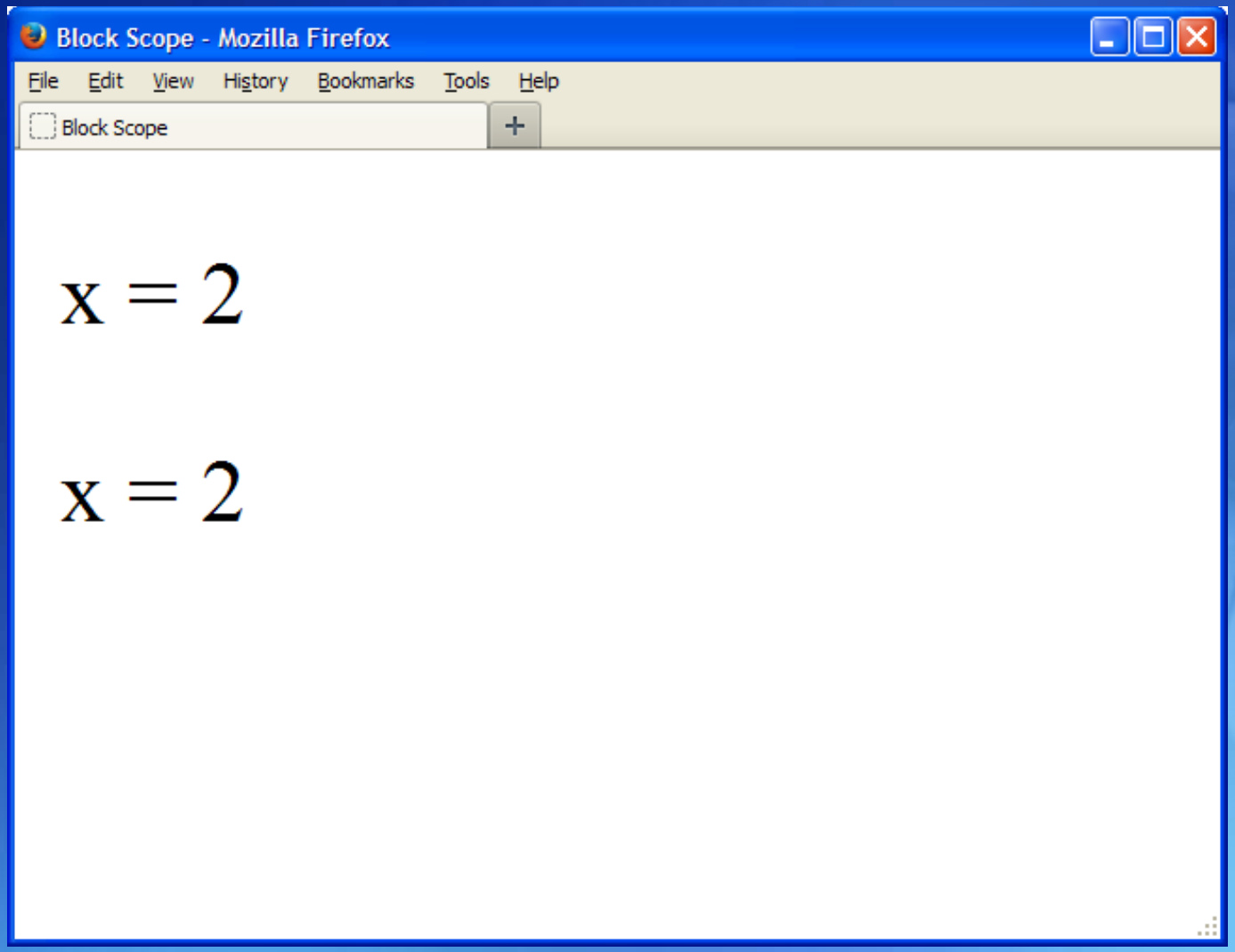
Lab3 : Block Scope

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - blockscope.html

Lab3 : blockscope.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> Block Scope </title>
5          <meta charset="utf-8">
6      </head>
7      <body>
8          <script>
9              var x = 1;
10             {
11                 var x = 2;
12                 document.write("<p>x = " + x + "</p>");
13             }
14             x = 2;
15             document.write("<p>x = " + x + "</p>");
16         </script>
17     </body>
18 </html>
```

Lab3 : Result


Block Scope - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Block Scope
x = 2
x = 2

Anonymous Functions

- Functions are objects.
- Can create them, just like a `String` or `Array`.
- By using a constructor and assigning the function to a variable.

```
<script type="text/javascript">

   var sayHi = new Function("toWhom", "alert('Hi ' + toWhom);");
   sayHi("World!");

</script>
```

Anonymous Functions (Cont.)

- Syntax

```
var variable =
            new Function ("param1",
            "param2", ... , "paramn",
            "function body");
```

- The first parameters are the arguments to the function.
- The last parameter is the *function body*.

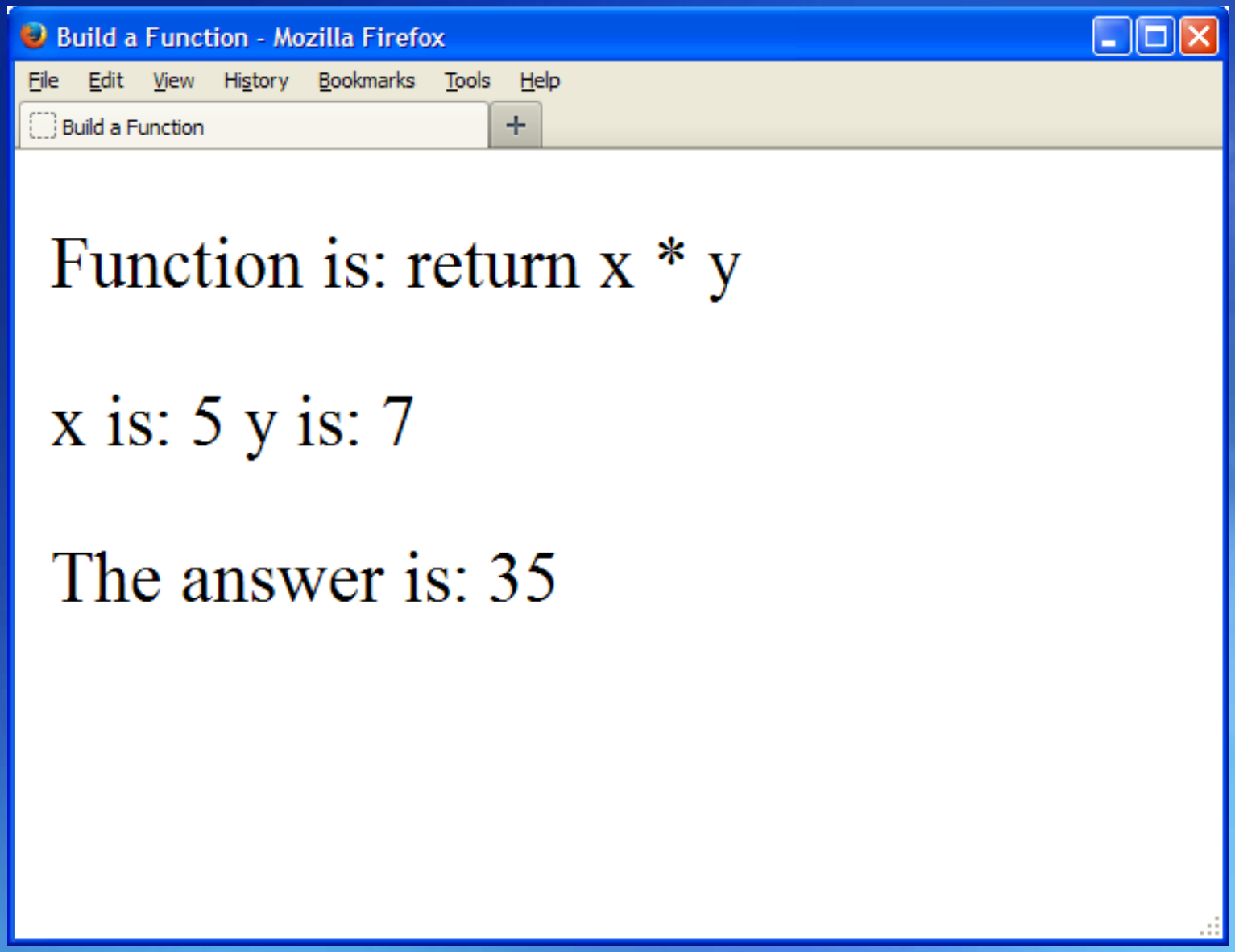
Lab4 : Anonymous Functions

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - anonymous.html

Lab4 : anonymous.html

