# CSI 3140 – Laboratory 5

# JavaScript

## *Exercise 1: getting JavaScript to run in your browser*

Using a text editor (like notepad) type up the example HiLow.js page 198 of the book and run it in your browser. Bring up your Browser JavaScript console and use it as needed to ensure that your code actually runs. Modify the code so that it is possible to exit without finding the solution!

## *Exercise 2: declaring your variables*

We have seen that you can avoid variable declaration using JavaScript. It is however bad practice, as it is confusing and error prone. In some (rare) cases, it even makes a difference! For example, declaring variable may help with optimization and therefore provide more efficient code. It may also help with exceptions. See for yourself with exercise 4.14, page 242 (implement and test the code).

## *Exercise 3: a first program*

Do exercise 4.22 page 245 of the book.

## *Exercise 4: more programming*

In September of 2003, the following information started to be send by email throughout the internet:

```
Aoccdrnig to a rscheearch at Cmabrigde Uinervtisy, it
deosn't mttaer in waht oredr the ltteers in a wrod are, the
olny iprmoetnt tihng is taht the frist and lsat ltteer be
at the rghit pclae. The rset can be a toatl mses and you
can sitll raed it wouthit porbelm. Tihs is bcuseae the
huamn mnid deos not raed ervey lteter by istlef, but the
wrod as a wlohe.
```

This translates into: *According to a researcher (sic) at Cambridge University, it doesn't matter in what order the letters in a word are, the only important thing is that the first and last letter be at the right place. The rest can be a total mess and you can still read it without problem. This is because the human mind does not read every letter by itself but the word as a whole.*

Of course, like almost every email chain, it turned out to not be true. But let see for ourselves:

### Question 1

Create a JavaScript function that takes a string as input and splits this string word by word. Create a small html page that prompts the user for some input and shows the result of your function in an alert box.
*Hint*: look at the method **split** from the object **String**.

### Question 2

Create a JavaScript function that takes a string as input and randomly "shuffles" all the letter of that string. Create a small html page that prompts the user for some input and shows the result of your function in an alert box.
*Hint*: look at the method **random** from the object **Math**. It might be easier to first put the string in an Array.

### Question 3

Use the functions you have created to write a JavaScript program that prompts he user for some input and then displays back this input with each word having its first and last letter at the right place, and the other letters of the word randomly shuffled around.

Once you are satisfied that your program is working, go to some web page (e.g. select one of the stories from google news, http://news.google.com/), copy one paragraph, past it into your program and see how easy it is to read the result.

## *Exercise 5: objects and regular expressions*

Create an Object that has five methods[1]:

- `Object.normalize(inputString)`

inputString is a string made of space separated words. “Normalize” returns a string that contains the same words, but with no leading spaces, no trailing spaces, and exactly one space between words.

Only use regular expressions and the String methods “replace” and “match” to implement “normalize”.

- `Object.hasString(inputString, testedString)`

Returns true if and only if “inputString” contains “testedString” as a sub-word (not as a substring! That is, testedString can be preceded by a space or be at the beginning of inputString only, and can be followed by a space or be at the end of inputString only)

[1] Note that you can actually achieve the same functionality more directly, e.g. by using some of the more advanced String methods. Also, in reality, nothing calls for an object in this case, a simple function would do. This is done this way for the sake of the exercise.

- `Object.add(inputString, addedString)`

Adds “addedString” to “inputString”, as a sub-word, if it doesn’t have it already

- `Object.remove(inputString, removedString)`

Removes every instance of “removedString” as a sub-word of  “inputString”

- `Object.toggle(inputString, firstString, secondString)`

If “inputstring” has instances of “firstString” as a sub-word, then these instances are replaced by a (unique) instance of the sub-word “secondString”. Else, if “inputstring” has instances of “secondString” as a sub-word, then these instances are replaced by a (unique) instance of the sub-word “firstString”.

Your object must be reasonably resistant, and should handle gracefully various cases (null/empty parameters etc…). Only use regular expressions and the String methods “replace” and “match” to implement most of the methods.