INFO 151
Web Systems and Services

Week 4 (T2)

Dr Philip Moore

Dr Zhili Zhao

Course Overview

Weeks 1 – 3

- Introduction to Web Systems and Services
- Creating Web-Pages and Web-Sites with a Markup Language
- Introductory HTML 4 and HTML 5 with CSS

Weeks 4 – 6

- Client-Side Web Programming
- Introductory JavaScript

Weeks 7 – 9

- Server-side Programming
- Introductory PHP
- Introduction to Database, SQL, and MySQL

Sources of Resources

- The sources of information and resources for JavaScript may be found at the:
  - w3schools.com web-site: (url: https://www.quanzhanketang.com/)
- The w3schools.com web-site available to you has limited resources for PHP
  - For information on PHP see the recommended course text book:
    - Sams Teach Yourself PHP, MySQL & JavaScript All-in-One Sixth Edition

JavaScript Functions
Variable scope

Session Overview

- In this session we will introduce JavaScript Functions including:
  - syntax / prototypes / invocation (calling) / return / functions as variables
- We will consider:
  - variable scope and lifetime
  - Variable scope and closure
- We will provide worked examples showing:
  - The JavaScript <script> embedded in an HTML file with documentation
  - The output achieved

Functions

JavaScript Functions

- Functions form an essential element in JavaScript programming
- A function is defined once and may be re-used (i.e., *executed* (or '*called*') multiple times. This promotes:
  - The same code (function) can be re-used multiple times with different *arguments* and *variable values* (to produce different results)
  - Code re-use which improves the code by reducing errors
  - Reduces the need to write a specific operation multiple times
- A JavaScript function
  - Is a block of code designed to perform a particular task.
  - Is executed when "something" ***invokes*** it (or '***calls***' it)

JavaScript Function Syntax

- A JavaScript function is defined with the *`function`* keyword, followed by a *`name`*, followed by parentheses (…)
- Function names can contain letters, digits, underscores, and dollar signs (the same rules as *`variables`*).
- The parentheses may include parameter names separated by commas: *`(parameter1, parameter2, ...)`*
- The code to be executed, by the function, is placed inside a block defined by curly brackets: `{ }`

JavaScript Function Prototype

- JavaScript functions follow a pre-defined pattern (termed a prototype)
- The function prototype is as follows:

```
function name(parameter1, parameter2, ...) {
    code to be executed
}
```

- Function *parameters* are listed inside the parentheses () in the function definition
- A function can contain no passed parameters
- Function *arguments* are the *values* received by the function when it is invoked
- Inside the function, the arguments (the parameters) behave as local variables

Function Invocation

- When a function is assigned to a property of an *object* it is termed a *method* of that *object*
  - Within the body of the method the keyword *this* refers to the *object*
- Within the body of a function the `arguments [] array` contain the complete set of `arguments` passed to the function
  - The JavaScript code inside the function will execute when "something" *invokes* (or *calls*) the function
- JavaScript is essentially an *event-driven* program, *events* can include:
  - When a user *clicks* a button in a form (or) when it is *called* from JavaScript code

Function Example

- The following example of a JavaScript function
- From the following slides we can see:
  - Lines 14 – 21: the JavaScript `<script>`
  - Line 15: the *variable declaration* and *assignment*
  - Line 16: the variable *result* is *declared* and *assigned* with the *value* returned by the *function*
  - Line 17: outputs the *result* to the web-browser
  - Line 18: the function prototype with the (two) variables passed to the function for processing
  - Line 19: calculates the *return result* and *returns* the *result* to line 16
- The output is shown in the embedded NetBeans web kit

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
  - Site Root
    - index.html
  - Unit Tests
- Precedence_Test
- Precedence test

Navigator | Browser DOM

- JavaScript
  - myFunction(n1, n2) : Number
  - document
    - getElementById
      - innerHTML : Number
  - n1 : Number
  - n2 : Number
  - result : Number
- CSS
  - Ids
    - #demo
- HTML
  - html
    - head
      - title
    - body
      - h2
      - p
      - p id=demo
      - script

index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example of a JavaScript function
4  The myFunction function takes two variables and returns a value
5  -->
6  <html>
7      <head>
8          <title>JavaScript Function</title>
9      </head>
10     <body>
11         <h2>JavaScript Function Example</h2>
12         <p>This example calls a function which performs a calculation and
13         <p id="demo"></p>
14         <script>
15             var n1 = 4; var n2 = 3;
16             var result = myFunction(n1, n2);
17             document.getElementById("demo").innerHTML = result;
18             function myFunction(n1, n2) {
19                 return n1 * n2;
20             }
21         </script>
22     </body>
23 </html>
24
25
26
27
```

CSS Styles

Selection | Document

<No Element Selected>

No Rule Selected

<No Properties>

Variables | Call Stack | Breakpoints | Output

24:1 | INS

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
  - Site Root
    - index.html
  - Unit Tests
- Precedence_Test
- Precedence test
- Return_Function
- Self_Invoke

Navigator | Browser DOM

http://localhost:8383/JS_Function_Ex...

- html
  - head
    - title
  - body
    - h2
    - p
    - p#demo
    - script

PrecedenceTest.java | index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example of a JavaScript function
4  The myFunction function takes two variables and returns a value
5  -->
6  <html>
7      <head>
8          <title>JavaScript Function</title>
```

Variables | Call Stack | Breakpoints

Name

JavaScript Function

http://localhost:8383/JS_Function_Example/index.html

100%

JavaScript Function Example

This example calls a function which performs a calculation and returns the result:

12

CSS Styles

<No Element Selected>

No Rule Selected

<No Properties>

1:1 INS

Function Return

Function Return

- JavaScript functions receive input parameters and process the inputs to produce an output (the result)
- To process the inputs the body of the function will implement variables and statements to compute the result
- If the function was invoked from a *statement*, JavaScript will "*return*" to execute the code after the invoking statement
- The result is returned to the caller by the return statement
  - When JavaScript reaches a *return statement* the function will stop executing
  - Functions often compute a *return value* the return value is "returned" back to the "caller":
- A function can return **only one result**

Simple Function Return Example

- The following example shows a simple JavaScript function
- From the following slides we can see:
  - Lines 10 – 14: is the JavaScript `<script>` located in the `<head>`
  - Lines 11 – 13: is the `total()` function (located in the `<head>`)
  - Lines 17 – 18: is the `<script>` in the `<body>`
  - Line 18: the `total()` function (located in the `<head>`) is called and the parameters passed to the function
  - Lines 24 – 26: outputs the result from the function
- The output is shown in the embedded NetBeans web kit

Return_Function - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Files | Services

- Return_Function
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- JavaScript
  - total(numA, numB)
- HTML
  - html
    - head
    - body
      - script
      - p
      - p

index.html

Source | History

```
<!DOCTYPE html>
<!--
Example HTML / JavaScript function which returns a value
-->
<html>
    <head>
        <title>Returning a Function</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
            <script type="text/javascript">
                function total(numA,numB) {
                    return numA + numB;
                }
            </script>
    </head>
    <body>
        <script type="text/javascript">
            document.write(total(2,3));
        </script>
        <p>The script in the body section calls a function with two arguments: 2 and 3.
        <p>The function returns the sum of these two arguments.
    </body>
</html>
```

html > body > p

21:64 INS

Returning a Function

localhost:8383/Return_Function/index.html

Apps Bookmarks Outlook Google Google Scholar Google Sites BBC News Amazon Kindle Amazon CC EDAS EasyChair Other bookmarks

5

The script in the body section calls a function with two arguments: 2 and 3.

The function returns the sum of these two arguments.

Function Return Example with Variables

- The following example shows a JavaScript function
- From the following slides we can see:
  - Lines 12 – 16: is the JavaScript `<script>` located in the `<head>`
  - Lines 13 – 15: is the `total()` function (located in the `<head>`)
  - Lines 21 – 27: is the `<script>` in the `<body>`
  - Line 23: the variable *result* is defined and is *assigned* the function call
  - Lines 24 – 26: outputs the result from the function
- The output is shown in the embedded NetBeans web kit

Return_Function - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
- Return_Function
  - Site Root
    - index.html
  - Unit Tests
  - Important Files
- Self Invoke

Navigator

- JavaScript
  - total(num1, num2) : Number
  - n1 : Number
  - n2 : Number
  - result : Number
- HTML
  - html
    - head
      - title
      - meta
      - meta
      - script
    - body
      - p
      - script

index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example JavaScript function which returns a value
4  In this example the function is located in the <head>
5  The function is passed two variables (values) and returns the result
6  -->
7  <html>
8      <head>
9          <title>Returning a Function</title>
10         <meta charset="UTF-8">
11         <meta name="viewport" content="width=device-width, initial-scale=1.0">
12         <script type="text/javascript">
13                 function total(num1,num2) {
14                     return num1 * num2;
15                 }
16         </script>
17     </head>
18     <body>
19         <p>The script in the body calls a function in the head <br>
20         The function receives two arguments: (n1, n2)</p>
21         <script type="text/javascript">
22             var n1 = 10; var n2 = 8;
23             var result = (total(n1, n2));
24             document.write("The two argument values are " + n1 + " and " + n2 + "<br>");
25             document.write("The function multiplies " + n1 + " and " +n2 + "<br>");
26             document.write("The result returned is: " + result);
27         </script>
28     </body>
29 </html>
30
```

Variables | Call Stack | Breakpoints

1:1 INS

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
- Return_Function
  - Site Root
    - index.html
  - Unit Tests
  - Important Files
- Self_Invoke

Navigator | Browser DOM

http://localhost:8383/Return_Functio...

- html
  - head
    - title
    - meta
    - meta
    - script
  - body
    - p
    - script
    - br
    - br

index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example JavaScript function which returns a value
4  In this example the function is located in the <head>
5  The function is passed two variables (values) and returns the result
6  -->
7  <html>
8      <head>
```

Variables | Call Stack | Breakpoints

Name

Returning a Function

http://localhost:8383/Return_Function/index.html

100%

The script in the body calls a function in the head
The function receives two arguments: (n1, n2)

The two argument values are 10 and 8
The function multiplies 10 and 8
The result returned is: 80

CSS Styles

Selection | Document

<No Element Selected>

No Rule Selected

<No Properties>

1:1 INS

Self-Invoking Functions

Self Invoking Functions

- A Function expression(s) can be made *self-invoking*
- A *self-invoking* expression is invoked automatically without being called
- Function expressions will execute automatically if the expression is followed by parentheses ()
- self-invoking can not be used for a function declaration.
- Parentheses () must be added around the function to indicate that it is a function expression

Self Invoking Function Example

```
<!DOCTYPE html>
<html>
<body>
<p>Functions can be invoked automatically without being called:</p>
<p id="demo"></p>
<script>
(function () {
    document.getElementById("demo").innerHTML = “self invoked";
})();
</script>
</body>
</html>
```

bower.json | Gruntfile.js | gulpfile.js | package.json | .bowerrc | index.html

Source | History

```html
<!DOCTYPE html>
<!--
An example HTML / JavaScript self-invoking function
-->
<html>
    <head>
        <title>Self-Invoking Function</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <p>Functions can be invoked automatically without being called:</p>
        <p id="demo"></p>
        <script>
            (function () {
            document.getElementById("demo").innerHTML = "Hello! I called myself";
            })();
        </script>
    </body>
</html>
```