```
9   <script>
10      function buildFunction() {
11
12          // prompt for function and args
13          var func= prompt("Enter function body:");
14          var x = prompt("Enter value of x:");
15          var y = prompt("Enter value of y:");
16
17          // invoke anonymous function
18          var op = new Function("x", "y", func);
19          var theAnswer = op(x, y);
20
21          // print out results
22          document.write("<p>Function is: " + func + "</p>");
23          document.write("<p>x is: " + x + " y is: " + y+ "</p>");
24          document.write("<p>The answer is: " + theAnswer+ "</p>");
25      }
26      buildFunction();
27  </script>
```

Lab4 : Result


Build a Function - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Build a Function
Function is: return x * y
x is: 5 y is: 7
The answer is: 35

Function Literals

- JavaScript objects can have a literal form.
- JavaScript function is a **`Object`**.
- Means don't have to use a function constructor to create a function and assign it to a variable.

```
var func = function (params) {
    statements;
}
```

Function Literals (Cont.)

```
<body>

  <p id="demo"></p>

  <script type="text/javascript">
    function myFunction(a,b) {
      return a*b;
    }
    document.getElementById("demo").innerHTML=myFunction(4,3);
  </script>

</body>
```

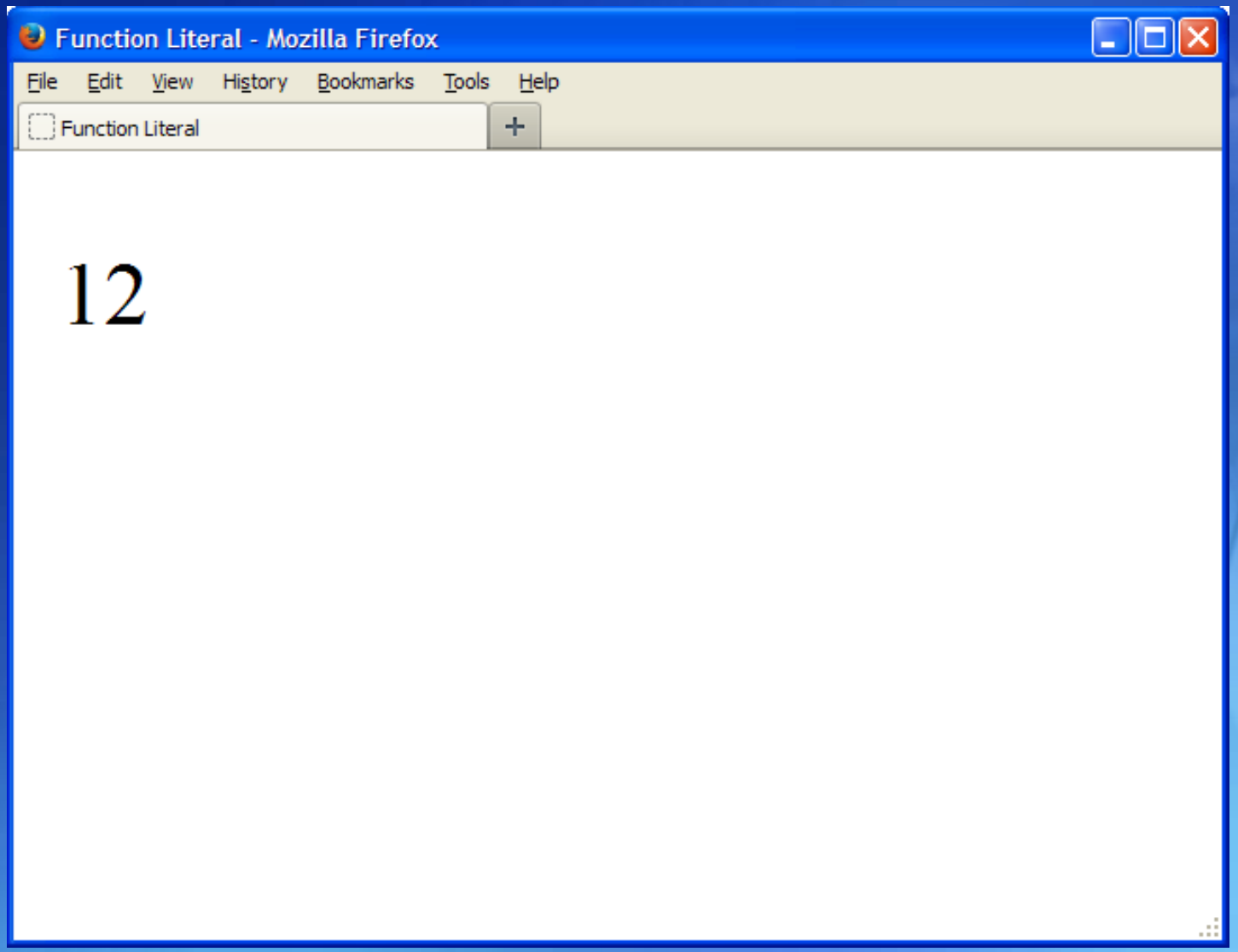
Lab5 : Function literals

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - funliteral.html

Lab5 : funliteral.html

```
4   <head>
5       <title> Function Literal </title>
6       <meta charset="utf-8">
7   </head>
8   <body>
9       <script type="text/javascript">
10          // invoking third argument as function
11          function funcObject(x,y,z) {
12              document.write("<p>" + (z(x,y)) + "</p>");
13          }
14
15          function testFunction() {
16              // third parameter is function
17              funcObject(3,4,function(x,y) { return x * y})
18          }
19
20          testFunction();
21      </script>
22  </body>
```

Lab5 : Result


Function Literal - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Function Literal
12

Functions and Recursion

- A function that invokes itself is called a *recursive* function.
- When a process must be performed more than once, with each new iteration of the process performed on the previously processed result.
- The use of recursion isn't common in JavaScript.
- Can also be memory- and resource-intensive, as well as complicated to implement and maintain.

Functions and Recursion (Cont.)

