Last Name: _____ First: _____ Netid: _____ Section _____

CS 1110 Prelim 2 November 14th, 2013

This 90-minute exam has 6 questions worth a total of 100 points. 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.

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.

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
```

You should not use while-loops on this exam. Beyond that, you may use any Python feature that you have learned about in class (if-statements, try-except, lists, for-loops, recursion and so on).

Question	Points	Score
1	2	
2	24	
3	20	
4	20	
5	20	
6	14	
Total:	100	

The Important First Question:

1. [2 points] Write your last name, first name, netid, and *lab section* at the top of each page.

Throughout this exam you will need to make use of both strings and lists. You are expected to understand how slicing works. In addition, the following functions and methods may be useful:

String Functions and Methods

Function or Method	Description
len(s)	**Returns**: number of characters in `s`; it can be 0.
s.isupper()	**Returns**: True if all letters in `s` are upper case, False otherwise.
s.upper()	**Returns**: a copy of `s` with all letters upper case. Non-letters are unaffected.
s.islower()	**Returns**: True if all letters in `s` are lower case, False otherwise.
s.lower()	**Returns**: a copy of `s` with all letters lower case. Non-letters are unaffected.
s.find(s1)	**Returns**: index of the first character of the FIRST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).
s.rfind(s1)	**Returns**: index of the first character of the LAST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).

List Functions and Methods

Function or Method	Description
len(x)	**Returns**: number of elements in list `x`; it can be 0.
y in x	**Returns**: True if `y` is in list `x`; False otherwise.
x.index(y)	**Returns**: index of the FIRST occurrence of `y` in `x` (an error occurs if `y` does not occur in x).
x.count(y)	**Returns**: the number of times `y` appears in list `x`.
x.pop()	**Returns**: The first element of `x`, removing it from the list.
x.append()	Adds `y` to the end of list `x`.
x.insert(i,y)	Inserts `y` at position `i` in list `x`. Elements after position `i` are shifted to the right.

2. [24 points] **Classes and Subclasses**

The next two pages have skeletons of two classes: `Cornellian` and `Student`. `Student` is a subclass of `Cornellian`, while `Cornellian` is only a subclass of `object`. You are to

1. Fill in the missing information in each class header.
2. Fill in the parameters of each method.
3. Implement each method according to its specification.
4. Enforce any preconditions in these methods using asserts.

There is one method completed for you: `_assignCUID`. You are to use it in the initializer of `Cornellian` to assign the initial value of attribute `_cuid`.

Pay close attention to the specification. There are no parameters beyond the ones listed. If a parameter has a default argument, it is clearly listed in the specification. Some method specifications are very explicit about the helper methods that you must use.

```
class Cornellian(          object          ):              # Fill in missing part
    """Instance represents someone at Cornell
    IMMUTABLE ATTRIBUTES:
        _cuid:  Cornell ID (not netid) [int > 0]
    MUTABLE ATTRIBUTES
        _name:  full name [str, not empty]"""
    # CLASS ATTRIBUTE TO ASSIGN IDS IN ORDER
    NEXT_CUID = 1

    # GETTERS/SETTERS HERE
    def getCUID(              self              ):        # Fill in parameters
        """Return: Cornell ID"""
        return self._cuid


    def getName(              self              ):        # Fill in parameters
        """Return: full name"""
        return self._name


    def setName(          self,     n           ):        # Fill in parameters
        """Set full name to n
        Precondition: n a nonempty string."""
        assert type(n) == str and n != ''
        self._name = n

    def _assignCUID(self):
        """Assigns _cuid to next available Cornell id"""
        self._cuid = Cornellian.NEXT_CUID
        Cornellian.NEXT_CUID = Cornellian.NEXT_CUID+1

    # INITIALIZER
    def __init__(          self,     n           ):       # Fill in parameters
        """Initializer: Make a Cornellian with name n.
        Initializer calls _assignCUID() to assign attribute _cuid
        Precondition: n a nonempty string."""
        self.setName(n)    # Handles precondition for us.
        self._assignCUID() # Creates CUID.


