COMP284 Practical 7
JavaScript (2)

Introduction

- This worksheet contains further exercises that are intended to familiarise you with JavaScript Programming. While you work through the tasks below compare your results with those of your fellow students and ask for help and comments if required.
- This worksheet can be found at

  http://cgi.csc.liv.ac.uk/~ullrich/COMP284/notes/practical07.pdf

  and you might proceed more quickly if you cut-and-paste code from that PDF file. Note that a cut-and-paste operation may introduce extra spaces into your code. It is important that those are removed and that your code exactly matches that shown in this worksheet.
- The exercises and instructions in this worksheet assume that you use the Department's Linux systems to experiment with JavaScript.
  If you want to use the Department's Windows systems instead, then you can do so.
- To keep things simple, we will just use a text editor, a terminal, and a web browser. You can use whatever text editor and web browser you are most familiar or comfortable with.
- If you do not manage to get through all the exercises during this practical session, please complete them in your own time.

Exercises

1. Let us start with an exercise related to *functions* in JavaScript.

   It is assumed that you have already created a sub-directory called `public_html` in your home directory and that the directory is both readable and executable by everyone. Make sure that this is so before you proceed.

   a. Open a text editor and enter the following JavaScript code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Functions</title>
    <script type="text/javascript">
      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
      }
    </script>
```

```
  </head>
  <body>
    <h1>Exercises with JavaScript Functions</h1>
    <p>Output:</p>
    <script type="text/javascript">
      document.writeln("sumAll() = "+sumAll()+"<br>")
      document.writeln("sumAll(2) = "+sumAll(2)+"<br>")
      document.writeln("sumAll(2,3) = "+sumAll(2,3)+"<br>")
      document.writeln("sumAll(2,3,4) = "+sumAll(2,3,4)+"<br>")
    </script>
    <noscript>JavaScript not supported or enabled</noscript>
  </body>
</html>
```

b. Save the code to a file named `js07A.html` in `$HOME/public_html/`.

c. Open a terminal, go to the directory in which the file has been stored and make sure the file is only readable by yourself and nobody else can read, write or execute the file:

```
chmod u+r,og-rwx $HOME/public_html/js07A.html
```

(Note: No space after the comma!) You will only have to do so once. File permissions should not change while you continue to edit the file.

d. Now open a web browser and access the URL

```
http://cgi.csc.liv.ac.uk/~<user>/js07A.html
```


where *<user>* should be replaced by your departmental user name.

If everything is working correctly, then you should be shown the following in your web browser:


```
Exercises with JavaScript Functions

Output:
sumAll() = null
sumAll(2) = 2
sumAll(2,3) = 5
sumAll(2,3,4) = 9
```

If not, open the error console in your web browser and track down the errors in your code.

e. In the definition of `sumAll` we have used a variable `sum`. We have learned that one of the distinctions that JavaScript makes with respect to variables that are introduced in a function is whether they are *local* (only accessible from within the function) or *global* (accessible from elsewhere in the code).

Is `sum` local or global?

Let us test that by adding the following code before the last `</script>`-tag in your file:

```
document.writeln("sum = "+sum+"<br>")
```

Save the file again. Clear the output shown in the error console, then refresh the page in your web browser.

What does the additional output tell you about whether `sum` is local or global?

f. If your conclusion is that `sum` is global, then change the code for the function `sumAll` so that it will be local.

Test that any change you make has the desired effect. Once you have confirmed that `sum` is local, comment out the line of code introduced in Exercise 1e.

2. We now move on to a more complex function, the `bubble_sort` function we discussed in the lectures.

a. Add the following function to the JavaScript code in the *head section* of `js07A.html` (where we have already placed the function `sumAll`):

```
function bubble_sort(array) {
  function swap(array,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(array,j, j+1)
  return array
}
```


Also add the following code at the end of the JavaScript code in the body section of `js07A.html`:


```
array = [2,4,3,9,6,8,5,1]
document.writeln("<table border='1px' cellpadding='5px'>")
document.writeln("<tr><td>array before sorting"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
sorted = bubble_sort(array.slice()) // slice creates copy
document.writeln("<tr><td>array after sorting of copy"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
sorted = bubble_sort(array)
document.writeln("<tr><td>array after sorting of itself"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
document.writeln("<tr><td>sorted array</td><td>"+
                 sorted.join(", ")+"</td></tr>")
document.writeln("</table>")
```


Save the file. Clear the output shown in the error console, then refresh the page in your web browser.

b. Compare the additional output produced by the code in Exercise 2a with that in the lecture notes. Check that it is the same.

Make sure that you understand what the difference between `bubble_sort(array)` and `bubble_sort(array.slice())` is.

c. In the lectures we have seen how we can use the prototype property to change the methods associated with an object. This not only works for user-defined objects, but also for objects that JavaScript already provides, for example, `Array`. We will explore this further with the help of the `bubble_sort` function.

In preparation for this exercise add the following code in the *head section* of `js07A.html`:

```
function bubble_sort2() {
  return this
}
```

Also add the following code at the end of the JavaScript code in the body section of `js07A.html`:

```
Array.prototype.bubble_sort = bubble_sort2
array = [2,4,3,9,6,8,5,1]
document.writeln("<p>bubble sort method</p>")
document.writeln("<table border='1px' cellpadding='5px'>")
document.writeln("<tr><td>array before sorting"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
sorted = array.slice().bubble_sort() // slice creates copy
document.writeln("<tr><td>array after sorting of copy"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
sorted = array.bubble_sort()
document.writeln("<tr><td>array after sorting of itself"+
                 "</td><td>"+array.join(", ")+"</td></tr>")
document.writeln("<tr><td>sorted array</td><td>"+
                 sorted.join(", ")+"</td></tr>")
document.writeln("</table>")
```


Save the file. Clear the output shown in the error console, then refresh the page in your web browser.

As the additional output indicates `array.bubble_sort()` does not (yet) return a sorted array. Make sure that you understand what the line

```
Array.prototype.bubble_sort = bubble_sort2
```

does and why the output is as it is.

Expand the definition of `bubble_sort2`, re-using code from `bubble_sort`, so that `array.bubble_sort()` does return a sorted array.

Hint: Take the code from `bubble_sort` and replace the appropriate occurrences of `array` by the keyword `this`.

d. Extend `Array` with a method `peek`: `array.peek()` returns the first item of `array` without changing `array` if `array` has at least one element. If `array` has length 0, then the value `undefined` should be returned.

   Test you method using arrays of varying length.

3. In the lectures we have also considered various aspects of objects in JavaScript, in particular, the way *instance variables* and *'class' variables* are declared and how these can be made *public* or *private*.

   The following exercise is intended to reinforce those considerations.

   a. Open a text editor and enter the following JavaScript code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Functions</title>
    <script type="text/javascript">
    </script>
  </head>
```

```
  <body>
  <script type="text/javascript">
    var e = []
    e[0] = new Employee("Hal Smith", 30000)
    e[1] = new Employee("Tim Peck", 20000)
    e[2] = new Employee("Ari Bell",18000)
    document.writeln("e[0].name = "+
                     e[0].name+"<br>")         // prints Hal Smith
    document.writeln("e[0].salary = "+
                     e[0].salary+"<br>")       // prints undefined
    document.writeln("e[0]'s salary = "+
                     e[0].getSalary()+"<br>")  // prints 30000
    document.writeln("e[1]'s name = "+
                     e[1].getName()+"<br>")    // prints Tim Peck
    document.writeln("e[1]'s salary = "+
                     e[1].getSalary()+"<br>")  // prints 20000
    e[1].name = "Tom Beck"
    e[1].setSalary(25000)
    document.writeln("e[1]'s name = "+
                     e[1].getName()+"<br>")    // prints Tom Beck
    document.writeln("e[1]'s salary = "+
                     e[1].getSalary()+"<br>")  // prints 25000
    document.writeln("Employees: "+e[1].getEmployeeCount()+
                     "<br>")                   // prints Employees: 3
</script></body></html>
```


