COL100: Introduction to Computer Science
Assignment 5

April 2, 2025

5.1 Instructions

Unlike previous assignments where we were string matching your output with the expected data. We will be checking if the classes and objects are implemented correctly. The first four parts of the assignment will require you to submit *classes only*. We will implement the objects and thus check if they have expected members and member values. To facilitate this please:

- Keep the names of classes and data members exactly same as in this document.
- In the case where any value does not pass the sanitation checks return -1.
- If a function reaches a fail condition it should return -1, like in the case of compute score where the student may not be eligible.
- We understand that Python does not have true private and protected members. By convention, private members are created with adding __ before the name of the member. Python uses something called name mangling, where it attaches the name of class to the member.
- Each class has a `constructor`, `get_values()`, `set_values(data_attributes)` and `show_values()`, even if not explicitly specified.
- The `set_values` function will be passing a subset of all available data members.
- The object of the class of your submitted class is created by running the following:

```
st_dict = {"name": "John Doe", "id": 1, "birth_year": 2005, "
    birth_month": 6, "birth_day": 15, "gender": "Male", "
    registration_id": 1001, "grade_level": 10, "class_assigned": "
    10A", "gpa": 6.0, "selected_activity": "Sports", "talent_score"
    : 8.5, "athletic_score": 7.0, "leadership_score": 9.0}
Student(**st_dict)
```

5.2 Problem Statement [100 Marks]

Your school is hosting its Annual Talent Hunt Festival where students demonstrate their skills in academics, sports, and artistic activities. As the coordinator for the event, you are responsible for organizing the participants, managing the logistics, and ensuring a fair evaluation of their performance.

The school has shared files containing details of participants and activities. Your task is to build a Python-based system to manage and analyse the data efficiently by applying

Object-Oriented Programming (OOP) principles. You will use this system to handle tasks such as identifying top artists, categorizing students based on their talents, and generating reports for the event.

Event Overview

The event involves participants and activities as discussed below.

Participants

1. Students participate in various activities and are categorized based on their interests, such as Athletes, Artists, or Scholars. Some students may take on leadership roles, such as TeamLeads.
2. Teachers act as Mentors or Judges, guiding or evaluating participants.

Activities

1. Activities are classified into three main types: Sports Tournaments, Talent Shows, and Academic Competitions.
2. Each activity type has further specializations:
   - Sports Tournament: Team-based and individual games.
   - Talent Shows: Activities such as Music, Singing, and Dance.
   - Academic Competitions: Events like Science, Mathematics, and Computers.

5.2.1 Participant Class [5 Marks]

Write a class for `Participant`. This class will be the building block for the rest of the roles. Note that information such as gender, birth date, month, and year should be made private. The Participant class should have the following elements for each of its instances:

```
name: str
idi: int
birth_year: int
birth_month: int
birth_day: int
gender: str
get_values(self) -> tuple
show_values(self) -> None
set_values(self, data_attributes: dict) -> None
calculate_age(self, curr_day: int, curr_month: int, curr_year: int) ->
    int
```

- `get_values(self)`: Returns all attributes of the participant as a tuple.
- `show_values(self)`: Prints all participant details for easy human-readable display.

- `calculate_age(self)`: Computes age using the current date. Returns the age in years. Performed using `math.floor()`
- `set_values(self, data_attributes:dict)`: Sets both inherited and new attributes.

5.2.2 Student and Teacher Class [10 Marks]

Create the following two subclasses of participants.

Student

Using the *Participant* class we need to build the `Student` class. If we look at our scenario again, we need to express the profile of a student fully. In order to accomplish this, we will be inheriting the `Participant` class and adding more features to it. Details other than name and idi should be private variables to the class. Accordingly, implement a `Student` class which has the following features:

```
age: int
grade_level: int
class_assigned: str
gpa: float
selected_activity: str
talent_score: float
athletic_score: float
leadership_score: float
eligible: bool
is_eligible(self) -> bool
get_values(self) -> tuple
show_values(self) -> None
set_values(self, data_attributes: dict) -> None
```

- *age* is a derived variable set using calculate_age function and with date 1st January 2025. This is an integer value which is derived at the time of object creation.
- `grade_level`: The grade level the teacher mentors (e.g., 10).
- `class_assigned`: The specific class assigned to the teacher (e.g., "10A").
- *selected_activity* type of activity selected by student from *"Sports"*, *"Talent"*, or *"Academic"*.
- `is_eligible()`: Calculates eligibility based on average scores and GPA and updates eligible feature. The `is_eligible` checks if the student has a GPA of at least 5.0, if not they are not eligible to participate in any activities. It also sets the value of the data member.
- `get_values()`: Returns all inherited and new attributes as a tuple.
- `show_values()`: Prints all student details, including those from Participant.
- `set_values(self, data_attributes: dict)`: Sets both inherited and new attributes. Not all data members will be present in data_attributes. Also updates the derived variables.