    def __str__(              self              ):        # Fill in parameters
        """Returns: Description of this Cornellian
        Description has form 'name [cuid]'
        Example:  'Walker White [1160491]' """
        return self._name+' ['+str(self._cuid)+']'
```

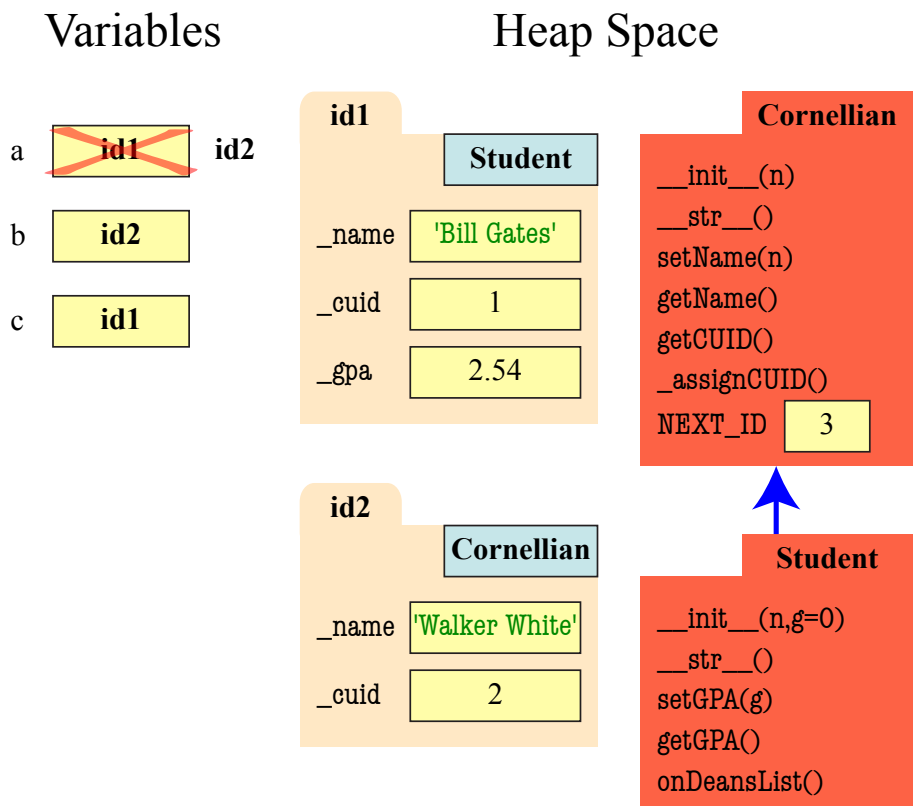
```
class Student(        Cornellian        ):                # Fill in missing part
    """Instance represents someone at Cornell
    Instance attributes are inherited from Cornellian.  Also,
    MUTABLE ATTRIBUTES
        _gpa:  grade point average [float between 0 and 4.3]"""

    # GETTERS/SETTERS HERE
    def getGPA(              self              ):        # Fill in parameters
        """Return: student GPA"""
        return self._gpa


    def setGPA(           self,     g           ):        # Fill in parameters
        """Set student GPA to g
        Precondition: g a float between 0 and 4.3."""
        assert type(g) == float and 0 <= g and g <= 4.3
        self._gpa = g


    # INITIALIZER
    def __init__(      self,     n,     g = 0      ):       # Fill in parameters
        """Initializer: Make a Student with name n, gpa g.
        Precondition: n a nonempty string, g a float between 0 and 4.3.
        g is 0 by default."""
        # Call initializer for superclass
        Cornellian.__init__(self,n)
        self.setGPA(g) # Handles precondition


    def onDeansList(             self             ):        # Fill in parameters
        """Return: True if GPA >= 3.5; False otherwise"""
        return self._gpa >= 3.5


    def __str__(              self              ):        # Fill in parameters
        """Returns: Description of this Student
        Description is same as Cornellian, plus '.  Dean\'s List'
        if the student is on the Dean's List.
        Implementation must use __str__ in Cornellian as helper.
        Example:  'Bob Roberts [234781]'
                  'Emma Towns [492886].  Dean\'s List' """
        prefix = Cornellian.__str__(self)
        if self.onDeansList():
            return prefix+'.  Dean\'s List'
        return prefix # Not on Dean's List
```

3. [20 points total] **Folders and Name Resolution**

   (a) [12 points] Suppose you were to execute the following commands in the interactive shell.

```
>>> a = Student('Bill Gates', 2.54)
>>> b = Cornellian('Walker White')
>>> c = a
>>> a = b
```

In the space below, create two columns: one for global space and another for heap space. Clearly show what is created in each, drawing folders for objects and classes, and boxes for variables. If the value of a variable or attribute changes, cross the old one out and write the new value out beside it. **You do not need to draw the folder of the `object` class.**



(b) [8 points] Consider the two (undocumented) classes below.

```
class A(object):
    x = 3
    y = 5

    def __init__(self,y):
        self.y = y

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

    def g(self):
        return self.x+self.y
```

```
class B(A):
    y = 4
    z = 10
    def __init__(self,x,y):
        self.x = x
        self.y = y

    def g(self):
        return self.x+self.z

    def h(self):
        return 42
```

Execute the following in the interactive shell:

```
>>> a = A(1)
>>> b = B(7,3)
```

The following expressions are either valid or the produce an error. Indicate which is which. If the expression is valid, tell us what it evaluates to.

i. `a.y` 1

ii. `a.z` Error

iii. `b.x` 7 (Instance Attribute)

iv. `B.x` 3 (Class attribute)

v. `a.f()` 4

vi. `b.f()` 17 (Inherited from A)

vii. `a.h()` Error

viii. `A.g(b)` 10 (Uses method in A)

4. [20 points] **Iteration**.

Use for-loops to implement both the functions specified below. You may wish to refer to the list methods on the first page. You do not need to enforce the function preconditions.

