CS 251: Python Outlab 4

Please refer to the general instructions and submission guidelines at the end of this document before submitting.

P1. ArgParse Prime

Create **prime.py** which takes two arguments "**check_prime**" and "**range**" through argparse library. At least one of the arguments should be provided. If both arguments are missing then display a Error message.

- ➔ "**check_prime**" argument if provided, takes integer as value and prints "Yes" or "No" depending on whether the value is prime or not respectively.
- ➔ "**range**" argument if provided, takes two more integer values named "**lower_bound**" and "**upper_bound**" and prints total number of primes between **lower_bound** and **upper_bound** (both inclusive).
- ➔ **Error Handling** : Show error message if any of the given integer values is not between 1 to 1000 (both inclusive).

Example 1 :
```
> $python prime.py --check_prime 7
> Yes
```

Example 2 :
```
> $python prime.py
> Error : At least one of the following arguments are required: --check_prime, --range
```

Example 3 :
```
> $python prime.py --check_prime 51 --range 35 367
> No 62
```

Example 4 :
```
> $python prime.py --check_prime 2 --range 24 1001
> Error : Please enter a value between 1 and 1000 only
```

Example 5 :
```
> $python prime.py --check_prime -1 --range 432 857
> Error : Please enter a value between 1 and 1000 only
```

P2. Random+Pickle

Create **choice.py** which contains 2 tasks.

Task 1

Create fill_choice() which generates 100 unique choices of integers in range 1-5000, give each integer generated a unique name(string). Store this data using pickle in **new_int.p** .

Task 2

Create ask_choice() which reads the file created above from command line argument and ask user to input a number in range 5000-7000. Return a pair of integers(A,B) from the data generated, such that A+B equals the number entered by the user. If there is no such pair available in the data, then return (A,B) such that A+B<input.

Example:
new_int contains:

```
123 AVDA
1542 FDWS
8 QWQ
```

```
$python choice.py new_int.p
Enter Number: 1665
FDWS 1542 AVDA 123
```

```
$python choice.py new_int.p
Enter Number: 150
QWQ 8 AVDA 123
```

P3. Numpy

You are given a csv file **info_day.csv**, which contains information about temperature, humidity, etc. The file has following fields -
Day, Temperature (Celsius), Humidity(relative %), Light(lux), CO2(ppm)
The data has been collected in the morning.

Your job is to complete the following tasks -

Task 1

- Create **analysis.py** to print the mean and standard deviation values for each field on stdout.
- Store the data from CSV file into a numpy array and calculate mean and standard deviation.

The format should be as follows -

Field	*Mean*	*Std. Dev.*
Temperature	...	...
Humidity	...	...
Light	...	...
CO2	...	...

Task 2

- Create **convert.py** to convert the temperature from Celsius to Fahrenheit.
- Read **info_day.csv**, convert it to a numpy array and perform the above operation.
- Create another CSV file **transformed.csv.**
- The format will be same as **info.csv**, except that the temperature would be in Fahrenheit.

Task 3

New data has been collected which includes the information observed after sunset. This data is stored in **info_night.csv**

- Create **merge.py** to combine **info_day.csv** and **info_night.csv**.
- Name the merged file as **info_combine.csv**
- The format should be as follows -

Day	*Temperature(Day)*	*Temperature(Night)*	*Humidity(Day) ... and so on*
Monday	...	...	...
...	...		

P4. KFC Again

You might have done CS152 which goes by the name Abstractions and Paradigms of Programming. How many of you have really understood what the title meant? What is an abstraction? If you did not, you might really want to understand the key word **abstraction**. Google it!

KFC store at Hiranandani wants to update their ordering system. Their ordering system previously consists of a Menu with MenuItems. Each MenuItem was a list of **Name, Cost, Rating**. And Menu was a list of MenuItems. So a menu was just a list of lists of MenuItems. But now due to seasonal sales, KFC decided to put different menus at different stores according to most bought items from that store. That would become difficult to manage as then the menus will then have a data structure of list of list of lists.

