Last Name: ____________ First: ____________ Netid: ____________

CS 1110 Prelim 2 December 5th, 2024

This 90-minute exam has 5 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, look at any reference material, or 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
```

Throughout this exam, you may use any Python feature that you have learned about in class *other than generators*.

Question	Points	Score
1	2	
2	22	
3	28	
4	24	
5	24	
Total:	100	

The Important First Question:

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

Last Name: ______________	First: ______________	Netid: ______________

Reference Sheet

String Operations

Operation	**Description**
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`a in s`	**Returns**: True if the substring `a` is in `s`; False otherwise.
`s.find(s1)`	**Returns**: Index of FIRST occurrence of `s1` in `s` (-1 if `s1` is not in s).
`s.count(s1)`	**Returns**: Number of (non-overlapping) occurrences of `s1` in `s`.
`s.lower()`	**Returns**: A copy of`s` with all letters converted to lower case.
`s.upper()`	**Returns**: A copy of`s` with all letters converted to upper case.
`s.islower()`	**Returns**: True if `s` is *has at least one letter* and all letters are lower case; it returns False otherwise (e.g. `'a123'` is True but `'123'` is False).
`s.isupper()`	**Returns**: True if `s` is *has at least one letter* and all letters are uppper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.isalpha()`	**Returns**: True if `s` is *not empty* and its elements are all letters; it returns False otherwise.
`s.isdigit()`	**Returns**: True if `s` is *not empty* and its elements are all digits; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or digits; it returns False otherwise.

List Operations

Operation	**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.
`del x[k]`	Deletes the element of the list at position k.
`x.index(y)`	**Returns**: Index of FIRST occurrence of `y` in `x` (error if `y` is not in `x`).
`x.count(y)`	**Returns**: the number of times `y` appears in list `x`.
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in `x`. Elements after `i` are shifted to the right.
`x.remove(y)`	Removes first item from the list equal to `y`. (error if `y` is not in `x`).

Dictionary Operations

Function or Method	**Description**
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key `d`; False otherwise.
`d[k] = v`	Assigns value `v` to the key `k` in `d`.
`del d[k]`	Deletes the key `k` (and its value) from the dictionary `d`.
`d.clear()`	Removes all keys (and values) from the dictionary `d`.

2. [22 points total] **Recursion**.

   Use recursion to implement the following functions according to their specification. Solutions using loops (for or while) will receive no credit. You **do not** need to enforce the preconditions.

   (a) [10 points]

```
def triplicate(s):
    """Returns a copy of s with each letter tripled

    Example: triplicate('abra') returns 'aaabbbrrraaa'

    Precond: s is a (possibly empty) string of lowercase letters"""
```

   (b) [12 points]

```
def prefix(s):
    """Returns the prefix (identical characters at the start) length of s

    Example: prefix('abc') returns 1 as the prefix is 'a'
             prefix('xxxxxxyzx') returns 6 as the prefix is 'xxxxxx'
             prefix('') returns 0 as the string is empty

    Precond: s is a (possibly empty) string of lowercase letters"""
```

3. [28 points] **Classes and Subclasses**

In this problem, you will create `Cornellian`, a class representing a person working or studying at Cornell. You will also create the subclass `Student`, which is a `Cornellian` with a GPA.

Each `Cornellian` must have a unique Cornell id. The class attribute `NEXTID` is used to automatically assign Cornell ids (a person cannot pick their id). When a `Cornellian` object is created, it gets `NEXTID` as its id and then `NEXTID` is incremented by one.

In summary, the attributes of these two classes are as follows:

`Cornellian`

Attribute	**Invariant**	**Category**
`NEXTID`	int > 0	Class attribute
`_cuid`	int > 0	**Immutable** instance attribute
`_name`	nonempty string	**Mutable** instance attribute

`Student`

Attribute	**Invariant**	**Category**
`_gpa`	float in 0.0 to 4.3	**Mutable** instance attribute

On the next three pages, you are to do the following:

1. Fill in the missing information in each class header.
2. Add getters and setters as appropriate for the instance attributes
3. Fill in the parameters of each method (beyond the getters and setters).
4. Implement each method according to the specification.
5. Enforce any preconditions in these methods using asserts

When enforcing type-based preconditions, you should use `isinstance` instead of `type`.

We have not added headers for any of the getters and setters. You are to write (and name) these yourself. **You are not expected to write specifications for the getters and setters**. For the other methods, pay attention to the provided specifications. The only parameters are those indicated by the preconditions.

Important: `Student` is not allowed to access any hidden attributes of `Cornellian`. As an additional restriction, `Student` may not access any getters and setters in `Cornellian`.

Last Name: ______________________ First: ______________________ Netid: ______________________

```
class Cornellian____________________________________                # Fill in missing part
   """A class representing a student at Cornell"""

   Attribute NEXTID: A CLASS ATTRIBUTE that is an int > 0.  Its initial value is 1."""
   # Attribute _cuid: The Cornell id. An int > 0 (IMMUTABLE)
   # Attribute _name: The full name of the person. A non-empty string (MUTABLE)

   # INITIALIZE THE CLASS ATTRIBUTE



   # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.



   # THE INTIALIZER
   def __init______________________________________                 # Fill in missing part
      """Initializes a Cornellian with name n.

