Last Name: ____________ First Name: ____________ Cornell NetID, all caps: ______

CS 1110 Final Solutions May 2017

1. [11 points] Implement the following function according to its specification.

```
def putSideBySide(two_line_strings):
"""Returns: a string with two lines. The first line of the return string
    should contain, in order, the first line of each string in two_line_strings,
    separated by a space. The second line of the return string should contain,
    in order, the second line of each string in two_line_strings, separated by a space.

    Input: two_line_strings, which is a non-empty list of two-line strings.
    Each string has exactly this form: xx\nxx, where x is a character
    in [a..z or A..Z or 0..9]. There will always be exactly two characters
    before and after the new line character \n.

    Remember that \n is a SINGLE special character indicating a new line
    (which causes the string to print over multiple lines).

    For example:
    putSideBySide(["AB\nCD", "EF\nGH"]) should return "AB EF\nCD GH".

    This corresponds to arranging strings that print as:

    AB
    CD

    and:

    EF
    GH

    into the following:

    AB EF
    CD GH

    Note: should NOT add a space before the first string or after the last string.
    This would be wrong because there is a space before the first AB\nCD:

     AB EF
     CD GH

    Another example:
    putSideBySide(["AB\nCD", "EF\nGH", "IJ\nKL"]) returns "AB EF IJ\nCD GH KL".
    """
```

Implement this function on the next page.

```
# Put your code for function putSidebySide below.
```

Solution:
Here are several alternatives.

Solution LL1:

```
def solLL1(two_line_strings):
    top = ""
    bottom =""
    for tls in two_line_strings:
        nindex = tls.find("\n")
        front = tls[:nindex]
        back = tls[nindex+1:]
        top += (front + " ")
        bottom += ( back + " ")
    return top.strip() + "\n" + bottom.strip()
```

Solution AP1:

```
def solAP1(two_line_strings):
    top_str = ''
    bottom_str = ''
    for x in two_line_strings:
        top_str += x[0:2] + ' '
```

Solution EA:

```
    """
# BEGIN REMOVE
    line1 = ""
    line2 = ""
    first = True
    for a_string in two_line_strings:
        parts = a_string.split("\n")
        if first:
            first = False
        else:
            line1 += " "
            line2 += " "
        line1 += parts[0]
```

Solution AP2:

```
def solAP2(two_line_strings):
    top_str = two_line_strings[0][0:2]
    bottom_str = two_line_strings[0][3:5]
    for x in two_line_strings[1:]:
        top_str += ' ' + x[0:2]
```

Solution AP3:

```
def solAP3(two_line_strings):
    # ima hacky haxor solution. Uses [::], which wasn't discussed in class
    twolist = '\n'.join(two_line_strings).split('\n')
    first = twolist[::2] #crazy python nonsense
```

Alternate: Solution LL2

```
def solLL2(two_line_strings):
    converted = map(splitem, two_line_strings)

    # top is output[0], bottom is output[1]
    output = [converted[0][0], converted[0][1]]

    for i innon range(1, len(converted)):
        for j in range(2):
            output[j] += (" " + converted[i][j])
    return output[0] + "\n" + output[1]


# helper for solLL2; to allow use of map
def splitem(tls):
    """Version of split() that takes the string to split as an argument"""
```

2. [21 points] Consider the following code to solve the "Towers of Hanoi" problem. We guarantee there are no errors.

```
class Tower(object):
  def __init__(self, name, disks):
    self.disks = disks
    self.name = name

  def topDisk(self):
    if self.disks == []:
      return None
    else:
      return self.disks[len(self.disks) - 1]

  def popTopDisk(self):
    return self.disks.pop(len(self.disks) - 1)

  def move(self, to):
    if self.topDisk() is not None:
      isEmpty = to.topDisk() is None

      if isEmpty or to.topDisk() > self.topDisk():
        to.disks.append(self.popTopDisk())

def plan(source, target, other, num_disks):
  if num_disks == 1:
    source.move(target)
  else:
    plan(source, other, target, num_disks-1)
    print "move disk"
    source.move(target)
    plan(other, target, source, num_disks-1)

left = Tower("left", [2, 1])
middle = Tower("middle", [])
right = Tower("right", [])
plan(left, right, middle, len(left.disks))
```

