בס"ד

Home Exercise Nr 4 - Functional Programming with Python

Important Notes:

A. You are allowed to use functional programming tools only.
B. Inside the code of the solution of every one of the questions, as short comments, you must write the following:
   - which programming pattern (or combination of patterns) you used in your solution.
   - in every recursive function that you will write, tell
     - which is (or are) the base case(s) of the recursion.
     - which is (or are) the general case(s) of the recursion.
     - for every recursive call, tell if it is a tail recursive call or a non-tail recursive call.
     - is the function as a whole, tail recursive or non-tail recursive?
     - is the function as a whole, a linear recursion (a recursion over a non-nested structure), or is it a tree recursion (a recursion over a nested structure)?

Question Nr. 1

In the Python file Targil4input.py (which may be downloaded from the course moodle site) you will find the definition of two lists:

- jctMarks: it includes information about students' grades. For example, the student whose ID number is 12345 got 75 in an English course.
- teacherNames: it includes information about teachers' names and the courses that they teach. For example, Dr. Aharoni teaches English. You may assume that there are no more than one teacher with the same name, and who teach that same course.

Those lists are nested lists.

Write a function, called **`myStudList (list1, list2)`**, which receives two parameters, which both are expected to be lists: the structure of the first list is like that of jctMarks, and the structure of the second list is like that of teacherNames.
The function will create a list L in which every element is a list whose first element will be the name of a teacher, and whose second element will be a list of size 2 whose first element will be a list of the IDs of the students of that the teacher, and whose second element is a tuple containing the average and the standard deviation of the grades of all those students.
In the case that there is no students in any of the courses of the teacher, both, the list of student IDs of that teacher, and the tuple of average and standard deviation of the grades, will be empty.
Additionally, the function will calculate the average and standard deviation of the grades of all the students, by calling a helper function the you have to write for this purpose.
The function will return a tuple containing three elements: the list L, the grades' average, and the grades' standard deviation of all the students.
An important note: you will have to import the module Targil4input.

Write another function, called **`myStudDict(Lst)`**, that receives a list Lst of the same kind as the list which is the first element of the tuple returned by the function **`myStudList`**. The function **`myStudDict`** will create and return a dictionary D in which every key will be the name of a teacher, and the value bound to it will be a list whose elements, except the last one, are the IDs of all the students of that teacher; the last element will be a tuple of size 2 containing the grades' average and the grades' standard deviation of those students.

Write a Python program that calls the function **`myStudList`**, passing the lists jctMarks and teacherNames as parameters. The program will call the function **`myStudDict`**, passing to it the list which is the first element of the tuple returned by the function **`myStudList`**. The program will print the dictionary returned by the function **`myStudDict`**; each printed line will have the following format:

Teacher name, list of IDs of his students, grades' average, grades' standard deviation

At the end of the printing, the program will print the grades' average and grades' standard deviation of all the students.
The dictionary returned by the function **`myStudDict`** for the data contained in the provided JctMarks and teacherNames, will be as follows:

{ 'Aharoni': [12345, 75.0], 'Kaner': [23415, 12345, 68.0], 'Melamed': [23452, 23459, 86.5], 'Korman': [], 'Zloti': [23560, 23452, 90.5] }

Question Nr. 2

Here we will write a mini-system for text file processing. For this purpose we will write the following functions:

1. **`treatline(lineNr, line)`**

   This function receives two parameters: **lineNr** (a positive integer number) and **line** (a string of any length, whose last character is '\n'). Each line may contain any number of English words, separated by spaces or any other punctuation marks. The function will use an appropriate Boolean function to check every word if it contains letters only; if any word contains some non-letter character, the function will return -1. If everything is OK, the function will create a dictionary in which each key is a word found in the string **line** and the value bound to the key is a tuple of size 3: (a) the first element of the tuple will be a list containing all the vowels that occur in the word, including duplications (this list will be empty if there is no vowels in the word); (b) the second element of the tuple will be a list of all the consonants between b and m, including duplications (this list will be empty if there is no consonants between b and m, in the word); (c) the third element of the tuple will be a list of all the consonants between n and z, including duplications (this list will be empty if there is no consonants between n and z, in the word).
   For example, if the text line received as parameter is "people enjoy programming", the function will create the following dictionary:

   {"people":(['e','o','e'], ['l'], ['p','p']), "enjoy":(['e', 'o'], ['j'], ['n', 'y']), "programming":(['o','a', 'i'], ['g','m','m'],['p', 'r', 'n'])}

   The function will return tuple containing two elements: the first element will be the value of lineNr, and the second element will the dictionary.

