Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 2 November 17th, 2022

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
```

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	23	
3	22	
4	27	
5	26	
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.
`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`.

Last Name: _____ First: _____ Netid: _____

2. [23 points total] **Iteration**.

   Implement the functions below according to their specification using for-loops. You **do not** need to enforce preconditions. But pay attention to all instructions.

   (a) [9 points]

```
def rotleft(nums):
    ""MODIFIES the given list nums by rotating each element one spot left

    Ex: If a = [1,2,3,4], rotleft(a) modifies a to [2,3,4,1] (first element goes last)

    Precond: nums is a nonempty list of numbers"""
```

   (b) [14 points]

```
def reflect(table):
    """Returns a COPY of table, reflected horizontally (each row is reversed)

    Ex: reflect([[1,2,3],[4,5,6],[7,8,9]]) returns [[3,2,1],[6,5,4],[9,8,7]]

    Precond: table is a nonempty 2d list of numbers (int or float)"""
    # You may NOT use the reverse method in this function
```

Last Name: _____ First: _____ Netid: _____

3. [22 points total] **Recursion**.

   Use recursion to implement the following functions. **Solutions using loops will receive no credit.** You **do not** need to enforce the preconditions. But pay attention to all instructions.

   (a) [10 points]

```
def x_out(s):
    """Returns a copy of s with every lower-case letter replaced by 'x'

    Ex: x_out('Hello World') returns 'Hxxxx Wxxxx'

    Precond: s is a string"""
```

   (b) [12 points]

```
def sumfold(nums):
    """Returns the accumulated sum of nums (as a new list)

    Each element at position i in the new list becomes the sum up to and including
    that position from the list nums. If nums is empty, so is the list returned.
    Ex: sumfold([0,1,2,3,4]) returns [0,1,3,6,10].
        sumfold([3]) returns the COPY [3]

    Precond: nums is a (nonempty) list of integers"""
```

Last Name: _____ First: _____ Netid: _____

4. [27 points total] **Classes and Subclasses**

A server is a machine that supports multiple users. Each user has to have an account with a username and password. User names must be unique, but account holders can often set their real name as well (which need not be unique). Indeed, this is how netids and student names work at Cornell.

Some of these accounts have special privileges, and are called *administrators*. Administrators are exactly like normal accounts, except that they have a security level that indicates how much power they have over the system.

In this question, you will create the class `Account` and its subclass `Administrator`. The attributes for these classes are as follows:

`Account`

Attribute	**Invariant**	**Category**
`IN_USE`	list of all user names (str)	Class attribute
`_username`	a nonempty string of only letters and digits	**Immutable** instance attribute
`_realname`	a nonempty string	**Mutable** instance attribute
`_password`	a string with at least 8 characters of only letters and digits; it cannot be only letters or only digits	**Mutable** instance attribute

`Administrator` (in addition to those inherited)

Attribute	**Invariant**	**Category**
`_level`	an int in the range 1..9 (inclusive)	**Immutable** instance attribute

Instructions: On the next four pages, you are to do the following:

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

We have not added headers for the getters and setters. You are to write these from scratch. However, **you are not expected to write specifications for them**. For the other methods, pay attention to the provided specifications. The only parameters are those in the preconditions.

You should enforce preconditions with `assert` unless you are given a specific error to use instead. Type-based preconditions should all be managed with `isinstance` and not the function `type`.

As one last restriction, the class `Adminstrator` **may not** use any attribute or getter/setter inherited from `Account`. It may only use `super()` to access overridden methods.

Last Name: _____ First: _____ Netid: _____

(a) [18 points] The class `Account`

```
class Account_____________________________________          # Fill in missing part
    """A class representing a user account on a server.

