CIS 345 – Business Information Systems Development - II

PE 8: Classes and Objects

Learning Outcomes

1. Implement a class with instance variables
2. Implement properties using decorators (setter)
3. Create object instances from classes using dunder init method
4. Use instance methods and properties

Program Overview

In this program, you will create a "model" of a college class called Course. A Course maintains the name of the course, the number of the course, the college name, etc. The "model" will be in the form of a Class. A Class is a model or a blueprint for the kinds of information that needs to be maintained for a real-world object. ***Classes become available as a new type!***

Blueprints or models are just designs. To use them in reality, you need to manufacture items based on those designs. In the programming environment, the equivalent is instantiating objects based on classes. You will create a Course class in which you will instantiate objects, based on the Course class, and utilize those objects.

This brief exercise demonstrates the reusability principle of Object Oriented Design – once you create the Class Course, you are able to seamlessly create as many objects of type Course without any additional work.

Sample Output

```
    Entering course information...

Enter course number: CIS340

Add another course (Y/N)? y
Enter 3 letter department for next course: CIS
Enter course number: CIS345

Add another course (Y/N)? y
Enter 3 letter department for next course: ACC
Enter course number: ACC444

Add another course (Y/N)? n

Printing Courses
Course: CIS340:
Exam avg is 54.3 - [12, 58, 93]
Quiz avg is 5.7 - [3, 10, 4]
Course: CIS345:
Exam avg is 54.3 - [12, 58, 93]
Quiz avg is 5.7 - [3, 10, 4]
Course: ACC444:
Exam avg is 54.3 - [12, 58, 93]
Quiz avg is 5.7 - [3, 10, 4]
```

Instructions

- Create a PyCharm Project using standard naming convention and add a Python file:
  - [ASUrite]_course.py
- When done, submit your completed python module file only.
- Don't forget to put your name, class, class time, and assignment number in **as a comment** on Line 1 of all modules.

PEP8 Conventions

- Class names are PascalCase otherwise known as CapWords convention
- Method and instance variables use snake_case
- Non_public (aka private) methods and instance variables use one leading underscore

In the **[ASUrite]_course.py**, you will develop code for the following UML **Class Diagram**, which models a Course. For this exercise, the instructions that follow are a written version of the UML diagram. You should be able to look at the diagram and develop the entire class in code without further instructions. *In the future assignments, you will be given only the UML Class diagram!*

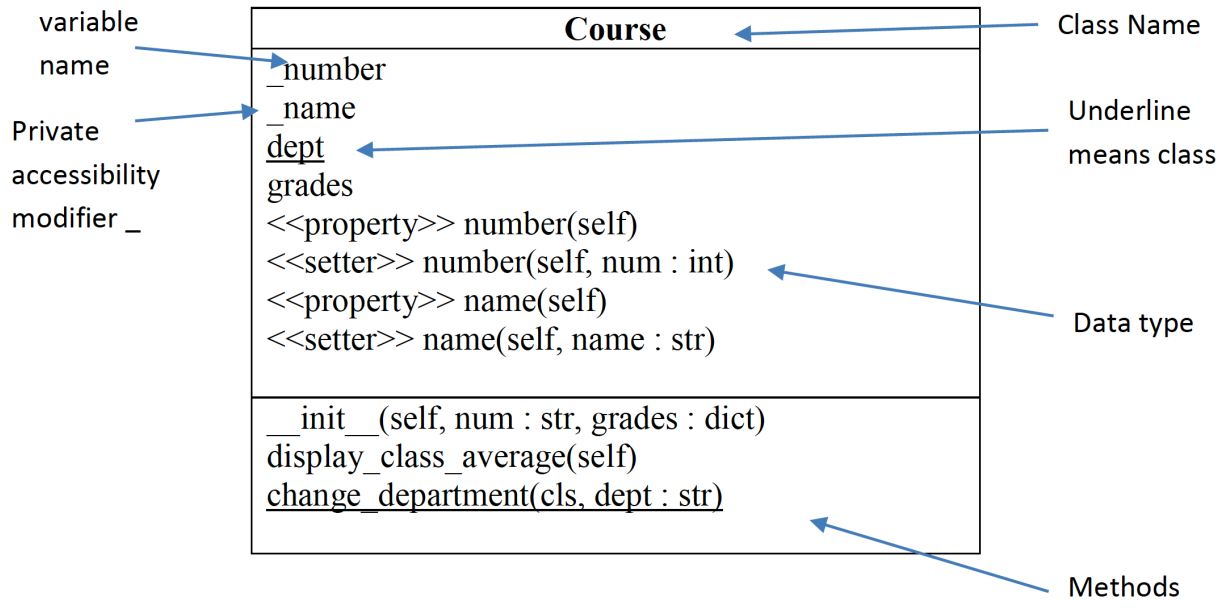

Figure 1. Course Class Diagram

1. Import two modules for use, random and statistics.

2. Declare a blank list called courses to store all course objects we create

3. Define class course with the items listed in UML above. The below are notes to assist in translating the UML into code:

    a. Leading underscores means a field is private and not to be accessed outside the class code. Create these items as instance variables inside the class's dunder init function.
    b. Underlined fields are either static or class methods or variables. Department is a class variable and is not defined in the init function but is declared under the class. Initialize dept to 'CIS'
    c. Any item that is not underlined or has a leading underscore is a normal instance variable which is grades in our UML.
    d. << >> these symbols are called **guillemets** and in UML are a stereotype telling us something special such as a field is a property or setter.

Define the `__init (self, num='', grades={})` **function.** Assign each of the incoming arguments to its corresponding instance variable listed in UML, but name will be dept + num. Example: self._name = self.dept + num

Create the following Properties to manage the "state" of the object. We create properties to get and set private **fields**. To create the property use the @property and @<propertyName>.setter decorators. Create a property for both number and name. Since we use decorators we will be defining methods (property acts like a getter and setter is a setter). Example:

1. **Create a property getter, for name and number using the below example**:

```
@property
def name(self):
    return self._name
```

2. **Create a property setter for name and number using this example and pseudo-code**:

```
@number.setter
def number(self, num):
    self._number = num
    self._name = _______
```

- Number setter: arguments are self and num.
    - Assign num to the instance variable _number

    - Update the name by appending the new number to the existing 3 letters in _name (how can string slicing help here?)
- Name setter: arguments are self and name (name will be values like CIS345).
    - Make name all caps and assign to the instance variable _name
    - Update the _number by extracting the new number from name the last 3 characters in name (how can string slicing using negatives help here?)

3. **Define a Class Method,** `change_departments(cls, dept):`

```
@classmethod
def change_department(cls, dept):
    """Class method to modify class variable dept"""
    cls.dept = dept
```

This method is how we will change the prefix of a course object such as CIS or ACC.

4. **Define an Instance Method,** `display_class_average(self):`
    - Instance functions are like normal functions, but within a class (no decorator on this one).
    - grades will hold a dictionary where the key is the grade item and the value is a list of scores. Example: {'Exam' : [80, 92, 73, 88, 77] }
    - The function body will loop through all **key**s and **value**s
        - Compute avg of scores using statistics.mean(**value**)
        - `print(f'{key} avg is {avg:.1f} - {value}')`

End of class course

Create a Boolean variable to control a while loop and initialize to True.

5. Main Program Logic,

    5.1. **Display** 'Entering course information…'.

    5.2. **Start** while loop.

    5.3. **Prompt** user for course number showing them the dept which should first time through be defaulting to CIS. Store input.

    5.4. **Construct** a course object passing stored course number, Course(course_number), and store returned object.

5.5. Assign random grades to a couple categories within your new object. Example:

```
new_course.grades['Exam'] = [random.randint(1, 100),
                             random.randint(1, 100),
                             random.randint(1, 100)]
```

5.6. **Append** the new course to your list of courses (the list you declared at the top of your module in a previous step)..

5.7. **Prompt** user 'Add another course (Y/N)? ' and deal with either upper of lowercase entries.

- If yes, then
  - prompt for a new 3 letter department and store
  - Call class method: Course.change_department() and pass user entered dept
  - Continue looping in while
- If no, then
  - Exit loop

After while loop

1. Display 'Printing Courses'
2. For each course in courses
   2.1. Display the course name
   2.2. Call each course object's display_class_average() method

Learning Points

- Write some additional test code and try to assign and or print the name and number properties within a course object.
- Now, look closely at your output and you should see a problem (this will match my sample output at the top of this document). Why do all courses have the same scores and averages? We will learn why in our next lecture.
- No need to fix the issue. Your program should produce the same output listed in the Sample Output screenshots using those entries.

Submission:

1. Submit your python module file only. No zip file for this PE.