COMP284 Practical 6
JavaScript (1)

Introduction

- This worksheet contains 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 document can be found at

  http://cgi.csc.liv.ac.uk/~ullrich/COMP284/notes/practical06.pdf

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

Exercises

1. Let us start by re-creating the 'Hello World' JavaScript example that you have seen in the lectures.

   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 HTML markup and JavaScript code:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Practical 6: JavaScript and Variables</title>
  </head>
  <body>
    <script id="s1"> // Script with id s1
    </script>
    <h1>Practical 6: JavaScript and Variables</h1>
    <script id="s2"> // Script with id s2
    user = "<your name>";
    document.writeln("<p><b>Hello " + user +
                     "<br>\nHello World!</b></p>")
    </script>
    <script id="s3"> // Script with id s3
    </script>
    <p>This is the rest of the page</p>
    <noscript>JavaScript not supported or disabled</noscript>
  </body>
</html>
```

Replace *<your name>* with your own name.

Note that there are three JavaScript scripts in this code. In order to be able to refer to each of these three scripts they have been given the ids `s1`, `s2` and `s3`. We will use these ids in the instructions below.

b. Save the code to a file named `js06A.html` in `$HOME/public_html/`.

c. Open a terminal, go to the directory in which the file has been stored and make sure that 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/js06A.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>/js06A.html
```

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

Check that the page looks as expected and also that the HTML code producing the web page you are shown is exactly the HTML code you have seen in 1a above, including the JavaScript code.

Hint: Your web browser can show you the HTML source of a web page.

e. If already at this point or in one of the later exercises the web browser does not show the expected result, you will have to debug your code. For that you will need access to some diagnostic output for the JavaScript code that you have written.

A lot of web browsers provide an 'error console'. For example, in Firefox the error console can be found in the 'Tools' menu. The shortcut CTRL+SHIFT+J in Firefox also opens the error console.

Figure out how to access the error console in your web browser and open it. If the web browser you use does not have an error console (or you cannot figure out how to open it), switch to a better browser.

f. To see how the error console works let us introduce an error into our JavaScript code. At the end of the script with id `s2` insert

```
document.writeln("(1) The value of userAge is: " +
                 userAge + "<br>")
document.writeln("(2) This statement is executed<br>")
```

into `js06A.html`. Save the file again and refresh the page in your web browser.

Note that the web page appears unchanged. In particular, the statements introduced in 1f seem to produce no output. However, in the error console you should now see an error message indicating that `userAge` has not been defined/declared.

g. We should correct the error by adding a declaration for `userAge`.

First, try the following: Add

```
var userAge = "27"
document.writeln("(3) The value of userAge is: " +
                 userAge + "<br>")
```

at the end of the script with id s2 in js06A.html (that is, after the document.writeln statement that already uses userAge). Save the file again. Clear the output shown in the error console, then refresh the page in your web browser.

The error console should no longer show an error message and the web page should include the lines:

```
(1) The value of userAge is: undefined
(2) This statement is executed
(3) The value of userAge is: 27
```

This shows you that in JavaScript a declaration of a variable does not have to precede its use in order to prevent a reference error, however the initialisation of a variable only affects code that is executed later (even if the initialisation is done as part of a declaration).

But this principle only applies to a particular JavaScript code fragment enclosed in the tags `<script>` and `</script>` as we will see next.

h. Move the declaration

```
var userAge = "27"
```

from the script with id s2 to the script with id s3.

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

Is the code correct now or does it produce an error?

i. Move the declaration

```
var userAge = "27"
```

from the script with id s3 to the script with id s1.

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

Is the code correct now or does it produce an error? Is the output the same as in Exercise 1g? If not, why not?

2. The following exercise concerns *types*, *equality* and *comparisons* in JavaScript.

a. Start a new file js06B.html in your public_html directory with the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <title>Practical 6: JavaScript Types, Equality and Comparisons</title>
  </head>
  <body>
    <h1>Practical 6: JavaScript Types, Equality and Comparisons</h1>
    <script type="text/javascript">
    var values = [0, 123, 1.23e2, "12.3e1", true, false, NaN,
                  Infinity, undefined, null, [0]]
    for (var i=0; i<values.length; i++) {
      document.writeln(i+") The type of "+values[i]+
                       " is: "+typeof(values[i])+"<br>")
    }
    document.writeln(i+") The type of "+values+" is: "+
                     typeof(values)+"<br>")
    </script>
    <noscript>JavaScript not supported or disabled</noscript>
  </body>
</html>
```

b. Save the file and make sure that its access rights are set correctly, that is, the access rights should be set as in 1c.

c. Open the file `js06B.html` in a web browser and inspect the output that our JavaScript script has produced. Make sure that you understand why each array element is printed in the way it is and why the types of each array element are what they are.

d. As we can see from the last line of the output that our script has produced, the way in which JavaScript converts arrays to strings leaves a lot to be desired: Some of the elements (`undefined` and `null`) in `values` are not shown at all, although each element converted into a string by itself produces a non-empty string, and the fact that the string representation of `values` is not enclosed in brackets causes problems with elements that are themselves arrays (for example, `[0]`).

   We want to improve on that by redefining the `toString`-function that JavaScript uses to convert an array to a string. To do so add the following code to `js06B.html` just after the opening `<script>`-tag.