Follow the instructions given in inlab question 5.
You should have created two classes named **Menu** and **MenuItem** in **menu.py** and **menuitem.py** respectively.

Implement **__str__** method in **Menu** class to print all the items in a menu in the format as specified below.

Item: Lime Krusher, Cost: 75, Rating: 4.300000
Item: Chicken Pizza, Cost: 298, Rating: 4.000000
...

As a test before continuing with the next part, copy **kfctest.py** to dir containing your python files and execute **kfctest1.py**, the output should match with **kfctest1.output** (both provided in P4 dir) file. You may have to change the import statements according to your dir structure. Once your output matches with **kfctest.output** you are ready to continue.

One problem with the above implementation of **Menu** and **MenuItem** is that a menu object doesn't detect duplicates in its menuitems. The underlying reason for this is that a *list* is used in **Menu** to store **MenuItem**s.

This can be easily solved by using *set* instead. Create a file **setmenu.py** which contains class **SetMenu**. **SetMenu** class is similar to **Menu** class except that the field **items** in SetMenu class is a set rather than a list. Note that parameter passed to **__init__** method of SetMenu is still a list. The **__init__** method in SetMenu has to convert that argument list into a *set* and store it in **items** field. Once that is done copy **kfctest2.py** in your dir, execute it and check your output with **kfctest2.output**. Submit 3 files **menu.py setmenu.py** and **menuitem.py**

P5. JSON

JSON is an abbreviation for Javascript Object Notation. It is simple and effective way of representing data structures. Due to its simplicity it has become a standard of data representation and transfer in the industry. One of the main advantage of JSON notation above other forms of data representation like CSV is that JSON need not have a fixed structure.

Task 1

Given a json file, create **parse.py** to parse the json data and store it into a csv file.
The json file is **infinity_stones.json.** Your task is to parse the json file, extract the data and store the data in a csv file **infinity_stones.csv**.

The format of the **infinity_stones.csv** is as follows

infinity_stones.csv

```
-------------------------------------------------------------
| Stone Name | Stone Color | Containment Unit |
| Space Stone | Blue           | Tesseract            |
| Mind Stone  | Yellow        | Scepter               |
        ...              ...                    ...
-------------------------------------------------------------
```

Task 2

Thanos has now retrieved all the infinity stones and they reside in his Infinity Gauntlet. Your task is to update the Containment Unit column of each stone with Infinity Gauntlet.
Create **update.py** to complete the above mentioned task. You need to write the updated json data to a file **update_stones.json**

P6. Exception Handling

Task 1

- Create a file named **teams.py** to solve the following problem.
- Create a function **makeTeams()** that takes number of teams and number of players available as arguments, divide the players amongst the teams such that each team has equal number of players. If this is not possible raise ValueError saying to remove the remaining number of players left from the process. Handle ValueError, ZeroDivisionError.

Task 2

- Create a json database of 100 players, assign them to 10 teams exclusively, assign unique jersey number to each player, assign loyalty to each player. Read a file named transfer.txt, which contains information of transfer of some players. When a transfer happens, requested player is taken away from a team and in return a player with minimum loyalty is returned. Players with loyalty greater than 7 cannot be transferred, if a transfer request of such player arrives, consider it as an exception and handle it accordingly by asking for a different transfer. At the end, count the number of players actually transferred. Handle exceptions like if a player with jersey number does not exist, or if a team with certain name does not exist.
  (Hint: json database can contain dictionary of dictionary also along with many other things.)

Constraints:

    0<Loyalty<=10.
    0<Jersey Number<=1000.
    Team_Name = [A...J].

