Inlab 4 - Python Advanced

Though the inlab tasks are un-graded, the outlab tasks may directly extend the inlab tasks. Therefore you may want to do in-lab questions properly.

You can do inlab in groups.

Python is a beautifully designed language. The Zen of Python tells the philosophy behind it.

Do the following tasks in PyCharm. Be sure to watch the intro video of PyCharm in its official website.

1. HighFive

- **Task 1**
  Create **highfive.py** which takes input from User in form of "High" or "Low". Your task is to generate random integer number from 1 to 9 (**except 5**) and compare it with user's input. Guessing of "High" or "Low" refers to whether the generated random number will be higher than 5 or lower than 5.
  (Hint: Import random module)

  Example 1:
  ```
  > python highfive.py
  > High or Low? : High
  > Generated random number was 6. You guessed Correct.
  ```

  Example 2:
  ```
  > $python highfive.py
  > High or Low? : Low
  > Generated random number was 8. You guessed Incorrect.
  ```

- **Task 2**
  Perform the same task 1 without using random library. Create your own random function.
  (Hint: Import time module and use slice notations)

2. Pickle (Data Serialization and Persistence)

Find out what data serialization means.
Similarly, find out what persistence of data means.

- **Task 1**
  Create **movies.py** where you create a class **Movie**. The class includes member variables 'movieName' and 'rating'.

- **Task 2**
  - Create **store.py** which imports Movie and creates objects.
  - Create function addMovie(movie-name, rating) which creates Movie objects.
  - Create a list of Movie objects.
  - Use pickle library to store the list. (Import pickle module)

- **Task 3**
  - Create **load.py** which imports Movie and reads from the pickled list.
  - Print the movie information on stdout in following format (Descending order based on the rating)-

```
Movie       Rating
Xyz         5
Abc         3
Pqr         3
```

- **Task 4**
  - Create **edit.py** which takes user input through stdout.
  - Get movie-name and new rating from user and update the rating of the mentioned movie.
  - Access the pickled list from your python code.
  - Print the movie information similar to the format in Task 3.

3. CSV

What is the difference between a regular file and a database?

- **Task 1**

  Your job is to create and manage a list of students table, which contains columns First Name, Last Name, Roll Number and Department stored as a Comma Separated File (CSV).

  Create **student_db.py** which takes input from the command line arguments and creates a CSV file named **student_db.csv**. Use argument parser for this task. If **student_db.csv** already exists then the new information should get appended. Display error message if any argument is missing.

  Constraints:
  Number of Arguments = 4
  Every argument will be of type "--argument_name=argument_value".

  Example 1:
  ```
  > $python student_db.py --first_name=abc --last_name=xyz --roll_no=17005
  --dept=CSE
  > Successfully added to student_db.csv
  ```

  Example 2:
  ```
  > $python student_db.py --roll_no=17016 --last_name=asd --first_name=zxc --dept=HSS
  > Successfully added to student_db.csv
  ```

  Example 3:
  ```
  > $python student_db.py --last_name=qwe --roll_no=17008 --first_name=rty
  > the following arguments are required: --dept
  ```

  CSV file structure should be as followed:
  ```
  First Name, Last Name, Roll Number, Dept
  abc, xyz, 17005, CSE
  zxc, asd, 17016, HSS
  ```

  (Hint: Import argparse, Overwrite parse.error)
  (Note: If you are using PyCharm then you can give the arguments in "run -> run configuration -> parameters")

- **Task 2**
  Create **alter.py** which reads the CSV file **student_db.csv** created in Task 1 and modify the dept name to CS from CSE. The structure of CSV file should remain the same.
  (Hint: Use csv reader)

4. CSE

- **Task 1**
  Create **cseuser.py** which returns the cse username of user. Use the URL https://www.cse.iitb.ac.in/page222?batch=BTech2 to get CSE username.
  (Hint: Use urllib library and regular expressions/import re)

  Example 1:

  ```
  > $python cseuser.py UG17 TEMPLATE
  > CSE Username of UG17 TEMPLATE is "ug17_template"
  ```

5. KFC

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.

So we are going to create a new architecture of Menu using abstractions.

1. Create a class **MenuItem** which has member variables (also called fields) **name, cost, rating** in **menuitem.py** in which constructor takes the same arguments and initializes the fields
2. Create a class **Menu** which contains a member variable **items** which is a list of objects of type **MenuItem** in **menu.py**
    a. The constructor of **Menu** takes a list of **MenuItems** as a parameter and initializes **items** field with that
    b. Now a collection of menus can be represented by a list of **Menu** objects in python instead of nested hell of lists
3. A new chef has an idea of a new menu item, but he wants to check if it already exists in the menu, to do this we need to define what equality between two **MenuItem** objects mean. This can be neatly done by overriding **__eq__** method of **MenuItem** class. Find what it does and how to use it.
    a. If you correctly implement **__eq__** **MenuItem** objects can be directly compared using == instead of writing another function which compares them. For ex

```
  i.   a = MenuItem("Chicken Popcorn", 100, 4.7)
 ii.   b = MenuItem("Chicken Zinger", 200, 4.8)
iii.   a == b # returns false
```

4. Store workers doesn't care what data structure you use, all they want is a proper print of Name and Cost on the screen. To do this you may write a function which takes in a **MenuItem** object as argument and prints the name and cost. But there is a better way of doing that by overriding **__str__** method in **MenuItem** class. Find how to do that. If you successfully implement it, you can do the following.

a. `a = MenuItem(“Chicken Popcorn”, 100, 4.7)`
b. `print(a) # Item: Chicken Popcorn, Cost: 100`

5. Store Manager wants to know how many items there are in a Menu. The trivial way of doing this may be finding the length of **items** field in **Menu** class. But as we are smarter than that by now we override **__len__** function in **Menu** class. If successful the following should be possible
   a. `m = Menu([a, b]) # a, b defined previously`
   b. `len(m) # prints 2`
6. At the end of a season, Mr. Kentucky wants to know the stats of sales. One important factor in sales is the **rating** field in **MenuItem** class. He wants to sort the menu according to rating with highest rating item first in the list. Traditionally you may sort using a separate function. But we both know that’s not gonna happen. To do this right, you will have to override **__hash__** and **__lt__** method of **MenuItem** class and call **sort()** (import from list module) on the **items** field. This might be useful. If done right the following will be possible

```
a. sort(m.items) # m defined previously
b.               # sorts and replaces m.items
c. print(m.items[0])# prints Item: Chicken Zinger, Cost: 200
d. print(m.items[1])# prints Item: Chicken Popcorn, Cost: 100
```