**15110 PRINCIPLES OF COMPUTING – LABEXAM 1A- Summer 2016**

**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 (up to 3 points in

total for questions 2, 3 and 4).

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

Consider the function below. Rewrite the function to accomplish the following:

* Change its functionality so that it accomplishes the **opposite** of its current purpose
* **Rename** the function and its parameter (***function1, parameter1***) based on its new functionality. 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).
* **Finally,** modify“#comment” with appropriate brief, descriptive comments.

def function1(parameter1): variable1 = 0 *#comment* i = 0 *#comment*

while i < len(parameter1):

if parameter1[i] % 2 > 0: *#comment*

variable1 = variable1 + 1 *#comment*

print(variable1, ":", parameter1[i])

i = i + 1 *#comment*

**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 histogram(input\_list) that takes a list of integers as a parameter. The function should print a vertical histogram that summarizes the information in the list, printing a count of the integers in groups of 10 (0-9, 10-19, 20-29….90-99 and 100 as a distinct category) as a histogram. The function should print asterisks equal to the count of integers in each category.

If the function were called with this list:

ip = [14, 45, 78, 89, 23, 12, 52, 43, 14, 76, 87, 88, 87, 76, 76, 90, 99, 100, 23, 32, 41, 20, 45, 54, 52, 67, 89, 43, 21, 43, 56, 67, 67, 78, 59, 93, 33, 11, 1, 43, 55, 45, 67, 65, 54, 76, 67, 59, 95, 20, 34, 32, 37, 38, 30,30, 45, 56, 56, 45]

it would provide this output:

0 | \*

10 | \*\*\*\*

20 | \*\*\*\*\*

30 | \*\*\*\*\*\*\*\*

40 | \*\*\*\*\*\*\*\*\*\*

50 | \*\*\*\*\*\*\*\*\*\*

60 | \*\*\*\*\*\*

70 | \*\*\*\*\*\*

80 | \*\*\*\*\*

90 | \*\*\*\*

100 | \*

Your solution must use a nested loop. Your solution should provide appropriate formatting between the category (0, 10, etc) and the separator bar (|). Your solution can assume that only a list of integers will be passed into the function.

**Bonus:**

An acceptable solution for this problem may iterate over the input\_list multiple times. However, a more efficient solution to this problem can be constructed by sorting the list and then iterating through the list only once. Up to three extra credit points may be awarded for a solution that sorts input\_list and iterates through the list only once.

**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*): loops for *v* = *x* through *y*-1, inclusive in steps of *z loop\_body* (*y* is optional, default 0. *z* is optional, default 1.)

while *condition*: loops while *condition* is True

*loop\_body*

if *condition1*: executes *instruction1* set once if *condition1* is True

*instruction1\_set*

elif *condition2*: otherwise executes *instruction2* set once if *condition2 instruction2\_set* is True. This part is optional, can be repeated.

else: otherwise executes *instruction3* set once if all

*instruction3\_set* previous conditions tested as False. Optional.

Lists: *listname* = [] An empty list.

*listname* = [ *item0* , *item1* , … , *itemn-1*] A list of n items, n >= 1.

*listname*[*i*] Evaluates to the ith 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*: performs instructions once for each item in list, no index is available

*loop\_body* (*item* can be referenced in loop body)