```
<script type="text/javascript">

   function factorial(n){
      if(n == 0 || n == 1)
         return 1;
      else return n * factorial(n-1);
   }
   document.write(factorial(1) + "<br />"); //print 1
   document.write(factorial(2) + "<br />"); //print 2
   document.write(factorial(3) + "<br />"); //print 6
   document.write(factorial(4));            //print 24
</script>
```

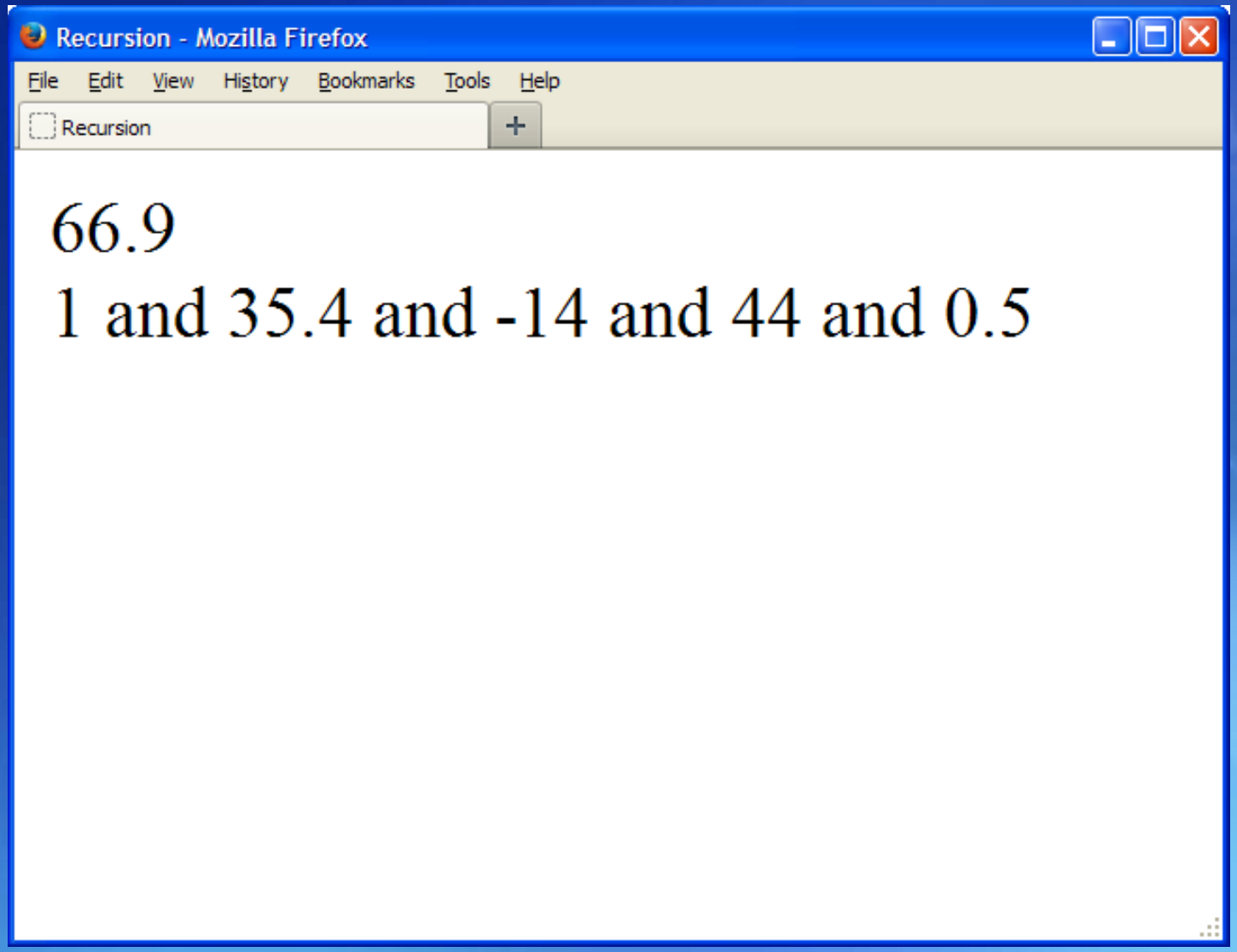
Lab6 : Recursion

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - recursion.html

Lab6 : recursion.html

```
<script>
    function runRecursion() {
        var addNumbers = function sumNumbers(numArray,indexVal,resultArray) {
            // recursion test
            if (indexVal == numArray.length)
                return resultArray;
            // perform numeric addition
            resultArray[0] += Number(numArray[indexVal]);
            // perform string concatenation
            if (resultArray[1].length > 0) {
                resultArray[1] += " and ";
            }
            resultArray[1] += numArray[indexVal].toString();
            // increment index
            indexVal++;
            // call function again, return results
            return sumNumbers(numArray,indexVal,resultArray);
        }
        // create numeric array, and the result array
        var numArray = ['1','35.4','-14','44','0.5'];
        var resultArray = new Array(0,''); // necessary for the initial case
        // call function
        var result = addNumbers(numArray,0, resultArray);
        // output
        document.writeln(result[0] + "<br />");
        document.writeln(result[1]);
    }
    runRecursion();
```

Lab6 : Result


Recursion - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Recursion
66.9
1 and 35.4 and -14 and 44 and 0.5

Callback Functions

- A *callback* is a reference to a piece of executable code that is passed as an argument to other code.*

```
<script type="text/javascript">

    function mySandwich(param1, param2, callback) {
        alert('Started eating my sandwich.\n\nIt has: ' + param1 + ', ' + param2);
        callback();
    }

    mySandwich('ham', 'cheese', function() {
        alert('Finished eating my sandwich.');
    });

</script>
```

* http://en.wikipedia.org/wiki/Callback_(computer_programming)

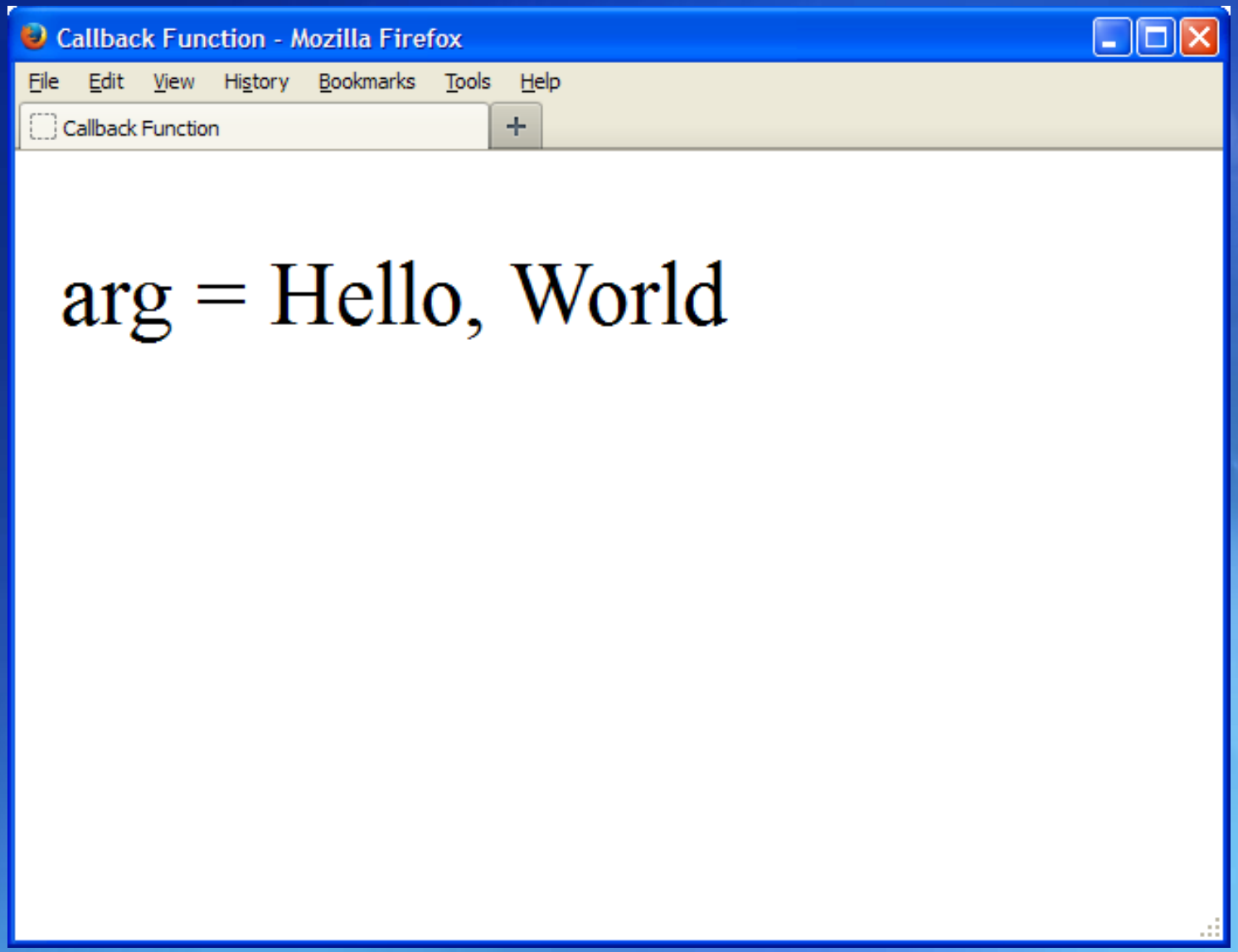
Lab7 : Callback Functions I

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - callback.html

Lab7 : callback.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> Callback Function </title>
5          <meta charset="utf-8">
6      </head>
7      <body>
8          <script>
9              function callbackTest(arg){
10                 document.write("<p>arg = " + arg + "</p>");
11             }
12             function test(args1, arg2, callback){
13                 if(typeof callback == "function") callback();
14             }
15             test('a', 'b', callbackTest("Hello, World"));
16         </script>
17     </body>
18 </html>
```