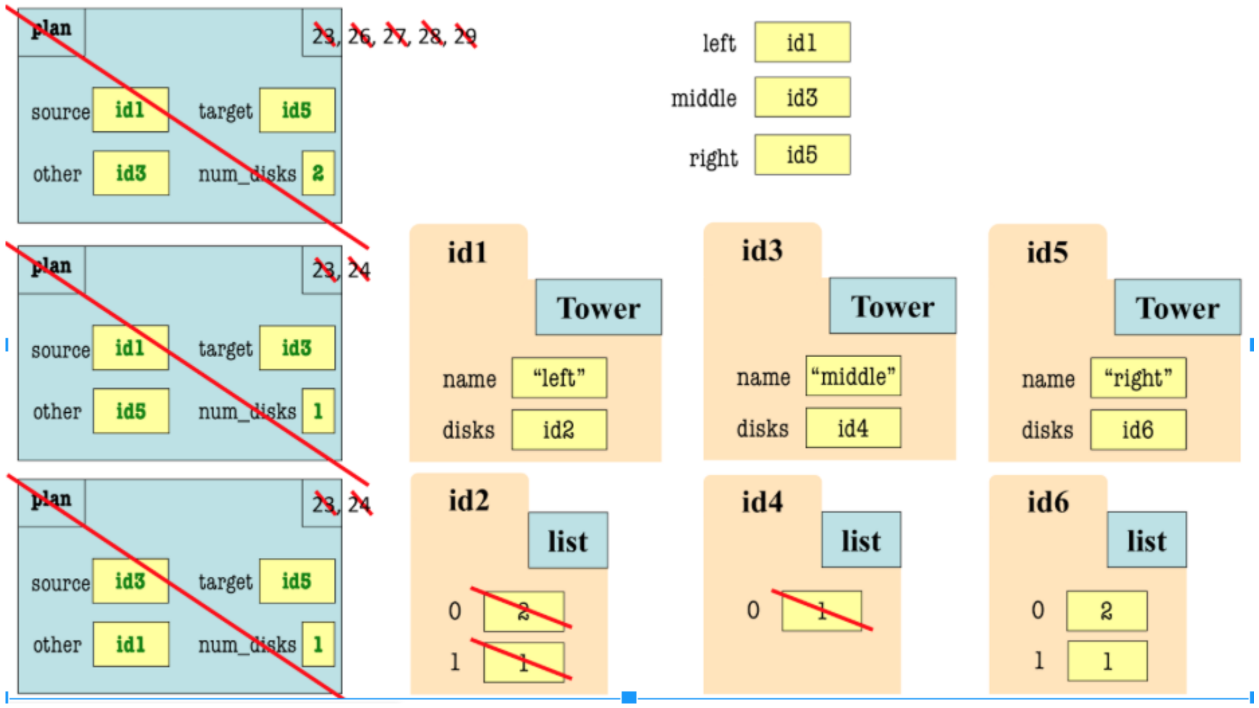
Draw the requested items here:

In the space above, draw the function call frames for `plan`, global variables, and object folders that result by running this code.

Do NOT draw class folders or folders for functions. You do NOT need to draw function call frames for any function other than `plan`.

Remember that lists are objects.

Solution:



It is OK to include line 25 (for the line `else:`) in the 1st call frame

3. For debugging purposes, Professor Andersen wants to count how many moves it takes to complete the Towers of Hanoi puzzle in the previous example. He added a `print` statement on line 27 to print "move disk" whenever a "disk" is moved. This may have been a poor choice.

   (a) [1 point] How many times will "move disk" get printed when the code is run?

   Solution:
   1

   (b) [1 point] How many moves were actually made (i.e., line 20 was actually executed)?

   Solution:
   3

   (c) [3 points] Suggest how Prof. Andersen can change the code so that "move disk" is printed each time a move is made. Refer to specific line numbers on which `print` statements should be inserted or deleted, and also be clear about the level of indentation of any added `print`s.