22:1 INS

Self-Invoking Function

localhost:8383/Self_Invoke/index.html

Apps Bookmarks Outlook Google Google Scholar Google Sites BBC News Amazon Kindle Amazon CC EDAS EasyChair Other bookmarks

Functions can be invoked automatically without being called:

Hello! I called myself

Self Invoking Function Example with a Variable

- The following example of a JavaScript function
- From the following slides we can see:
  - Lines 19 – 27: the JavaScript **`<script>`**
  - Lines 20 – 24: the *function()*
  - Line 21: the variable (n) declaration
  - Lines 23 – 24 and 24 – 25: the output to a web-browser
  - Line 26: the line of code (**`})();`**) *invokes* the *function*
    - The invocation process is similar to inner classes in the Java programming language
- The output is shown in the embedded NetBeans browser

Self_Invoke - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
- Self_Invoke
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator

- JavaScript
  - document
    - getElementById
      - innerHTML : String
  - n : Number
- CSS
  - Ids
    - #demo
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - p
      - p id=demo
      - script

index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example JavaScript self-invoking function
4  The function() has no passed variables
5  The function is self-invoked and calls itself
6  The output  is sent tot a web-browser
7  One output is the string "A self-invoking function which calls itself"
8  A second output is the string "The value of the variable n is: " + n = 47
9  -->
10 <html>
11     <head>
12         <title>Self-Invoking Function</title>
13         <meta charset="UTF-8">
14         <meta name="viewport" content="width=device-width, initial-scale=1.0">
15     </head>
16     <body>
17         <p>Functions can be invoked automatically without being called:</p>
18         <p id="demo"></p>
19         <script>
20             (function () {
21                 var n = 47;
22                 document.getElementById("demo").innerHTML =
23                     "A self-invoking function which calls itself <br>";
24                 document.getElementById("demo").innerHTML =
25                     "The value of the variable n is: " + n;
26             }) ();
27         </script>
28     </body>
29 </html>
30
```

Output | index.html saved. | 24:17 | INS

Self_Invoke - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Function_Example
- Precedence_Test
- Precedence test
- Self_Invoke
  - Site Root
    - index.html
  - Unit Tests
  - Important Files

Navigator | Browser DOM

http://localhost:8383/Self_Invoke/ind...

- html
  - head
    - title
    - meta
    - meta
  - body
    - p
    - p#demo
    - script

index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  An example JavaScript self-invoking function
4  The function() has no passed variables
5  The function is self-invoked and calls itself
6  The output  is sent tot a web-browser
7  One output is the string "A self-invoking function which calls itself"
8  A second output is the string "The value of the variable n is: " + n = 47
```

Variables | Call Stack | Breakpoints

Name	Type	Value
<Enter new watch>		

Self-Invoking Function

http://localhost:8383/Self_Invoke/index.html

100%

Functions can be invoked automatically without being called:

The value of the variable n is: 47

CSS Styles

<No Element Selected>

No Rule Selected

<No Properties>

Output

24:17 INS

Functions as Variables

Function as Variables

- Functions can be used the same way as variables are used
  - For example in all types of: formulas / assignments / calculations
- In the JavaScript code:
  - `function () { })();` (self-invoking)
  - `function();`
  - The `()` operator *Invokes* (or *calls*)the function
- Accessing a function *without* the () will return the function definition instead of the function result

Function as Variables

- In the worked examples shown in the NetBeans IDE we have demonstrated
  - The function prototype
  - Examples of functions and the passing of variables
  - The use of a function() as a variable
  - The assignment of a function() return result to a variable
  - The output to a web-browser with string operators
- Functions form the basis for JavaScript programming and the control of variables.

Worked Function Examples

Functions

- In the following worked example:
  - I show a function located in the <head> of the HTML file
  - The function is invoked (or called) in the <script> located in the <body> of the HTML file
  - The following slide shows:
  - The JavaScript program
  - The output
  - The program is written using EMACS
  - Note: the comments documenting the code and its purpose

function_example.html

```html
<!DOCTYPE html>
<!--
An example of a JavaScript function
Two variables are created and initialised
The variable values are passed to the function
The function comp[utes the calculation
The result is passed back to the calling function
-->
<html>
    <head>
      <title>JavaScript function example</title>
      <script>
        //calc function
        function calc(n1, n2) {
            return (n1 + n2);
        }
      </script>
    </head>
    <body>
      <p>
      An example of a JavaScript function. Two variables are
      created and initialised. The variable values are passed
      to the function. The function comp[utes the calculation.
      The result is passed back to the calling function.
      </p>
      <p>start of JavaScript script</p>
      <script>
        var a  = 4, b = 2;
        //call calc function and pass 2 data values
        document.write("Call the (calc(a, b)) <br>");
        var result = calc(a, b);
        document.write("<br>");
        document.write("The value of variable a is: " + a + "<br>");
        document.write("The value of variable b is: " + b + "<br><br>");
        document.write("Return the result from  (calc(a, b)) <br>");
        document.write("The result of adding " + a + " and " + b + " is: " + result + "<br>");
      </script>
        <p>end  of JavaScript script</p>
    </body>
</html>
```

```
U:---  function_example.html   All L32    (HTML+JS)
Wrote /Users/philip/Documents/Academic/LZU_Teaching_2018/151/Session_Slides/Week_5/html/function_exa
mple.html
```

file:///Users/philip/Documen

JavaScript function example | JavaScript function example

An example of a JavaScript function. Two variables are created and initialised. The variable values are passed to the function. The function comp[utes the calculation. The result is passed back to the calling function.