Lab7 : Result


Callback Function - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Callback Function
arg = Hello, World

Lab8 : Callback Functions II

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - callback1.html

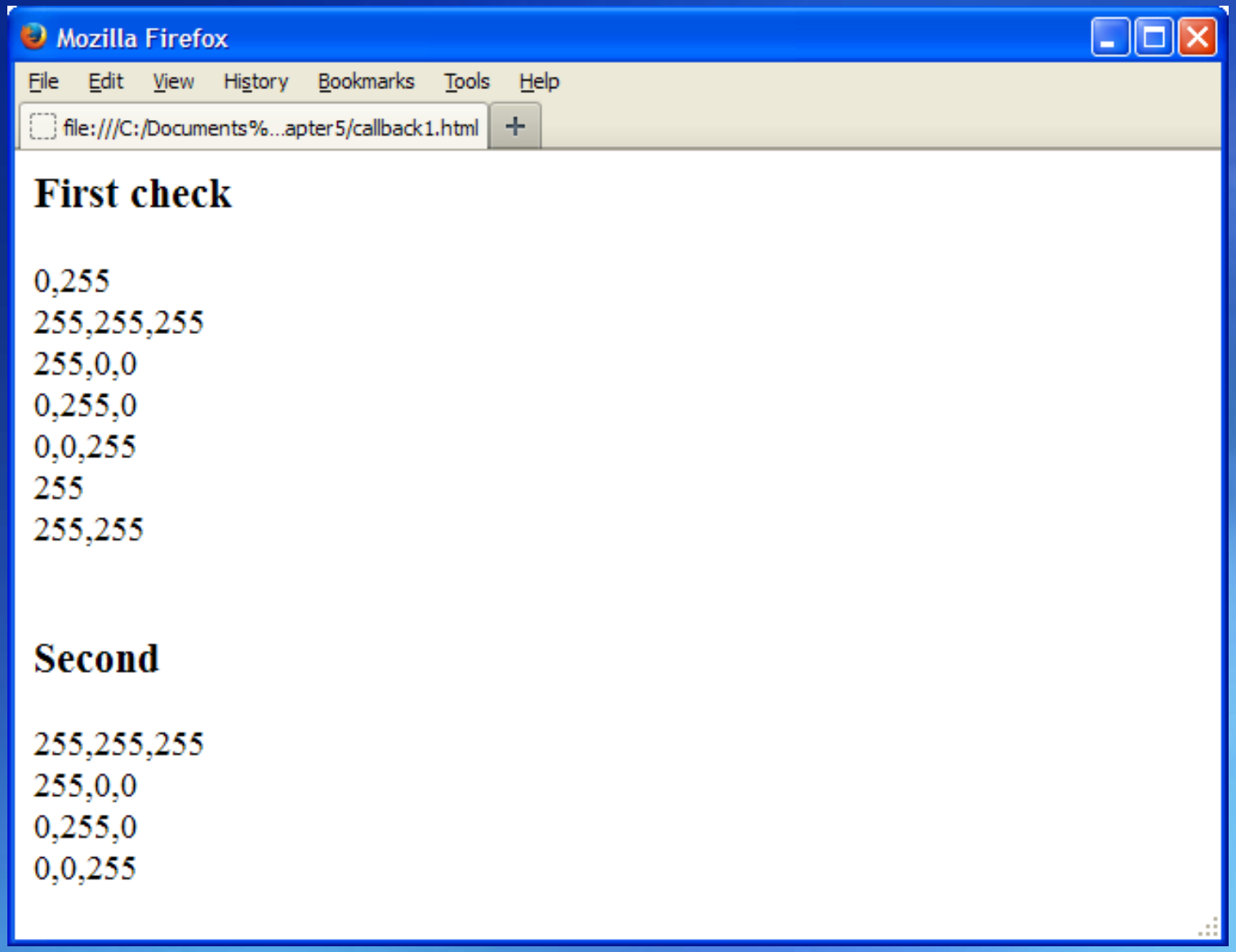
Lab8 : callback1.html (1/2)

```
8   <body>
9       <script>
10          // check color range callback function
11          function checkColor(element,index,array) {
12              return (element >= 0 && element < 256);
13          }
14          // check to ensure three RGB colors
15          function checkCount(element,index,array) {
16              return (element.length == 3);
17          }
```

Lab8 : callback1.html (2/2)

```
18          function testingCallbacks() {
19              // color array
20              var colors = new Array();
21              colors[0] = [0,262,255];
22              colors[1] = [255,255,255];
23              colors[2] = [255,0,0];
24              colors[3] = [0,255,0];
25              colors[4] = [0,0,255];
26              colors[5] = [-5,999,255];
27              colors[6] = [255,255,1204556];
28              // filter on color range
29              var testedColors = new Array();
30              for (var i in colors) {
31                  testedColors[i] = colors[i].filter(checkColor);
32              }
33              // print results of first round
34              document.writeln("<h3>First check</h3>");
35              for (i in testedColors) {
36                  document.writeln(testedColors[i] + "<br />");
37              }
38              // filter fewer than three values
39              var newTested = testedColors.filter(checkCount);
40              document.writeln("<br /><h3>Second</h3>");
41              // print survivors
42              for (i in newTested) {
43                  document.writeln(newTested[i] + "<br />");
44              }
45          }
46          testingCallbacks();
47      </script>
```

