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

CS 1110 Prelim 2, April 2017

This 90-minute exam has 6 questions worth a total of 74 points. You may tear the pages apart; we have a stapler at the front of the room.

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 _____

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

For reference:

`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.append(item)`	Adds `item` to the end of list `lt`
`lt.remove(item)`	Removes the first occurrence of `item` from list `lt`.
`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**.

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

Question	Points	Score
1	10	
2	16	
3	24	
4	18	
5	5	
6	1	
Total:	74	

1. [10 points] **For-loops**. Implement the following specification. Your implementation must make effective use of a for-loop.

```
def followers(wordlist, starter):
    """Returns a list of all words that immediately follow starter in wordlist,
        in the order they appear in wordlist.
        (Returns the empty list if there aren't any).
        Does not alter wordlist.

    Preconditions:
       wordlist is a list of nonempty strings, none of which contain spaces
          wordlist might be itself empty
       starter is a nonempty string with no spaces

    Example: if wordlist is ["a", "man", "a", "plan", "a"],
        if starter is "a", returns ["man", "plan"]
        if starter is "flower", returns [] """
    ### Your implementation must make effective use of a for-loop


    result = []
    for s in wordlist:
        result.append(wordlist[wordlist.index(starter)+1])
        wordlist = wordlist[wordlist.index(starter):]
    result
```

2. [16 points] **Recursion.** Make effective use of recursion to implement the following new method, **games**, for class Outcome. Some relevant specifications are given on the next page.

```
class Outcome(object):
    """Class invariant given on next page, for space reasons."""

    def games(self, team):
        """Returns an int representing the number of games that team played in
        this Outcome and all its sub-Outcomes.

        Precondition: team is a non-empty string.

        For examples of output for games(), see the next page."""


        ### You may NOT use the teams() method from A4.
        ### You MUST use _extract_name() --- specification on the next page.

        ### Look at the examples for games() on the next page before you start.
```

Reference material for previous page — There are no questions on this page.

Example for games():

```
Example: for the Outcome below:    Here are some inputs & outputs for games():
D                                |
    D                            |
        A                        |    "A" -> 2
            A                    |    "B" -> 3
            B                    |    "C" -> 2
        D                        |    "D" -> 3
            C                    |    "E" -> 0
            D                    |
    B                            |
        B                        |
        C                        |
```

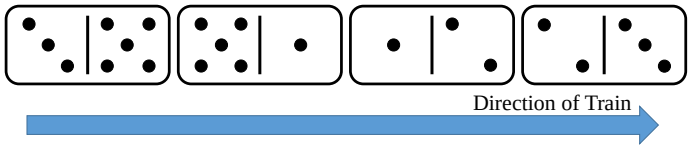
Class invariant for Outcome:

```
"""An instance is an outcome in a tournament tree.

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

input1 [Outcome 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 an Outcome, then the name of input1 is its winner attribute.

input2 [Outcome 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 an Outcome, 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.
"""
```

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

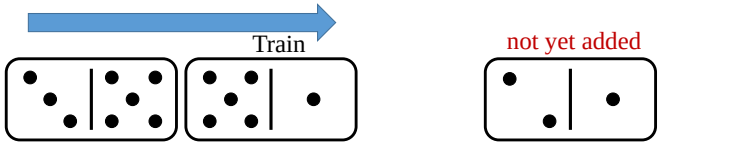
```
def _extract_name(x):
    """Returns: string that is the name of x, defined as follows:
    If x is an Outcome, then the name is x's winner; otherwise, the name is x itself.

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

3. [24 points] **Classes**. Dominoes are rectangular tiles with one number on each side. There are multiple games in which players place dominoes on a table in a row called a "train". A domino can be placed in a train next to another domino if their adjacent numbers match:



There is a skeleton of a Domino class below and on the following pages. Complete this class by adding code under each function specification.

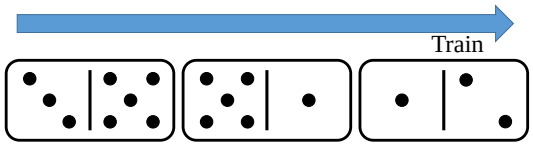
When you have finished, `addDomino` should have the following behavior. Before adding a domino [2 | 1], the train might look like this:



side1 = 5
side2 = 1
prior = 
next = None
outwardFacingSide = 1

side1 = 2
side2 = 1
prior = None
next = None
outwardFacingSide = -1

After [2 | 1] has been added (and flipped), the train will look like this:



side1 = 5
side2 = 1
prior = 
next = 
outwardFacingSide = 1

side1 = 2
side2 = 1
prior = 
next = None
outwardFacingSide = 2

Note that `side1` and `side2` for [2 | 1] have not changed.