start of JavaScript script

Call the (calc(a, b))

The value of variable a is: 4
The value of variable b is: 2

Return the result from (calc(a, b))
The result of adding 4 and 2 is: 6

end of JavaScript script

Memoization (1)

- In the following worked examples I show the calculation of Fibonacci numbers
- The first example works but there are 453 function calls: I call the function 11 times and it calls itself 442 times
- In the second example there are 29 function calls: again I call the function 11 times but it calls itself only 18 times
  - The second example improves on the first example using a process termed *"memoization"* where the values of the function calls previously calculated values are remembered by the program

Memoization (2)

- In the following slides we can see the **`memo[]`** array hidden in a *closure* (introduced later in this tutorial)
- The reduction in the number of function calls results from the storage of previously calculated values in the **`memo[]`** array
  - When the program is run the function carries out a 'lookup' of the array and if the result exists it is immediately returned
- The following slides show:
  - The recursive function
  - The improved recursive function with *closure* using *memoization*

index.html | index.html

Source | History

```html
<!DOCTYPE html>
<!--
A JavaScript program to calculate fibonacci numbers
The program passes a value to an anonymous function
-->
<html>
    <head>
        <title>JavaScript Fibonacci Numbers</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div style="color: blue"><b>Fibonacci numbers</b></div>
        <script>
            var fib = function(n) { //anonymous function
                return n < 2 ? n : fib(n - 1) + fib(n - 2);
            };
            for(var i = 0; i <= 10; i++) {
                document.write('// ' + i + ': ' + fib(i) + '<br>');
            }
        </script>
    </body>
</html>
```

eb Browser | Web Browser | Web Browser | Web Browser | Improved JavaScript Fibon... | 24:1 | INS

index.html | index.html

Source | History

```
1  <!DOCTYPE html>
2  <!--
3  A JavaScript program to calculate fibonacci numbers
4  The program passes a value to an anonymous function
5  -->
6  <html>
7      <head>
8          <title>JavaScript Fibonacci Numbers</title>
9          <meta charset="UTF-8">
10         <meta name="viewport" content="width=device-width, initial-scale=1.0">
11     </head>
12     <body>
```

JavaScript Fibonacci Numb...

http://localhost:8383/JS_Fib/index.html

100%

Fibonacci numbers

// 0: 0
// 1: 1
// 2: 1
// 3: 2
// 4: 3
// 5: 5
// 6: 8
// 7: 13
// 8: 21
// 9: 34
// 10: 55

eb Browser | Web Browser | Web Browser | Web Browser | 24:1 | INS

index.html | index.html

Source | History

```html
<!DOCTYPE html>
<!--
A n improved JavaScript program to calculate fibonacci numbers
The program is more efficient (less function calls) than JS_Fib
-->
<html>
    <head>
        <title>Improved JavaScript Fibonacci Numbers</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div style="color: blue">
            <b>Improved JavaScript Fibonacci numbers program</b>
        </div>
        <script>
            var memoizer = function(memo, formula) {
                var recur = function(n) {
                    var result = memo[n];
                    if(typeof result !== 'number') {
                        result = formula(recur, n);
                        memo[n] = result;
                    }
                    return result;
                };
                return recur;
            };
            var fib = memoizer([0, 1], function(recur, n) {
                return recur(n - 1) + recur(n - 2);
            });
            for(var i = 0; i <= 10; i++) {
```

eb Browser | Web Browser | Web Browser | Web Browser | Improved JavaScript Fibon... | 37:1 | INS

index.html | index.html

Source | History

```html
<head>
    <title>Improved JavaScript Fibonacci Numbers</title>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
</head>
<body>
    <div style="color: blue">
        <b>Improved JavaScript Fibonacci numbers program</b>
    </div>
    <script>
        var memoizer = function(memo, formula) {
            var recur = function(n) {
                var result = memo[n];
                if(typeof result !== 'number') {
                    result = formula(recur, n);
                    memo[n] = result;
                }
                return result;
            };
            return recur;
        };
        var fib = memoizer([0, 1], function(recur, n) {
            return recur(n - 1) + recur(n - 2);
        });
        for(var i = 0; i <= 10; i++) {
            document.write('// ' + i + ': ' + fib(i) + '<br>');
        }
    </script>
</body>
</html>
```

eb Browser | Web Browser | Web Browser | Web Browser | Improved JavaScript Fibon... | 37:1 | INS

index.html | index.html

Source | History

```html
<!DOCTYPE html>
<!--
A n improved JavaScript program to calculate fibonacci numbers
The program is more efficient (less function calls) than JS_Fib
-->
<html>
    <head>
        <title>Improved JavaScript Fibonacci Numbers</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
```

Improved JavaScript Fibon...

http://localhost:8383/JS_Fib_Mem/index.html

100%

Improved JavaScript Fibonacci numbers program

// 0: 0
// 1: 1
// 2: 1
// 3: 2
// 4: 3
// 5: 5
// 6: 8
// 7: 13
// 8: 21
// 9: 34
// 10: 55

eb Browser | Web Browser | Web Browser | Web Browser | 37:1 | INS

Variable Scope
and
Static Variables

Scope

- Scope in a computer program controls the **`visibility`** and **`lifetime`** of variables
- In many high-level languages' variables defined within a block can be released when execution of the block is finished
- JavaScript does **`not`** include **`block`** scope (*even thought the syntax suggests it does*)
- In JavaScript ES6 (introduced in a later tutorial) the **`let`** keyword (shown later in this tutorial) has been introduced to provide a measure of block scope