Lab8 : Result


Mozilla Firefox
File Edit View History Bookmarks Tools Help
file:///C:/Documents%...apter5/callback1.html
First check
0,255
255,255,255
255,0,0
0,255,0
0,0,255
255
255,255
Second
255,255,255
255,0,0
0,255,0
0,0,255

Nested Functions

- JavaScript allows for the nesting of functions.
- Inner function is active only in the outer function scope.

```
function outer (args) {
    function inner (args) {
        inner statements;
    }
}
```

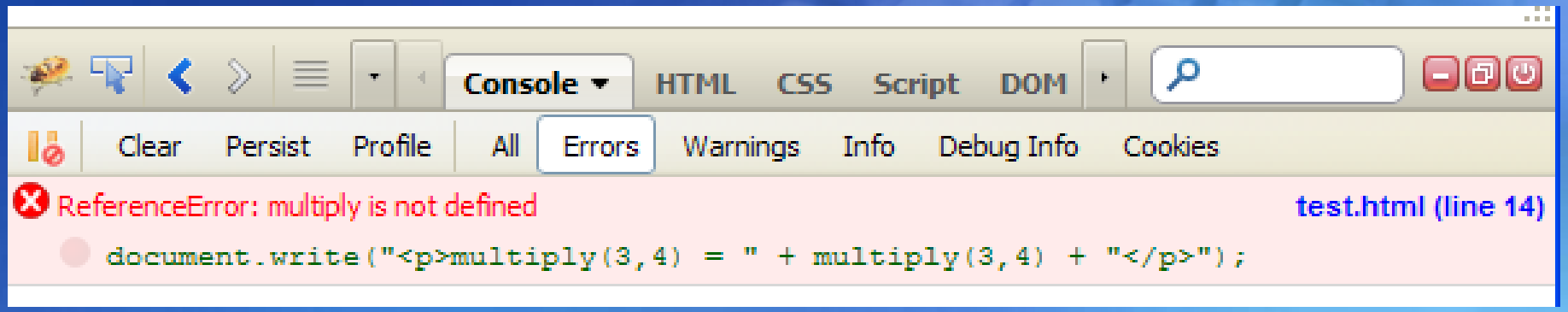
- Inner function can access to outer function's variable & arguments.
- Outer function cannot access to inner function's variables.
- Inner function cannot be called from the function's outer.

Nested Functions (Cont.)

```
<script type="text/javascript">
  function addSquare(a, b){
     function multiply(x, y){
        return x * y;
     }
     return multiply((a + b), (a + b));
  }
  document.write(addSquare(3, 4));  //print 49
</script>
```

Nested Functions (Cont.)

```
8   <script>
9       function addSquare(a, b){
10          function multiply(x,y){
11              return x * y;
12          }
13      }
14      document.write("<p>multiply(3,4) = " + multiply(3,4) + "</p>");
15  </script>
```



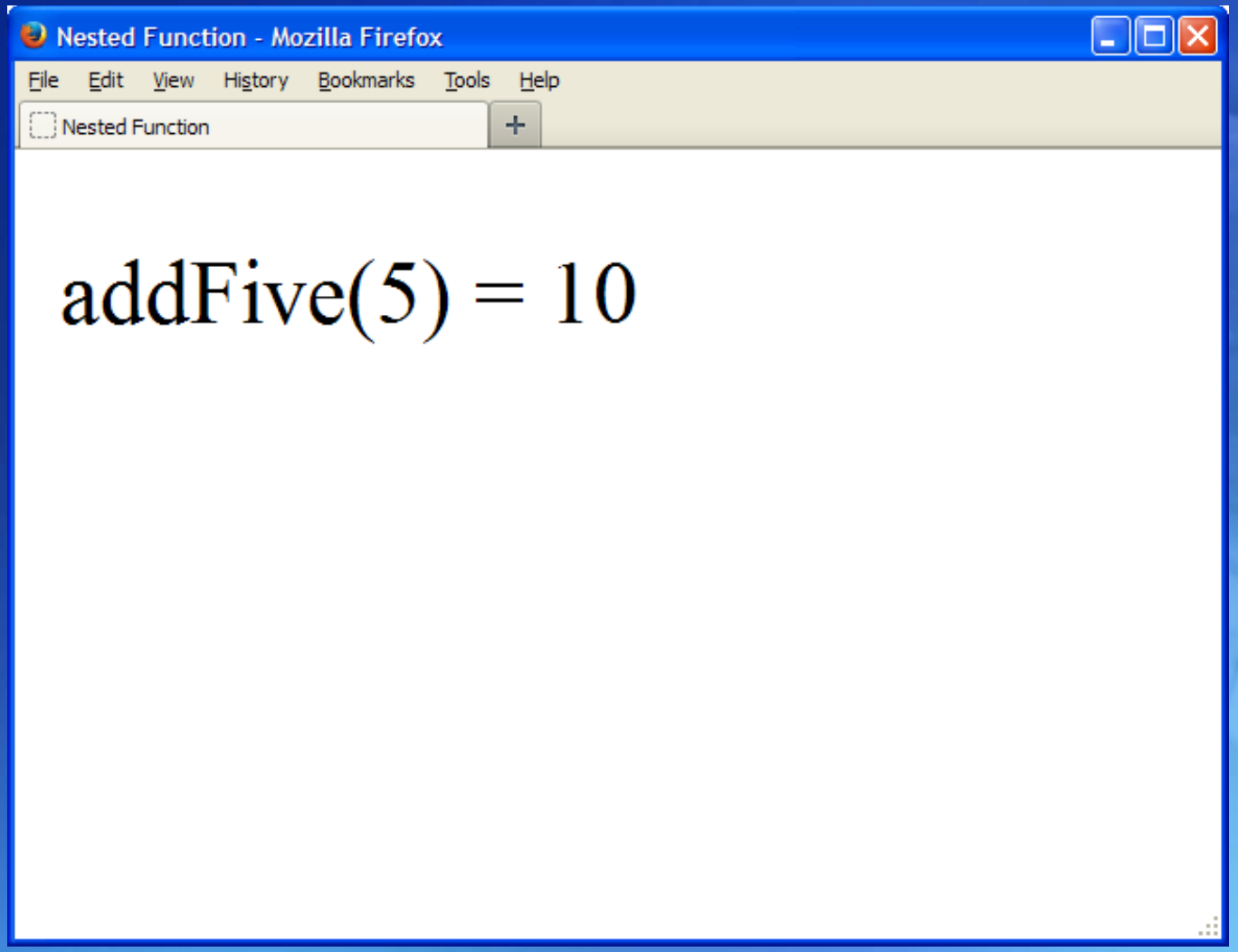
Lab9 : Nested Function I

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - nested.html

Lab9 : nested.html

```
1  <!DOCTYPE html>
2  <html>
3      <head>
4          <title> Nested Function </title>
5          <meta charset="utf-8">
6      </head>
7      <body>
8          <script>
9              function addFive(theVal){
10                 function addNumber(howmany){
11                     return theVal + howmany;
12                 }
13                 return addNumber(5);
14             }
15             document.write("<p>addFive(5) = " + addFive(5) + "</p>");
16         </script>
17     </body>
18 </html>
```

Lab9 : Result


Nested Function - Mozilla Firefox
File Edit View History Bookmarks Tools Help
Nested Function
addFive(5) = 10