This code will serve as test code for Exercise 3c.

b. Save the code to a file named `js07B.html` in `$HOME/public_html/`. Make sure that the access rights `js01B.html` are set correctly.

c. We want to define an *employee* object. To keep the exercise simple, we assume that the only attributes of an *employee* are a *name* and a *salary*. The first should be *public*, the second *private*. In addition, we need a method to obtain information on an employee's salary as well as a method that allows us to change it. Finally, we want to keep track of how many employees there are in total and we want to keep that number private. The total number of employee's should be automatically incremented each time a new employee object is created.

   Create a *constructor* for employee objects that satisfies these requirements and add it to the *head section* of `js07B.html`.

d. Test your definition of the *employee* constructor by opening `js07B.html` in your web browser and observing that the output is as expected.

e. Since the total employee count is not really an attribute of a particular employee, it is a bit odd that we obtain that count by using an expressions like `e[1].employeeCount()`.

   Is it possible to define `employeeCount()` in such a way that we could use the expression `Employee.employeeCount()` instead? If so, modify your code accordingly and test your solution by replacing

```
document.writeln("Employees: "+e[1].getEmployeeCount()+
                 "<br>")          // prints Employees: 3
```

   with

```
document.writeln("Employees: "+Employee.getEmployeeCount()+
                 "<br>")          // prints Employees: 3
```

and checking that you get the correct output after you have saved the file and refreshed the page in the browser.

f. Being able to 'create' new employees is obviously nice, but sometimes we also have to 'delete' an existing employee.

Can we extend our definition of `Employee` by a method `remove` that deletes a particular employee object and at the same time decrements the total employee count? If so, modify your code accordingly and test your solution by adding the code

```
e[2].remove()
document.writeln("Employees: "+Employee.getEmployeeCount()+
                 "<br>")          // prints Employees: 2
```

and checking that you get the correct output after you have saved the file and refreshed the page in the browser.

Hint: Refer to

http://stackoverflow.com/questions/684575/how-to-quickly-clear-a-javascript-object

g. We want to prompt the user to enter a new salary for one of our employees. Add the following code to the body section of `js07B.html` (after the already existing code):


```
do {
  string  = prompt("Enter a new salary for "+
                   e[0].name+"?",e[0].getSalary())
  newSalary = parseInt(string)
} while (isNaN(newSalary) || newSalary <= 0)
e[0].setSalary(newSalary)
alert("The new salary for "+e[0].name+" is "+
      e[0].getSalary())
```


Save the file. Clear the output shown in the error console, then refresh the page in your web browser. Check that the code is working correctly. If it does you will first see a dialog box that prompts you to enter a new salary for 'Hal Smith' and then another dialog box will inform you what the salary of 'Hal Smith' has been changed to which should be the new salary that you have just entered.

h. Obviously, that dialog would be much more useful if you could change the salary of an employee whom you specify by providing his/her name.

As a first step you would need an additional dialog box that prompts the user for the name of an employee. Then you would need to define a function that given a name finds which of the *employee* objects in the array `e` stores data for an employee with that name and returns that particular oject; for comparison of names you could either use string equality or a regular expression search. Finally, you need to execute that function for the name that the user entered, and pass the returned object to the code in the previous step.

Implement that functionality and test it by entering various names that do or do not concur with the names for one of the existing employees and change their salaries.

4. Create a new file `jsDemo7.html` with HTML and JavaScript code that provides the following functionality. Initially, the page shows the user a two-dimensional table with 3 columns and 3 rows where every cell of the table contains the number zero. Below the table should be a clickable HTML element with the label 'Calculate'.

Whenever the user clicks on a cell, the number currently in the cell is replaced by a new random number between 1 and 9.

If the user clicks on ‘Calculate’ a message box will be shown with the message ‘The sum of all the numbers on the board is X’ where X is the sum of all the numbers currently in the cells of the table.