Important

1. Due Date: **Before 11:55pm on Monday April 18th, 2016**.
2. This homework is graded out of 100 points.
3. This is an Individual Assignment. You may collaborate with other students in this class. Collaboration means talking through problems, assisting with debugging, explaining a concept, etc. Students may only collaborate with fellow students currently taking CS 1301, the TA's and the lecturer. You should not exchange code or write code for others. For individual assignments, each student must turn in a unique program. Your submission must not be substantially similar to another student's submission. Collaboration at a reasonable level will not result in substantially similar code.
4. For Help:
    - TA Helpdesk (Schedule posted on class website.)
    - Email TA's or use Piazza Forums Notes:
    - How to Think Like a Computer Scientists [http://openbookproject.net/thinkcs/python/english3e/]
    - CS 1301 Python Debugging Guide [http://www.cc.gatech.edu/classes/AY2016/cs1301_spring/CS-1301-Debugging-Guide/index.html]
5. Don't forget to include the required collaboration statement (outlined on the syllabus).
6. Do not wait until the last minute to do this assignment in case you run into problems.
7. **Read the entire specifications document before starting this assignment.**

Object-oriented programming

Python is an **object-oriented programming language**. That means it provides features that support object-oriented programming (**OOP**).

Object-oriented programming has its roots in the 1960s, but it wasn't until the mid 1980s that it became the main programming paradigm used in the creation of new software.

In object-oriented programming the focus is on the creation of **objects**, which contain both data and functionality together. Usually, each object definition corresponds to some object or concept in the real world and the functions that operate on that object correspond to the ways real-world objects interact. We've already seen classes like `Window`, `Picture`, `Myro`, and many others. We are now ready to create our own user-defined class.

[http://openbookproject.net/thinkcs/python/english3e/classes_and_objects_I.html]

Introduction

The goal of this homework is for you to get practice and understand the basics of Object Oriented Programming. Your mission, should you choose to accept it, involves the creation of

three classes: Student, Course, and Instructor. Each class will contain several methods for you to implement. What you should do for each method is described below and in the docstrings of the file. Below you will find more information to complete your assignment. Read it thoroughly before you begin. The key to this homework is to see the bigger picture of how the 3 classes are interconnected i.e., a Course contains an Instructor, a Student can have many Courses, and an Instructor can change a Student's course grade. You have until – to complete this mission. This message will self-destruct in five seconds.

Student

You are to code a `Student` object. Your `Student` should have the following properties:

```
 |  Attributes:
 |      name: A string representing the student's name.
 |      gtid: A string representing  the student's GT id.
 |      year: An integer representing the student's class (default: 1)
 |          1 -> Freshman
 |          2 -> Sophomore
 |          3 -> Junior
 |          4 -> Senior
 |      courses: A dictionary representing the courses a student is enrolled.
 |          Key -> A string representing a Course code
 |          Value -> [A Course object, grade] (a list 0th index a Course
Object, 1st index an int representing the grade of the course (0-4) )
 |
 |  Methods defined here:
 |  __init__(self, name, gtid, year=1)
 |      Initialize a Student object whose name is *name*, gtid is *gtid*, gpa
is *gpa*, class standing is *year* (default: 1); courses start empty
 |
 |  calculate_gpa(self)
 |      Returns a float of the student's gpa. To calculate GPA:
 |      1. If student has no courses his gpa must be 0.0
 |      2. Multiply the grade (0-4) by the number of credit to get quality
poins.
 |      3. Total the credit hours. Total the total quality points.
 |      4. Divide the total quality points by the total credit hours
 |
 |  drop(self, course)
 |      Drop a course if student is registered to it. Return True if
successfully dropped and False if student is not registered for that course.
 |
 |  get_real_year(self)
 |      return a string representation of the student's class standing
 |      1 -> Freshman, 2 -> Sophomore, 3 -> Junior, 4 -> Senior
 |
 |  get_total_credits(self)
 |      return the amount of credits a student is taking as an int
 |
 |  in_dean_list(self)
 |      return true if gpa greater than or equal 3, false otherwise
 |
 |  is_taking(self, course)
 |      Returns true if student is registered for a course
```

```
 |
 |  register(self, course)
 |      Add a course to the student's courses. No duplicate courses are
allowed. If student is registered for X course with X professor, it should not
be allowed to registered for X course with Y professor. The initial grade for
the course should be 0.
 |
 |  register_many(self, courses)
 |      register courses from a list of Course objects
 |
 |  set_year(self, year)
 |      change student's year. year is now *year*
```

Course

You are to code a `Course` object. Your `Course` should have the following properties:

```
 |  Course have the following properties:
 |  Attributes:
 |      code: A string representing the course code (i.e CS1301)
 |      credits: An integer representing the course's amount of credits (i.e 3)
 |      instructor: An Instructor object representing the instructor of the
course
 |
 |  Methods defined here:
 |
 |  __init__(self, code, credits, instructor)
 |      Initialize a Course object whose code is *code*, credits is *cresits*,
and instructor is *instructor*.
 |  get_code(self)
 |      returns string representing the course code
 |
 |  get_credits(self)
 |      returns int representing the course credits
```