      The initializer assigns the NEXTID value as the Cornell ID.
      It it also increments the class attribute NEXTID by one.

      Precondition: n is a nonempty string"""
```

```
    # Class Cornellian (CONTINUED).
    def __str____________________________________           # Fill in missing part
        """Returns a description of this Cornellian

        The description has form 'name [cuid]'
        Example:  'Walker White [1160491]' """




    def __eq__________________________________              # Fill in missing part
        """Returns True other is a Cornellian object equal to this one.

        Two Cornellians are equal if they have the same Cornell ID EVEN
        THOUGH their names may be different (people can change names).

        Precondition: NONE (other can be anything)"""




class Student__________________________________             # Fill in missing part
    """A class representing a student at Cornell"""
    # Attribute _gpa: Student's grade point average. Float between 0.0 and 4.3 (MUTABLE)

    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    # Class Student (CONTINUED).
    def __init______________________________________                  # Fill in missing part
        """Initializes a Student with name n and gpa g.

        Precondition: n is a nonempty string
        Precondition: g is a float between 0.0 and 4.3 (DEFAULT value 0.0)"""










    def onDeansList____________________________________               # Fill in missing part
        """Return True if the student's GPA >= 3.5; False otherwise"""







    def __str_______________________________________                  # Fill in missing part
        """Returns a description of this Student

        The description is the same as Cornellian, except that it adds
        "Dean's List" (after a comma) if the Student is on the Dean's List.

        Examples: 'Bob Roberts [234781]' or "Emma Towns [492886], Dean's List" """
```

4. [24 points total] **Iteration**. Implement the functions on the next two pages, according to their specification, using loops (either for-loops or while loops). While we do not tell you which kind of loop to use, we have given you rules to follow in class. Use those to guide your choice of loop. For both of these questions, you **do not** need to enforce preconditions.

   (a) [12 points]

```
def merge(dict1,dict2):
    """Returns a new dictionary merging (joining keys) dict1 and dict2.

    If a key appears in only one of dict1 or dict2, the value is the value
    from that dictionary. If it is in both, the value is the sum of values.

    Examples:
        merge({'a':1,'b':2},{'b':3,'c':4}) returns {'a':1,'b':5,'c':4}
        merge({'a':1,'b':2},{'c':3,'d':4}) returns {'a':1,'b':2,'c':3,'d':4}
        merge({'a':1,'b':2},{}) returns {'a':1,'b':2}
        merge({},{'b':3,'c':4}) returns {'b':3,'c':4}

    Precond: dict1, dict2 are (possibly empty) dictionaries with int values"""
```

Last Name: ____________________ First: ____________________ Netid: ____________________

(b) [12 points] For the problem below you are only allowed to remove or delete elements from `nlist`. You are ***not allowed*** to append or extend `nlist` (so you cannot clear the list and then add the elements back one-by-one). **However**, you are allowed to create any additional lists that you want and append to those. You may find a second list helpful for keeping track of duplicates.

```
def remdups(nlist):
    """MODIFIES nlist to remove all duplicates.

    A value is a duplicate if it appears twice in the list. Duplicates do not
    have to be adjacent to each other. If a number is duplicated, the
    first instance is preserved and all later versions are removed.

    Examples:
        If a = [1,2,1,3,2,4], remdups(a) modifies a to [1,2,3,4]
        If a = [1,2,1,2,1], remdups(a) modifies a to [1,2]
        If a = [1,2,3,4], remdups(a) leaves a unchanged
        If a = [], remdups(a) leaves a unchanged

    Precond: nlist is a (possibly empty) list of ints"""
```

Last Name: ______________ First: ______________ Netid: ______________

5. [24 points total] **Call Frames and Name Resolution**

Consider the two (undocumented) classes below, together with their line numbers.

```
1  class A(object):
2      x = [1]
3      y = 5
4
5      def __init__(self,x):
6          self.x = self.x + A.x
7          self.y = x
8
9      def f(self,y):
10         return 1-self.g(y)
11
12     def g(self,x):
13         return x*self.y
14
```

```
15 class B(A):
16     x = [2,3]
17     z = 3
18
19     def __init__(self,x):
20         super().__init__(x+1)
21         z = x+2
22
23     def f(self,y):
24         return super().f(y)+2
25
26     def g(self,x):
27         self.x = x
28         return 3+self.y
```

(a) [9 points] Draw the *entire contents* of the heap for these two class definitions. That means you should draw the class folders, but also any object folders that might be associated with the class definition.

(b) [15 points] On the next two pages, diagram the call

```
>>> x = B(5)
```

You will need **eight diagrams**. Draw the call stack, global space and heap space. If the contents of any space are unchanged between diagrams, you may write *unchanged*. While you will need to use the folders from part (a) in your answer, **you do not need to draw any of those folders again**. Your heap should only contain the folders that are newly creted by this part. However, make sure that any new folders do not share ids with folders in part (a).

When diagramming a constructor, you should follow the rules from Assignment 5. Remember that `__init__` is a helper to a constructor but it is not the same as the constructor. In particular, there is an important **first step** before you create the call frame.

Last Name:		First:		Netid:	

Call Stack	Global Space	The Heap
①		
②		
③		
④		

Last Name: ______________ First: ______________ Netid: ______________

Call Frames	Global Space	The Heap
⑤		
⑥		
⑦		
⑧		