15110 PRINCIPLES OF COMPUTING – LABEXAM 1A- Summer 2015

Name: _____Section: _____Andrew Id: _____Machine:_____

Directions:

1. In your home directory, **create a folder** named **labexam1**
2. Write a function in Python for each of the following problems and store these functions in the labexam1 folder. **Test your functions** by running in IDLE or calling them with python3 –i. Although we give you example/test runs, your function should work on all legal inputs based on the specifications given, and your output should match the examples as closely as possible for full credit. Remember that we will run your code on additional test cases that are not shown on the exam.
3. These problems can be done using for **loops, while loops, or recursion**: your choice (unless otherwise specified).
4. Once you are finished, **compress the labexam1 folder into a zip file and submit** it to **AutoLab** (http://autolab.cs.cmu.edu) by the end of lab. **Do not delete the labexam1 folder** from your home directory.

Below is Python3 syntax reminder for ***`for`*** *and* ***`while`*** *loops. If we call the functions below with an argument that is a list of numbers they both print the odd items such that each item is printed on a separate line. Note that the* ***`print`*** *function can be called with the keyword arguments* ***`sep`*** *and* ***`end,`*** *defining respectively, the string to be placed between every two printed values and the string to be printed at the end of the print function. For example, using* ***`print(list[i], end='')`*** *in the examples below would print the values on the same line.*

```
def example1(list):
   for i in range(0,len(list)):
      if list[i]%2 != 0 :
         print(list[i])
```

```
def example2(list):
   i = 0
   while i < len(list):
      if list[i]%2 != 0:
         print(list[i])
      i = i + 1
```

BONUS: if you write appropriate necessary comments, use meaningful variable names and use necessary spaces to improve the readability of your code you can get a bonus point (upto 3 points in total for questions 2, 3 and 4).

Question 1 (save the file as q1.py in your labexam1 folder) [20 points]

Rewrite the function below by **renaming** the function and its parameter (***function1, parameter1***). Also rename **variable1** according to its aim. As data type Parameter1 is a list of integers. So this function could be called such as "function1([3,5,6,8,9,22])" (before renaming function1). In addition change the sentences like "#bla bla bla" with appropriate brief comments.

```
def function1(parameter1):
   variable1 = 0        #bla bla bla bla
   i = 0                #bla bla bla bla

   while i < len(parameter1):
      if parameter1[i] % 2 == 0:                #bla bla bla
         variable1 = variable1 + 1              #bla bla bla
         print(variable1, ":", parameter1[i])
      i = i + 1                                 #bla bla bla
```

Question 2 (save the file as q2.py in your labexam1 folder) [25 points]

Write a python function ***squaresBetween(firstNum, lastNum)*** that prints the squares of the numbers between integers ***firstNum*** and ***lastNum*** **(inclusive).** You can assume that *firstNum* is less than *lastNum*.

As shown below, first print a string like "Squares of numbers from ____ to ____" and then print the numbers on the same line and use " | " between (instead going to the new line).

Sample usage:

```
>>> squaresBetween(2,5)
Squares of numbers from 2 to 5
4 | 9 | 16 | 25 |
>>> squaresBetween(10,15)
Squares of numbers from 10 to 15
100 | 121 | 144 | 169 | 196 | 225 |
```

Question 3 (save the file as q3.py in your labexam1 folder) [25 points]

Write a python function ***first_num_greater_than(NumbersList, Key)*** that takes a list of integers(*NumbersList*) and a key number (*Key*) in order to find and return the first number in the list that is greater than the key number taken. You may assume that the list you have taken has at least one element. If you cannot find a number greater than the key then you should return None.

Sample usage:

```
>>> sampleList = [3, 7, 18, 9, 18, 42, 4, 35, 45]
>>> print(first_num_greater_than(sampleList, 2))
3
>>> print(first_num_greater_than(sampleList, 18))
42
>>> print(first_num_greater_than(sampleList, 100))
None
```

Question 4 (save the file as q4.py in your labexam1 folder) **[30 points]**

Write a function **convert(List1**, **LetterList)** that takes

1. a list (*List1*) of other lists of integers and
2. another list of characters (*LetterList*).

