COMP519 Practical 8
JavaScript (3)

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.
- You might proceed more quickly if you cut-and-paste code from this 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.
- 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 before the next practical takes place.

Exercises

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

   a. Create a new file `jsDemo08A.html` in your `public_html` director with the following HTML markup and JavaScript code:

```
<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <title>JavaScript 08A: Functions</title>
    <script id="s1">
      function loop(max) {

        function applyAndWrite(f,y) {
          document.writeln(f.name,'(',y,') = ',f(y),"<br>")
        }

        function times2(x) { return 2*x }

        // Code for Exercise 1f here

        for (var i = 0; i <= max; i++) {
          applyAndWrite(times2,i)
        }
      }
    </script>
  </head>
```

```
  <body>
    <script id="s2">
      loop(2)
      // Code for Exercise 1e here
    </script>
    <noscript>JavaScript not supported or enabled</noscript>
  </body>
</html>
```

b. 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/jsDemo08A.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.

c. Now open a web browser and access the URL

```
https://student.csc.liv.ac.uk/~<user>/jsDemo08A.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:

```
times2(0) = 0
times2(1) = 2
times2(2) = 4
```

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

d. Note that the definitions of `applyAndWrite` and `times2` are nested inside that of `loop`. What happens if you try to call `times2` outside of `loop`? Add

```
    times2(6)
```

at the end of the script with id `s2`. Save the file and reload it in the web browser. What happens?
Remove the function call `times2(6)` from the code again.

e. Just below the definition of `times2`, at the point indicated, add the definition of the following function

```
        function scope(x) { return i * x }
```

Also, inside the for-loop add a call `applyAndWrite(scope,i)`. Save the file and reload it in the web browser. The output should now be

```
times2(0) = 0
scope(0) = 0
times2(1) = 2
scope(1) = 1
times2(2) = 4
scope(2) = 4
```

Why does this work? To what variable `i` does the function `scope` refer to in its definition?

f. Extend the code with the definition of another function `power2` that takes a single argument x and returns x^2. Extend the for-loop with code that executes that function in analogy to `times2` and `scope`. Test your solution.

2. We continue with another exercise related to *functions* in JavaScript.

   a. Create a new file `jsDemo08B.html` in your `public_html` directory with the following HTML markup and JavaScript code:

```
<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <title>JavaScript 08B</title>
    <script>
    // Function sumAll has no minimum number of arguments,
    // but accepts an arbitrary number of arguments
    function sumAll() {
      sum = 0
      for (var i = 0; i < arguments.length; i++)
          sum = sum + arguments[i]
      return sum
    }
    </script>
  </head>
  <body>
    <script>
    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. Make sure that the file permissions for this file are set as in Exercise 1b, then open it in a web browser.

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

```
JavaScript Functions

Output:
sumAll() = 0
sumAll(2) = 2
sumAll(2,3) = 5
sumAll(2,3,4) = 9
```

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

   c. 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 the 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?

d. 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 2c.

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

a. Create a new file `jsDemo08C.html` in your `public_html` directory with the following HTML markup and JavaScript code:

```
<!DOCTYPE html>
<html lang="en-GB">
  <head>
    <title>JavaScript 08C</title>
    <style>
      table {
        border:     1px;
        cellpadding: 5px;
      }
    </style>
    <script>
    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
    }
    </script>
  </head>
  <body>
    <h1>Sorting Arrays</h1>
    <script>
    numbers = [20,4,3,9,6,8,5,10]
    document.writeln("<h2>bubble_sort function</h2>")
    document.writeln("<table>\n<tbody>")
    document.writeln("<tr><td>array before sorting" +
                     "</td><td>"+numbers.join(", ") + "</td></tr>")
    // Sort a copy of array
    sorted = bubble_sort(numbers.slice())
```

```
    document.writeln("<tr><td>array after sorting of copy" +
                     "</td><td>" + numbers.join(", ") + "</td></tr>")
    sorted = bubble_sort(numbers)
    document.writeln("<tr><td>array after sorting of itself" +
                     "</td><td>" + numbers.join(", ") + "<\/td></tr>")
    document.writeln("<tr><td>sorted array</td><td>" +
                     sorted.join(", ") + "</td></tr>")
    document.writeln("</tbody>\n</table>")
    </script>
  </body>
</html>
```

b. Make sure that the access rights for the file are set correctly, then open it in a web browser.

c. Compare the output produced by the script in with that in the lecture notes.

