In-class Exercises for Lecture 11

ICCS 101 – Introduction to Programming

Problem 1: Word Count

A word is a consecutive sequence of English letters (i.e., a through z, and A through Z) without any other character. Using this definition, the string “High-land tribes in hBoro are: meme; mae-mae” have 9 words, namely: High, land, tribes, in, hBoro, are, meme, mae, mae.

Write a function `wordCount(filename)` that takes a filename (as a string) and returns the number of words in that file.

Save the following text to a file named “p1.txt”

```
Lorem ipsum dolor sit amet, consectetur adipiscing elit. Susp
odio. In in condimentum mauris. Integer sollicitudin vel feli
nulla id mauris posuere commodo vitae sit amet dolor. In puru
dolor ut, maximus ultrices nunc. Donec mollis pulvinar nulla.
aliquet ipsum quis iaculis. Phasellus eleifend est massa, id
pellentesque auctor. Nam laoreet dignissim nulla lobortis sol
nunc egestas, dapibus lectus a, lobortis mi. Duis vitae matti
eros, gravida a urna et, pellentesque tincidunt velit.

Mauris viverra, lorem et dictum fermentum, libero odio vulput
scelerisque turpis tortor id ante. Aenean ultrices lectus nec
Quisque aliquam vestibulum sodales. Vestibulum diam orci, var
cursus ornare odio. Maecenas nulla dolor, pretium vitae sapie
gravida ipsum. Praesent cursus risus sem, ut ullamcorper ipsu
elit dui, suscipit vel suscipit id, pretium a quam. Aliquam n
odio vel, facilisis risus. Fusce ultrices et massa ut blandit
facilisis. Vestibulum sodales libero ac odio euismod ultricie
dolor, venenatis non neque in, accumsan viverra dui. Fusce so
```

Example:

```
assert wordCount('p1.txt') == 200
```

Problem 2: Most Common Names

Part 1

Write a function `mostCommonName(names)` that takes a list of names (such as `["Jane", "Aaron", "Cindy", "Aaron"]`, and returns the most common name in this list (in this case, “Aaron”). If there is more than one such name, return a list of the most common names, in any order. That is, `mostCommonName(["Jane","Aaron", "Jane", "Cindy", "Aaron"])` returns, for example, the list `["Aaron" , "Jane"]`. If the list is empty, return `None`. Also, treat names case sensitively, so “Jane” and “JANE” are different names.

Part 2

Write another function `mostCommonNameToFile(names, filename)` that has the same functionality as `mostCommonName` but, instead of returning a list, `mostCommonNameToFile` writes the most common name(s) to a file instead. The name of the output file is specified by the string `filename`.

Problem 3: SumAll

Write a function, `sumAll(filename)`, that should read a file containing only integers and return the sum of all these integers. *Hint: Use str.split() to split a line containing multiple integers.*

Example usage:

```
print sumAll('sumAllExample.txt')
>> 1262
```

where `sumAllExample.txt` contains the following content:

```
12  234 23  23  392
1 4 523 2
2
19
2 23 2
```

Problem 4: Matrix Transpose

Write a function, `transpose(matrix)`, that returns the transpose of an input matrix given by a nested list `matrix`. Recall that the transpose of

```
[[1,2],
 [3,4]]
```

is

```
[[1,3],
 [2,4]]
```

Example:

```
transpose([[1,2],[3,4]]) == [[1,3],[2,4]]
transpose([['a','b','c'],['d','e','f'],['g','h','i']]) ==
   [['a','d','g'],['b','e','h'],['c','f','i']]
```

Problem 5: Short Circuit

Save the following code to a file called `short_curcuit.py`

```
# Long calculation time
# Often return False
def longCheck(n):
    total = 0
    for i in range(n*1000):
        total += i
    return (total % 10) == 0

# Short calculation time
# Often return False
def shortCheck(n):
    total = 0
    for i in range(n):
        total += i
    return (total % 10) == 0

def doWork1():
    counter = 0
```

```
    for i in range(500):
        if longCheck(i) and shortCheck(i):
            counter += 1
    return counter

def doWork2():
    counter = 0
    for i in range(500):
        if shortCheck(i) and longCheck(i):
            counter += 1
    return counter
```

Part 1

Run `short_circuit.py` in Spyder and type in the following in the Spyder IPython console.

```
%time doWork1()
```

Write down the “wall time”, which is the total time it takes to run the function `doWork1()`.

Part 2

Now run:

```
%time doWork2()
```

Compare the wall time with the previous one.

- What is the difference between `doWork1` and `doWork2`? *
- Why `doWork2` runs significantly faster than `doWork1`?