CS 1110 Spring 2021, Assignment 6: Pre-enroll-apalooza*

1 Overview

Like all the assignments this semester, this assignment is motivated by a real-life[1] task: making a course schedule from courses listed on a roster, where the time of a course might change, and where you aren't allowed to add courses that conflict.

Managing the structure of the Python classes and subclasses in this assignment should not be too challenging.

You will need to keeping straight the difference between dictionary keys and dictionary values.

We anticipate that the biggest hurdle will be determining whether two courses overlap. *We leave it to you to design a representation for times and a way to use your chosen representation to check for courses having time overlaps.* We highly recommend that you first write down a high-level plan. Then, implement your plan using several helper function and helper classes that you create and test bit by bit.

Download the zip file of the files you will need: `http://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment6/a6_skeleton.zip`.

Contents

2 Previous Rules That Still Apply. Remember To Acknowledge All Sources Of Help, Allowed Or Not.

Except you do not need to acknowledge the course staff (although it's not bad if you do).

*Authors: Lillian Lee

[1] Or at least "real for Cornell".

See Sections 1.1-1.3 of Assignment 1[2]; Sections 3.1-3.2 of Assignment 2[3]; Section 2 item 3 of Assignment 3[4]; Section 2 item 2 of hreffnhttps://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment5/a5.pdfAssignment 5

3 New rules

1. You may use `break` and `continue` commands. You may use dictionary methods `keys()` and `values()`.
2. You may introduce new variables, helper functions, or even classes as long as you document them well enough that a reader can easily understand how they should be used.[5]
3. As usual, you cannot modify the headers or specifications we have supplied. Also, leave the pre-condition-enforcing `assert` statements for each method alone, and write your method bodies underneath them.[6]

4 Timeline and Deadlines

(a) If you are partnering:[7] do so well *before* submitting.

(b) By 2pm Ithaca time on Thu May 13, submit whatever you have done at that point on to CMS.[8]

(c) By **11:59pm (Ithaca time) on Thu May 13**, make your final submission.[9]

5 Overview of the given Python classes

5.1 The Python class Schedule

We've written almost all of this class, which is used to represent both individual schedules and course rosters, for you in file schedule.py. Because of the latter application, one is allowed to add conflicting courses to an arbitrary Schedule via the `add()` method.

```
"""
An instance represents a set of Courses.

An instance can represent a course roster.
So, Courses in this Schedule can overlap in time.

Instance attributes:

    * name [non-empty str]: the name of this Schedule, e.g.,
        "Cornell Roster Fall 2021" or (for a student schedule) "My Plan A"

    * courses [dictionary with string keys]: maps a course component name,
        like "CS 1110 LEC 001", to a Course object

    * warnings [list of str]: any warnings about this Schedule, such as a
        Course time change
```

[2] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment1/a1.pdf
[3] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment2/a2.pdf
[4] https://www.cs.cornell.edu/courses/cs1110/2021sp/assignments/assignment3/a3.pdf
[5] For variables, a good name may be sufficient. Helper functions and classes should have proper docstrings, including preconditions.
[6] These assertions are to help you debug, but they don't serve that function if you put your function bodies above them.
[7] Reminder: Both parties need to act on CMS in order for the grouping to take effect. See the "How to form a group" instructions at https://www.cs.cornell.edu/courses/cs1110/2021sp/resources/cms.html .
[8] It is OK if you haven't finished the assignment yet; the 2pm checkpoint provides you a chance to alert us if any problems arise, and us to alert you if your submission seems to be missing and of the deadline that day. Since you've been warned to submit early, do not expect that we will accept work that doesn't make it onto CMS on time, for whatever reason. There are no so-called "slipdays" and there is no "you get to submit late at the price of a late penalty" policy. Of course, if some special circumstances arise, contact the instructor(s) immediately.
[9] And, as usual, perform steps 1-3 in the "Updating, verifying, and documenting assignment submission" section of https://www.cs.cornell.edu/courses/cs1110/2021sp/resources/cms.html .

```
"""
```

The `warnings` attribute simulates messages that might be passed to interested parties when, for example, a Course is rescheduled.

You may find the following observations useful when you work with a Schedule's dictionary attribute `courses`. If `d` is a dictionary whose keys are strings and whose values are Course objects, then:

- You can write for-loops over `d`, and the loop variable will be assigned each key of `d` in turn.
- `d.values()` returns a list of the values stored in `d`. See our implementation of `Schedule.__str__()` for an example. Don't forget the parentheses!

5.2 The Python subclass StudentSchedule

A StudentSchedule is a special kind of Schedule that doesn't allow a Course to be added to it if that addition would cause a time conflict — instead, a warning is generated. This Python class thus overrides the `add()` method of Schedule. But it can inherit all the other Schedule methods as is, so there are no other methods defined within this subclass!

