

COMP284 Scripting Languages

Lecture 16: JavaScript (Part 3)

Handouts (8 on 1)

Ullrich Hustadt

Department of Computer Science
School of Electrical Engineering, Electronics, and Computer Science
University of Liverpool

Functions

Calling a function

Calling a function

A function is **called** by using the function name followed by a list of **arguments** in parentheses

```
function identifier(param1, param2, ...) {  
    ...  
}  
... identifier(arg1, arg2, ...) ... // Function call
```

- The **list of arguments** can be shorter as well as longer as the **list of parameters**
- If it is shorter, then any parameter without corresponding argument will have value **undefined**

```
function sum(num1, num2) { return num1 + num2 }  
  
sum1 = sum(5, 4)           // sum1 = 9  
sum2 = sum(5, 4, 3)        // sum2 = 9  
sum3 = sum(5)              // sum3 = NaN
```

COMP284 Scripting Languages

Lecture 16

Slide L16 – 4

Contents

- 1 Functions
 - Defining a function
 - Calling a function
 - Variable-length argument lists
 - Static variables
 - Example
 - Nested function definitions
- 2 JavaScript libraries
- 3 (User-defined) Objects
 - Object Literals
 - Object Constructors
 - Definition and use
 - Prototype property
 - Public and private static variables
 - Pre-defined objects

COMP284 Scripting Languages

Lecture 16

Slide L16 – 5

Functions

Calling a function

'Default values' for parameters

- JavaScript does **not** allow to specify **default values** for function parameters
- Instead a function has to check whether a parameter has the value **undefined** and take appropriate action

```
function sum(num1, num2) {  
    if (num1 == undefined) num1 = 0  
    if (num2 == undefined) num2 = 0  
    return num1 + num2  
}  
  
sum3 = sum(5)           // sum3 = 5  
sum4 = sum()            // sum4 = 0
```

COMP284 Scripting Languages

Lecture 16

Slide L16 – 5

Functions

Function definitions can take several different forms in JavaScript including:

```
function identifier(param1, param2, ...) {  
    statements  
}
```

```
var identifier = function(param1, param2, ...) {  
    statements  
}
```

- Such **function definitions** are best placed in the **head section** of a HTML page or in a **library** that is then imported
- **Function names** are **case-sensitive**
- The **function name** must be followed by parentheses
- A **function** has zero, one, or more **parameters** that are variables
- **Parameters** are not typed
- **identifier.length** can be used inside the body of the function to determine the number of parameters

COMP284 Scripting Languages

Lecture 16

Slide L16 – 2

Functions

Variable-length argument lists

Variable-length argument lists

Every JavaScript function has a property called **arguments**. The **arguments** property consists of an array of all the arguments passed to a function

- As for any JavaScript array, **arguments.length** can be used to determine the number of arguments

```
function sumAll() { // no minimum number of arguments  
    if (arguments.length < 1) return null  
    sum = 0  
    for (var i=0; i<arguments.length; i++)  
        sum = sum + arguments[i]  
    return sum  
}
```

```
sum0 = sumAll()           // sum0 = null  
sum1 = sumAll(5)          // sum1 = 5  
sum2 = sumAll(5, 4)       // sum2 = 9  
sum3 = sumAll(5, 4, 3)    // sum3 = 12
```

COMP284 Scripting Languages

Lecture 16

Slide L16 – 6

Functions

Defining a function

Functions

Function definitions can take several different forms in JavaScript including:

```
function identifier(param1, param2, ...) {  
    statements  
}
```

```
var identifier = function(param1, param2, ...) {  
    statements  
}
```

- The **return statement**
return value
can be used to terminate the execution of a function and to make **value** the return value of the function
- The **return value** does **not** have to be of a primitive type
- A function can contain more than one return statement
- Different return statements can return values of different types
~ there is no **return type** for a function

COMP284 Scripting Languages

Lecture 16

Slide L16 – 3

Functions

Static variables

JavaScript functions and Static variables

- JavaScript does not have a **static** keyword to declare a variable to be static and preserve its value between different calls of a function
- The solution is to use a **function property** instead

```
function counter() {  
    counter.count = counter.count || 0 // function property  
    counter.count++  
    return counter.count  
}  
  
document.writeln("1: static count = "+counter())  
document.writeln("2: static count = "+counter())  
document.writeln("3: global counter.count = "+counter.count)  
  
1: static count = 1  
2: static count = 2  
3: global counter.count = 2
```