Example:

Structure of transfer.txt:

Team_name(to which a player is transferred) Jersey_number(#of player to be transfered)

Team_name	Jersey_number
A	100
D	1741
X	1

Output:

```
Transfer Complete
```

Try another transfer(Wrong player number)
Try another transfer(Wrong team name)

P7. Orientations

Orientations of objects in 3D space can be expressed in many forms. Of course each has its own advantages and drawbacks in terms of ease of computation and simplicity in visualization. Two such drastically different representations shall be dealt with, one Euler Angles (pronounced `oiler`) and the other Quaternions. One is easy to capture and the other was designed to deal with the famous Gimbal Lock problem (read at your own risk) among other things.

Euler Angles are three angles when applied to an object in a neutral orientation, takes up the represented orientation.
Quaternions are 4-dim vectors whose normalized versions represent an orientation.

The task is to build a module that performs the interconversion between Euler Angles and Quaternions.

- **Task:**
  - ➔ Create a file named **qe.py** which contains two functions **quaternion_to_euler** and **euler_to_quaternion**.
  - ➔ **quaternion_to_euler (quaternion)**

    Input: [N,4] shaped ndarray with each vector in the second dimension a quaternion
    Output: [N,3] shaped ndarray with each vector in the second dimension the corresponding quaternion.
    Function:
        Normalize the input quaternion for them to represent orientations.
        Compute the euler angles as per the algorithm in the reference.
        Be sure to convert the angles into degrees
  - ➔ **euler_to_quaternion (euler)**

    Input: [N,3] shaped ndarray with each vector in the second dimension representing euler angles in degrees
    Output: [N,4] shaped ndarray with each vector in the second dimension the corresponding unit quaternion.
    Function:
        Convert the angles in radians into degrees
        Compute the quaternion as per the algorithm in the reference.

Normalize the quaternions

- ➔ Euler Angles are to be in degrees in the range [-180°,180°]
- ➔ You are expected to implement vectorized versions of the algorithms and thus are allowed to use neither for loops nor 'np.apply_along_axis' (FYI., np.apply_along_axis can be potentially used to apply a function independently to each vector along an axis).

  Almost all common operators are overloaded by numpy by default to deal with ndarrays The functions in numpy do so anyway. Refer to a part of this to understand better.
- ➔ Write a program in file **use_qe.py** which imports these functions from **qe.py** (present in the same directory), reads input from a file and outputs into a csv file based on a flag which determines the direction of conversion.

  <q2e> being 0 implies conversion from quaternion to euler and 1 implies conversion the other way (note the C-like semantics)

  All data is stored in files in the csv format (you might want to use \`np.savetxt\` and \`np.loadtxt\`)
- ➔ If the input is not consistent with the direction of conversion, set the exit code to 1 and exit.
- ➔ Your functions might be used in bigger programs to test their working and validity. Do not deter from the function specifications.
- ➔ To submit: **qe.py**, **use_qe.py**
- ➔ A script **lsref.py** that saves/loads data to/from csv files has been provided for reference.

Usage:

```
$ python use_qe.py <input_filename> <output_filename> <q2e>
```

General Instructions

- Make sure you know what you write, you might be asked to explain your code at a later point in time
- Your code may be tested on hidden test cases
- Grading is done automatically, so please make sure you stick to naming conventions
- The deadline for this lab is **Sunday, 12th August, 23:55.**

Submission Instructions

After creating your directory, package it into a tarball **`<rollno1>-<rollno2>-<rollno3>-outlab4.tar.gz`** in ascending order. Submit once only per team from the moodle account of smallest roll number.

The directory structure should be as follows (nothing more nothing less)

```
<rollno1>-<rollno2>-<rollno3>-outlab4
    ├── P1
    │   └── prime.py
    ├── P2
    │   └── choice.py
    ├── P3
    │   ├── analysis.py
    │   ├── convert.py
    │   └── merge.py
    ├── P4
    │   ├── menuitem.py
    │   ├── menu.py
    │   └── setmenu.py
    ├── P5
    │   └── parse.py
    ├── P6
    │   └── teams.py
    └── P7
        ├── qe.py
        └── use_qe.py
```