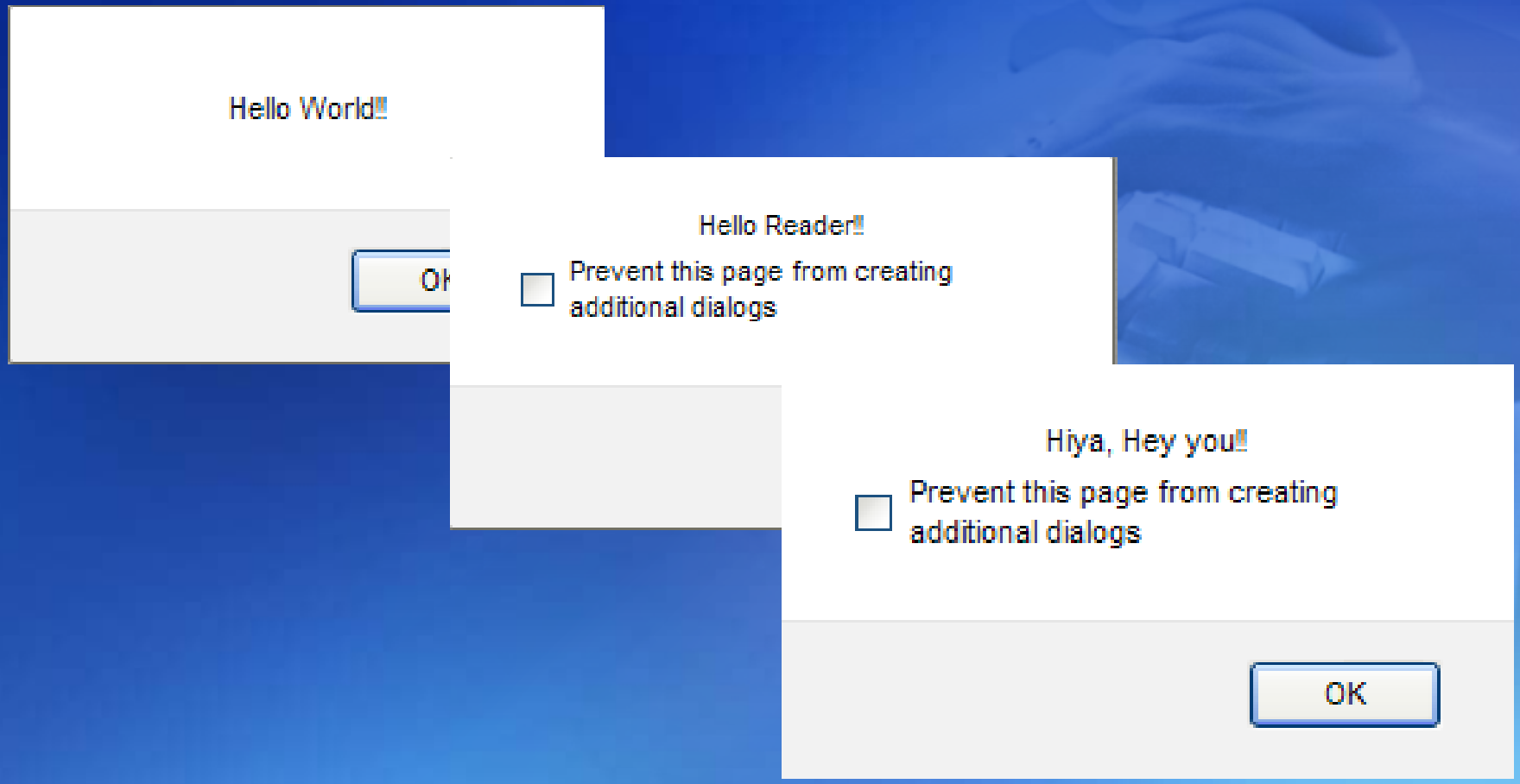
Lab10 : Nested Function II

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - nested1.html

Lab10 : nested1.html

```
<script>
    // outer function
    function outerFunc(base) {
        var punc = "!";
        // return inner function
        return function (ext) {
            return base + ext + punc;
        }
    }
    function processNested() {
        // create access to inner function
        var baseString = outerFunc("Hello ");
        // inner function still has access to outer function argument
        var newString = baseString("World!");
        alert(newString);
        // and still
        var notherString = baseString("Reader!");
        alert(notherString);
        // create another instance of inner function
        var anotherBase = outerFunc("Hiya, Hey ");
        // another local string
        var lastString = anotherBase("you!");
        alert(lastString);
    }
    processNested();
</script>
```

Lab10 : Result



Closures

- Is an expression (typically a function) that can have free variables together with an environment that binds those variables (that *closes* the expression).

```
<script type="text/javascript">
 function greetMe(name){
    return function(greeting){
       return greeting + " " + name;
    }
 }
 var greetSujan = greetMe("Sujan");
 document.write(greetSujan("Hi"));  //Hi Sujan
</script>
```