   Solution:
   Delete print at 27; add print at 21, indented under the second if.
   OR: Add a print on Line 24
   OR: Delete print at 27; add print at line 16
   OR: Delete print at 27; add print at line 13 before return

4. Consider the following *subclass* `Loutcome` of class Outcome. It has *almost* the same class invariant as Outcome. The differences are shown in orange below, so you should be to just skim the orange parts.

```
class Loutcome(Outcome):
    """ An instance is a loser-annotated outcome in a tournament tree.

    Attributes:
    winner [nonempty str]: name of the winner in this Loutcome
        Must be the same as the name of *exactly one* of attributes input1 or
            input2, defined next.
    loser [nonempty str]: name of the loser in this Loutcome

    input1 [Loutcome or nonempty string]:
        If a nonempty string, the name of a competitor in the tournament, and
            we say that the name of input1 is that string.
        If a Loutcome, then the name of input1 is its winner attribute.

    input2 [Loutcome or nonempty string]:
        If a nonempty string, the name of a competitor in the tournament, and
            we say that the name of input2 is that string
        If a Loutcome, then the name of input2 is its winner attribute.

    Note that the constraints (invariants) on winner imply that the names of
        input1 and input2 must be different. """
```

(a) [12 points] Implement the `__init__` method of Loutcome below.

```
def __init__(self, in1, in2, one_won=True):
    """Same as for the __init__ for Outcome, except that:

        If one_won is True, loser is the name of in2's winner
        (if in2 is an Loutcome) or in2 itself (if in2 is a string)

        Otherwise, loser is in1's winner (if in1 is an Loutcome) or
            in1 itself (if in1 is a string)

    Preconditions: same as for the __init__ for Outcome, except substitute
    the word "Loutcome" for "Outcome" """

     ### You MUST effectively call the __init__ method of Outcome.
     ### Its header is: def __init__(self, in1, in2, one_won=True)
     ### You are allowed to use _extract_name (see below for specification).
```

Solution:

We can use `_extract_name` on Loutcomes because Loutcomes *are* Outcomes (since Loutcome is a subclass of Outcome).

```
    Outcome.__init__(self, in1, in2, one_won) # CAN'T say "one_won=True"
    if one_won: # "if one_won == True" OK
        self.loser = _extract_name(in2)
    else:
        self.loser = _extract_name(in1)
```

Specification for `_extract_name()`. It is *not* a method of Outcome.

```
def _extract_name(x):
    """Returns: string that is the name of x, defined as follows:
    If x is an Outcome, then the name is x's winner; otherwise, the name is x itself.

    Precondition: x is either a non-empty string or an Outcome."""
```

Last Name: ______________________ First Name: ________________ Cornell NetID: ______

(b) [17 points] Implement the following method of **Loutcome** according to its specification.

```
def winsAndLosses(self, team):
    """Returns: a two-item list where the item at index 0 is the number of
      games that team won, and the item at index 1 is the number of games
      that team lost.

      Precondition: team [str] is a team that played in this Loutcome.

     Example: for the Loutcome below:  |    Here's the desired output
                                       |    for various teams:
     D beat B                          |
         D beat A                      |
             A beat B                  |     "A" --> [1,1]
                 A                     |     "B" --> [1,2]
                 B                     |     "C" --> [0,2]
             D beat C                  |     "D" --> [3,0]
                 C                     |
                 D                     |
         B beat C                      |
             B                         |
             C                         |
     """
     ### Hint: We are dealing with Loutcomes, so you don't need _extract_name.
```

Solution:

```
    # BEGIN REMOVE
    if team == self.winner:
        output = [1, 0]
    elif team == self.loser:
        output = [0, 1]
    else:
        output = [0,0]

    for sub in [self.input1, self.input2]:
        if isinstance(sub, Loutcome):
            suboutput = sub.winsAndLosses(team)
            output[0] += suboutput[0]
            output[1] += suboutput[1]
```

