CS251 Outlab 3 : Basic Python Programming

CS251 Outlab 3:
Basic Python Programming

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time.
- The submission will be graded automatically, so stick to the naming conventions strictly.
- The deadline for this lab is **Sunday, 25th August, 11:55 PM.**

Submission Instructions

```
outlab3-<team_name>/
│
├── Q1.py
├── Q2.py
├── Q3.py
├── Q4.py
├── Q5.py
└── references.txt
```

After creating your directory, package it into a tarball **outlab3-<team_name>.tar.gz**
Command: `tar czvf outlab3-<team_name>.tar.gz outlab3-<team_name>`

Submit once only per team from the moodle account of the smallest roll number.
The directory structure should be as follows (nothing more nothing less). Quote major references in `references.txt`.

TASK 1 - Command Line Arguments (15 Marks)

Write a python program (`Q1.py`) to compress multiple files or folders into a gzip compressed tar archive (`.tar.gz`) by taking the output archive filename followed by the names of files and folders to be compressed on the command-line as arguments.
The twist in the task is to create a *"flat archive"*, ie. it has only file members even if directories are to be included -- by flattening the input directories out. **See the example below.**

At least two arguments must be given: The name of the output file (not necessary to check if the

name ends in `.tar.gz`) and the names of files or folders that are to be compressed.

1. If fewer arguments are given, you should output the error message:
   `Too few arguments`
2. If all the files or folders are missing, you should print the message:
   `All files are missing`
3. If some of the files are missing, then compress the available files.
   Print the message:
   `Successfully compressed`
   followed by the relative paths of the files that were missing.

Your program will be invoked like:

```
python3 Q1.py <output-path> <input-path-1> <input-path-2> ...
```

Example:

Consider the following directory structure:

```
/home/labuser/universe
|-- one/
|   |-- a.txt
|   |-- b.txt
|   +-- c.txt
|-- d.txt
+-- INHERE/
    |-- three/
    |   +-- e.txt
    |-- Q1.py
    +-- f.txt
```

And since your script is in `INHERE/`, the shell is also in `INHERE` and the following invocation is made:

```
> pwd
> /home/labuser/universe/INHERE/
> python3 Q1.py foo.tar.gz three/ f.txt ../
> Successfully compressed
```

Produces an archive with only files in it, with fully qualified (absolute) names (use '-' instead of '/'):

```
-home-labuser-universe-one-INHERE-f.txt
-home-labuser-universe-one-a.txt
-home-labuser-universe-one-b.txt
-home-labuser-universe-one-c.txt
-home-labuser-universe-d.txt
-home-labuser-universe-INHERE-Q1.py
-home-labuser-universe-INHERE-three-e.txt
```

Clearly, files are not unnecessarily duplicated even when compressing overlapping directories with this simple strategy!

Hint: ***import tarfile, import os***

TASK 2 - Regular Expressions (15 Marks)

Your significant other (SO) has a habit of writing a diary every day in a text file. Now, you

somehow found the `MyDiary.txt` **(a sample is attached with this document)** from their system. You are curious to find who they contact or email regularly.
Write a Python program (`Q2.py`) to find all the email IDs and phone numbers and their occurrence frequencies from the text file. Dump all the contacts and their frequency (separated by a space in no particular order), including your own contact.
It is clear to you that you should have a serious discussion with your "SO" if you detect more conversations (same email/phone) with someone other than yourself (consider using a list comprehension here!).
First print your own occurrence frequency on a line:

```
my frequency: <frequency>
```

The program must report all these trespassers (in no particular order) if they exist:

```
Cheater alert! <trespasser's contact> <frequency>
Cheater alert! ...
...
```

else state,

```
It's all good yo!
```

You have to use regular expressions to find the email IDs and phone numbers.
Description of email IDs (tip: don't use something off the internet, use regexr.com or regex101.com for practice):-

<email id> = *<local part>* @ *<domain>*
<local part> = one or more *<alphanumeric>*s separated by *<dot_or_us>*
<alphanumeric> = one or more letters [a-zA-Z] or digits [0-9]
<dot_or_us> = The character `.` or the character _
<domain> = one or more *<alphanumeric>*s separated by *<dot>*. The last *<alphanumeric>* should be a *<alphabetic>*
<alphabetic> = one or more letters.

Sample email id:- fxps_ho.4@anhthu.org

Format of phone number:- 10 consecutive digits

Note:

1. 0123456789 is not a valid phone number as it starts with 0.
2. Also, 98765432100 is not valid as its length is 11.
3. Email and Number will be at a word boundary but may be surrounded or adjacent to punctuations.

Your program will be invoked like:

```
python3 Q2.py <path-to-diary> <contact>
# contact could be an email or phone no.
```

Sample output *(for 7758648932)***:**

```
my frequency: 7
Cheater alert! emokid@niceguys.com 10
```

TASK 3 - Operator Overloading (15 Marks)

Define a class **Dimension** which has 4 attributes: x, y, z, and d.
x, y and z represent the point in 3-dimensions. (All three are integers)
d is the 4th dimension and is used for storing the date. It is your choice on how to store the date

(dd, mm, yyyy). You can use list, tuple, etc.

Given that you are at a point, you can move to the next point in the dimension only if it takes you to the future (i.e. your current points time is less than the time of the point to which you want to move).

Create a constructor that will take the string as input and will assign the appropriate values to the attributes of the class.
Example:
String: `3,10,2 - 15/05/2019`
Assign: `x = 3, y = 10, z = 2, d = (2019,05,15)`

- Coordinates (x,y,z) can be negative too, but they will be integers.

