Solution: CS 1110 Prelim 2 — April 22, 2014

1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of *each* page.

   Solution: Every time a student doesn't do this, somewhere, a kitten weeps.

   More seriously, we sometimes have exams come apart during grading, so it is actually important to write your name on each page.

2. [10 points] **Recursion.** Recall the Node class from A3. Each node has a contacted_by attribute consisting of a (possibly empty) list of nodes that have contacted it, and we know that anything in a node's contacted_by list is from an earlier generation. This question asks you to add a new method for class Node; implement it according to its specification. Your solution must be recursive, though it can involve for-loops as well.

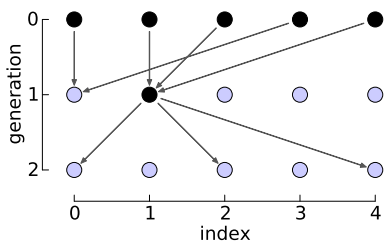
```
class Node(object):

    ...
    def is_downstream_from(self, older):
        """Returns True if: older is in this node's contacted_by list, OR if
           at least one of the nodes in this node's contacted_by list is
           downstream from older.  Returns False otherwise.
           Pre: older is a node.
        """
        # Do NOT compute the legacy of older (it doesn't even help to do so if
        # self is not converted).  You do NOT need to do any caching or check
        # if nodes are converted or not.
```

Solution:

```
    if self.contacted_by == []:
        return False
    elif older in self.contacted_by:
        return True
    else:
        for contacter in self.contacted_by:
            if contacter.is_downstream_from(older):
                return True
    # If we get to this line, no contacter was downstream of older
    return False
## +1 for first base case concept
## +1 for second base case concept
## +1 for using self correctly throughout
## +1 looping through each contacter
## +2 recursive call idea [this should be all or nothing, probably]
```

```
## +2 correct syntax (method means contacter-dot, older should be the argument again)
## +1 return True in correct recursive case
## +1 return False when all checks fail
```

Example: In the figure below, (2,0) is downstream from (0,1), (0,2), (0,4), and (1,1), but no other nodes.



3. [6 points] **While-loops.** Write a function that does the same thing as `product_for` but uses a while-loop.

```
def product_for(x):
    """Return: the product of the numbers in x.
       Pre: x is a list of integers.
    """
    p = 1
    for n in x:
        p *= n
    return p

def product_while(x):
    """Same specification as above."""
```

Solution:

```
    i = 0
    p = 1
    while i < len(x):
        p *= x[i]
        i += 1
    return p
## Counting backwards would also be fine.
## +1 for each initialization, the condition, the body, the increment,
##    and the return (that is, 1 for each line)
```

4. [8 points] **While-loops.** Implement the `strip` function so that it meets its specification, using two *non-nested* while-loops: *one starting from the beginning of the string and moving right*, and then *one starting from the end of the string and moving left.*

   Your implementation may *not* use the Python built-ins `strip`, `lstrip`, or `rstrip`.

```
def strip(s1, s2=' '):
    """Return a new string that is s1 but with the occurrences of characters in s2
       removed from the ends.
       Pre: s1 contains at least one character not in s2.
       Examples: strip(' te st ') == 'te st'
                 strip('batestb', 'ab') == 'test'
                 strip('test    ') == 'test'
                 strip('banana', 'nab') violates the precondition.
    """
    # Hint: the precondition means your loops can't "fall off" the other end.
```

Solution:

```
    h = 0
    # inv: s[0..h-1] is in s2
    while s1[h] in s2:
        h += 1
    # We now know that s1[h] is not in s2.

    k = len(s1)
    # inv: s[k..len(s)-1] is in s2
    while s1[k-1] in s2:
        k -= 1
    # We now know that s1[k-1] is not in s2.

    return s1[h:k] # returns s[h..k-1]
## Also possible to have j mean "next thing to check" , in which case
## j should start at len(s1)-1, loop condition is while s1[j] in s2,
## return s1[h:j+1]

## For each while loop:
## +1 for correct init, +1 for correct loop condition, +1 for increment
## The "return" line is worth two points, one for each increment.
##


# ALTERNATE SOLUTION WITHOUT EXPLICIT INDEXING
    while s1[0] in s2:
        s1 = s1[1:]
    while s1[len(s1)-1] in s2:
        s1 = s1[:len(s1)-1]
    return s1
```

see last pages of this file

5. [12 points] **Classes and objects.** The three classes `Course`, `Student`, and `Schedule` that are printed on a separate handout are part of the Registrar's new course enrollment database, which keeps track of which courses each student is enrolled in, and also which students are enrolled in each course. Two methods are not implemented: `Student.add_course` (line 96), which updates the database to reflect a student enrolling in a course, and `Student.validate` (line 106), which checks a student's schedule to make sure it follows the rules.

