**Python assignment three and four**

This Assignment is designed to take you through creating classes, aggregation, and manipulating arrays of objects.

**Scenario:** A University likes to have a simple system to keep track of all the students (graduate and undergrads). You have to create a menu-driven program for the user to use the system through the console. The following classes are needed for this object-oriented database.

1. Create a class called Student that has the following stored properties:

**Student**

* StudentID : Integer
* stdFirstName: String
* stdLastName: String
* stdMarks : Double []
* stdAddress: Address

\*\* Class Student should have set/get properties for its private parameters, constructor and have following methods:

Average() - that returns the average grade for students

\_\_str\_\_() method that returns the above information as a String

\*\*\* student ID has to be self generated and increment for every student you are adding

1. Create a class called Address which can be **aggregated** into the class student (aggregation means use an object of one class as a properties of other)

Address

* streetInfo: String
* city: String
* postalCode: String
* province: String
* country: String
* \*\* Class Student should have set/get properties for its private parameters, constructor and have following methods:

\_\_str\_\_() method that returns the above information as a String

1. Create a class called UndergraduateStudent that **inherits** from Student and has the following members:

**Undergrad Student**

* subject: String
* yearOfEntry :Integer
* \*\* Class Student should have set/get properties for its private parameters, constructor and have following methods:

Graduate() – Boolean that returns true if the Student is eligible to graduate when the average of their marks is greater than 50.

\_\_str\_\_() method that returns the above information as a String

1. Create a class called GraduateStudent that **inherits** from Student and has the following members:

**Graduate Student**

* subject : String
* yearOfEntry :Integer
* thesisTopic: String
* \*\* Class Student should have set/get properties for its private parameters, constructor and have following methods:

Graduate() – Boolean that returns true if the Student is eligible to graduate when the average of their marks is greater than 70.

\_\_str\_\_() method that returns the above information as a String

**Summary of Operations**

**System Menu:**

1. Add undergraduate student
2. Add graduate student
3. View all the students
4. View only eligible students for graduation
5. exit

Overview:

* You may use a list to store **all your students**(graduate and undergrad) into **one** list of objects.
* Your data has to be stored into a file of your choice using serialization which the user can view after existing the program.

1 –**Add undergraduate Student:** this menu should accept all the necessary parameters for undergraduate students and create an instance from undergraduate class and store it into students array.

2 -**Add graduate student:** this menu should accept all the necessary parameters for graduate students. It should create an instance from the graduate class and store it in the students' array.

3- **View all the students:** view all the relevant information of students (graduate and undergraduate) from students lists

4- **View only eligible students for graduation:** view all the relevant information (graduate and undergraduate) from the students' array only if they are eligible to graduate.

5 – **Exit:** exit the running menu (program)

**Submission Requirements:**

* Submission: on the Moodle
* Required files:
  + You may upload all the python files, including main or/and zip the project and upload.
  + If you use online compilers, you may copy-paste your code into this document.

CAREFUL NOTE:

- Please safeguard your code work.

- If two or more assignments are the same (or very much alike) they will all get 0 marks, so be cautious not to share your application with others.

**Marking Scheme:**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Trait** | **Excellent (85-100)** | **Good (70-85)** | **Satisfactory(50-70)** | **Unsatisfactory (< 50)** |
| **Delivery**  **(5 marks)** | * Submitted on time and in the correct format. * Completed 90 - 100 percent of the program requirements | * Submitted on time and in the correct format. * Completed 75 -90 percent of the program requirements | * Submitted on time and in the correct format. * Completed 70 -80 percent of the program requirements | * Submitted late or in the wrong format. * Completed less than 70% of the program requirements |
| **Coding Standards and Documentation (10 marks)** | * Includes name, date and assignment number. * Excellent variable names used (no global variables, or vague naming). * Useful documentation descriptions. * All functions commented. * Indented to standard. | * Includes name, date and assignment number. * Appropriate variable names used (little use of global variables, or vague naming). * Useful documentation descriptions. * Most functions commented. Indented to standard | * Includes name, date and assignment number. * Appropriate variable names used (a few use of global variables, or vague naming). * Basic documentation descriptions including purpose for functions. * Mostly indented well | * No name, program description included * Poor or misleading variable names used. Little or no indentation. * Regular use of global variables |
| **Specification and Runtime (75 marks)** | * The program meets all of the specifications required and works. * No errors in output. Output is formatted excellently. * All requirements met | * No errors in output. Output is formatted. * All requirements met. * It also meets most of the other specifications. | * No errors in output. * Output has basic formatting or meets core specifications only. | * Does not run due to errors, data read incorrectly. * Little or no requirement met. * Output is poorly formatted or does not follow specifications. |
| **Efficiency**  **(10 marks)** | * Algorithm is easy to understand and efficient. * Can be maintained or modified with minimal changes | * Algorithm is easy to understand and efficient | * Algorithm is easy to understand and but inefficient (excessive use of variables, loops or conditionals) | * Algorithm is hard to understand and very inefficient (excessive use of variables, loops or conditionals) |