    Attribute IN_USE: A CLASS ATTRIBUTE list of all user names on the server.
    This list starts off empty, as there are no accounts to start with."""
    # IMMUTABLE ATTRIBUTES
    # ATTRIBUTE _username: Unique account name. A nonempty string of letters and digits.
    # MUTABLE ATTRIBUTES
    # ATTRIBUTE _realname: Account holder's name. A nonempty string.
    # ATTRIBUTE _password: The account password. A string of size >= 8.
    # _password has only letters and digits but it cannot be only letters or only digits.

    # CLASS ATTRIBUTE. NO GETTERS OR SETTERS.



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

```
# Class Account (CONTINUED).

def __init__(______________________________________)          # Fill in missing part
    """Initializes a new account with given user name and password

    The parameter name is used to set both the user name and the real name.
    No account can share the same user name as another. On creation, the
    user name is added to the class attribute IN_USE.

    Precondition: name is a nonempty string of letter/digits, and not already in use.
    Precondition: pass is a string with at least 8 characters. It contains only letters
    and digits but cannot be all letters or all digits"""

def __str__(______________________________________)          # Fill in missing part
    """Returns a string representation of the user account.

    The format is '<username> (<realname>)' UNLESS username and realname are
    the same. In that case, the format is just '<username>'.

    Example: returns 'wmw2' if both username and realname are 'wmw2', but
    'wmw2 (Walker)' if username is 'wmw2' and real name is 'Walker'."""

def __eq__(______________________________________)          # Fill in missing part
    """Returns True if self and other are equal. False otherwise.

    An account object is equal to this one (self) if it has the same user name.
    Objects that are not instances of Account cannot be equal to this one.

    Precondition: NONE. other can be ANYTHING """
```

(b) [9 points] The class `Administrator`.

```
class Administrator_____                                    # Fill in missing part
    """A class representing  an administrator account."""
    # This page MAY NOT use any attributes or getters/setters from Account
    # IMMUTABLE ATTRIBUTES
    # ATTRIBUTE _level: The access level. An int in the range 1..9 (inclusive)

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









    def __init__(_____                                      # Fill in missing part
        """Initializes a new administrator with given level and password

        The parameter name is allowed to be None. If so, the user name
        (and real name) are set to 'adminX' where X is the level.

        Precondition: level is an int in the range 1..9 (inclusive)
        Precondition: pass is a string with same restrictions as in Account
        Precondition: name is either None or a nonempty string of letters/digits
        (OPTIONAL PARAMETER; name is None by default)"""









    def __str__(_____                                       # Fill in missing part
        """Returns a string representation of this administrator

        Format is '<username>: Level <level>' or '<username> (<realname>): Level <level>'
        depending on whether or not username and realname are the same.

        Example: 'wmw2 (Walker): Level 9' or 'abc3: Level 1' """
```

Last Name: _____ First: _____ Netid: _____

5. [26 points total] **Call Frames and Name Resolution**

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

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

```
13 class B(A):
14     x = 5
15
16     def __init__(self,x,y):
17         super().__init__(x)
18         self.y = y
19
20     def f(self,n):
21         if n == 0:
22             return 3*self.y
23         return self.x+self.g(n)
24
```

(a) [5 points] Draw the class folders in the heap for these two classes.

(b) [21 points] Assume that you have **already created** the following object, which is shown in the diagram on the next page.

```
>>> p = B(3,-1)
```

