CMPT 898 HOME EXERCISE 5
BUILDING ON HIGHER-ORDER FUNCTIONS BASICS

Building on the examples used in class, please attempt to perform the following using the file "document.txt" posted to the moodle site.

For the sake of the below, term "word" is defined as a contiguous series of alphabetic characters in the range a-z (or A-Z). Two words are considered the same if they differ only in terms of case (and thus should both be counted as the same word). You will throw away other characters.

- **Number of occurrences of words:** Count the number of times that a given word appears in the document (ignoring the case of the word).
- **Number of occurrences of word lengths:** Count the number of times that words of a given length appear in the document.
- Please redo the above, sorting by the counts of occurrences. Please note that a dictionary can be converted into a sequence using *toSeq* and then sorted by the above toSeq.sortWith((pair1, pair2) => pair1._2 > pair2._2)

The solution should be able to execute in a single line, and should use the higher order functions that we discussed, particularly map, mapValues, filter and groupBy.

Recall that:

- Applying *map* passed a unary function *f* to a collection *coll* (via coll.map(f)) will yield a new collection, each of whose element consists of applying *f* to the corresponding element of coll.
- Applying *filter* passed a predicate (a unary function *p* returning a boolean) to a collection *coll* (via coll.filter(p)) will yield a new collection consisting of just those elements of coll that match the predicate (i.e., elements *elt* for which the predicate returns the boolean true, i.e., when *p(elt)* is true).
- Applying *groupBy* given a function f on a collection coll (via coll.groupBy(f)) will create a dictionary, whose keys consist of all of the values of f applied to the elements of the collection *coll*. For a given key *k*, the value of the dictionary entry for that k consists of a collection of all of the elements *elt* of *coll* for which *f(elt)* is *k*.
- A *dictionary* (*Map)* is an abstract data type which keeps track of the values associated with different keys. (For example, a dictionary might map a string representing a word to a collection of occurrences of that word count of times that this word has appeared in a document. Alternatively, a dictionary hold keys that are double precision values, each of which represents the beginning of a bin, and the value associated with each such count indicates the count of times values in some collection have fallen within that bin). Applying mapValues(f) to a dictionary *dict* with a function f returns a new dictionary with the same keys, with the value associated with each such key k

For the above, the following hints may be helpful:

- Given a string str, you can
  - convert that string to lower case via the call str.toLowerCase
  - Split that string into distinct words (as described above) using str.split("[^a-zA-Z]+"), where the item inside the quotes is a regular expression pattern that characterizes where to split the string (here, on any character that is not in either the range a-z or A-Z).
- to create a function that takes an argument a and returns some expression expr (typically involving) a, you can simply write
  a => expr
- You can get an array of lines as follows (please note that getLines returns an iterator over lines):
  Source.fromFile("document.txt")("ISO-8859-1").getLines.toArray

Due in class January 31, 2017

- Before pursuing the above and using "Source", you will need to issue the following command in the Scala Read-Eval-Print Loop (REPL – also called the "Scala command line"):
  import scala.io._

Among other matters, you will want to eliminate lines that are blank ("") from this list.