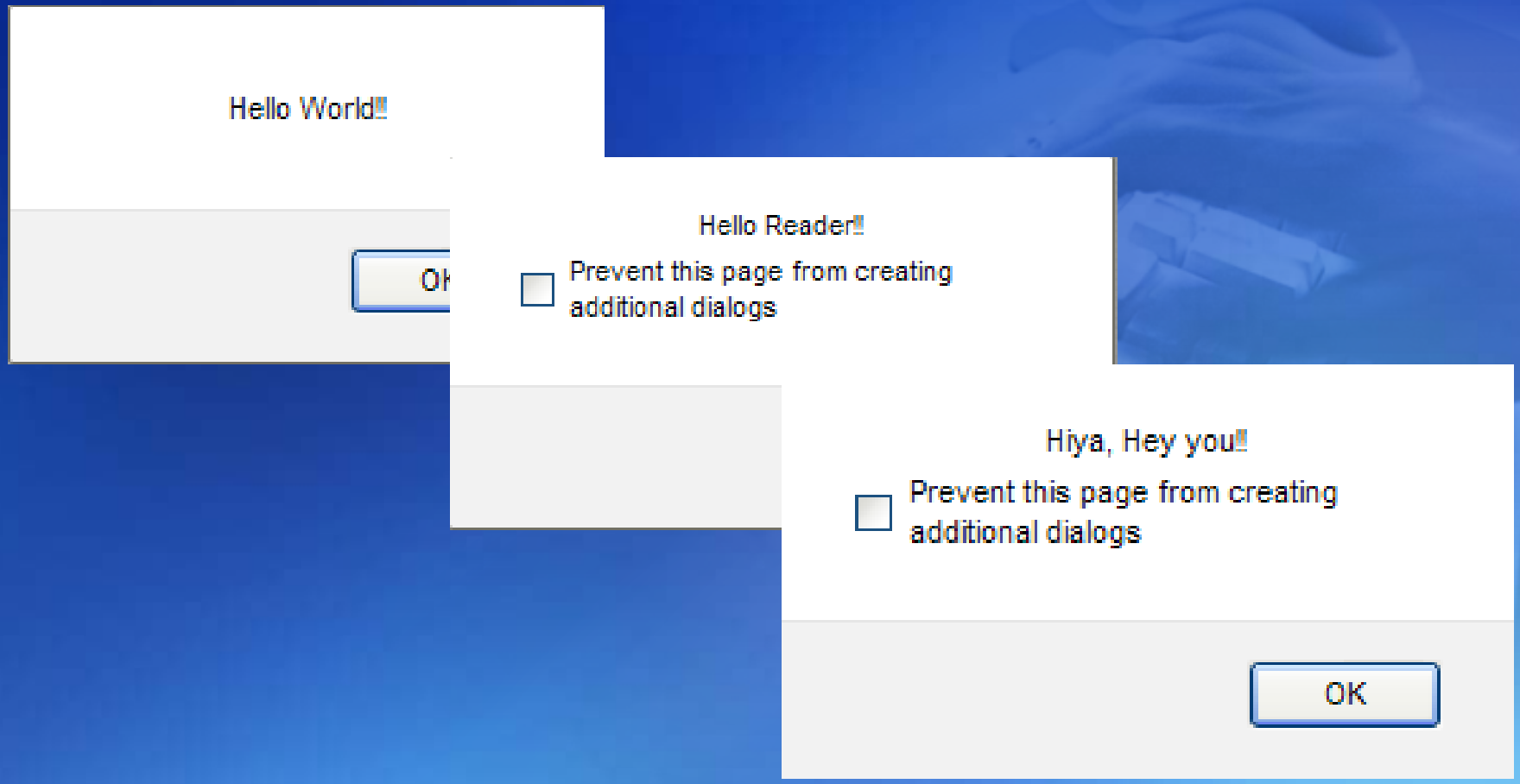
Lab10 : Closures I

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - closure.html

Lab10 : closure.html

```
8   <script>
9       // outer function
10      function outerFunc(base) {
11          var punc = "!";
12          // return inner function
13          return function (ext) {
14              return base + ext + punc;
15          }
16      }
17      function processNested() {
18          // create access to inner function
19          var baseString = outerFunc("Hello ");
20          // inner function still has access to outer function argument
21          var newString = baseString("World!");
22          alert(newString);
23          // and still
24          var notherString = baseString("Reader!");
25          alert(notherString);
26          // create another instance of inner function
27          var anotherBase = outerFunc("Hiya, Hey ");
28          // another local string
29          var lastString = anotherBase("you!");
30          alert(lastString);
31      }
32      processNested();
33  </script>
```

Lab10 : Result



Closures (Cont.)

```
<script type="text/javascript">
  function makeFish(color){
     function fish(){
        return color + " fish";
     }
     return fish;
  }

  var fish0 = makeFish('red');
  var fish1 = makeFish('blue');
  var str = fish0() + '; ' + fish1();
  document.write(str);   //red fish; blue fish
</script>
```

http://jrdodds.blogs.com/blog/2006/05/javascript_clos.html

Closures (Cont.)

```
<script type="text/javascript">

    var pet = function(name) {
        var getName = function() {
            return name;
        }
        return getName;
    }, myPet = pet("Vivie");
    document.write(myPet());

</script>
```

Lab11 : Closures II

- Web Browsers
  - IE10, Firefox, Google Chrome, Opera, Safari
- Text Editors
  - Notepad++ or Editplus
- Files
  - closure1.html

Lab11 : closure1.html

```
<script type="text/javascript">
  var createPet = function(name) {
     var gender;
     return {
        setName: function(newName) {   name = newName;  },
        getName: function() {   return name;   },
        getGender: function() {   return gender;   },
        setGender: function(newGender) {
           if(typeof newGender == "string" &&
              (newGender.toLowerCase() == "male" || newGender.toLowerCase() == "female")) {
                 gender = newGender;
           }
        }
     }
  }
  var pet = createPet("Vivie");
  document.write(pet.getName() + "<br />");   // Vivie
  pet.setName("Oliver");
  pet.setGender("male");
  document.write(pet.getGender() + "<br />");  // male
  document.write(pet.getName());               // Oliver
</script>
```

Lab11 : Result

Optional Arguments

```
<script type="text/javascript">
  function addFiveNumbers(a, b, c, d, e) {
      var result = 0 ;
      if(a) result += a;
      if(b) result += b;
      if(c) result += c;
      if(d) result += d;
      if(e) result += e;
      return result;
  }
  document.write(addFiveNumbers(1,2,3))
</script>
```

```
<script type="text/javascript">
  function addFiveNumbers(a, b, c, d, e) {
      if(!a) a = 0;
      if(!b) b = 0;
      if(!c) c = 0;
      if(!d) d = 0;
      if(!e) e = 0;
      return a + b + c + d + e;
  }
  document.write(addFiveNumbers(1,2,3));  //6
```

```
<script type="text/javascript">
  function addFiveNumbers(a, b, c, d, e) {
      a = a || 0;
      b = b || 0;
      c = c || 0;
      d = d || 0;
      e = e || 0;
      return a + b + c + d + e;
  }
  document.write(addFiveNumbers(1,2,3));  //6
</script>
```

The `arguments` Object

- Is a local variable available within all functions.
- Contains an entry for each argument passed to the function.
- The first entry's index starting at `0`.

```
<script type="text/javascript">
 function myConcat(separator) {
    var result = "";
    for (var i = 1; i < arguments.length; i++) {
       result += arguments[i] + separator;
    }
    return result;
 }
 document.write(myConcat(", ", "red", "orange", "blue") + "<br />");
 document.write(myConcat("; ", "elephant", "giraffe", "lion", "cheetah") + "<br />");
 document.write(myConcat(". ", "sage", "basil", "oregano", "pepper", "parsley"));
</script>
```

Function `length` Property

- Specifies the number of arguments expected by the function.

```
<script type="text/javascript">

function funcObject(x,y,z) {
    for (var i = 0; i < funcObject.length; i++) {
        document.writeln("argument " + i + ": " + arguments[i] + "<br />");
    }
}
funcObject(1,2,3);

</script>
```

Using Array

```
<script type="text/javascript">

  var a = [
     function(y) {
        return y;
     },
     function(y) {
        return y * y;
     },
     function (y) {
        return y * y * y;
     }
  ];
  document.write(a[0](5) + "<br />");  //returns 5,
  document.write(a[1](5) + "<br />");  //returns 25
  document.write(a[2](5) + "<br />");  //returns 125.

</script>
```

Predefined Functions

- **eval()**
  - Evaluates a string and executes it as if it was script code.
  - **eval(string)**

```
<script type="text/javascript">

  eval("x=10;y=20;document.write(x*y)");
  document.write("<br />" + eval("2+2"));
  document.write("<br />" + eval(x+17));

</script>
```

Predefined Functions (Cont.)

- **isFinite()**
  - Determines whether a value is a finite, legal number.
  - **isFinite(value)**

```
<script type="text/javascript">

 document.write(isFinite(123)+ "<br />"); //true
 document.write(isFinite(-1.23)+ "<br />"); //true
 document.write(isFinite(5-2)+ "<br />"); //true
 document.write(isFinite(0)+ "<br />");  //true
 document.write(isFinite("Hello")+ "<br />"); //false
 document.write(isFinite("2005/12/12")+ "<br />"); //false

</script>
```

Predefined Functions (Cont.)

- **isNaN()**
  - Determines whether a value is an illegal number.
  - **isNaN(value)**

```
<script type="text/javascript">

 document.write(isNaN(123)+ "<br />"); //false
 document.write(isNaN(-1.23)+ "<br />"); //false
 document.write(isNaN(5-2)+ "<br />");  //false
 document.write(isNaN(0)+ "<br />");  //false
 document.write(isNaN("Hello")+ "<br />");  //true
 document.write(isNaN("2005/12/12")+ "<br />"); //true

</script>
```

Predefined Functions (Cont.)