5. [12 points] Implement the following function according to its specification.

```
def merge_records(d1, d2):
    """Input: d1 and d2 are (possibly empty) dictionaries representing win-loss records:
        Each key is a non-empty string representing a team name.
        The value for each key is a two-item list of ints, where the first is
            the number of wins and the second is the number of losses for that
            team in some tournament/outcome tree.

    This function adds the win-loss records of d2 into d1.

    WARNINGS: It does NOT return anything; it changes d1 but not d2.
    And, the values in the altered d1 should be different list objects than the
    list objects that are values in d2, even if they have the same numbers in them.

    For example:

    if d1 is                 and d2 is                then d1 becomes
      {"Cornell": [10,1],      {"Cornell": [2,0],       {"Cornell":[12,1],
        "Harvard": [4,3]}        "Stanford": [0,3]}      "Harvard": [4,3],
                                                         "Stanford": [0,3]}

            where the new d1's "Stanford" list [0,3] is a DIFFERENT list object
            than d2's "Stanford" list object """

     ### HINT: for a list mylist, list(mylist) or mylist[:] returns a copy of mylist.
```

Solution:

```
    for team in d2:
        if team in d1:
            d1[team][0] += d2[team][0]
            d1[team][1] += d2[team][1]
        else:
            d1[team] = list(d2[team]) # d2[team][:] also OK
```

Alternate solution:

```
    for team in d2:
        if team not in d1: # "if not team in d1" also OK
            # make it so team *is* in d1
            d1[team] = [0,0]

        d1[team][0] += d2[team][0]
        d1[team][1] += d2[team][1]
```

Alternate solution with more loops than necessary:

```
for team in d1:
    if team in d2:
        d1[team][0] += d2[team][0]
        d1[team][1] += d2[team][1]

for team in d2:
    if team not in d1:
        d1[team] = list(d2[team])
```

6. A progression is a sequence of integers that are separated by the same distance; for instance, 5, 10, 15, 20 is a progression with 3 steps of step-size 5, and -15, -12, -9 is a progression with 2 steps of step-size 3.

   We would like to use a while-loop to write a function `has_progression(x, step, n)` that returns the starting index `start` in list of ints `x` for a progression of step-size `step` that has at least `n` steps. (It should return `-1` if there is no such `start`, and we require that `n>=1`).

   For example, suppose that `x` is `[1, 2, 14, 16, 18, 20, 22]`. Then,

```
has_progression(x, 2, 3) -->  2 b/c x[2]==14, x[3]==16, x[4]==18, x[5]==20
has_progression(x, 2, 4) -->  2 b/c the step-2 progression 14,16,18,20,22
                                starts at x[2] and has 4 steps.
has_progression(x, 2, 5) --> -1
has_progression(x, 1, 1) --> 0
```

   We give you the following invariant, documenting variables `start`, `m`, and `i`.

```
                                start                      i
x | the progression can't start here | progression of m steps | ??? |
```

   In words, `x[start..i-1]` forms a progression with `m steps`; `x[..start-1]` does not contain an `n`-step progression, and if `start-1` is a valid index, then `x[start-1] != x[start] - step`.

   (a) [4 points] We initialize `i` to be 1, because there is necessarily a progression in `x[0..0]`, albeit one with 0 steps.

   What should we then initialize `m` and `start` to be, *according to the invariant?*

   Solution:
   `m = 0, start = 0`

   (b) [3 points] Is the following a correct while-loop condition *according to the invariant*? If yes, write the word "YES" below it; otherwise, cross it out and write a correct while-loop condition below it.

   `while m < n and i < len(x) - 1:`

   Solution:
   It should be `while m < n and i < len(x):`, or loop ends too early.

   (c) [5 points] Suppose the invariant is true, and `m` is much less than `n` and `i` is much less than the length of `x`.

   Now, suppose we find out that `x[i] == x[i-1]+step`. *According to the invariant...*

   ...should `m` and/or `i` be updated? If so, give Python code making the update(s); if not, write "No changes".

   Solution:
   `m += 1` or `m = m+1`. `i += 1` or `i = i+1`. Luckily, here, the order of i and ms update doesn't matter.

...and, should `start` be updated? If so, give Python code making the update; if not, write “No change to `start`”.