```
def min(thelist):
    """Return: the least element of thelist.
    Example: min([3,0,-1,5]) is -1
    Precondition: thelist is a non-empty list of numbers"""

    # As list not empty, first element is place to start
    minimum = thelist[0]
    for x in thelist:
        if x < minimum:
            minimum = x


    return minimum
```

```
def insert(thelist,x):
    """MODIFIES thelist, putting x into the correct, ordered position.
    List thelist is sorted (e.g. the elements are in order).  The function
    puts x into thelist at the right position so it is still ordered.  If x is
    already in thelist, this function inserts x before the first occurrence.
    You may NOT use the method sort().
    Example: if a = [0,2,4,5], insert(a,3) makes a into [0,2,3,4,5]
             if a = [1,2,3,7], insert(a,-1) turns a into [-1,1,2,3,7]
             if a = [1,2,2,7], insert(a,2) turns a into [1,2,2,2,7]
             if a = [], insert(a,4) turns a into [4]
    Precondition: thelist is a sorted (possibly empty) list of numbers.
    x is a number (int or float)."""
    # Find position of first element greater than x
    pos = 0
    for k in range(len(thelist)):
        if thelist[k] < x:
            pos = pos+1


    # Shift greater elements to the right.
    thelist.insert(pos,x)
```

5. [20 points] **Recursion**.

Use recursion to implement both the functions specified below; **do not use for-loops or while-loops**. You may wish to refer to the string methods on the first page. You do not need to enforce the function preconditions.

```
def swapcase(s):
    """Return: a copy of s where letter case is swapped.

    Upper case letters are replaced by lower case letters.
    Lower case letters are replaced with upper case letters.

    Example: swapcase('Hello World!')  is 'hELLO wORLD!'

    Precondition: s is a string (possibly empty)."""
    if s == '':
        return ''

    if s[0].isupper():
        a = s[0].lower()
    else:  # Lower or not a letter
        a = s[0].upper()

    return a+swapcase(s[1:])
```

```
def split(s,delimiter):
    """Return: list of substrings of s separated by the delimiter.

    This function breaks up s into several substrings according to
    the delimiter (a single character separator like ',' or ':').

    Example: split('ab,c ,, d', ',') is ['ab', 'c ', '', ' d']
             split('ab::cd ::  ef', '::')  is ['ab', 'cd ', ' ef']
             split('ab::cd ::  ef', ',') is ['ab::cd ::  ef']
             split('', ',') is ['']

    Precondition: s is a string (possibly empty).
    delimiter is a nonempty string"""
    # Need to break up at delimiters, not first character
    pos = s.find(delimiter)
    if pos == -1:
        return [s]

    front = [ s[:pos] ]
    back = s[pos+len(delimiter):]  # Do not include delimiter
    return front+split(back,delimiter)
```

6. [14 points total] **Poutporri**.

(a) [6 points] Consider the following assignment statements.

```
>>> a = [[1,2,3],[4,5,6],[7,8,9]]
>>> b = a[1:]
>>> b[0] = [10,11]
>>> b[1][0] = 99
```

What are the values `a` and `b` after all these assignments? Explain your answer.

`a` is `[[1,2,3],[4,5,6],[99,8,9]]`

`b` is `[[10,11],[99,8,9]]`

The second line (the slice) puts `[[4,5,6],[99,8,9]]`. The third line replaces the first element of this 2D list with another list. The last line goes inside of the list `[7,8,9]` and changes the first element. Because of how slicing works, the folder for this list is shared by both `a` and `b`, so changes to one affect another.

(b) [4 points] Rewrite the command

```
assert type(x) == int, str(x)+' is not an int'
```

so that the user gets a `TypeError` instead of an `AssertionError`
(You may need more than one line).

```
if type(x) != int:
    msg = str(x)+' is not an int'
    raise TypeError(msg)
```

(c) [4 points] Explain the difference between an invariant and a precondition.
How do they relate to one another?

Okay, wow; this question was a mistake. We intended to ask a question about *class* invariants and *function* preconditions. However, we did not realize that the exam was on the same day as the lecture on *loop* invariants and *general* preconditions. In the end, we accepted either answer, as long it was correct.
A function precondition is a property that must be true of a function or method parameter for the function to work properly. An class invariant is a property that must be true of an object attribute that must be true for the object to work properly.
For the relationship, it is not enough to note that these are similar. A similarity is not the same thing as a relationship. The relationship is that the preconditions of the methods in a class are written so that they preserve invariants when called.