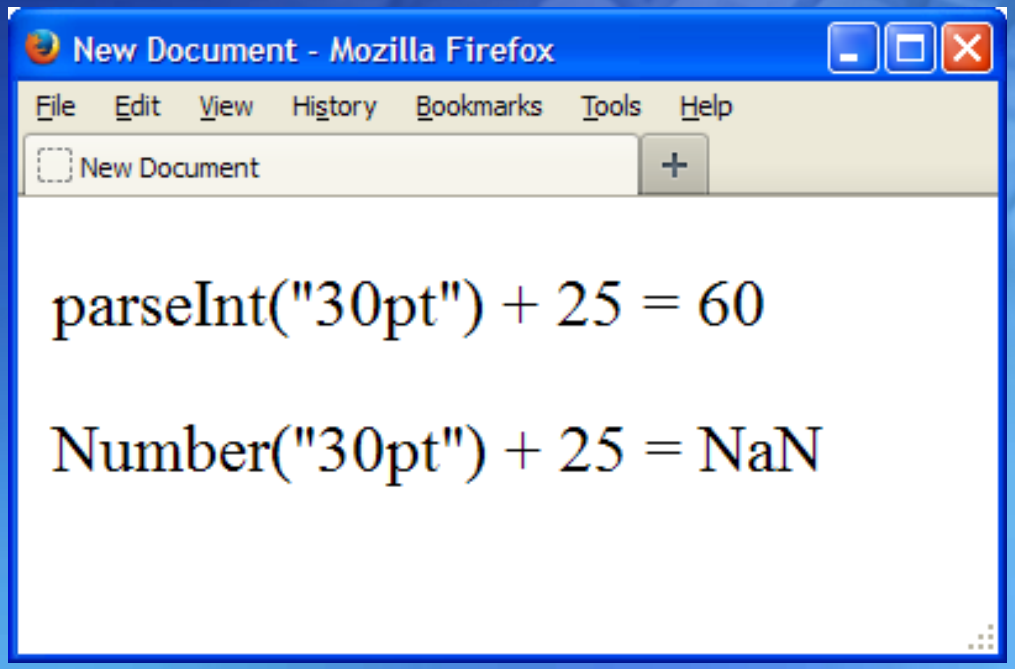
- **Number()**
  - Converts an object's value to a number.
  - If the value cannot be converted to al legal number, **NaN** is returned.
  - **Number(object)**

```
<script type="text/javascript">

  var test1= new Boolean(true);
  var test2= new Boolean(false);
  var test3= new Date();
  var test4= new String("999");
  var test5= new String("999 888");

  document.write(Number(test1)+ "<br />"); //1
  document.write(Number(test2)+ "<br />"); //0
  document.write(Number(test3)+ "<br />"); //1350578194406
  document.write(Number(test4)+ "<br />"); //999
  document.write(Number(test5)+ "<br />"); //NaN

</script>
```

Predefined Functions (Cont.)

- **Number()vs parseInt()**

```
8   <script>
9       document.write("<p>parseInt(\"30pt\") + 25 = " +
        (parseInt("35pt") + 25) + "</p>");
10      document.write("<p>Number(\"30pt\") + 25 = " +
        (Number("35pt") + 25) + "</p>");
11  </script>
```



Predefined Functions (Cont.)

- **encodeURI(uri)**
  - Encodes special characters, *except* :

    , / ? : @ & = + $ #

```
<script type="text/javascript">
   var uri="my test.asp?name=ståle&car=saab";
   document.write(encodeURI(uri)+ "<br />");
   //var uri="my%20test.asp?name=st%C3%A5le&car=saab";
</script>
```

Predefined Functions (Cont.)

- **encodeURIComponent(uri)**
  - Encodes special characters.
  - Encodes the following characters.

    , / ? : @ & = + $ #

```
<script type="text/javascript">
   var uri="http://w3schools.com/my test.asp?name=ståle&car=saab";
   document.write(encodeURIComponent(uri));
   //http%3A%2F%2Fw3schools.com%2Fmy%20test.asp%3Fname%3D...
</script>
```

Predefined Functions (Cont.)

```
function encodeStrings() {
    var uri = "http://burningbird.net/index.php?pagename=$1&page=$2";
    var sOne = encodeURIComponent(uri);
    var uri1 = "http://someapplication.com/?catsname=Zöe&URL=";
    var sTwo = encodeURI(uri1);
    var sOutput = "<p>Link is " + sTwo + sOne + "</p>";
    document.write(sOutput);
    var sOneDecoded = decodeURI(sTwo);
    var sTwoDecoded = decodeURIComponent(sOne);
    var sOutputDecoded = "<p>" + sOneDecoded + "</p>";
    sOutputDecoded += "<p>" + sTwoDecoded + "</p>";
    document.write(sOutputDecoded);
    /*
    Link is
    http://someapplication.com/?catsname=Z%C3%B6e&URL=http%3A%2F%2Fb
    urningbird.net%2Findex.php%3Fpagename%3D%241%26page%3D%242
    http://someapplication.com/?catsname=Zöe&URL=
    http://burningbird.net/index.php?pagename=$1&page=$2
    */
```

Predefined Functions (Cont.)

- **escape(uri)**
  - Encodes a string, *except* :

    * @ - _ + . /

```
<script type="text/javascript">

document.write(escape("Need tips? Visit W3Schools!"));
//print Need%20tips%3F%20Visit%20W3Schools%21

</script>
```

Predefined Functions (Cont.)

- **`unescape(string)`**
  - Decodes an encoded string.

```
<script type="text/javascript">

var str = "Need tips? Visit W3Schools!";
var str_esc = escape(str);
document.write(str_esc + "<br />");
//print Need%20tips%3F%20Visit%20W3Schools%21
document.write(unescape(str_esc));
//print Need tips? Visit W3Schools!

</script>
```

Test Your Knowledge : Quiz

1. A factorial is a number, *n*, that is the product of all numbers from 1 to *n*, usually written as 3! (1 × 2 × 3 or 6). Write a JavaScript function that uses recursion to figure out the factorial of a number given the number.

Test Your Knowledge : Quiz

1. A factorial is a number, *n*, that is the product of all numbers from 1 to *n*, usually written as 3! (1 × 2 × 3 or 6). Write a JavaScript function that uses recursion to figure out the factorial of a number given the number.

```
function findFactorial(n) {
   if (n == 0) return 1;
   return (n * findFactorial(n-1));
}
var num = findFactorial(4); // returns 24
```

Test Your Knowledge : Quiz (Cont.)

2. How can a function modify variables outside its scope? Write a function that takes an array of numbers from 1 to 5 and replaces the entries with string representations of the numbers (i.e., “one”, “two”, etc.).

Test Your Knowledge : Quiz (Cont.)

2. How can a function modify variables outside its scope? Write a function that takes an array of numbers from 1 to 5 and replaces the entries with string representations of the

```
function makeArray() {
    var arr = [1,5,3];
    alert(arr);
    alterArray(arr);
    alert(arr);
}
function alterArray(arrOfNumbers) {
    for (var i = 0; i < arrOfNumbers.length; i++) {
       switch(arrOfNumbers[i]) {
           case 1 : arrOfNumbers[i] = "one"; break;
           case 2 : arrOfNumbers[i] = "two"; break;
           case 3 : arrOfNumbers[i] = "three"; break;
           case 4 : arrOfNumbers[i] = "four"; break;
           case 5 : arrOfNumbers[i] = "five"; break;
       }
    }
}
```

Test Your Knowledge : Quiz (Cont.)

3. Create a function that takes a data object and a function as parameters and invokes the function using the data object.

Test Your Knowledge : Quiz (Cont.)

3. Create a function that takes a data object and a function as parameters and invokes the function using the data object.

```
function invokeFunction(dataObject, functionToCall) {
    functionToCall(dataObject);
}
var funcCall = new Function('x','alert(x)');
invokeFunction ('hello', funcCall);
```