Teacher

Using the `Participant` class, we will build the `Teacher` class next. As per our scenario, we need the teacher to also be participant. This can be done by inheritance of the `Participant` class. Data members of the class Teacher should be made private. Implement the `Teacher` class with the following attributes:

```
subject: str
mentor_grade: int
mentor_class: str
judge: bool
get_values(self) -> tuple
show_values(self) -> None
set_values(self, data_attributes: dict) -> None
```

- `subject`: The subject taught by the teacher (e.g., "Mathematics").
- `mentor_grade`: The grade level the teacher mentors (e.g., 10).
- `mentor_class`: The specific class assigned to the teacher (e.g., "10A").
- `judge`: A boolean indicating if the teacher serves as a judge.
- `get_values()`: Retrieves all the teacher's details along with inherited participant details.
- `show_values()`: Displays all the teacher's details in a user-friendly format.
- `set_values(self, data_attributes: dict)`: Sets both inherited and new attributes. Not all data members will be present in data_attributes. Method also updates all derived variables.

5.2.3 Artist, Athlete and Scholar Class [15 Marks]

Create the following specialized subclasses of students.

Artist

This class represents students who excel in specific artistic talents (e.g., music, dance, drama). Add attributes like performance level and additional methods related to their artistic talents. The data should be made private and only accessible through helper functions. Implement the following:

```
performance_level: float
talent: str
compute_scores() -> float
is_eligible() -> bool
get_values() -> tuple
show_values() -> None
set_values(data_attributes: dict) -> None
```

- `talent`: The selection made by the student (e.g., "Singing").
- `performance_level`: The numerical indicator of current performance, the same as talent score.

- `is_eligible()`: A student is eligible to participate in in performance based activities if their GPA is over 6.0, they need to be of age 16 or older on January 1, 2025. It also sets the value of the data member *eligible*.
- `get_values()`: Retrieves all artist details along with inherited participant details.
- `compute_scores()`: If the student is eligible to take part in the activity, their performance level is their score.
- `show_values()`: Displays all attributes details in a user-friendly format.
- `set_values(self, data_attributes: dict)`: Sets both inherited and new attributes. Not all data members will be present in data_attributes. Method also updates all derived variables.

Athlete

This class represents students participating in sports. Add attributes like sports_category (e.g., "Team" or "Individual") and fitness_score. The data should be made private and only accessible through helper functions. Include methods to calculate average performance across multiple sports.

```
sports_category : str
fitness_score : float
performance_level: float
compute_scores() -> float
is_eligible() -> bool
get_values() -> tuple
show_values() -> None
set_values(data_attributes: dict) -> None
```

- `sports_category`: The sport selection made by the student (e.g., "Football").
- `fitness_score`: The numerical estimate of athletic ability.
- `performance_level`: The numerical modifier of current performance, the same as athletic score.
- `is_eligible()`: Calculates if the student is eligible for a given activity. A student is eligible to participate in in performance based activities if their GPA is over 5.5, they need to be of age 12 or older on January 1, 2025. It also sets the value of the data member *eligible*
- `compute_scores()`: If the student is eligible to take part in the activity, calculate their score as `fitness_score` * `performance_level`.
- `get_values()`: Retrieves all athlete instance details along with inherited participant details.
- `set_values(self, data_attributes: dict)`: Sets both inherited and new attributes. Not all data members will be present in data_attributes. Method also updates all derived variables.
- `show_values()`: Displays all attributes details in a user-friendly format.