On the next two pages, diagram the evolution of the call

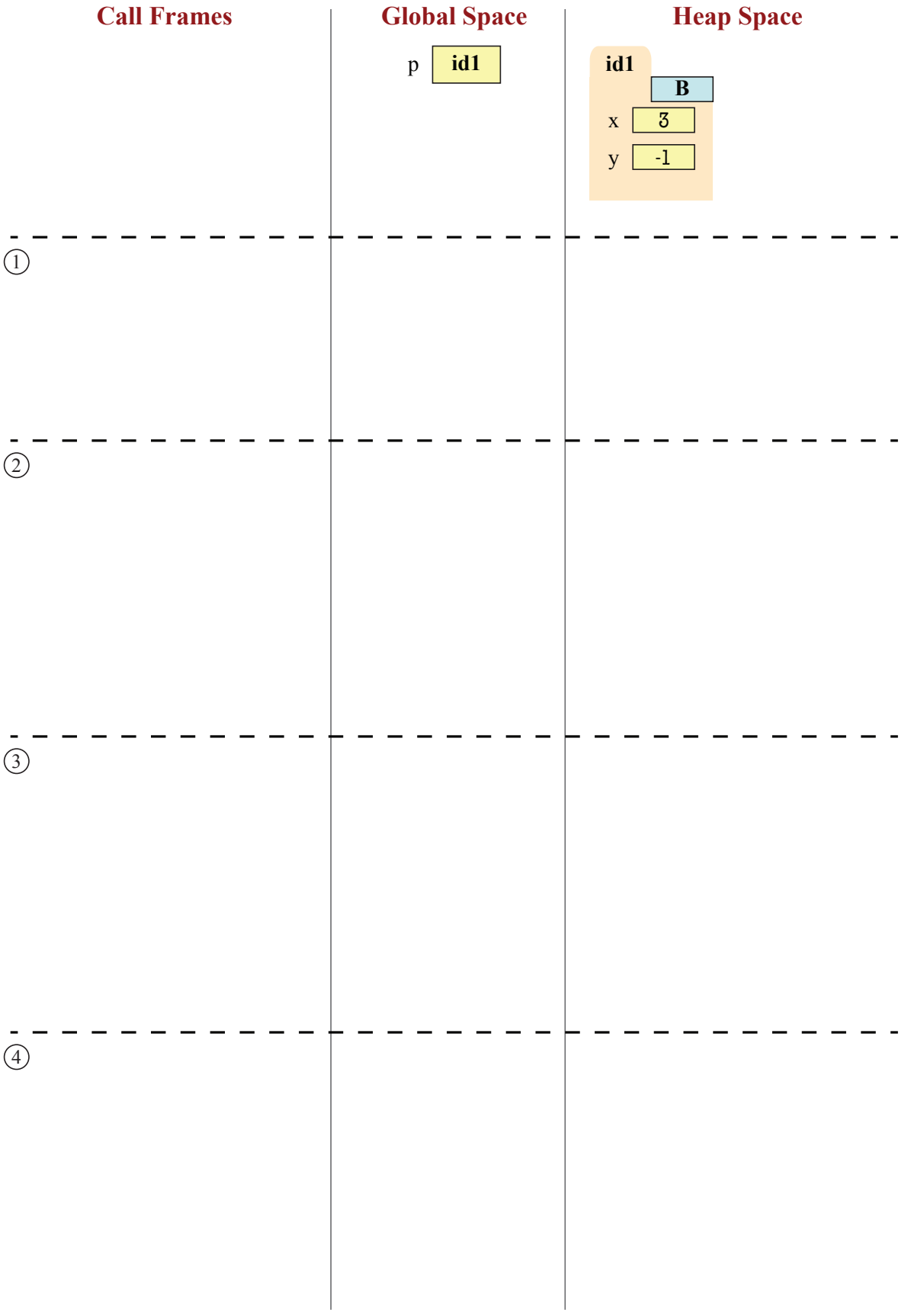
```
>>> z = p.f(1)
```

Diagram the state of the *entire call stack* for the method call when it starts, for each line executed, and when the frame is erased. If any other methods are called, you should do this for them as well (at the appropriate time). This will require a total of **nine diagrams**, excluding the initial (pre-call) diagram.

You should draw also the state of global space and the heap at each step. You are allowed to write "unchanged" if no changes were made to either global or the heap.

Hint: Do *not* diagram the constructor call. There should be a call frame in your very first diagram.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

Call Frames	Global Space	Heap Space
⑤		
⑥		
⑦		
⑧		
⑨		