MIT AITI
Python Software Development
Lab 06: Object-Oriented Programming

In this lab, you will implement two versions of a program in order to better understand the difference between imperative and object-oriented programming. Please read the instructions carefully.

Part I – Imperative Programming
Open the Python file called "Lab06_part1.py"
Do all your work for part I in this file

1. **Scoring**
There are a few ways that we could store information about the number of goals a player has scored on a particular date. We will give you a few options, but think about what types of functions you might call on the data structure before making your final decision. Look ahead to question 2 to see some functions that you will have to implement with your chosen data structure.

Some possibilities:

- List of (player,date,number_of_goals) tuples. For example, ('Beckham',datetime.date(2011,06,24),2)
- Dictionary where the player name is the key and a list of tuples with (date,number_of_goals) as the value
- Dictionary where the date is the key and a list of tuples with (player_name,number_of_goals) as the value.

The date should be stored as a datetime.date object and the score value should be stored as an int. Lookup datetime.date for more information on it.

Call this data structure `player_stats,` and create it using the data table described in the comment

Explain your choice of data structure in 2-3 sentences with another comment.

2. You now have the data structure in place for a very basic sport statistics program. The next step is to create functions that return the data you want when called. Implement the following functions, in any order.

a. `highest_score(player_stats)` - returns the highest score in a single match by anyone in the system in a tuple: (player name, date of score, score).

b. `highest_score_for_player(player_stats,player)` - returns the highest score by the supplied player in the same tuple as above, or None if the player is not in the system.

c. `highest_scorer(player_stats)` – returns the name of the player with the highest scoring sum, i.e. the total of all the goals/runs/etc. stored in the system.

Part II – Object-Oriented Programming I – Data Encapsulation
Create a new file Lab06.py and do the rest of your work in that file!

By now you should see that designing, maintaining, and extending code based around loosely coupled data held in lists is tedious and not straightforward to understand. We now reimplement the same problem using object oriented methods in order to see the difference between the two approaches.

1. Create a new class called `Player` as follows

```
class Player:
      def __init__(self,firstname,lastname,team=None):
      self.first_name = firstname
      self.last_name = lastname
      self.scores = []
      self.team = team
```

2. Add a method to the class Player `add_score(self,date,score)` which will append new scores as they occur.

3. Add methods `total_score(self)` and `average_score(self)` to the class `Player`. These methods calculate the total and average score of the player.

4. Create the player Fernando Torres (use the variable name `torres`) and add the following five scores: 0, 0, 1, 0, and 1. Here's an example player:

5. Call the `average_score` method for on `torres` you created in the previous part and print out Torres' average score.

Checkpoint!
Call over a Lab Assistant to check and talk about your code.

The player class you've created will be reused and augmented in the rest of the lab, so please make sure it is correct. Ask an instructor or assistant for help if you are lost.

Part III – Object-Oriented Programming II – Your First Classes

While we provided the skeleton for the Player class in the previous lab exercise, we are leaving the initial implementation of subsequent classes completely to you, though we do specify certain behaviors.

1. Create a class called Team. The class should contain the following attributes:
   `name`
   `league`
   `manager_name`
   `points`
   `players` – a list of players, initialized to an empty list

2. All of these attributes except 'players' should be required in the `__init__` method. Note that none of the code of Player class needs to be changed even though Team is now a class and not a string – this is one of the benefits of dynamically typed languages.

3. Write the method `add_player(self,player)` in the Team class, which adds the given player to the list of the teams player

4. Initialize two teams, in variables `portugal` and `spain`, using the constructor you defined in the previous part.

5. Define the `__str__(self)` method for the Team class, which defines what will be called when someone tries to `print` a Team object. What makes sense here? When I write `print spain`, what would be a sensible output?

6. Recreate the player `torres`, this time using the `spain` Team object in the constructor instead of the string 'spain'. Similarly, create the player `ronaldo`, using `portugal` as the Team object in the constructor instead of the string 'portugal'.

7. Use the `add_player` method of `portugal` and `spain` to add `ronaldo` and `torres` to the proper teams. (Ronaldo plays for Portugal, and Torres, Spain)

8. Create a class called Match. The class should contain the following instance attributes:
   `home_team`
   `away_team`
   `date` (an instance of the `datetime.date` class)
   `home_scores`
   (dictionary with player_last_name : player_score key:value pairs)
   `away_scores`
   (dictionary with player_last_name : player_score key:value pairs)

For example, if Spain is the home team and Torres scored one goal in the game, `home_scores` would be { 'torres' : 1 }

9. Implement methods `home_score(self)` and `away_score(self)` which return the sum of the player scores in `home_scores` and `away_scores`. Each should return 0 if no one on the team has scored.

10. Implement the method `winner(self)` which returns the Team which has the higher score, as determined by the sum of the scores of each team.

11. Implement the method `add_score(self,player,score)` which takes a Player object and an integer score and:

    a. Determines which team the Player belongs to using his `team` attribute
    b. Determines whether that team is the home or away team
    c. Adjusts `home_scores` or `away_scores` to reflect the new goal by this player – if he has already scored, incrementing the value stored at his last name, and otherwise, inserting his last name with a score of 1

12. Create the match `euro_semi_final`, played between `portugal` and `spain` (use Spain for the home team) on June 27, 2012

13. Use the `add_score` method to add a score by `torres`, a score by `ronaldo`, and then another score by `torres`.

14. Now, print the winner of the game using the winner method you defined earlier.

GOOD JOB!!!! THIS STUFF IS HARD!