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

CS 1110 Prelim 2 Solutions April 2018

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]
    """
```

Solution:

```
    for ind in range(len(from_list)):
        if from_list[ind] == v:
            to_list[ind] = v
            ##### Also OK to do (if properly indented), but above is preferred
            # to_list.pop(ind)
            # to_list.insert(ind, v)
```

The following involves more book-keeping (and so more possibilites for bugs), so we prefer the solution above.

```
    ind = 0
    for item in from_list:
        if from_list[ind] == v:
            to_list[ind] = from_list[ind]
        ind += 1
```

An interesting alternate solution from a student that first figures out what indices v occurs at in from_list:

```
places_with_v = []
for ind in range(len(from_list)):
    if from_list[ind] == v:
        places_with_v.append(ind)
for index_value in places_with_v:
    to_list[index_value] = v
```

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]"""
```

Solution:
This is essentially the same problem as track_topics in Spring 2018 A3.

```
    outlist = []
    for datalist in datalists:
        count = 0
        for num in datalist:
            if num in targets:
                count += 1
        outlist.append(count)
    return outlist
```

Alternate solutions:

```
def counts2(datalists, targets):
    # Alternate solution iterating through indices
    outlist = []
    for i in range(len(datalists)):
        count = 0
        for num in datalists[i]:
            if num in targets:
                count += 1
        outlist.append(count)
    return outlist


def counts3(datalists, targets):
    # Alternate solution adapted from one by Trey Driskell:
```

```
# Iterates through targets in inner loop
outlist = []
for datalist in datalists:
    count = 0
    for target in targets:
        count += datalist.count(target)
    outlist.append(count)
return outlist
```

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

    """
```

Solution:

Note that count_from can start negative.

```
    if count_from < 0:
        return
    print(count_from)
    countdown_by_n(count_from - count_by, count_by)
```

Alternate solution:

```
if count_from >= 0:
    print(count_from)
    countdown_by_n(count_from - count_by, count_by)
```

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']
        """
```

Solution:

Note: in general, if a specification does *not* explicitly say that you should be changing an input argument, *don't change it* — it's not safe to do so because the caller may not expect this.

```
if self.prereqs == []:
    return []
reqs = []
for req in self.prereqs:
    reqs.append(req.course_num)
    reqs += req.get_all_prereqs()
return reqs
```

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.
```

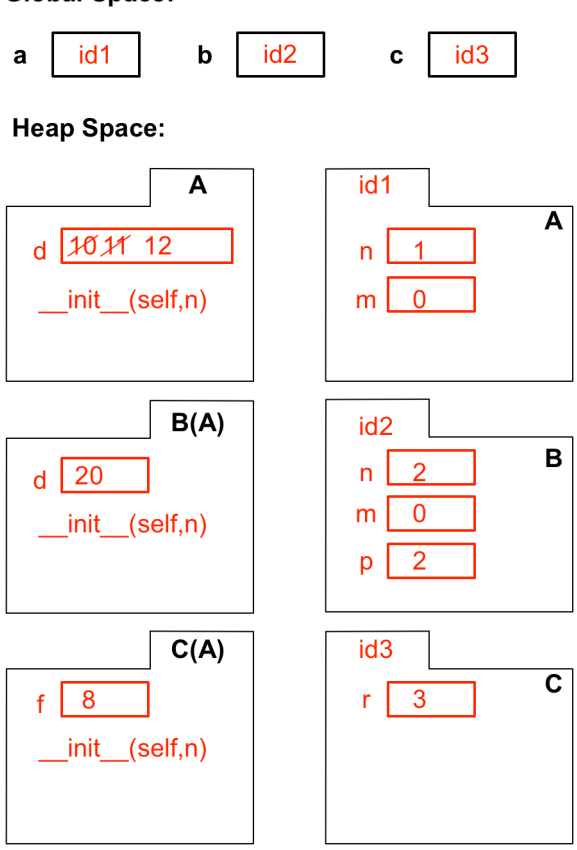
Solution:

```
        newsup = self.sups[0]  # local variable for A for convenience
        for sub in self.subs:
            if newsup not in sub.sups:
                sub.sups.append(newsup)  # job 1
                newsup.subs.append(sub)  # job 2
            sub.sups.remove(self)  # job 4

        newsup.subs.remove(self)  # job 3
        self.sups = []  # job 5
        self.subs = []  # job 6
```

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)
```

Solution:



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.