- As the example shows the **function property** is global/public
- **Private static variables** require more coding effort

COMP284 Scripting Languages

Lecture 16

Slide L16 – 7

<div> <div>Functions</div> <div>Example</div> <div>JavaScript functions: Example</div> <div> <pre>function bubble_sort(array) { if (!(array && array.constructor == Array)) throw("Argument not an array") for (var i=0; i<array.length; i++) { for (var j=0; j<array.length-i; j++) { if (array[j+1] < array[j]) { // swap can change array because array is // passed by reference swap(array, j, j+1) } } } return array } function swap(array, i, j) { var tmp = array[i] array[i] = array[j] array[j] = tmp }</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 8</div> </div> </div>	<div> <div>JavaScript libraries</div> <div>JavaScript libraries: Example</div> <div> <pre>~ullrich/public_html/sort.js function bubble_sort(array) { ... swap(array, j, j+1) ... return array } function swap(array, i, j) { ... }</pre> </div> <div> <pre>example.html <html><head><title>Sorting example</title> <script type="text/javascript" src="http://cgi.csc.liv.ac.uk/~ullrich/sort.js"> </script></head> <body> <script type="text/javascript"> array = [2,4,3,9,6,8,5,1]; sorted = bubble_sort(array.slice(0)) </script> </body></html></pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 12</div> </div> </div>
<div> <div>Functions</div> <div>Example</div> <div>JavaScript functions: Example</div> <div> <pre>function bubble_sort(array) { ... } function swap(array, i, j) { ... } array = [2,4,3,9,6,8,5,1] document.writeln("array before sorting" + array.join(", ") + "
") array before sorting 2, 4, 3, 9, 6, 8, 5, 1
 sorted = bubble_sort(array.slice(0)) // slice creates copy document.writeln("array after sorting of copy" + array.join(", ") + "
") array after sorting of copy 2, 4, 3, 9, 6, 8, 5, 1
 sorted = bubble_sort(array) document.writeln("array after sorting of itself" + array.join(", ") + "
") array after sorting of itself 1, 2, 3, 4, 5, 6, 8, 9
 document.writeln("sorted array" + sorted.join(", ") + "
") sorted array 1, 2, 3, 4, 5, 6, 8, 9
</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 10</div> </div> </div>	<div> <div>(User-defined) Objects</div> <div>Object Literals</div> <div>Object Literals</div> <div> <ul style="list-style-type: none"> JavaScript is an object-oriented language, but one without <code>classes</code> Instead of defining a class, we can simply state an <code>object literal</code> <pre>{ property1: value1, property2: value2, ... }</pre> where <code>property1</code>, <code>property2</code>, ... are variable names and <code>value1</code>, <code>value2</code>, ... are values (expressions) </div> <div> <pre>var person1 = { age: (30 + 2), gender: 'male', name: { first : 'Bob', last : 'Smith' }, interests: ['music', 'skiing'], hello: function() { return 'Hi! I\'m ' + this.name.first + ' ' + this.name.last; } }; person1.age --> 32 // dot notation person1['gender'] --> 'male' // bracket notation person1.name.first --> 'Bob' person1['name']['last'] --> 'Smith'</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 13</div> </div> </div>
<div> <div>Functions</div> <div>Nested function definitions</div> <div>Nested function definitions</div> <div> <ul style="list-style-type: none"> Function definitions can be <code>nested</code> in JavaScript <code>Inner functions</code> have access to the variables of <code>outer functions</code> By default, <code>inner functions</code> can not be invoked from outside the function they are defined in </div> <div> <pre>function bubble_sort(array) { function swap(i, j) { // swap can change array because array is // a local variable of the outer function bubble_sort var tmp = array[i]; array[i] = array[j]; array[j] = tmp; } if (!(array && array.constructor == Array)) throw("Argument not an array") for (var i=0; i<array.length; i++) { for (var j=0; j<array.length-i; j++) { if (array[j+1] < array[j]) swap(j, j+1) } } return array }</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 10</div> </div> </div>	<div> <div>(User-defined) Objects</div> <div>Object Literals</div> <div>Object Literals</div> <div> <pre>var person1 = { name: { first : 'Bob', last : 'Smith' }, greet: function() { return 'Hi! I\'m ' + name.first + ' ' + name.last; }, full1: this.name.first + " " + this.name.last, full2: name.first + " " + name.last }; person1.greet() --> "Hi! I'm Bob."</pre> </div> <div> <ul style="list-style-type: none"> Every part of a JavaScript program is executed in a particular <code>execution context</code> Every <code>execution context</code> offers a keyword <code>this</code> as a way of referring to itself In <code>person1.greet()</code> the <code>execution context</code> of <code>greet()</code> is <code>person1</code> <ul style="list-style-type: none"> <code>this.name.first</code> is <code>person1.name.first</code> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 14</div> </div> </div>
<div> <div>JavaScript libraries</div> <div>JavaScript libraries</div> <div> <ul style="list-style-type: none"> Collections of JavaScript functions (and other code), <code>libraries</code>, can be stored in one or more files and then be reused By convention, files containing a JavaScript <code>library</code> are given the file name extension <code>.js</code> <code><script></code>-tags are <code>not</code> allowed to occur in the file A JavaScript library is imported using <pre><script type="text/javascript" src="url"></script></pre> where <code>url</code> is the (relative or absolute) URL for library <pre><script type="text/javascript" src="http://cgi.csc.liv.ac.uk/~ullrich/jsLib.js"></script></pre> One such import statement is required for each library Import statements are typically placed in the <code>head</code> section of a page or at the end of the <code>body</code> section Web browsers typically cache libraries </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 11</div> </div> </div>	<div> <div>(User-defined) Objects</div> <div>Object Literals</div> <div>Object Literals</div> <div> <pre>var person1 = { name: { first : 'Bob', last : 'Smith' }, greet: function() { return 'Hi! I\'m ' + name.first + ' ' + name.last; }, full1: this.name.first + " " + this.name.last, full2: name.first + " " + name.last }; person1.greet() --> "Hi! I'm undefined." person1.full1 --> "undefined undefined" person1.full2 --> "undefined undefined"</pre> </div> <div> <ul style="list-style-type: none"> In <code>person1.greet()</code> the <code>execution context</code> of <code>greet()</code> is <code>person1</code> <ul style="list-style-type: none"> but <code>name.first</code> does <code>not</code> refer to <code>person1.name.first</code> In the (construction of the) object literal itself, <code>this</code> does <code>not</code> refer to <code>person1</code> but its <code>execution context</code> (the window object) <ul style="list-style-type: none"> none of <code>name.first</code>, <code>name.last</code>, <code>this.name.first</code>, and <code>this.name.last</code> refers to properties of this object literal </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 15</div> </div> </div>

