Last Name: _____ First Name: _____ Cornell NetID, all caps: _____

CS 1110 Prelim 2, April 2018

This 90-minute exam has 6 questions worth a total of 52 points. You may tear the pages apart; we have a stapler at the front of the room.

You are expected to track the announcements displayed at the front of the exam room yourself, so that we can minimize audible interruptions during the test.

You may use any Python introduced so far in this course, but not Python constructs that we have not introduced.

When a question asks for a particular technique to be used, do not expect to receive credit for questions based on a different technique. **For example, if we ask you to make effective use of recursion, your solution must be fundamentally based on recursion, even if a recursion-less for-loop would also solve the problem.**

The second page of this exam gives you the specifications for some useful functions and methods.

You will be expected to write Python code on this exam. We recommend that you draw vertical lines to make your indentation clear, as follows:

```
def foo():
    if something:
        do something
        do more things
    do something last
```

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 _____

`s.find(substr)`	Returns: index of first occurrence of string `substr` in string `s` (-1 if not found)
`s.strip()`	Returns: copy of string `s` where all whitespace has been removed from the beginning and the end of `s`. Whitespace not at the ends is preserved.
`s.split(sep)`	Returns: a list of the "words" in string `s`, using `sep` as the word delimiter (whitespace if `sep` not given)
`s.join(slist)`	Returns: a string that is the concatenation of the strings in list `slist` separated by string `s`
`s[i:j]`	Returns: if `i` and `j` are non-negative indices and `i` $\leq$ `j-1`, a new string containing the characters in `s` from index `i` to index `j-1`, or the substring of `s` starting at `i` if `j` $\geq$ `len(s)`

`lt.insert(i,item)`	Insert `item` into list `lt` at position `i`
`lt.append(item)`	Adds `item` to the end of list `lt`
`lt.count(item)`	Returns: count of how many times item occurs in list
`list(range(n))`	Returns: the list `[0 .. n-1]`
`lt.remove(item)`	Removes the first occurrence of `item` from list `lt`; raises an error if `item` not found.
`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` is not found. (There's no "find" for lists.)
`lt[i:j]`	Returns: A new list `[lt[i], lt[i+1], ..., lt[j-1]]` under ordinary circumstances. Returns `[]` if `i` and `j` are not both sensible indices.
`lt.pop(i)`	Returns: element of list `lt` at index `i` **and also removes that element from the list** `lt`. Raises an error if `i` is an invalid index.

`list(map(func, lt))`	Returns: A list obtained by applying function `func` to each element in list `lt` and concatenating the results of each application.

Question	Points	Score
1	5	
2	7	
3	5	
4	10	
5	10	
6	15	
Total:	52	

1. [5 points] Implement the following, making effective use of **for-loops**, so that it obeys its specification.

```
def overlay_value(to_list, from_list, v):
    """Copy each v in from_list to the corresponding position in to_list.

    (Does not change anything else in to_list, or anything in from_list.
    Does not return anything.)

    Precond: to_list and from_list are lists of the same length (possibly 0).
             v is a string, int, float, bool, or None.

    Examples:
        if to_list is ['_', '_', '_']
        from_list is  ['m', 'o', 'o']
        v is 'o'
           ---> to_list becomes ['_', 'o', 'o']

        if to_list is ['x', 'x', 'x']
        from_list is  ['m', 'o', 'o']
        v is 'e'
           ---> to_list stays as it was

        if to_list is [0, 9, 4]
        from_list is  [4, 0, 2]
        v is 0
           ---> to_list becomes [0, 0, 4]
    """
```

2. [7 points] Implement the following, making effective use of **for-loops**, so that it obeys its specification.

```
def counts(datalists, targets):
    """Returns: a new list where each element is how many elements of
    the corresponding list in datalists occur in targets.

    (Does not change datalists, any list in datalists, or targets.)

    Preconditions: Every element in datalists is a list of ints, possibly empty.
                   targets is a non-empty list of ints, no repeats.

    Example: If targets is [0, 12, -7, 13]
             and datalists is [[4000, 1100, 3600, 1000],
                               [12, -10000],
                               [13, 0, 13],
                               []]
             then the output is [0, 1, 3, 0]"""
```