Scholar

This class represents students excelling in academics. Add attributes like subject_specialization (e.g., "Mathematics", "Science") and olympiad_scores. The data should be made private and only accessible through helper functions. Include methods to compute cumulative scores for different subjects.

```
subject_specialization  : str
olympiad_scores : list[float]
performance_level: float
is_eligible() -> bool
compute_scores() -> float
get_values() -> tuple
show_values() -> None
set_values(data_attributes: dict) -> None
```

- `subject_specialization`: The subject selection made by the student (e.g., "Mathematics").
- `performance_level`: The numerical modifier of current performance. This is defined by GPA * 10.
- `olympiad_scores`: A list of scores achieved by the student.
- `is_eligible()`: Calculates if the student is eligible for a given activity based on Olympiad scores and GPA. A student is eligible to participate in in performance based activities if their GPA is over 8.0, they need to be of age 10 or older on January 1, 2025. They should also have achieved an `olympiad_scores` over 80 at least once. It also sets the value of the data member *eligible*
- `compute_scores()`: If the student is eligible to take part in the activity, calculate their score as `olympiad_scores` * `performance_level`.
- `get_values()`: Retrieves all academic instance's details along with inherited participant details.
- `set_values(self, data_attributes: dict)`: Sets both inherited and new attributes. Not all data members will be present in data_attributes. Method also updates all derived variables.
- `show_values()`: Displays all attributes details in a user-friendly format.

5.2.4 Activity Class [20 Marks]

Activity

We will be writing the `Activity` class next. This would be later used to make subclasses for different sports, artistic, and academic activities. Remember all data attributes other than ID and name should be private. While making the `Activity` class, give it the following attributes:

```
activity_id: int
activity_name: str
activity_type: str
```

```
max_participants: int
grade_level: int
is_active: bool
participants: list[Participants]
organizers: list[Teachers]
get_values() -> tuple
show_values() -> None
set_values(data_attributes: dict) -> None
```

- `activity_id`: Public unique ID for the activity.
- `activity_type`: Public type of the activity. Types of activity (e.g., "Sports", "Talent", or "Academic").
- `max_participants`: Maximum number of participants allowed.
- `is_active`: Whether the activity is currently active or not.
- `participants`: List of Student objects involved in the activity.
- `organizers`: List of Teacher objects organizing the activity.
- `get_values()`: Returns all attributes as a tuple. Converts participants and organizers to their names for readability.
- `set_values()`: Updates private attributes.
- `show_values()`: Prints all the activity details in a user-friendly format.

SportsTournament

Using the above `Activity` class, let's make `SportsTournament` specialization. With the following data and method attributes. game_type and duration_minutes should be private members.

```
game_type: str
duration_minutes: int
determine_winner() -> Athlete
show_values() -> None
```

- `game_type`: Specifies the type of game ("Team" or "Individual").
- `duration_minutes`: Duration of the game.
- `show_values()`: Prints all the activity details in a user-friendly format.
- `determine_winner()`: Looks through the participants' data to find one with the maximum score. In the case of a team sport, the score is calculated as the team average and the team with the best average wins. Return the athlete object with the highest score in the winning team
- Only `Athlete` class objects are allowed to participate in this activity.

TalentShow

Using the above `Activity` class, let's make `TalentShow` specialization with the following data attributes and methods. talent_categories should be a private member.

```
talent_categories: list
```

```
evaluate_talent() -> Artist
show_values() -> None
```

- `talent_categories`: List of artistic talent categories expressed as strings.
- `show_values()`: Prints all the activity details in a user-friendly format.
- `evaluate_talent`: Looks through the participants data member to find one with the maximum score.
- Only `Artist` class objects are allowed to participate in this activity.

AcademicCompetition

Using the above `Activity` class, let's make `AcademicCompetition` specialization with the following data attributes and method. subjects and max_marks should be private members.

```
subjects: list
max_marks: float
determine_winner() -> Scholar
```

- `subjects`: Subjects involved in the competition.
- `max_marks`: Maximum achievable marks in the competition.
- determine_winner (): Looks through the participants' data to find one with the maximum score.
- Only `Scholar` class objects are allowed to participate in this activity.

5.2.5 Reading Data [10 Marks]

Using the above data structures, read comma-separated files, `participant_data.csv` and `activity_data.csv`. Checkout Appendix on how to read the file.

Reading Participant Data

Create a function `load_participant_data(filepath:str) -> tuple(list[Student], list[Teacher])` that:

- Reads the CSV file.
- Validates and converts data into appropriate object types.
- Returns a tuple with two members: list of teachers objects and list of student objects.