<div> <div>(User-defined) Objects</div> <div>Object Constructors</div> <div>Objects Constructors</div> <div> <ul style="list-style-type: none"> JavaScript is an object-oriented language, but one without classes Instead of defining a class, we can define a function that acts as object constructor <ul style="list-style-type: none"> variables declared inside the function will be instance variables of the object <ul style="list-style-type: none"> each object will have its own copy of these variables it is possible to make such variables private or public inner functions will be methods of the object it is possible to make such functions/methods private or public private variables/methods can only be accessed inside the function public variables/methods can be accessed outside the function Whenever an object constructor is called, prefixed with the keyword new, then <ul style="list-style-type: none"> a new object is created the function is executed with the keyword this bound to that object </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 16</div> </div>	<div> <div>(User-defined) Objects</div> <div>Prototype property</div> <div>Objects: Prototype property</div> <div> <ul style="list-style-type: none"> The prototype property can be modified 'on-the-fly' <ul style="list-style-type: none"> all already existing objects gain new properties / methods manipulation of properties / methods associated with the prototype property needs to be done with care </div> <div> <pre>function SomeObj() { ... } obj1 = new SomeObj() obj2 = new SomeObj() document.writeln(obj1.instVar4) // undefined document.writeln(obj2.instVar4) // undefined SomeObj.prototype.instVar4 = 'A' document.writeln(obj1.instVar4) // 'A' document.writeln(obj2.instVar4) // 'A' SomeObj.prototype.instVar4 = 'B' document.writeln(obj1.instVar4) // 'B' document.writeln(obj2.instVar4) // 'B' obj1.instVar4 = 'C' // creates a new instance variable for obj1 SomeObj.prototype.instVar4 = 'D' document.writeln(obj1.instVar4) // 'C' !! document.writeln(obj2.instVar4) // 'D' !!</pre> </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 20</div> </div>
<div> <div>(User-defined) Objects</div> <div>Definition and use</div> <div>Objects: Definition and use</div> <div> <pre>function SomeObj() { instVar2 = 'B' // private variable var instVar3 = 'C' // private variable this.instVar1 = 'A' // public variable this.method1 = function() { // public method // use of a public variable, e.g. 'instVar1', must be preceded by 'this' return 'm1[' + this.instVar1 + ']' + method3() } this.method2 = function() { // public method // calls of a public method, e.g. 'method1', must be preceded by 'this' return 'm2[' + this.method1() + ']' } method3 = function() { // private method return 'm3[' + instVar2 + ']' + method4() } var method4 = function() { // private method return 'm4[' + instVar3 + ']' } } obj = new SomeObj() obj.instVar1 --> "A" obj.instVar2 --> undefined obj.instVar3 --> undefined obj.method1() --> "m1[A] m3[B] m4[C]" obj.method2() --> "m2[m1[A] m3[B] m4[C]]" obj.method3() --> error obj.method4() --> error</pre> </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 17</div> </div>	<div> <div>(User-defined) Objects</div> <div>Prototype property</div> <div>Objects: Prototype property</div> <div> <ul style="list-style-type: none"> The prototype property can be modified 'on-the-fly' <ul style="list-style-type: none"> all already existing objects gain new properties / methods manipulation of properties / methods associated with the prototype property needs to be done with care </div> <div> <pre>function SomeObj() { ... } obj1 = new SomeObj() obj2 = new SomeObj() SomeObj.prototype.instVar5 = 'E' SomeObj.prototype.setInstVar5 = function(arg) { this.instVar5 = arg } obj1.setInstVar5('E') obj2.setInstVar5('F') document.writeln(obj1.instVar5) // 'E' !! document.writeln(obj2.instVar5) // 'F' !!</pre> </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 21</div> </div>
