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.

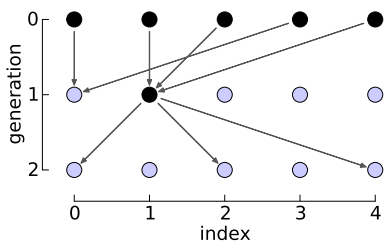
```
class Node(object):

    ...
    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
```

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?