   Reminder: the str data type has a method "split"; for example:

   ```
   >>> s1 = "this is an example"
   >>> s1.split()
   ['this', 'is', 'an', 'example']
   ```

2. **`treatxtfile(flname)`**

   This function receives one parameter: **flname** (a string that is expected to be the full pathname of the text file. The function will read all the file, a line at-a-time and will process each line by calling the function **`treatline`**. The function will create a list of tuples of the kind that the function **`treatline`** returns. After eading all the text file, it will be closed, and the function will create a dictionary based on that list.

3. **occursumary(fldict**)

   This function receives one parameter: **fldict** (a dictionary of the kind that the function **`treatxtfile`** returns). The function will create a new dictionary in which every key is the number of the corresponding line in the text file, and the value bound to it is a tuple containing three elements: the first element is the the number of vowels that occur in that line, the second element is the number of consonants between b and m that occur in that line, and the third element is the number of consonants between n and z that occur in that line.

Write a Python program that requests from the user to enter the name (or full pathname) of a text file, and prints lines according to the following format:

LineNr nr of vowels nr of b-m consonants nr of n-z consonants

Note that each data item in each printed line must be centered relative to their corresponding title.

After printing all those lines, the program will print a summary line according to the following format:

Nr of Lines in text total nr of vowels total nr of b-m consonants total nr of n-z consonants

Note that each data item in that summary line must be centered relative to their corresponding title.

Question Nr. 3

Write the following functions:

1. **sortedzip(L)**
   This function receives a list of lists, sorts each one of the sub-lists in ascending order (using the **sorted** built-in function, and *not* the **sort** method), applies the built-in zip function upon them, and returns the resulting list. For example:

   ```
   >>> list(sortedzip([[3,1,2],[5,6,4],['a','b','c']]))
   [(1,4,'a'), (2,5,'b'), (3,6,'c')]
   ```

2. **reversedzip(L)**
   This function receives a list of lists, reverses each one of the sub-lists (using the **reversed** built-in function, and *not* the **reverse** method), applies the built-in zip function upon them, and returns the resulting list. For example:

   ```
   >>> list(reversedzip([[3,1,2],[5,6,4],['a','b','c']]))
   [(2, 4, 'c'), (1, 6, 'b'), (3, 5, 'a')]
   ```

3. **funczip(func, L)**
   This function receives two parameters: a functional object **func** (which behaves similarly to **sortedzip** and **reversedzip**), and a list of lists. The function will call **func** and pass **L** to it, and returns the result returned by **func**.

4. **unzippy(L)**
   This function receives a list of tuples **L**, of the kind that **funczip** returns. The function builds a list of lists of the same kind that **sortedzip** or **reversedzip** expects to receive as parameter.
   Note: At the beginning of the solution, you should create a list of empty lists. You could think that such a list can be created by using N*[ ], but this does not work as you may expect; try to create that list of empty lists using some alternative way.
   For example:

   ```
   >>> unzippy([(1, 4, 'a'), (2, 5, 'b'), (3, 6, 'c')])
   [[1,2,3],[4,5,6],['a','b','c']]
   ```

Write a Python program that requests from the user to enter a list **L** whose all elements are lists of the same size **N.** The program will show the user a menu of functions to apply upon the list **L**. The function will call **funczip** and pass it the function that the user chose and the list **L**. After the return from that function call, the program will do the following:

- print the result returned by the function that was called.
- apply the function **unzippy** upon that result
- print the result returned by **unzippy**

Note: Create a dictionary in which every key is the number of a menu option, and the value bound to it is the corresponding functional object.

Question Nr. 4

Let's try to write a program that every time that it is ran, it prints a "random poem" like the following example:

Iosi sees now small flowers

A stone sings suddenly black oranges

…..

In order to do this, you will need to create several global variables whose values are lists of words, each one having a different syntactic role in English sentences or phrases. For example:

peopleNames = ("Iosi", "Ety", …. (
verbs = ("sees", "plays", "sings", …..)
adjectives = ("tall", "small", "red", …..)
adverbs = ("slowly", "tomorrow", "now", "soon", "suddenly", …..)
animateObjects = ("flowers", "oranges", …))
inanimateObjects = '("a stone", "a chair", …..)

You are allowed to fill those lists with the words that you wish.

Note that those global variables should be initialized only once, and used only to retrieve data from them.

In order to implement this Python program, you will have to write the following functions:

1. function **theHumbletPoet(N)**
   This function receives a positive integer N as parameter, it will call the function crPoem (see below) in order to generate the list of lines of the random poem, and prints the poem one line at-a-time.
2. function **crPoem(N)**

This function receives a positive integer N as parameter, generates a list of N syntactically correct English sentences (actually, the random poem)

Write a Python program that requests the user to enter the number of lines that he wants the random poem to have. The program will generate and print the random poem by calling the function theHumbletPoet, passing it the number of lines that the poem should have.