Each list in *List1* consists of a series of integers. Each of these integers is the index of another list which is taken as the second parameter (*LetterList*). Aim of the function is to get the series of numbers from the items of *List1*, to generate **strings** and append them to a new list. And at the end to return that new list of strings as a result. Follow the algorithm below to write this function:

1. Create a new empty list (lets say newList. You can give different names.)
2. For each item of *List1*
   a. Generate a string using the list of integers of this item *(detailed explanation is below)*
   b. Append this string to the newList
3. Return the new list

In order to Generate a string using the list of integers
1. Create a new empty string
2. For each number in the list of integers
   a. Add relevant letter to the end of the new string

Hint:

- if a is a list then a.append(x) appends the element c to the list a
- if s is a string s = s + 'X' will concatenate string s and 'X'

Sample usage:

```
>>> letters = [' ', 'D', 'G', 'M', 'O', 'R', '!']
>>> a = convert( [ [2, 4, 4, 1],
                   [5, 4, 4, 3, 0, 1, 4, 4, 5],
                   [1, 4, 5, 3, 0, 5, 4, 4, 3],
                   [2, 4, 4, 1, 0, 2, 4, 1, 6],
                   [1, 4, 2]
                 ], letters)
>>> a
['GOOD', 'ROOM DOOR', 'DORM ROOM', 'GOOD GOD!', 'DOG']
```

15110 PYTHON REFERENCE SHEET

Arithmetic Operations:	`**`	`*`	`/`	`//`	`%`	`+`	`-`
Relational Operations:	`==`	`!=`	`<`	`<=`	`>`	`>=`	
Logical Operations:	`and`	`or`	`not`				

Variable Names: All variable names must start with a letter (lowercase recommended). The remainder of the variable name (if any) can consist of any combination of uppercase letters, lowercase letters, digits and underscores (_). Variables are case sensitive.

Assignment Statement: *variable* = *expression*

Defining a function: `def` *functionname* `(` *parameterlist* `):`
function_body
A *parameterlist* may be empty or may include one or more variables representing data required for the function, separated by commas.

Calling a function: *functionname* `(` *argumentlist* `)`
An *argumentlist* may be empty or may include one or more expressions representing data required for the function to use, separated by commas.

Importing module: `import` *modulename*
Using module: *modulename*`.`*functionname* `(` *argumentlist* `)`

`print(data)`	prints data to screen and moves cursor to next line
`print(data, end=" ")`	prints data to screen and keeps cursor on same line
`print()`	moves cursor to next line
`return(data)`	returns data to instruction that called this function

`for` *v* `in range(`*x, y, z*`):` *loop_body*	loops for *v* = *x* through *y*-1, inclusive in steps of *z* (*y* is optional, default 0. *z* is optional, default 1.)
`while` *condition*`:` *loop_body*	loops while *condition* is `True`
`if` *condition1*`:` *instruction1_set*	executes *instruction1* set once if *condition1* is `True`
`elif` *condition2*`:` *instruction2_set*	otherwise executes *instruction2* set once if *condition2* is `True`. This part is optional, can be repeated.
`else:` *instruction3_set*	otherwise executes *instruction3* set once if all previous conditions tested as `False`. Optional.

Lists:	*listname* = `[]`	An empty list.
	listname = `[` $item_0$, $item_1$, … , $item_{n-1}$ `]`	A list of n items, n >= 1.
	listname`[`*i*`]`	Evaluates to the i^{th} element of the list

`len(`*listname*`)`	returns the number of items in the list
item `in` *listname*	returns `True` if the item is in the list, `False` otherwise.
listname`[`*i*`:`*j*`]`	returns a sublist of list from index *i* to *j*-1
listname = `[` *item*`]` `*` *n*	creates a list with *n* copies of the item
listname`.append(`*item*`)`	appends item to end of the list
listname`.remove(`*item*`)`	removes the first occurrence of the item in the list
listname`.insert(`*pos, item*`)`	insert the *item* to the item in so that its index will be equal to *pos*

`for` *item* `in` *listname*`:` *loop_body*	performs instructions once for each item in list, no index is available (*item* can be referenced in loop body)