5.3 The Python class Course

As stated in file schedule.py, Course objects must have attributes `name` and `in_schedules`, but *you will add more attributes and methods, and even Python classes as you see fit* to identify and handle time conflicts.

Attribute `in_schedules` tracks which Schedules a given Course is part of, so that if the Course gets rescheduled, all those Schedules can be updated, or at least warned.

```
    Instance attributes:
        name [non-empty string]: name of the course
        in_schedules: list of Schedules this Course is in

        # STUDENTS: below this comment block (but within the docstring for class
        # Course), document any attributes you add.
```

6 Goal

Upon completion of this assignment, you should be able to run file `a6_in_action.py`, which demonstrate how your code might be used: it creates a class roster, and then a personal schedule, and then tries adding courses to that personal schedule. The following run is also available to you in file `output_of_a6_in_action.txt`.

```
Here is our running example of a course roster
Schedule Cornell Fall 21 roster
CS 1110 LEC 001: TR 9:05am-9:55am
INFO 2450 LEC 001: TR 9:40am-10:55am
BIOG 1140 DIS 201: R 9:05am-9:55am
ENGL 2650 SEM 101: TR 11:25am-12:40pm
HADM 1150 LEC 001: TR 11:25am-12:40pm
ARCH 1120 STU 520: TBA
AEM 2010 LEC 001: TR 8:05am-9:20am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKE 0000 LEC 001: T 9:30am-9:45am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
Warnings:


Creating my own schedule, adding 3 non-conflicting courses to it.
Here is my_schedule so far:
Schedule My Plan A
CS 1110 LEC 001: TR 9:05am-9:55am
```

```
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
ARCH 1120 STU 520: TBA
Warnings:

Adding INFO2450 should fail due to starting during CS1110.
*******
CHECK that it isn't in my_schedule, but a warning about it is:
*******

Schedule My Plan A
CS 1110 LEC 001: TR 9:05am-9:55am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
ARCH 1120 STU 520: TBA
Warnings:
Could not add INFO 2450 LEC 001 due to conflict with CS 1110 LEC 001

Above, we've requested that you perform a check (see the stars).
Enter q to quit checks, anything else to keep going:

Adding BIOG1140 should fail due to complete conflict with CS1110 on R.
*******
CHECK that it isn't in my_schedule, but a warning about it is:
*******

Schedule My Plan A
CS 1110 LEC 001: TR 9:05am-9:55am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
ARCH 1120 STU 520: TBA
Warnings:
Could not add INFO 2450 LEC 001 due to conflict with CS 1110 LEC 001
Could not add BIOG 1140 DIS 201 due to conflict with CS 1110 LEC 001

Above, we've requested that you perform a check (see the stars).
Enter q to quit checks, anything else to keep going:

Adding AEM2010 should fail due to running into CS1110.
*******
CHECK that it isn't in my_schedule, but a warning about it is:
*******

Schedule My Plan A
CS 1110 LEC 001: TR 9:05am-9:55am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
ARCH 1120 STU 520: TBA
Warnings:
Could not add INFO 2450 LEC 001 due to conflict with CS 1110 LEC 001
Could not add BIOG 1140 DIS 201 due to conflict with CS 1110 LEC 001
Could not add AEM 2010 LEC 001 due to conflict with CS 1110 LEC 001

Above, we've requested that you perform a check (see the stars).
Enter q to quit checks, anything else to keep going:

Change time of ARCH1120 so it now conflicts with AEP3560 and FAKER0000
*******
CHECK that the roster has the new time and a new warning
*******

Schedule Cornell Fall 21 roster
```

```
CS 1110 LEC 001: TR 9:05am-9:55am
INFO 2450 LEC 001: TR 9:40am-10:55am
BIOG 1140 DIS 201: R 9:05am-9:55am
ENGL 2650 SEM 101: TR 11:25am-12:40pm
HADM 1150 LEC 001: TR 11:25am-12:40pm
ARCH 1120 STU 520: MWF 8:00am-8:00pm
AEM 2010 LEC 001: TR 8:05am-9:20am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKE 0000 LEC 001: T 9:30am-9:45am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
Warnings:
ARCH 1120 STU 520: time changed from TBA to MWF 8:00am-8:00pm

Above, we've requested that you perform a check (see the stars).
Enter q to quit checks, anything else to keep going:
*******
CHECK that my_schedule still has the course, but now warns of 2 conflicts
*******

WWWWWWWWW
Schedule My Plan A
CS 1110 LEC 001: TR 9:05am-9:55am
AEP 3560 LEC 001: MWF 9:05am-9:55am
FAKER 0000 LEC 002: F 1:30pm-2:30pm
ARCH 1120 STU 520: MWF 8:00am-8:00pm
Warnings:
Could not add INFO 2450 LEC 001 due to conflict with CS 1110 LEC 001
Could not add BIOG 1140 DIS 201 due to conflict with CS 1110 LEC 001
Could not add AEM 2010 LEC 001 due to conflict with CS 1110 LEC 001
ARCH 1120 STU 520: time changed from TBA to MWF 8:00am-8:00pm
ARCH 1120 STU 520: time change causes conflict with AEP 3560 LEC 001
ARCH 1120 STU 520: time change causes conflict with FAKER 0000 LEC 002

Above, we've requested that you perform a check (see the stars).
Enter q to quit checks, anything else to keep going:

That's the end of A6 in action!
(But this script can run to completion even if your code isn't correct,
so also run the schedule_test.py script.
```

