MIT AITI
Python Software Development

Lab 06A: 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.

➔ **Create a new project named "lab06"**
➔ **Create a new Python file called "lab06-1.py"**

Part I – Imperative Programming

Your goal is to create a program which can store and return various statistics about sports players, without the use of inheritance or class structures. In order to accomplish this, you will create a series of lists, each containing a property of the person.

1. The system you're building should, at minimum, know each player, what team he or she plays for, and some sense of his or her scoring. In a new python file, define three `List` data structures called `nameList`, `teamList`, and `scoreList`.
    a. Populate `nameList` and `teamList` with five players in whatever sport you want your system to support, e.g:

    ```
    nameList=["Asamoah Gyan","Michael Essien","Didier
    Drogba","Emmanuel Adebayor","Samuel Eto'o"]
    ```

    and

    ```
    teamList = ["Sunderland","Chelsea","Chelsea","Man
    City","Inter"]
    ```

2. Consider how you might store the information regarding scoring. For football, you could record total goals scored but that doesn't give you any information about whether the goal was for a club or national team, when it was scored, against whom, etc. Similarly for cricket, you could record total runs scored, or strike rate, but you would be missing much of the important information. For simplicity, assume you want to record some score (goals scored, runs scored, wickets taken, saves, etc.) and the date it occurred.

    You may implement this as you like, using tuples, dictionaries, or lists, as long as you can return the date as a `datetime.date` and the score value as an `int`.

3. You now have the data 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:
    a. `def highestScore()` - returns the highest score by anyone in the system in a `tuple`: (player name, team, date of score, score).
    b. `def highestScoreForPlayer(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. **Hint:** You can find the index of an item in a list by using the `index()` function, e.g. [“a”,”b”,”c”].index(“b”). You will, however, need to catch an exception if the item is not in the list, so it may be more straightforward to iterate.

c. `def highestScorer()` – 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.
d. `def highestAverageScorer()` – returns the name of the player with the highest average, i.e. the total of all the goals/runs/etc divided by the number of matches.
e. `def addScore(player,date,score)` – adds a new score element to the score datastructure.

4. Now imagine your application has caught the attention of a local web company who wants to use it for their sports reporting. However, the information provided has to be much deeper; in addition to scoring they want (using football as an example) things like dates with different teams, minutes played in each match, player birthdays and age, height, nationality, appearances, injuries, etc. This is, of course, represents many, many lists.

   Choose two additional statistics, of which one must be dynamic – i.e. changing as time passes - and add new data structures to hold their values. Also add functions to get and set these values.

Part II – Object-Oriented Programming I – Structs and Data Encapsulation

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.

➔ **Create a new Python file called “lab06-2.py”**

1. Create a new class called `Player` as seen in the lecture:

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

   Object oriented programming in Python is typically getter/setter-free, so this can serve as the entirety of the definition. Note that `__scores` has the leading underscores so that Python will ‘disguise’ its name to make it less convenient to directly access, as we don’t want the scores directly meddled with. **Hint:** please note that attributes defined outside of the scope of `__init__` are **class** rather than **instance** variables (i.e. like `static` variables in Java).

2. Add a function `def addScore(self,date,score)` which will append new scores as they occur.

3. Add functions `def totalScore(self)` and `def averageScore(self)` which calculate the total and average score of the player. Create a player and add some scores using `addScore`, then check that the results are correct before moving on.

4. To keep things simple, we will define the remainder of the functions and variables at the module scope.
    a. Add a list for players: `__players = []` **Hint:** You can use another datatype here as a `List` is inefficient for this purpose. (Why?)
    b. Add a function for adding new players `def addPlayer(player)` which just appends the player to the end of the list, or adds it to your chosen datastructure.
    c. Re-add the players you defined in lists in 1a. from Part I as `Player` objects using `addPlayer`. You can also choose to initialize them in the initial variable assignment statement, e.g.
    ```
    __players = [Player(”Didier”,”Drogba”,”Chelsea”),...]
    ```
    d. Reimplement the functions in 3a.-3e. from Part I. For 3e., the function signature should change to `def addScore(firstname,lastname,date,score)`

5. Your solutions to this lab will be evaluated both for correctness and for style. The `Player` class you’ve created will be reused and augmented in Lab 06B, so please make sure it is correct. Ask an instructor or assistant for help if you are lost.