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

d. 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 element* of the file `jsDemo08C.html`:

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

   Also add the following code at the end of the JavaScript code in the body element of `jsDemo08C.html`:

```
Array.prototype.bubble_sort = bubble_sort2
numbers = [20,4,3,9,6,8,5,10]
document.writeln("<h2>bubble sort array prototype method</h2>")
document.writeln("<table>\n<tbody>");
document.writeln("<tr><td>array  before sorting" +
                 "</td><td>" + numbers.join(", ") + "</td></tr>")
// Sort a copy of array
sorted = numbers.slice().bubble_sort()
document.writeln("<tr><td>array  after  sorting of copy" +
                 "</td><td>" + numbers.join(", ") + "</td></tr>")
sorted = numbers.bubble_sort()
document.writeln("<tr><td>array  after  sorting of itself" +
                 "</td><td>" + numbers.join(", ") + "</td></tr>")
document.writeln("<tr><td>sorted array</td><td>" +
                 sorted.join(", ") + "</td></tr>")
document.writeln("</tbody>\n</table>")
```

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

   As the additional output indicates `numbers.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 for any array `a0`, `a0.bubble_sort()` returns a sorted array.

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

e. Bubble sort is incredibly inefficient and nobody in their right mind would use it. Plus, JavaScript arrays already have a more efficient pre-defined `sort` method. Let us compare it to our `bubble_sort` method.

Add the following code at the end of the JavaScript code in the body element of `jsDemo08C.html`:

```
numbers = [20,4,3,9,6,8,5,10]
document.writeln("<h2>pre-defined array sort method</h2>")
document.writeln("<table>\n<tbody>")
document.writeln("<tr><td>array  before sorting" +
                 "</td><td>"+ numbers.join(", ")+"</td></tr>")
// Sort a copy of array
sorted = numbers.slice().sort()
document.writeln("<tr><td>array  after  sorting of copy" +
                 "</td><td>"+ array.join(", ") + "</td></tr>")
sorted = numbers.sort()
document.writeln("<tr><td>array  after  sorting of itself"+
                 "</td><td>"+ numbers.join(", ")+"</td></tr>")
document.writeln("<tr><td>sorted array</td><td>" +
                 sorted.join(", ") + "</td></tr>")
document.writeln("</tbody>\n</table>")
```

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

f. You should see that the pre-defined `sort` method puts the elements of the array in a different order than our `bubble_sort` method.

Using resources on the web, try to find out why this is so.

g. Again, using resources on the web, find out how the code introduced in Exercise 3e needs to be modified so that we get numerically sorted arrays. Make sure that you document the source of your solution as expected by the COMP519 Coding Standard [1].

h. Extend `Array` with a method `sum`: `array.sum()` returns the sum of all the elements of `array`. If `array` has length 0, then the value `0` should be returned.

```
document.writeln("<h2>array prototype sum function</h2>")
// For array0 the expected result is 0
array0 = [];
document.writeln("<div>[" + array0 + "].sum() = "+array0.sum()+"</div>")
// For array1  the expected result is 6
array1 = [3,2,1];
document.writeln("<div>[" + array1 + "].sum() = "+array1.sum()+"</div>")
// For array2  the expected result is 6
array2 = [3,"2M",[1,0]];
document.writeln("<div>[" + array2 + "].sum() = "+array2.sum()+"</div>")
```

i. 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.

4. We conclude with an exercise on JavaScript *arrays* and *array operations*.

   a. Replicate the array example we have seen in the lecture, that is, create a web page `jsDemo08D.html` that executes the following JavaScript code:

```
planets = ["earth"]
planets.unshift("mercury","venus")
planets.push("mars","jupiter","saturn")
document.writeln("planets\@1: " + planets.join(" ") + "<br>")
last = planets.pop()
document.writeln("planets\@2: " + planets.join(" ") + "<br>")
first = planets.shift()
document.writeln("planets\@3: " + planets.join(" ") + "<br>")
document.writeln("    \@4: " + first + " " + last + "<br>")

home = ["mercury","venus","earth"].pop()
document.writeln("    \@5: " + home + "<br>")
number = ["earth"].push("mars")
document.writeln("    \@6: " + number + "<br>")
```

   b. Save the file in your `public_html` directory and make sure that its access rights are set correctly. Check that the output produced by the script is as expected.

   c. Extend the script with the following code:

```
planets.length = 2
document.writeln("planets\@7: " + planets.join(" ") + "<br>")
list = planets // Modify this line in Exercise 4g
document.writeln("list \@8: " + list.join(" ") + "<br>")
planets[1] = "midgard"
document.writeln("list \@9: " + list.join(" ") + "<br>")
```

   d. Save the file and refresh the page in your web browser. The last line of the output should now be:

```
list @9: venus midgard
```

   How does this come about?

   e. Extend the script with the following code:

```
planets = ["Bellerophon","Ymir","Osiris"]
document.writeln("list \@10: " + list.join(" ") + "<br>")
document.writeln("planets\@10: " + planets.join(" ") + "<br>")
```

   f. Save the file and refresh the page in your web browser. The last two lines of the output should now be:

```
list @10: venus midgard
planets @10: Bellerophon Ymir Osiris
```

   How does this come about?

   g. Modify the code in 4c at the indicated point so that changes to `planets` do not affect `list`.

References

[1] U. Hustadt. *COMP519 Coding Standard*. Department of Computer Science, University of Liverpool. 24 January 2023. URL: http://cgi.csc.liv.ac.uk/~ullrich/COMP519/notes/COMP519CodingStandard.pdf (accessed 24 January 2023).