3. [5 points] **Simple Recursion.** Make effective use of recursion to implement `countdown_by_n`. Your solution *must be recursive* to receive points.

```
def countdown_by_n(count_from, count_by):
    """Prints a count down from count_from by count_by.
    Stops printing before the result goes negative.
    Note: this function does not return anything.

    count_from: the number you're counting down from [int]
    count_by: the amount you're counting down by [int > 0]

    Examples:

    coundown_by_n(16, 5) should print:
       16
       11
       6
       1

    coundown_by_n(21, 7) should print:
       21
       14
       7
       0

    """
```

4. [10 points] **Recursion with Classes.** Make effective use of recursion to implement `get_all_prereqs` for class `Course`. Your solution *must be recursive* to receive points. We should have specified the method should return a *new* list.

```
class Course():
    """An instance represents a course offered by a university.
    Courses are uniquely identified by their course number.

    Instance variables:
        course_num [str] -- unique non-empty course number, e.g., 'CS1110'
        prereqs [list of Course] -- courses that one must complete
        before one may enroll in this course. possibly empty. """

    def __init__(self, course_num, prereqs):
        """A new course called course_num with prerequisites prereqs.
        Pre: course_num: a non-empty string (e.g., 'CS1110')
             prereqs: a (possibly empty) list of Course"""
        self.course_num = course_num
        self.prereqs = prereqs

    def get_all_prereqs(self):
        """Returns a list of str: ALL the course_nums that one would
        have taken before taking this class.
        Note: (1) duplicates are fine
              (2) order of list items in does not matter
        Example:
        c1 = Course('CS1110', [])
        c2 = Course('CS2800', [c1])
        c3 = Course('CS2110', [c1])
        c4 = Course('CS4820', [c2,c3])

        c1.get_all_prereqs() should return []
        c2.get_all_prereqs() should return ['CS1110']
        c4.get_all_prereqs() should return ['CS1110','CS2800','CS1110','CS2110']
        """
```

5. [10 points] **Handling objects.** For this question, all you need to recall from A4 about the attributes of Position objects is:

   - sups (list of Positions, possibly empty): The list of Positions that are direct supervisors of this Position. There are no repeats in it.
   - subs (list of Positions, possibly empty): The list of Positions that this Position directly supervises, i.e., its direct subordinates. There are no repeats in it.
   - Class invariant: If Position pos1 has Position pos2 in its sups list, then pos2 has pos1 in its subs list, and vice versa.