   Read the code to become familiar with the design and operation of these classes. Note that helper methods and a unit test included, which may help in understanding how these classes are used and in solving the problems below.

   After reading the code, implement the two incomplete methods by filling in your code below. Write your answers on this sheet, not on the code printout (where there is no space to fit your answer).

```
class Student(object):
    ...

    def add_course(self, course):
        """See the code for the specification."""
```

Solution:

```
        course.students.append(self)
        self.schedules[0].courses.append(course)
## 2 for first line:
##   +1 for accessing students list
##   +1 for appending correctly.
## 3 for second line:
##   +1 for accessing schedules list
##   +1 for then accessing courses list
##   +1 for appending correctly.
```

```
    ...

    def validate(self, credit_limit):
        """See the code for the specification."""
```

Solution:

```
        valid = True
        for sched in self.schedules[1:]:
            if sched.overlaps(self.schedules[0]):
                valid = False
        return valid and (self.schedules[0].total_credits() <= credit_limit)
## Various other ways to structure the logic are possible.
## 5 for computing overlap boolean
##    +1 for loop over the right part of schedules
##    +1 for calling Schedule.overlaps correctly
##    +1 for calling it with the right argument
##    +2 for logic that returns False if any are true
## 2 for enforcing total credits limit
##    +1 for calling Schedule.total_credits on the right thing
##    +1 for logic that returns False if limit is exceeded
```

6. [8 points] **Loop invariants.** Each of the following can be fixed with a one-line change to the code. Fix each method by crossing out **only one line** and rewriting it to the right, so that the code is consistent with the invariant.

```
def partition(b, z):
    i = 0
    j = len(b)-1
    # inv: b[0..i-1] <= z and b[j..] > z
    while i != j:
        if b[i] <= z:
            i += 1
        else:
            j -= 1
            b[i], b[j] = b[j], b[i]
    # post: b[0..j-1] <= z and b[j..] > z
    return j
```

Solution: Change `j = len(b)-1` to `j = len(b)`. 2 pts (all or none) for correcting the right line; 2 pts (all or none) for the right correction.

```
def partition2(b, z):
    i = -1
    j = len(b)
    # inv: b[0..i] <= z and b[j..] > z
    while i != j:
        if b[i+1] <= z:
            i += 1
        else:
            b[i+1], b[j-1] = b[j-1], b[i+1]
            j -= 1
    # post: b[0..j-1] <= z and b[j..] > z
    return j
```

Solution: Change `i != j` to `i != j-1`. 2 pts (all or none) for correcting the right line; 2 pts (all or none) for the right correction.