The following is the header for the input CSV file:

```
idi, name, birth_year, birth_month, birth_day, gender, athletic_score,
    leadership_score, talent_score, gpa, class_assigned,
    selected_activity, grade_level, talent, olympiad_scores,
    subject_specialization, fitness_score, sports_category, subject,
    mentor_grade, mentor_class, judge
```

Reading Activity Data

Create the following function:

```
load_activities_data(filepath:str) -> (list[SportsTournament], list[
    TalentShow], list[AcademicCompetition])
```

that

- Reads the given CSV file
- Validates and converts data into appropriate object types
- Returns a a tuple with three members: list of `SportsTournament` objects, list of `AcademicCompetition` objects, and list of `TalentShow` objects.

The following is the header for the input CSV file:

```
activity_id, activity_name, activity_type, max_participants,
    grade_level, game_type, duration_minutes, talent_categories,
    subjects, max_marks
```

5.2.6 Participant Data Analysis [10 Marks]

Find the students interested in each activity by writing the function that:

```
specialised_students(filepath:str) -> tuple(list[Artist], list[Athlete
    ], list[Scholar])
```

- Reads the participant CSV file.
- Generates `[class_assigned]-artist.csv`, `[class_assigned]-athlete.csv`, and `[class_assigned]-scholar.csv` for each `grade_level` and `class_assigned` where the text `[class_assigned]` should be replaced with the 10A, 9B, etc.

The CSV file will have the following headers:

```
participant_id, name, grade_level, class_assigned, selected_activty,
    score
```

Here, the score is calculated by the respective `compute_scores` function.

5.2.7 Activity Registration [15 Marks]

The school has decided to host events for grades 6 to 12. These events include two SportsTournament: cricket and chess, three AcademicCompetition quizzes in Mathematics, Science, and Computers, and lastly, a multidisciplinary TalentShow called BlackRose. Write a function `register_activity` that takes two inputs, the path for `participant_data.csv` and path for `activity_data.csv`. Using this and the previous functions, it returns a tuple with six lists:

1. `cricket : list[SportsTournament]`
2. `chess : list[SportsTournament]`
3. `mathematics: list[AcademicCompetition]`
4. `science: list[AcademicCompetition]`

5. computers: list[AcademicCompetition]
6. blackrose: list[TalentShow]

```
register_activity(participant_filepath: str, activity_filepath: str) ->
    tuple (list[SportsTournament], list[SportsTournament], list[
   AcademicCompetition], list[AcademicCompetition], list[
   AcademicCompetition], list[TalentShow])
```

The function should do the following.

- Adds the participants to the activity they are associated with.
  - Registers participants for an activity.
  - Updates the participants' attribute of the activity using the `set_values` function.
  - Ensures that only participants meeting certain criteria (e.g., `is_eligible()`) are registered.
- Assigns a judge who has to be a teacher conducting classes in the same grade.
- Updates the `is_active` status of the activity.
  - If the number of registered students is more than allowed, `is_active` should be set to False.
  - There should be at least two participants / teams in an activity.
  - In the case of team sports, the `class_assigned` is considered as a team.
- There should be 7 elements, one for each grade (6 to 12), in each of the lists returned by this function.

5.2.8 Find Winners [15 Marks]

Write a function `winners()` which takes the tuple from the previous question as the input, and creates seven files grade6.csv, grade7.csv, grade8.csv, grade9.csv, grade10.csv, grade11.csv and grade12.csv. In each file we would like to print the winner of each competition, and NA if the competition could not be held. The fuction signature should be

```
winners(activities: tuple) -> tuple:
# where activities can be unpacked into:
# cricket, chess, math, science, computers, blackrose = activities
```

The file would look like this:

```
cricket, chess, mathematics, science, computers, blackrose
[class_assigned], [id], [id], [id], [id], [id],
```

Sample Output

```
cricket, chess, mathematics, science, computers, blackrose
9B, NA, NA, 1006, 1002, 1009
```

5.3 Appendix

5.3.1 How to read a CSV

```python
import csv

# Reading the CSV file
with open("example.csv", "r") as file:
    reader = csv.reader(file)
    data = [row for row in reader]  # Each row is a list of strings

print(data)
```

5.3.2 How to write a CSV

```python
import csv

# Data to write (list of lists)
data = [
    ['Name', 'Age', 'Country'],
    ['Alice', '30', 'USA'],
    ['Bob', '25', 'Canada']
]

# Writing the CSV file
with open("output.csv", "w", newline="") as file:
    writer = csv.writer(file)
    for row in data:
        writer.writerow(row)  # Writes a single row at a time
```