```
class Domino(object):
    """ An instance represents a Domino game piece

    Attributes:
        side1: the value on side 1 of the domino piece [int 1..6]
        side2: the value on side 2 of the domino piece [int 1..6]
        prior: the domino immediately preceeding this domino [Domino or None]
        next: the domino that follows this domino [Domino or None]
        outwardFacingSide: the value on the side that is facing out,
            or -1 if unattached [int -1 or 1..6] """
```

```
def __init__(self, n1, n2):
    """ Makes new domino have value n1 on side 1 and n2 on side2
    and not attached to any domino train (prior and next are None).
    outwardFacingSide should default to -1.

    Precondition: n1, n2 are integers in [1..6]"""
```

```
def __str__(self):
    """Returns: The string representation of this Domino
    in the form: "Domino: side1|side2". For example:
    if side1 is 5, side2 is 6, this returns "Domino: 5|6"
    Note: | is a key on your keyboard; just draw a vertical bar"""
```

```
def canExtend(self):
    """Returns: True if no domino follows this one in a train,
    False otherwise."""
```

```
def addDomino(self, d):
    """Returns: True if domino d can be added to the train ending at
    the current domino, and False otherwise. If d can be added, this function
    sets this domino's next to be d, sets the prior of d to be this
    domino, and updates the outwardFacingSide attribute of d.

    A domino d can be added to the current domino if: 1) d has side1 or side2
    equal to this domino's outwardFacingSide value, 2) the domino on which this
    is being called is the end of a train (canExtend returns True for this domino),
    and 3) the prior of d is None.

    Precondition: d is a Domino """
```

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

4. [18 points] **Name resolution and inheritance**.

```
class A(object):
    c = 3

    def f(self):
        self.c = 5
        return 10

    def g(self):
        return self.f()

class B(A):

    def f(self):
        c = 4
        return 14


a = A()
b = B()

print a.g()
print b.g()
print a.c
print b.c
```

Put your object and class folder diagrams here (do *not* draw any function frames):

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

1. **In the space above**, draw the object and class folders that result by running this code. You should include method names and class variables in the class folders, and your object folders must have the type label in the upper right corner.

   You only need to draw two class folders, in addition to any folders for objects.

   Do *not* draw any frames for function calls.

   Remember that class folders have their tab on the right-hand side, whereas object folders have their tab on the left-hand side.

2. **In the space below**, write down what will get printed by each of the four print statements when they are executed in the order shown. (No credit for the print-statement output if you do not provide the diagrams requested above.)

5. [5 points] **Loop correctness/invariants.** Here is a specification:

```
def first_below(thelist, limit):
  """Returns: index of leftmost item in thelist that has value less than
     limit.  (Returns -1 if no such item exists in thelist.)

     Precondition: thelist is a possibly empty list of ints. limit is an int.

   Example input/output pairs:

   first_below([4, 2, 5, -2], 3) -->  1     first_below([4, 10, 5,-2], 3) -->  3
   first_below([4,  2, 5,-2], 5) -->  0     first_below([4, -2, 5, 2],-3) --> -1
   first_below([4, -2, 5, 2],-2) --> -1     first_below([],5) --> -1 """
```

Below are two attempted implementations. **Exactly one** is correct; the other one fails its test cases, and one potential issue is that it fails to initialize or maintain its invariant.

Your job: correct the wrong implementation so that it agrees with the written invariant and thereby works correctly. Do so by changing **exactly one line of (non-commented) code**: **circle the offending line and then write the correct version next to it.**

```
    i = 0
    # INVARIANT: thelist[0..i-1] are all >= limit. So i is next place to check
    while i < len(thelist) and thelist[i] >= limit:
        i = i + 1
    if i < len(thelist):
        return i
    else:
        return -1
```

```
    j = 0
    # INVARIANT: thelist[0..j] are all >= limit. So j+1 is next place to check
    while j+1 < len(thelist) and thelist[j+1] >= limit:
        j = j + 1
    if j+1 == len(thelist):
        return -1
    else:
        return j+1
```

6. [1 point] **Fill in your last name, first name, and Cornell NetID at the top of each page.**

Did you write your name and netID on each page, and re-read all specs, and check your code works against test cases?