```
Array.prototype.toString = function () {
  var arrStr = "["
  // Add your own code here
  return (arrStr += "]")
}
```

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

```
11) The type of [] is: object
```

f. Add code to the `toString`-function at the indicated point so that the last line of the output will be

```
11) The type of [0, 123, 123, 12.3e1, true, NaN, Infinity,
    undefined, null, [0]] is: object
```

   (without a line break).

g. Extend the script with code that compares every element of `values` with every other element of `values` and itself using each of the four operators ==, ===, >, and <. The code should print out the elements it compares, their types and the result of the comparison.

   The first nine lines of the output produced by your code should look as follows:

```
(number)0 == (number)0: true
(number)0 === (number)0: true
(number)0 < (number)0: false
(number)0 > (number)0: false

(number)0 == (number)123: false
(number)0 === (number)123: false
(number)0 < (number)123: true
(number)0 > (number)123: false
```

h. Change the code developed in the previous step so that the results of the comparison are presented as an HTML table. For example, the nine lines above should produce two rows in the table:

`(number)0 == (number)0: true`	`(number)0 === (number)0: true`	`(number)0 < (number)0: false`	`(number)0 > (number)0: false`
`(number)0 == (number)123: false`	`(number)0 === (number)123: false`	`(number)0 < (number)123: true`	`(number)0 > (number)123: false`

3. It is often better to separate JavaScript code from HTML markup. This is what we want to do in this exercise. At the same time we have a quick look at mathematical operators in JavaScript.

   a. Start a new file `js06C.html` in your `public_html` directory with the following content:

```
<!DOCTYPE html>
<html>
  <head>
    <title>JavaScript Libraries</title>
  </head>
  <body>
    <h1>JavaScript Libraries</h1>
    <script src="js06C.js"></script>
  </body>
</html>
```

   Then create a separate file `js06C.js` in your `public_html` directory with the following content:

```
for (var ii = 4, jj = 3; jj >= 0; ii++, jj--) {
   document.writeln(ii + " * " + jj + " = " +
                    ii*jj + "<br>")
   document.writeln(ii + " / " + jj + " = " +
                    ii/jj + "<br>")
   document.writeln("log(" + jj + ") = " +
                    Math.log(jj) + "<br>")
   document.writeln("sqrt(" + jj-1 + ") = " +
                    Math.sqrt(jj-1) + "<br><br>")
}
```

   b. Save the files and make sure that their access rights are set correctly, that is, the access rights should be set as in Exercise 1c.

   c. Open the file `js06C.html` in a web browser and inspect the output that our JavaScript script has produced. Make sure that you understand how results like `NaN`, `Infinity`, and `-Infinity` come about.

   d. The first four lines of output produced by our JavaScript script are:

```
4 * 3 = 12
4 / 3 = 1.3333333333333333
log(3) = 1.0986122886681098
sqrt(2) = 1.4142135623730951
```

   As you can see the results of the mathematical operations are displayed with quite a high number of decimal places. We would like to limit that to just three decimal places, that is, the output should be

```
4 * 3 = 12.000
4 / 3 = 1.333
log(3) = 1.099
sqrt(2) = 1.414
```

Hint: Search on-line for a JavaScript function that limits the number of decimal places in the conversion of a number to a string.

e. Extend `js06C.js` with the following code:

```
x = 1.275
y = 1.27499999999999991118
document.writeln(x + " and " + y + " are " +
             ((x == y) ? "equal" : "not equal") + "<br>")
```

f. Save the file and refresh the page `js06C.html` in your web browser. The last line of the output should now be:

```
1.275 and 1.275 are equal
```

While this statement in itself it is true, something strange seems to have happened to the value of the variable `y`. What is going on?

g. Extend `js06C.js` with the following code:

```
mode = Math.floor((Math.random()*4)+1)
switch (mode) {
 case 1: randvar = Math.round(Math.random()*100); break;
 case 2: randvar = String(Math.random()); break;
 case 3: randvar = Math.random()/(Math.random()+1); break;
 default: randvar = Boolean(Math.random()+0.5); break;
}
document.writeln("Random value: (" + typeof(randvar) + ")"
               + randvar + "<br>\n")
```

h. Save the file and execute the script several times by repeatedly refreshing the page in your web browser. See how the value of `randvar` changes every time.

4. Your JavaScript code is accessible and readable by everyone who can access your web page. If you want to make it a bit more difficult for others to read, copy, and/or modify your code, then you should obscure it, for example, using `uglifyjs`.

   a. In a terminal, go to the directory in which the files `js06C.js` and `jsDemo3.html` has been stored and execute the commands

   ```
   cp js06C.js js06C.js.bak
   mv js06C.js js06C.pretty.js
   uglifyjs js06C.pretty.js -m -c > js06C.js
   ```

   (The option "`-m`" is for `mangle` and the option "`-c`" is for `compress`; to see a full list of options, use `uglifyjs -h`.)

   b. Check that `js06C.html` still produces the same output as before. Compare the files `js06C.js` and `js06C.pretty.js` to see how they differ.

   c. If anything goes wrong, then the file `js06C.js.bak` should still contain your original code, just copy it back to `js06C.js`.

   If you use this method for the COMP284 JavaScript assignment, then make sure that you submit the readable/pretty code via the departmental submission system and place the unreadable/uglified only code in your `public_html` directory.

5. 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 `js06D.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 4e
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. Modify the code in 5c at the indicated point so that changes to `planets` do not affect `list`.