Overload the subtract method of the class Dimension such that when called like: `Obj1 - Obj2` then it would return the Euclidean distance between the points only if its possible to move from `Obj1` to `Obj2` otherwise returns `-1`

Given the current point and the list of other points, your task is to find the nearest point to which you can move using the overloaded subtract method.

You also need to overload the `__str__` method to return the following kind of string:
`Coordinates: (3, 1, 2) Time: (2017, 08, 12)`

You need to print all the points other than the current point using the overloaded `__str__` method

Input 1:
```
3,10,2 - 15/05/2019                 # x,y,z - dd/mm/yyyy    the current point
5                       # n the number of points other than the current point
1,10,2 - 15/05/2022
3,1,2 - 12/08/2017
9,6,7 - 01/01/2000
-3,0,1 - 15/05/2025
1,0,0 - 28/02/2035
```

Output 1:
```
Coordinates: (1, 10, 2) Time: (2022, 5, 15)
Coordinates: (3, 1, 2) Time: (2017, 8, 12)
Coordinates: (9, 6, 7) Time: (2000, 1, 1)
Coordinates: (-3, 0, 1) Time: (2025, 5, 15)
Coordinates: (1, 0, 0) Time: (2035, 2, 28)
Closest point is:
Coordinates: (1, 10, 2) Time: (2022, 5, 15)
```

Input 2:
```
3,10,2 - 15/05/2019
3
1,10,2 - 15/05/2000
3,1,2 - 12/08/2017
9,6,7 - 01/01/2000
```

Output 2:

```
Coordinates: (1, 10, 2) Time: (2000, 5, 15)
Coordinates: (3, 1, 2) Time: (2017, 8, 12)
Coordinates: (9, 6, 7) Time: (2000, 1, 1)
Can't move to any point
```

TASK 4 - Lists and Dictionaries (15 Marks)

To keep track of scores during IPL matches, a dictionary is used as follows:

```
{
  "ipl1": { "rohit": 57, "virat": 38 },
  "ipl2": { "smith": 9, "warner": 42 },
  "ipl3": { "rahane": 41, "tare": 63, "russel": 91 }
}
```

Each player and match is represented by a string and the scores by integers.

Input:

Stats of each match will be presented on lines over `stdin`:
`<match_name>:<player_name>-<runs>,<player_name>-<run>,...`

Output:
Construct and print the dictionary with the structure mentioned above, followed by a list containing tuples (`name, total-score across all matches`) for each player sorted by the total-score in descending order, (and decreasing lexicographic order of player names).
Assume that,

1. match_name will be unique for all matches
2. player_name will be unique in a given match
3. runs will always be greater than equal to 0

For instance:
Input:

```
3
match1:p1-9,p2-38
match2:p3-19,P1-49
m3:p3-1,p4-6,p1-91
```

Output:

```
{'match1':{'p1':9, 'p2':38}, 'match2':{'p3':19, 'P1':49},
'm3':{'p3':1, 'p4':6, 'p1':91}}
[('p1', 100), ('P1', 49), ('p2', 38), ('p3', 20), ('p4', 6)]
```

NOTE: Here P1 and p1 are different!

TASK 5 - Argument Parser (20 Marks)

You have to create a database of students, which will contain First Name, Last Name, Roll Number, Gender, Mobile, Dept & CGPA. Write a Python program, which in the first run should take input from command line arguments and create a CSV file named **student_database.csv**. Subsequent runs should append the data to the same file.

The argument parser should be used for this task, because the order of arguments passed in the command line will be random. If any of the arguments is not present, your program should display an error message as shown in the following example.

Constraints:-

0 ≤ Number of arguments ≤ 7
Every argument will be of type “--<ARG_NAME>=AAA”.
“ARG_NAME” will be one of the 7 mentioned below (no need to worry about spellings)
Don’t worry about type of the “AAA” part, consider it as string.

Example1:-

```
> python3 Question5.py --first_name=abc --last_name=xyz
--roll_no=17305 --gender=male --mobile=1234567890 --dept=CSE
--CGPA=8.4

> Successfully Added!!
```

Example2:-

```
> python3 Question5.py --CGPA=8.0 --gender=male
--mobile=4321567890 --roll_no=17306 --last_name=asd
--first_name=zxc --dept=HSS

> Successfully Added!!
```

Example3:-

```
> python3 Question5.py --dept=IE --last_name=qwe --CGPA=9.4
--roll_no=17308 --gender=female --mobile=5684167890
--first_name=rty

> Successfully Added!!
```

Example4:-

```
> python3 Question5.py --dept=IE --last_name=qwe --CGPA=9.4
--roll_no=17308 --mobile=5684167890 --first_name=rty

> the following arguments are required: --gender
```

Example5:-

```
> python3 Question5.py --last_name=qwe --CGPA=9.4 --roll_no=17308
--mobile=5684167890 --first_name=rty

> the following arguments are required: --gender, --dept
```

Even though the order of command arguments changed it should create the CSV file **student_database.csv** containing the three lines that have been successfully added:

First Name, Last Name, Roll Number, Gender, Mobile, Dept, CGPA
abc, xyz, 17305, male, 1234567890, CSE, 8.4
zxc, asd, 17306, male, 4321567890, HSS, 8.0
rty, qwe, 17308, female, 5684167890, IE, 9.4

Hint:-
1. import **argparse**, import **csv**
2. Overwrite the parse.error method
If you are using PyCharm then you can give the arguments in “run -> run configuration -> parameters”