Instructor

You are to code an `Instructor` object. Your `Instructor` should have the following properties:

```
 |  Instructor have the following properties:
 |      Attributes:
 |          name: A string representing the instructor name (i.e CS1301)
 |
 |  Methods defined here:
 |
 |  __init__(self, name)
 |      Initialize an Instructor object whose name is *name*.
 |  assign_grade(self, student, course, grade)
 |      If student is taking a course, change the student's grade in a course
by *grade*. grade is a number between 0 and 4 inclusively. The max a student
can get in a course is 4.
```

college_test.py

We have provided you with a python file called `college_test.py`. In this file we have created a series of tests for your usage. We understand you probably have never been exposed to testing code so you are not expected to do anything to this file or even use this file if you don't want to. However, we encourage you to use it as it will be highly beneficial in testing our your code. Feel free to add your own tests to the file to cover any additional cases you would like to test.

If you do desire to test your code, all you have to do is have the `college.py` and the `college_test.py` files in the same directory. Open and run `college_test.py`. After running the tests you should see their results. Check the results and start debugging if needed. If you pass all the tests you should see something like this in your output window:

```
test_add_a_course (__main__.CollegeTest) ... ok
test_add_a_duplicate_course (__main__.CollegeTest) ... ok
test_add_many_courses (__main__.CollegeTest) ... ok
test_add_many_courses_with_duplicates (__main__.CollegeTest) ... ok
test_calculate_gpa_1 (__main__.CollegeTest) ... ok
test_calculate_gpa_2 (__main__.CollegeTest) ... ok
test_calculate_gpa_3 (__main__.CollegeTest) ... ok
test_default_year (__main__.CollegeTest) ... ok
test_drop_course (__main__.CollegeTest) ... ok
test_drop_course_2 (__main__.CollegeTest) ... ok
test_empty_courses (__main__.CollegeTest) ... ok
test_get_course_code (__main__.CollegeTest) ... ok
test_get_course_credits (__main__.CollegeTest) ... ok
test_get_real_year (__main__.CollegeTest) ... ok
test_get_total_credits (__main__.CollegeTest) ... ok
test_get_total_credits_2 (__main__.CollegeTest) ... ok
test_get_total_credits_3 (__main__.CollegeTest) ... ok
test_in_dean_list (__main__.CollegeTest) ... ok
test_in_dean_list_2 (__main__.CollegeTest) ... ok
test_init_course_code (__main__.CollegeTest) ... ok
test_init_course_credits (__main__.CollegeTest) ... ok
test_init_course_instructor (__main__.CollegeTest) ... ok
test_init_id (__main__.CollegeTest) ... ok
test_init_instructor (__main__.CollegeTest) ... ok
test_init_name (__main__.CollegeTest) ... ok
test_set_year (__main__.CollegeTest) ... ok

----------------------------------------------------------------------
Ran 26 tests in 0.363s

OK
```

Tips & Hints

1. Get familiar with OOP before you start coding. Revise your class notes and read section 15 of the book again.
2. Even if you do not want to use the test file, open it and read it, as this might help you understand what we are looking for and it might give you great hints on how to implement your classes and methods. Huge spoilers in this file.
3. Use the methods you are writing inside other methods. **Hints**:
   a. In Instructor's `assign_grade()` you have a `student` object that has an `is_taking()` method. So an instructor can check that the student is taking the course using this method and then change the grade.
   b. In student's `calculate_gpa()` you have access to the `get_total_credits()` method, use it.
4. In `college_test.py` the module `setUp()` is always called before any test. Each test is supposed to be independent to all others. Read more about unittest here: [https://docs.python.org/3/library/unittest.html]

Provided

The following file(s) have been provided to you. There are several, but you will only edit one of them:

1. `college.py`
   This is the file you will edit and implement. All instructions for what the methods should do are in the docstrings.

2. `college_test.py`
   This is a test file that contains a set of tests if you wish to test your code. This part is completely optional and it's provided just for your benefit**. It is not intended to be exhaustive and does not guarantee any type of grade**. It is specifically missing some test cases that may be important! **Write your own tests if you wish to ensure you cover all edge cases.**

Deliverables

You must submit all of the following file(s). Please make sure the filename matches the filename(s) below. Be sure you receive the confirmation email from T-Square, and then download your uploaded files to a new folder and run them.

1. `college.py`

Grading Rubric

Student

`__init__`	5pts
`calculate_gpa`	10pts
`set_year`	5pts
`in_dean_list`	5pts
`get_real_year`	5pts
`get_total_credits`	5pts
`register`	10pts
`register_many`	5pts
`drop`	10pts
`is_taking`	5pts
TOTAL	**65pts**

Course

`__init__`	5pts
`get_code`	5pts
`get_credits`	5pts
TOTAL	**15pts**

Instructor

`__init__`	15pts
`assign_grade`	5pts
---	---
TOTAL	**20pts**

TOTAL
100pts