Global and Local Scope

- In JavaScript there are two levels of scope:
  - **`Global`** scope
  - **`Local`** scope
- Any variable:
  - Declared outside of a function has **`Global`** scope
  - An identifier (variable) created without the **`var`** keyword is automatically converted into a **`Global`** variable
  - **`Global`** variables are accessible from anywhere in JavaScript code
  - All scripts and functions in a web-page can access global variables

Function Scope

- JavaScript does include function scope
- A *function variable* has *function scope*
  - Variables declared within a function has function scope
  - A variable declared within that function is only accessible from that function and any nested functions
  - Function *arguments* (parameters) work as *local* variables *inside* functions
  - An identifier (variable) created in a function without the **`var`** keyword is automatically converted into a **`Global`** variable

Local Scope

- Variables declared within a JavaScript function, become **`local`** to the function.
- **`Local`** variables have Function scope
  - They can only be accessed within the function
- Since local variables are only recognized inside their functions
  - Variables with the same name can be used in different functions
- **`Local`** variables are:
  - Created when a function executes
  - Deleted when the function is terminates
- Function arguments (*parameters*) are local variables inside functions

Global Variables and Scope

- **`Global`** JavaScript Variables:
  - A variable declared outside a function is a **`GLOBAL`** variable
  - A **`Global`** variable has **`Global`** scope: all scripts and functions on a web page can access it
  - Assigning a value to a variable that has not been declared will automatically create a **`Global`** variable
- There are some significant changes implemented in JavaScript ES:
  - This will be addressed in another tutorial
- The following worked examples show variable scope and the resulting output in range of JavaScript programs

index.html

Source History

```
15  <script>
16      var foo = function() { //anonymous function
17          var a = 3, b = 5;
18          document.write("Here: * a = " + a + " b = " + b + "<br>");
19          var bar = function() { //anonymous function
20              var b = 7, c = 11;
21              document.write("Here: ** b = " + b + " c = " + c + "<br>");
22              a += b + c;
23              document.write("Here: *** a = " + a + " b = " + b + " c = " + c + "<br>");
24          };
25          document.write("Here: **** a = " + a + " b = " + b + "<br>");
26          bar();
27          document.write("Here: ***** a = " + a + " b = " + b +"<br>");
28      };
29      document.write("Invoke foo(): " + foo());
30  </script>
```

A script showing the effect of scope on the variables and their values

Scope in JavaScript

http://localhost:8383/JS_Scope_Test/index.html

100%

Scope in JavaScript
Here: * a = 3 b = 5
Here: **** a = 3 b = 5
Here: ** b = 7 c = 11
Here: *** a = 21 b = 7 c = 11
Here: ***** a = 21 b = 5
Invoke foo(): undefined

index.html

Source History

```html
<!--
Variables declared within a JavaScript function become LOCAL to the function
Local variables have Function scope:
They can only be accessed from within the function
-->
<html>
    <head>
        <title>JavaScript variable scope example</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>JavaScript scope example</h2>
        <p>Outside myFunction() carName is undefined</p>
        <p id="demo1"></p>
        <p id="demo2"></p>
        <script>
            myFunction();
            function myFunction() {
                var carName = "Volvo";
                document.getElementById("demo1").innerHTML =
                                typeof carName + " " + carName;
                            }
                document.getElementById("demo2").innerHTML =
                                typeof carName;
        </script>
    </body>
</html>
```

index.html

Source History

```
2  <!--
3  Variables declared within a JavaScript function become LOCAL to the function
4  Local variables have Function scope:
5  They can only be accessed from within the function
6  -->
7  <html>
8      <head>
```

JavaScript variable scope...

http://localhost:8383/JS_Scope/index.html

100%

JavaScript scope example

Outside myFunction() carName is undefined

string Volvo

undefined

Variables | Call Stack | Breakpoints

Name	Type	Value
<Enter new watch>		

30:2 INS

index.html

Source History

```html
<!DOCTYPE html>
<!--
A variable declared outside a function, becomes GLOBAL
A global variable has global scope:
All scripts and functions on a web page can access it
-->
<html>
    <head>
        <title>JavaScript Global Scope</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
   <body>
       <h2>JavaScript Scope</h2>
       <p>A GLOBAL variable can be accessed from any script or function.</p>
       <p id="demo"></p>
       <script>
           var carName = "Volvo";
           myFunction();
           function myFunction() {
               document.getElementById("demo").innerHTML =
                                   "The display is: " + carName;
            }
        </script>
    </body>
</html>
```

27:1 INS

index.html

Source History

```
1  <!DOCTYPE html>
2  <!--
3  A variable declared outside a function, becomes GLOBAL
4  A global variable has global scope:
5  All scripts and functions on a web page can access it
6  -->
7  <html>
8      <head>
```

JavaScript Global Scope

http://localhost:8383/JS_Global_Scope/index.html

100%

JavaScript Scope

A GLOBAL variable can be accessed from any script or function.