7 Testing

You can run a6_in_action.py; the "intent" of that file is more obvious than our full testing code. A good/bad thing about a6_in_action.py is that it will run without crashing even before you have handled time conflicts — it just needs a complete Schedule method `add()` to work, and will tell you *checks-by-eyeball* that you should do. **But this file not crashing doesn't mean you're done.**

We've also provided test script schedule_test.py. Lines 285ff. define a list of test functions to run. Comment out those you don't want to run at a given point in your code development. This script makes multiple calls to function `make_sample_roster()` from file roster_creation.py to set up a 10-course pre-built roster Schedule that you see in the printouts in Section 6

Finally, some examples of quick-and-dirty testing code is commented out at the bottom of schedule.py.

8 Tasks

There are two files to submit: schedule.py and explanation_of_time_handling.py

The `STUDENTS` comments in file schedule.py include indications of what you must implement. We highly recommend you proceed in the order suggested by the subheadings below and by the order of the tests in file `schedule_test.py`.

8.1 Complete Schedule's `add()` method.

When this method is called with a Course `c` as argument, it needs to be able to access `c`'s `name` and `in_schedules` attributes, and you haven't written the initializer for Course objects yet.[10] But, we already wrote enough of the `__init__` method for Python class Course that you can assume these are already available to you.[11]

8.2 Write an initial version of StudentSchedule's `add()` method.

Make sure to make effective use of `super()` to (eventually) call the superclass Schedule's `add()` method. In this initial version, don't try to handle time conflicts; testing function `schedule_tests.test_studentschedule_add_noconflicts()` can test it in its initial form.

8.3 Before doing any more coding, write your plan for handling times and time conflicts in file `explanation_of_time_handling.py`.

Decide how the class Course should represent a meeting pattern like "MWF 9:05am-9:55am" or "R 4:00pm-7:30pm" or the other type of time input we'll allow, "TBA". (We refer to these patterns as *roster format*.) Considerations:

1. Course method `getTime()` requires you to be able to convert your representation of the meeting pattern back into a string in roster format.
2. Your representation should make it fairly easy to implement Course method `conflictsWith()`

```
def conflictsWith(self, other):
    """Returns True if this Course conflicts with `other`, False otherwise.
    It is not a conflict if one course ends when the other begins.
    Precondition: `other` is a Course object.
    """
```

We suggest this step having a written plan to consult while coding will make the process much, much smoother.

8.4 Complete the remaining StudentSchedule and Course methods.

You need to add attributes to Course to represent a Course's meeting times. Document these attributes clearly in the docstring for class Course.

We recommend you write your own helper functions, to break up the task into manageable, easily-debuggable pieces and keep your code organized. For instance, some separate jobs might be: converting a meeting-pattern string into your time representation, determining whether two times overlap, and converting time overlap into course overlap.

You may also want to create new classes to represent times or other useful pieces of information. While lists can suffice for just about anything, it's easier to understand an expression like `<object>.meaningful_attribute_name` than `list[2][3]`.

Make sure to document any additional functions and classes you create.

Don't forget to have your Course methods add warning messages to the relevant schedules.

8.5 Restore any commented-out tests in list `tests_to_run` in the script code of file `schedule_tests.py`

And then rerun that testing file!

A Optional further challenges (a.k.a., things we sacrificed to reduce this assignment's workload)

1. We didn't handle deleting a Course from the main Cornell roster. Note that doing so should also have the side effect of deleting the course from all student schedules containing that course. On the other hand, deleting a

[10] Unless you didn't follow these directions. But if you didn't, how are you reading this footnote?
[11] You're welcome.

Course from a StudentSchedule should not delete it from the main Cornell roster. How might this asymmetry be handled. [12]

2. You might find it interesting or satisfying later on to integrate the A1 code, which pulls and parses info from the live roster pages, into your A6 project, thus "closing the loop" on this semester.

[12] The "right" thing to do would probably be to make course rosters be a subclass of Schedule.