Solution:
No change to `start`

Helper function for question on next page.

```
def swap(b,h,k):
    """ Swaps element h and k in list b """
    tmp = b[h]
    b[h] = b[k]
    b[k] = tmp
```

Last Name: ______________________ First Name: ________________ Cornell NetID: ______

7. [15 points] Implement string_list_sort according to its specification.

```
def string_list_sort(b,h,k):
    """Swaps items in the PORTION of list b from index h up to and including
    index k so that strings are in front and lists are in back.
    Returns: index i such that b[h..i] are strings and b[i+1..k] are lists.

    Example: if b = [['Prospero', 'and', 'Ariel'], 'how', ['are', 'you']]
    string_list_sort(b,0,2) returns 0,
      and could change b to :
      ['how', ['are', 'you'], ['Prospero', 'and', 'Ariel']]
      or
      ['how', ['Prospero', 'and', 'Ariel'], ['are', 'you']]

    Your solution MUST use a while loop, MUST NOT create a copy of the list,
    and MUST satisfy the invariants set out in the code below.

    Precondition: b is a nonempty list containing only lists and strings; h and k
    are integers that are valid indices in list b"""
    # INVARIANT: b[h..i] are strings, b[t..k] are lists
    # PRECONDITION: no strings or lists yet identified (b[h..i] and b[t..k] both empty)

    # FIRST, FIX THESE INITIALIZATIONS
    i = None
    t = None

    # Hint: you may use the swap function from the previous page
```

Solution:

```
    i = h - 1
    t = k + 1
    while t != i+1:   # i < t-1 also OK
        if type(b[i+1]) == str:
            i = i + 1
        else: # type(b[i+1]) == list
            swap(b,i+1,t-1)
            t = t - 1
```

Alternate solution:

```
    i = h - 1
    t = k + 1
    while t != i+1: # i < t-1, t >= i also OK
        if type(b[t-1]) == list: #isinstance also good here
```

```
            t = t - 1
        else: # type(b[t-1]) == str
            swap(b,i+1,t-1)
            #may also use python b[i+1],b[t-1] = b[t-1],b[j-1]
            i = i + 1
    return i

    return i
    #POSTCONDITION: b[h..i] are strings, b[i+1..k] are lists
```

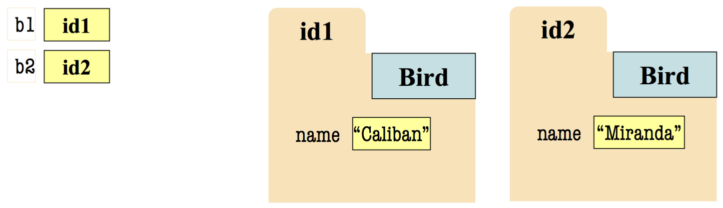
8. (a) [3 points] Consider the following class definition:

```
1 class Bird(object):
2   def __init__(self, n):
3       self.name = n
4   def tweet(self, punc):
5       output = self.name + " says tweet"
6       return output + str(punc)
```

Write a Python assignment statement that stores in variable `x` the ID of a new Bird object whose name is the string “Caliban”.

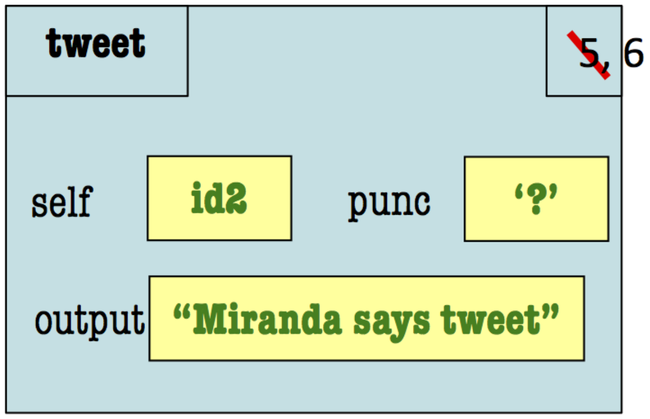
Solution:
`x = Bird("Caliban")`

(b) [8 points] Consider the following objects and global variables:



Draw the call frame that results from the call `b2.tweet('?')`, stopping execution just before line 6 is executed. Include any crossed-off variable contents or line numbers.

Solution:



9. [1 point] **Fill in your last name, first name, and Cornell NetID at the top of each page.**

Solution:
Always do this! It prevents disaster in cases where a staple fails.