The display is: Volvo

Variables | Call Stack | Breakpoints

Name	Type	Value
<Enter new watch>		

27:1 INS

index.html

```html
<!DOCTYPE html>
<!--
If you assign a value to a variable that has not been declared:
it will automatically become a GLOBAL variable
This code example will declare a global variable carName:
even if the value is assigned inside a function.
-->
<html>
    <head>
        <title>JavaScripe Global Variable Scope</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
   <body>
       <p>
           If you assign a value to a variable that has not been declared,
            it automatically becomes a GLOBAL variable:
        </p>
        <p id="demo"></p>
        <script>
            myFunction();
            // code here can use carName as a global variable
            document.getElementById("demo").innerHTML = "The display is: " + carName;
            function myFunction() {
                carName = "Volvo";
            }
        </script>
    </body>
</html>
```

Note: **carName = "Volvo"** has not used **var** : **carName** is automatically converted into a **Global** variable

index.html

Source History

```
1  <!DOCTYPE html>
2  <!--
3  If you assign a value to a variable that has not been declared:
4  it will automatically become a GLOBAL variable
5  This code example will declare a global variable carName:
6  even if the value is assigned inside a function.
7  -->
8  <html>
9      <head>
10         <title>JavaScripe Global Variable Scope</title>
```

JavaScripe Global Variabl...

http://localhost:8383/JS_Global_Scope_1/index.html

100%

If you assign a value to a variable that has not been declared, it automatically becomes a GLOBAL variable:

The display is: Volvo

Variables Call Stack Breakpoints

Name

30:1 INS

Variable Scope

- In JavaScript functions we must consider variable *scope*
- We have introduced:
  - **`Global`** variables
  - **`Local`** variables
  - Variables with **`function`** scope
  - Variables with **`block`** scope (a block is defined by the { … })
    - For example: a block can be a for loop (which includes nested for loops) or an IF statement (with nested IF statements)
- Developments in the JavaScript ES6 standard
  - The ECMA standard supports block-level scope using the **`let`** keyword
  - The following slides demonstrate block scope and **`let`**

JavaScript Functions and Scope

- In the JavaScript ES5 language standard there is block scope support
- However: JavaScript ES6 has introduced block scope using **`let`**
- JavaScript ES6 has a large number of revisions and additions when compared to ES5:
  - For comprehensive details of the Es6 JavaScript language specification see:
  - https://www.ecma-international.org/ecma-262/10.0/index.html#Title
  - https://developer.mozilla.org/en-US/docs/Web/javascript
- The following worked example shows the ES6 block scope and **`let`** and the use of the **`strict`** running mode shown used globally and locally

JS_block_let - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_block_let
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - blockDem() : undefined
- CSS
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - div
        - h3
      - div
      - script

index.html

Source | History

```
<!DOCTYPE html>
<!--
In JavaScript ES5 there is no block scope (even if it appears there is!)
In JavaScript ES6 there is block scope
This is a JavaScript program to demonstrate block scope and the let keyword
Also shown is the use of the "strict" running mode
-->
<html>
    <head>
        <title>JavaScript ES6 and Block Scope in Strict Mode</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <div style="color: blueviolet">
            <!-- Note: the style preference rules introduced in HTML and CSS -->
            <h3 style="color: olive"> <!-- the color used -->
                A Demonstration of block scope and let implemented using the struct mode
            </h3>
        </div>
        <div style="color: brown">
            <!-- Note: the style preference rules introduced in HTML and CSS -->
            <h4>
                Note: there is no output for (c) in the blockDem() function scope <br>
            </h4>
        </div>
        <script>
            "use strict"; //Note: the us of the "strict" running mode
                          //for the whole sctipt
            function blockDem() {
                //"use strict"; //Note: the us of the "strict" running mode
```

Web Browser | JavaScript ES6 and Block ... | 55:1 | INS

JS_block_let - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_block_let
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - blockDem() : undefined
- CSS
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - div
        - h3
      - div
      - script

index.html

Source | History

```
25            </h4>
26        </div>
27        <script>
28            "use strict"; //Note: the us of the "strict" running mode
29                          //for the whole sctipt
30            function blockDem() {
31                //"use strict"; //Note: the us of the "strict" running mode
32                            //just for the function
33                var a = 1;
34                if(a <= 1) {
35                    let b = 2;
36                    document.write("while loop <br>");
37                    while(b < 5) {
38                        let c = (b * 2);
39                        b++;
40                        document.write((a + c) + "*");
41                    }
42                    document.write("<br>");
43                    document.write("Output for (a) in if: " + (a) + "<br>");
44                }
45                document.write("Output for (a) in blockDem(): " + (a) + "<br>");
46                document.write("Output for (b) in blockDem(): " + (b) + "<br>");
47                document.write("Output for (c) in blockDem(): " + (c) + "<br>");
48            }
49            //the blockDem() function invocation
50            blockDem();
51
52        </script>
53    </body>
54 </html>
55
```

- Note: the use of **strict**
- It (**strict**) can be used globally for the whole script (or) locally just for a function
- Note: the use of **let** and **var** in the **if** statement and the **while** loop

Web Browser | JavaScript ES6 and Block ... | 55:1 | INS

Apache NetBeans 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

118.9/269.0MB

Projects | Services | Files

- JS_block_let
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - blockDem() : undefined
- CSS
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - div
        - h3
      - div
      - script

index.html

Source | History

```
<!DOCTYPE html>
<!--
In JavaScript ES5 there is no block scope (even if it appears there is!)
In JavaScript ES6 there is block scope
This is a JavaScript program to demonstrate block scope and the let keyword
Also shown is the use of the "strict" running mode
-->
```

JavaScript ES6 and Block ...

http://localhost:8383/JS_block_let/index.html

100%

A Demonstration of block scope and let implemented using the struct mode

Note: there is no output for (c) in the blockDem() function scope

while loop
5*7*9*
Output for (a) in if: 1
Output for (a) in blockDem(): 1

Web Browser 55:1 INS

Variable Lifetime

- The lifetime of a JavaScript variable is defined the scope
- **`Global`** JavaScript variables
  - Are available across a JavaScript program while the program is running
    - The program is running while the web-page is open
- In a web browser
  - Global variables are *destroyed* (deleted) when the web-page (or tab) is closed
  - However: **`global`** variables remain available to new web-pages loaded into the same *window object*
- Local (function scope) variables
  - The function scope variables are *created* when the function **`object`** is *created* (called)
  - Function scope variables are *destroyed* (deleted) on function *exit*

Why is Variable Scope Important?