Did you write your name & netID on each page, and carefully re-read all instructions and specifications? Did you mentally test your code against the examples, where provided?

```
1   # enroll.py
2   # Steve Marschner (srm2) and Lillian Lee (ljl2)
3   """CS1110 Prelim 2: Module for tracking student enrollment in courses."""
4
5   class Course(object):
6       """An instance represents an offering of a course at Cornell.  There is a
7       separate Course instance for each semester in which a course is offered.
8       Each course also keeps track of the students who are enrolled.
9
10      Instance variables:
11         title [str] -- title of course
12         credits [int] -- number of credits
13         students [list of Student] -- list of students enrolled in this course
14      """
15
16      def __init__(self, title, credits):
17          """A new course with the given title and number of credits.
18          The course starts out with no students enrolled.
19          Pre: title is a string (e.g., 'CS1110: Awesome Introduction to Python')
20               credits is a positive integer
21          """
22          self.title = title
23          self.credits = credits
24          self.students = []
25
26
27  class Schedule(object):
28      """Instances represent a student's schedule for one semester.
29
30      Instance variables:
31         student [Student] -- the student whose schedule this is
32         semester [str] -- the semester this schedule is for
33         courses [list of Course] -- the Courses in this schedule
34      """
35
36      def __init__(self, student, semester):
37          """A schedule for <student> in <semester>, which starts with no courses.
38          """
39          self.student = student
40          self.semester = semester
41          self.courses = []
42
43      def total_credits(self):
44          """Return: the total number of credits in this schedule.
45          """
46          total = 0
47          for course in self.courses:
48              total += course.credits
49          return total
50
```

```
51          def overlaps(self, other_schedule):
52              """Return: True if this schedule contains any course with the same title
53              as a course contained in <other_schedule>.
54              Pre: other_schedule is a Schedule.
55              """
56              for course in self.courses:
57                  if other_schedule.contains_course(course):
58                      return True
59              return False
60
61          def contains_course(self, query_course):
62              """Return: True if this schedule contains a course with the same title
63              as <query_course>.
64              """
65              for course in self.courses:
66                  if course.title == query_course.title:
67                      return True
68              return False
69
70
71      class Student(object):
72          """Instances represent students at Cornell.  For each student, we keep track
73          of their schedules for each semester they've been at Cornell.
74
75          Instance variables:
76             name [str] ---  Name of student
77             schedules [list of Schedule] -- the student's schedules from all semesters,
78                in reverse chronological order.  The Schedule for the current semester
79                is at position 0 in this list.
80          """
81
82          def __init__(self, name):
83              """A new student named <name>, who starts with no schedules.
84              Pre: <name> is a string.
85              """
86              self.name = name
87              self.schedules = []
88
89          def start_semester(self, semester):
90              """Set up for a new semester by adding an empty Schedule at the head
91              of the schedules list.
92              Pre: <semester> is a string, such as '2014sp'
93              """
94              self.schedules.insert(0, Schedule(self, semester))
95
96          def add_course(self, course):
97              """Add a course for the current semester.  This means the course is added
98              to the student's current schedule, and the student is added to the
99              enrollment of the course.
100             Pre: <course> is a Course, the student has a current schedule, and <course>
101                  is not already on the current semester's schedule.
```

```
102             """
103             # TODO: implement this method
104             # Write your answer on the main exam sheet, not on this printout.
105
106         def validate(self, credit_limit):
107             """Return: True if the student's schedule for the current semester is
108             valid, which means that
109                (a) the total number of credits in the current semester is not over
110                    <credit_limit> (credits from prior semesters don't matter)
111                (b) the student is not taking any courses in the current semester that
112                    they already took in a previous semester.  Course titles are used
113                    to determine when a course is repeated; see Schedule.overlaps.
114             Pre: credit_limit is an integer, and the student has a current schedule.
115             """
116             # TODO: implement this method
117             # Write your answer on the main exam sheet, not on this printout.
118             # Be sure to take the time to read through all the methods in Schedule --
119             # using them makes this method much shorter to implement.
120
121
122     def test_enrollment():
123         """Test the enrollment system, making sure particularly that validation of
124         schedules works properly and that students get enrolled in the courses
125         that go on their schedules."""
126
127         # Four courses, offered in each of two semesters
128         c1_s14 = Course('CS1110: Awesome Python', 4)
129         c2_s14 = Course('CS2110: Jolly Java', 4)
130         c3_s14 = Course('CS4740: Natural Language Processing', 4)
131         c4_s14 = Course('CS4620: Computer Graphics', 3)
132         c1_f14 = Course('CS1110: Awesome Python', 4)
133         c2_f14 = Course('CS2110: Jolly Java', 4)
134         c3_f14 = Course('CS4740: Natural Language Processing', 4)
135         c4_f14 = Course('CS4620: Computer Graphics', 3)
136
137         # A student whose course enrollment validates OK
138         ljl = Student('Lillian Lee')
139         ljl.start_semester('Spring 2014')
140         ljl.add_course(c1_s14)
141         ljl.start_semester('Fall 2014')
142         ljl.add_course(c2_f14)
143         assert ljl.schedules[1].contains_course(c1_s14)
144         assert not ljl.schedules[1].contains_course(c2_f14)
145         assert not ljl.schedules[0].overlaps(ljl.schedules[1])
146         assert ljl.schedules[0].total_credits() == 4
147         assert ljl.validate(5)
148
149         # A student who is trying to re-take a course
150         srm = Student('Steve Marschner')
151         srm.start_semester('Spring 2014')
152         srm.add_course(c1_s14)
```

```
153         srm.start_semester('Fall 2014')
154         srm.add_course(c1_f14)
155         assert srm.schedules[1].contains_course(srm.schedules[0].courses[0])
156         assert srm.schedules[1].overlaps(srm.schedules[0])
157         assert not srm.validate(5)
158 
159         # A student who is trying to take too many credits
160         mcp = Student('Mary Pisaniello')
161         mcp.start_semester('Fall 2014')
162         mcp.add_course(c1_f14)
163         mcp.add_course(c2_f14)
164         mcp.add_course(c3_f14)
165         mcp.add_course(c4_f14)
166         assert mcp.schedules[0].total_credits() == 15
167         assert not mcp.validate(14)
168 
169         # Check that enrollments came out OK
170         assert set(c1_s14.students) == set([ljl, srm])
171         assert set(c2_f14.students) == set([ljl, mcp])
172 
173 
174     if __name__ == '__main__':
175         test_enrollment()
176 
```