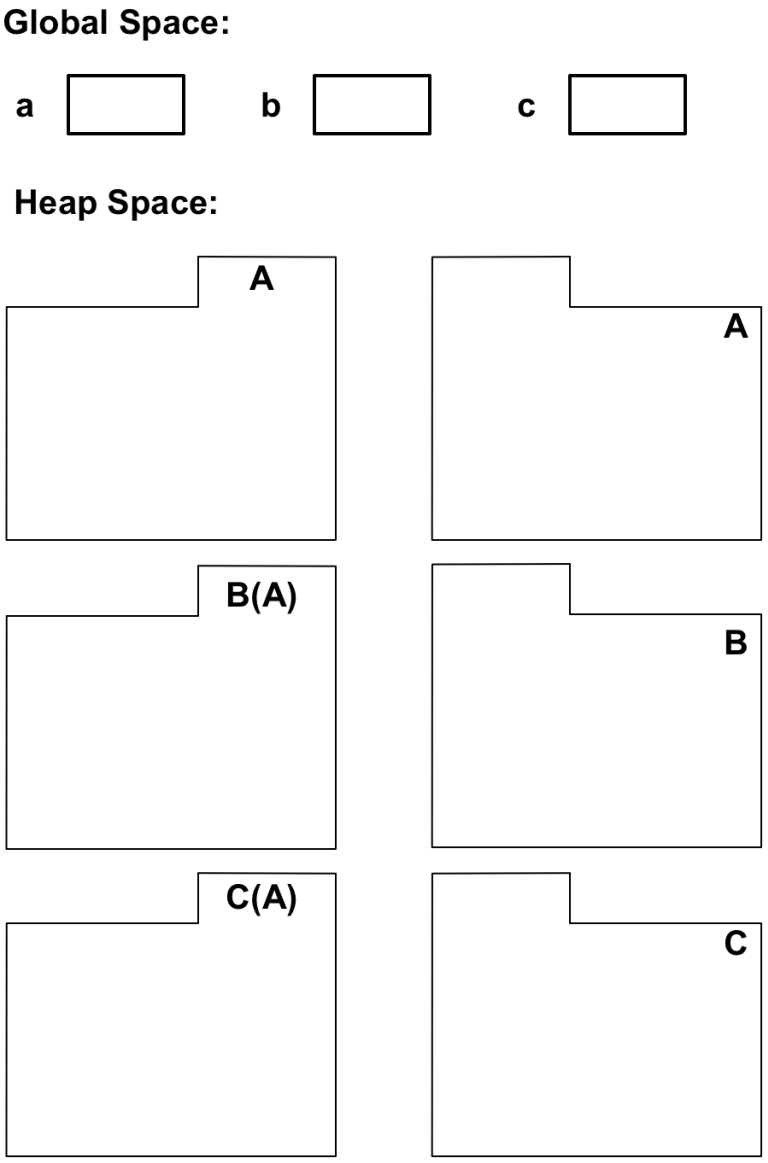
   Implement this method for class Position. **Don't use recursion**; it's not needed.

```
def collapse_upwards(self):
    """ Collapses this Position into its supervisor Position.

    Preconditions: this Position has exactly one direct supervisor.

    Example: suppose this Position has title 'B',
    its single direct supervisor has title 'A',
    and its direct subordinates have titles 'C1' and 'C2'.
            A
            | \
       E    B  D?  [maybe there are other subs of A, who knows]
        \  / \
         C1  C2 [note that subordinates can have other supervisors]

    This method changes the situation to
      E     A
       \  / | \
        C1  C2  D

    (The order of the subordinates doesn't matter.)

    Thus, in our example,
      1. A is added to the sups list of C1 and C2
         (remember to avoid adding repeats)
      2. C1 and C2 are added to the subs list of A
         (remember to avoid adding repeats)
      3. B is removed from the subs list of A
         (recall that mylist.remove(x) removes first x from mylist)
      4. B is removed from the sups list of C1 and C2
      5. B's sups list becomes the empty list
      6. B's subs list becomes the empty list
      (Don't change anything else about B.)

    YOU MAY NOT MAKE USE OF PREDEFINED HELPERS FROM A4, but you may write
    your own, as long as you specify them carefully.
```

```
class Position():
    # [other stuff about this class omitted]

    def collapse_upwards(self):
        """Specification on previous page."""
        # DON'T USE RECURSION.  It's not needed.
```

6. [15 points] **Inheritance**.

```
class A(object):
    d = 10

    def __init__(self, n):
        self.n = n
        self.m = 0
        A.d += 1

class B(A):

    d = 20

    def __init__(self, n):
        super().__init__(n)
        self.p = 2

class C(A):

    f = 8

    def __init__(self, n):
        self.r = n

a = A(1)
b = B(2)
c = C(3)
```



We guarantee that no errors result from running the code above.

Complete the object and class folders above that result by running this code. Include method names and class variables in the class folders. Add id #s and attributes to the object folders. If any values change as the code is executed, show *all* values by crossing out old values and putting new values next to the crossed out ones.

Did you re-read all specs, and check your code works against test cases?