- As we have seen JavaScript is based on *a Global object*
  - This can be a problem for JavaScript programming
  - Programmers must know *how* and *when* variable is modified
- Why is this important?
  - Unrestricted access to a variable (a feature of the *Global object*) may be required and useful
  - However: *uncontrolled* changes to the variable (data) can be a significant problem
  - Uncontrolled (*Global*) variables may result in unreported errors
    - Such errors are not syntax errors but logic errors

Static Variables

- **`Global`** variables can be a problem for JavaScript
- There may be times when a **`static`** variable is needed in a JavaScript function
  - **`Static`** variables have restricted (local) scope when used within a function and their values cannot be modified outside a function / and maintain their value between function calls
  - **`Global`** variables can be modified outside a function
- Why is this important?
  - Using global variables in a function may be required (or) it may result in (unreported) logic errors

Variable Scope and Static Variables

- **`Global`** variables can be modified outside a function
- In contrast consider where variables are declared as function parameters:
  - Variables declared **`static`** are destroyed on the function's *exit*
  - A **`static`** variable will not lose its value when the function exits and will still hold that value should the function be called again
- Why is this important?
  - A global variable may be required (such as a constant)
  - However: local (function) scope may be required as there are potential issues with variable naming and variable name re-use (with logic errors)

Variable Scope and Static Variables

- Why is variable scope and static variables important?
- A **`Global`** variable (e.g., **`PI`**) can be modified outside a function
  - This is a problem for a *constant* where a variable used multiple times (such as a *pre-set value for PI*) will must not be modified
- However: *local* (function) scope may be required
  - To avoid potential issues with *variable naming*
  - To restrict the *visibility* of a variable (the scope)
  - To prevent problems with *name re-use*
  - Such problems may result in *logic errors* (correct syntax but incorrect result)
  - Logic errors are not reported and are very hard to identify and correct

Variable Scope and Static Variables

- As we have seen functions provide a means by which variable scope and access may be controlled
- Where variables are declared as function parameters:
  - A `static` variable has restricted local scope in a function
  - The values for a `static` variable when used in a function cannot be modified outside the function
  - A `static` variable will not lose its value when the function exits and will still hold that value if the function is called again
- These properties help to
  - Understand the program logic
  - Control variable visibility

JavaScript Objects and Functions as Variables

- In JavaScript: **`objects`** and **`functions`** are also variables
- When assigning a value to an undeclared variable:
  - The variable will automatically become a **`Global`** variable
- The reason for this is:
  - In JavaScript: all top-level variables are grouped in a common *namespace* (the **`Global object`**)
  - Global scope is the complete JavaScript environment
- The Global design of JavaScript is potentially an issue

Global Scope in HTML

- In JavaScript controlling the scope and lifetime of a JavaScript variable can be achieved using functions
- Controlling **`Global`** Variables in HTML
  - In JavaScript **`global`** scope is the complete JavaScript environment
  - In HTML **`global`** scope is the window object
  - All **`global`** variables belong to the window object)
- In web browsers
  - **`Global`** variables are deleted when the browser window (or tab) is closed
  - **`Global`** variables remain available to new pages loaded into the same window
  - This is clearly a potential source of a broad range of problems

JS_Global_Scope_HTML - NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Global_Scope
- JS_Global_Scope_1
- JS_Global_Scope_HTML
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - document
    - getElementById
      - innerHTML : String
  - carName : String
- CSS
  - Ids
    - #demo
- HTML
  - html
    - head
      - title
      - meta
      - meta
    - body
      - h2
      - p
        - b
      - p id=demo
      - script

index.html

Source | History

```
<!DOCTYPE html>
<!--
Global Variables in HTML
With JavaScript, the global scope is the complete JavaScript environment
In HTML, the global scope is the window object
All global variables belong to the window object
-->
<html>
    <head>
        <title>Global Variables in HTML</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
    </head>
    <body>
        <h2>JavaScript Scope: Global Variables in HTML</h2>
        <p>
            In HTML, global variables defined with <b>var</b>
            global variables will become window variables.</p>
        <p id="demo"></p>
        <script>
            var carName = "Volvo";
            // code here can use window.carName
            document.getElementById("demo").innerHTML = "The display is: " + window.carName;
        </script>
    </body>
</html>
```

Variables | Call Stack | Breakpoints

27:1 INS

NetBeans IDE 8.2

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_Global_Scope
- JS_Global_Scope_1
- JS_Global_Scope_HTML
  - Site Root
    - index.html
  - Unit Tests

Navigator | Browser DOM

http://localhost:8383/JS_Global_Scop...

- html
  - head
    - title
    - meta
    - meta
  - body
    - h2
    - p
    - p#demo
    - script

index.html

Source | History

```
4   With JavaScript, the global scope is the complete JavaScript environment
5   In HTML, the global scope is the window object
6   All global variables belong to the window object
7   -->
8   <html>
9       <head>
10          <title>Global Variables in HTML</title>
11          <meta charset="UTF-8">
12          <meta name="viewport" content="width=device-width, initial-scale=1.0">
```

Global Variables in HTML

http://localhost:8383/JS_Global_Scope_HTML/index.html

100%

JavaScript Scope: Global Variables in HTML

In HTML, global variables defined with **var** global variables will become window variables.