<div> <div>(User-defined) Objects</div> <div>Definition and use</div> <div>Objects: Definition and use</div> <div> <pre>function SomeObj() { this.instVar1 = 'A' // public variable instVar2 = 'B' // private variable var instVar3 = 'C' // private variable this.method1 = function() { ... } // public method this.method2 = function() { ... } // public method method3 = function() { ... } // private method var method4 = function() { ... } // private method }</pre> </div> <div> <ul style="list-style-type: none"> Note that all of instVar1 to instVar3, method1 to method4 are instance variables (properties, members) of someObj The only difference is that instVar1 to instVar3 store strings while method1 to method4 store functions </div> <div> every object stores its own copy of the methods </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 18</div> </div>	<div> <div>(User-defined) Objects</div> <div>Public and private static variables</div> <div>'Class' variables and 'Class' methods</div> <div> Function properties can be used to emulate Java's class variables (static variables shared among instances) and class methods </div> <div> <pre>function Circle(radius) { this.r = radius } // 'class variable' - property of the Circle constructor function Circle.PI = 3.14159 // 'instance method' Circle.prototype.area = function () { return Circle.PI * this.r * this.r; } // 'class method' - property of the Circle constructor function Circle.max = function (cx,cy) { if (cx.r > cy.r) { return cx } else { return cy } } c1 = new Circle(1.0) // create an instance of the Circle class c1.r = 2.2; // set the r instance variable c1.area = c1.area(); // invoke the area() instance method x = Math.exp(Circle.PI) // use the PI class variable in a computation c2 = new Circle(1.2) // create another Circle instance bigger = Circle.max(c1,c2) // use the max() class method</pre> </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 22</div> </div>
<div> <div>(User-defined) Objects</div> <div>Prototype property</div> <div>Objects: Prototype property</div> <div> <ul style="list-style-type: none"> All functions have a prototype property that can hold shared object properties and methods <ul style="list-style-type: none"> objects do not store their own copies of these properties and methods but only store references to a single copy </div> <div> <pre>function SomeObj() { this.instVar1 = 'A' // public variable instVar2 = 'B' // private variable var instVar3 = 'C' // private variable SomeObj.prototype.method1 = function() { ... } // public SomeObj.prototype.method2 = function() { ... } // public method3 = function() { ... } // private method var method4 = function() { ... } // private method }</pre> </div> <div>Note: prototype properties and methods are always public!</div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 19</div> </div>	<div> <div>(User-defined) Objects</div> <div>Public and private static variables</div> <div>Private static variables</div> <div> In order to create private static variables shared between objects we can use a self-executing anonymous function </div> <div> <pre>var Person = (function () { var population = 0 // private static 'class' variable return function (value) { // constructor population++ var name = value // private instance variable this.setName = function (value) { name = value } this.getName = function () { return name } this.getPop = function () { return population } } })(); person1 = new Person('Peter') person2 = new Person('James') person1.getName() --> 'Peter' person2.getName() --> 'James' person1.name --> undefined Person.population person1.population --> undefined person1.getPop() --> 2 person1.setName('David') person1.getName() --> 'David'</pre> </div> <div>COMP284 Scripting LanguagesLecture 16Slide L16 – 23</div> </div>

