Last Name: _____ First Name: _____ Cornell NetID: _____

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` $\geq$ `len(s)` or `i` $\geq$ `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` $\geq$ `len(lt)` or `i` $\geq$ `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).

Last Name: _____ First Name: _____ Cornell NetID: _____

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

Last Name: _____ First Name: _____ Cornell NetID: _____

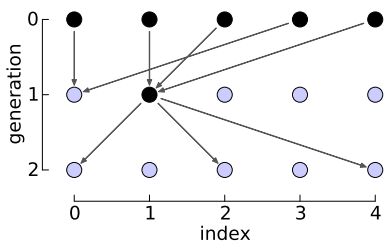
1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of *each* page.

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.

```
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`.

```
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).

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

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









    ...

    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?