The display is: Volvo

Variables | Call Stack | Breakpoints

Name	Type	Value
<Enter new watch>		

23:74 INS

Variable Scope
and
Closure

Closure

- In JavaScript *closure* is an important (and least understood) concept
- *Closure* is a way to remember and continue to access a function's scope (the variable(s)) when a function has terminated
- In the following worked example:
  - The **`reference`** to the inner **`add(…)`** is returned with each call to the outer **`makeAdder(…)`**
  - **`var plusOne = makeAdder(1)`** : sets the value of (**`x`**) to (**`1`**)
  - **`plusOne(3)`** : now sets the value of (**`y`**) to (**`3`**)
  - The inner **`add(…)`** function now **`adds x + y`** (1 + 3) and **`returns 4`**

JS_MakeAdder - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_MakeAdder
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - makeAdder(x) : makeAdder.add
    - add(y) : x
  - plusOne : makeAdder.add
- CSS
  - Elements
    - body
    - div
    - h4
  - Rules
    - body
    - div
    - h4
- HTML
  - html
    - head
      - title
      - meta
      - meta
      - style
    - body
      - div
        - h4
      - script

index.html

Source | History

```
<!DOCTYPE html>
<!--
A JavaScript script to test demonstrate closure in JavaScript
-->
<html>
    <head>
        <title>JavaScript Closure</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            body {background-color: powderblue;}
            h4 {font-style: italic}
            div    {color: red;}
        </style>
    </head>
    <body>
        <div><h4>A demonstration of closure</h4></div>
        <script>
            function makeAdder(x) { //outer function
                document.write("* x = " + x + "<br>");
                function add(y) {   //inner function
                    document.write("** y = " + y + "<br>");
                    var result = (x + y);
                    return result;
                };
                return add;
            }
            document.write("call to makeAdder(1) where ");
            var plusOne = (makeAdder(1));
            plusOne(3);
            document.write("*** plusOne(3) = " + plusOne(3) + "<br>");
```

Web Browser | Web Browser | Web Browser | Web Browser | Web Browser | Web Browser | JavaScript Closure

36:1 INS

JS_MakeAdder - Apache NetBeans IDE 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

Projects | Services | Files

- JS_MakeAdder
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - makeAdder(x) : makeAdder.add
    - add(y) : x
  - plusOne : makeAdder.add
- CSS
  - Elements
    - body
    - div
    - h4
  - Rules
    - body
    - div
    - h4
- HTML
  - html
    - head
      - title
      - meta
      - meta
      - style
    - body
      - div
        - h4
      - script

index.html

Source | History

```
6   <head>
7       <title>JavaScript Closure</title>
8       <meta charset="UTF-8">
9       <meta name="viewport" content="width=device-width, initial-scale=1.0">
10      <style>
11          body {background-color: powderblue;}
12          h4 {font-style: italic}
13          div    {color: red;}
14      </style>
15  </head>
16  <body>
17      <div><h4>A demonstration of closure</h4></div>
18      <script>
19          function makeAdder(x) { //outer function
20              document.write("* x = " + x + "<br>");
21              function add(y) {   //inner function
22                  document.write("** y = " + y + "<br>");
23                  var result = (x + y);
24                  return result;
25              };
26              return add;
27          }
28          document.write("call to makeAdder(1) where ");
29          var plusOne = (makeAdder(1));
30          plusOne(3);
31          document.write("*** plusOne(3) = " + plusOne(3) + "<br>");
32          document.write("*** plusOne(6) = " + plusOne(6) + "<br>");
33      </script>
34  </body>
35 </html>
36
```

Web Browser | Web Browser | Web Browser | Web Browser | Web Browser | Web Browser | JavaScript Closure

36:1 INS

Apache NetBeans 11.1

File Edit View Navigate Source Refactor Run Debug Profile Team Tools Window Help

141.2/348.0MB

Projects | Services | Files

- JS_MakeAdder
  - Site Root
    - index.html
  - Unit Tests

Navigator

- JavaScript
  - makeAdder(x) : makeAdder.add
    - add(y) : x
  - plusOne : makeAdder.add
- CSS
  - Elements
    - body
    - div
    - h4
  - Rules
    - body
    - div
    - h4
- HTML
  - html
    - head
      - title
      - meta
      - meta
      - style
    - body
      - div
        - h4
      - script

index.html

Source | History

```
<!DOCTYPE html>
<!--
A JavaScript script to test demonstrate closure in JavaScript
-->
<html>
    <head>
        <title>JavaScript Closure</title>
        <meta charset="UTF-8">
        <meta name="viewport" content="width=device-width, initial-scale=1.0">
        <style>
            body {background-color: powderblue;}
            h4 {font-style: italic}
            div    {color: red;}
        </style>
```

JavaScript Closure

http://localhost:8383/JS_MakeAdder/index.html

100%

A demonstration of closure

call to makeAdder(1) where * x = 1
** y = 3
** y = 3
*** plusOne(3) = 4
** y = 6
*** plusOne(6) = 7

Web Browser | Web Browser | Web Browser | Web Browser | Web Browser | Web Browser

20:55 INS

Review

- We have introduced the basics of JavaScript functions
- We have shown the JavaScript function:
  - syntax / prototype / invocation (calling)
- We have shown worked examples for:
  - functions / self-invoking functions / function return / functions as variables / variable scope, lifetime, and closure
- The worked examples are shown in the NetBeans IDE
  - The output is shown in the NetBeans embedded web kit
- We have provided an overview of variable scope and lifetime