<div> <div> <div>(User-defined) Objects</div> <div>Pre-defined objects</div> </div> <div>Pre-defined objects: String</div> <div> <ul style="list-style-type: none"> JavaScript has a collection of pre-defined objects, including Array, String, Date A String object encapsulates values of the primitive datatype string Properties of a String object include <ul style="list-style-type: none"> length the number of characters in the string Methods of a String object include <ul style="list-style-type: none"> charAt(<i>index</i>) the character at position <i>index</i> (counting from 0) substring(<i>start</i>,<i>end</i>) returns the part of a string between positions <i>start</i> (inclusive) and <i>end</i> (exclusive) toUpperCase() returns a copy of a string with all letters in uppercase toLowerCase() returns a copy of a string with all letters in lowercase </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 24</div> </div> </div>	
<div> <div> <div>(User-defined) Objects</div> <div>Pre-defined objects</div> </div> <div>Pre-defined objects: String and RegExp</div> <div> <ul style="list-style-type: none"> JavaScript supports (Perl-like) regular expressions and the String objects have methods that use regular expressions: <ul style="list-style-type: none"> search(<i>regexp</i>) matches <i>regexp</i> with a string and returns the start position of the first match if found, -1 if not match(<i>regexp</i>) <ul style="list-style-type: none"> without <i>g</i> modifier returns the matching groups for the first match or if no match is found returns null with <i>g</i> modifier returns an array containing all the matches for the whole expression replace(<i>regexp</i>,<i>replacement</i>) replaces matches for <i>regexp</i> with <i>replacement</i>, and returns the resulting string <pre>name1 = 'Dave Shield'.replace(/(\w+)\s(\w+)/, "\$2, \$1") regexp = new RegExp("(\\w+)\\s(\\w+)") name2 = 'Ken Chan'.replace(regexp, "\$2, \$1")</pre> </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 25</div> </div> </div>	<div>Assignment Project Exam Help</div> <div>https://powcoder.com</div>
<div> <div> <div>(User-defined) Objects</div> <div>Pre-defined objects</div> </div> <div>Pre-defined objects: Date</div> <div> <ul style="list-style-type: none"> The Date object can be used to access the (local) date and time The Date object supports various constructors <ul style="list-style-type: none"> new Date() current date and time new Date(<i>milliseconds</i>) set date to milliseconds since 1 Januar 1970 new Date(<i>dateString</i>) set date according to <i>dateString</i> new Date(<i>year</i>, <i>month</i>, <i>day</i>, <i>hours</i>, <i>min</i>, <i>sec</i>, <i>msec</i>) Methods provided by Date include <ul style="list-style-type: none"> toString() returns a string representation of the Date object getFullYear() returns a four digit string representation of the (current) year parse() parses a date string and returns the number of milliseconds since midnight of 1 January 1970 </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 26</div> </div> </div>	<div>Add WeChat powcoder</div>
<div> <div> <div>(User-defined) Objects</div> <div>Pre-defined objects</div> </div> <div>Revision</div> <div> <div>Read</div> <ul style="list-style-type: none"> Chapter 16: JavaScript Functions, Objects, and Arrays Chapter 17: JavaScript and PHP Validation and Error Handling (Regular Expressions) <div>of</div> <div>R. Nixon:</div> <div>Learning PHP, MySQL, and JavaScript.</div> <div>O'Reilly, 2009.</div> <ul style="list-style-type: none"> http://coffeeonthekeyboard.com/private-variables-in-javascript-177/ http://coffeeonthekeyboard.com/javascript-private-static-members-part-1-208/ http://coffeeonthekeyboard.com/javascript-private-static-members-part-2-218/ </div> <div> <div>COMP284 Scripting Languages</div> <div>Lecture 16</div> <div>Slide L16 – 27</div> </div> </div>	

Assignment Project Exam Help

<https://powcoder.com>

Add WeChat powcoder