CS 1110 Prelim 2 — April 22, 2014

This 90-minute exam has 6 questions worth a total of 46 points. When permitted to begin, scan the whole test before starting. Budget your time wisely. Use the back of the pages if you need more space. You may tear the pages apart; we have a stapler at the front of the room.

When asked to write Python code on this exam, unless otherwise stated, you may use any Python feature that you have learned about in class (if-statements, for-statements, map, lists, and so on).

It is a violation of the Academic Integrity Code to look at any exam other than your own, to look at any other reference material, or to otherwise give or receive unauthorized help. We also ask that you not discuss this exam with students who are scheduled to take a later makeup.

Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: _____   Date _____

For reference:

s[i:j]	Returns: A new string s[i] s[i+1] ... s[j-1] under ordinary circumstances. Returns '' if i ≥ len(s) or i ≥ j.
s.find(s1)	Returns: index of the first character of the first occurrence of s1 in s, or −1 if s1 does not occur in s.
s.index(s1)	Like find, but raises an error if s1 is not found.
lt[i:j]	Returns: A new list[lt[i], lt[i+1], ..., lt[j-1]] under ordinary circumstances. Returns [] if i ≥ len(lt) or i ≥ j.
lt.index(item)	Returns: index of first occurrence of item in list lt; raises an error if item is not found.
range(n)	Returns: the list [0, 1, 2, ..., n-1]
x in lt	Returns: True if x is in list lt, False otherwise.
lt.append(x)	Append object x to the end of list lt.
lt.sort()	Sort the items of lt, in place (the list is altered).

Question	Points	Score
1	2	
2	10	
3	6	
4	8	
5	12	
6	8	
Total:	46	

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

Example: In the figure below, (2,0) is downstream from (0,1), (0,2), (0,4), and (1,1), but no other nodes.
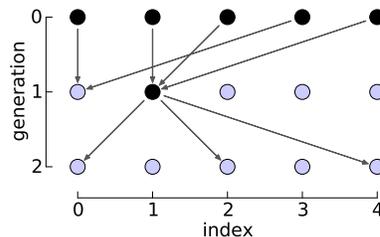
3. [6 points] **While-loops.** Write a function that does the same thing as `product_for` but uses a while-loop.

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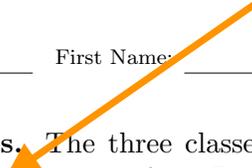
4. [8 points] **While-loops.** Implement the `strip` function so that it meets its specification, using two *non-nested* while-loops: *one starting from the beginning of the string and moving right*, and then *one starting from the end of the string and moving left*.

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

```python
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.
```

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).

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

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

```python
    ...

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

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
```

```
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
```

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?

```python
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
```

```python
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
```