Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 2 December 4th, 2025

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, as long as you keep everything together when you turn it in.

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	24	
3	22	
4	29	
5	23	
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.
`s.isidentifier()`	**Returns**: True if `s` is a valid variable identifier (e.g. starts with a letter or underscore, only has letters, digits and underscores).

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`.

Last Name: _____ First: _____ Netid: _____

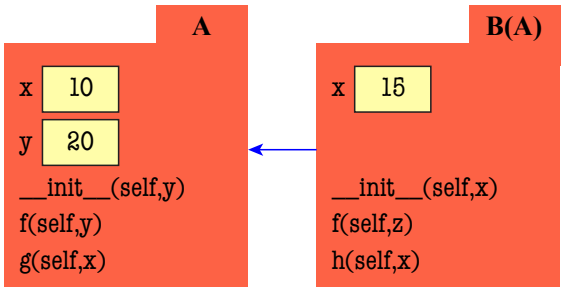
2. [24 points total] **Call Frames and Name Resolution**

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

```
1  class A(object):
2      x = 10
3      y = 20
4
5      def __init__(self,y):
6          self.z = y + self.y
7          self.f(y)
8
9      def f(self,y):
10         return self.g(y)+1
11
12     def g(self,x):
13         return 2*x
14
```

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

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



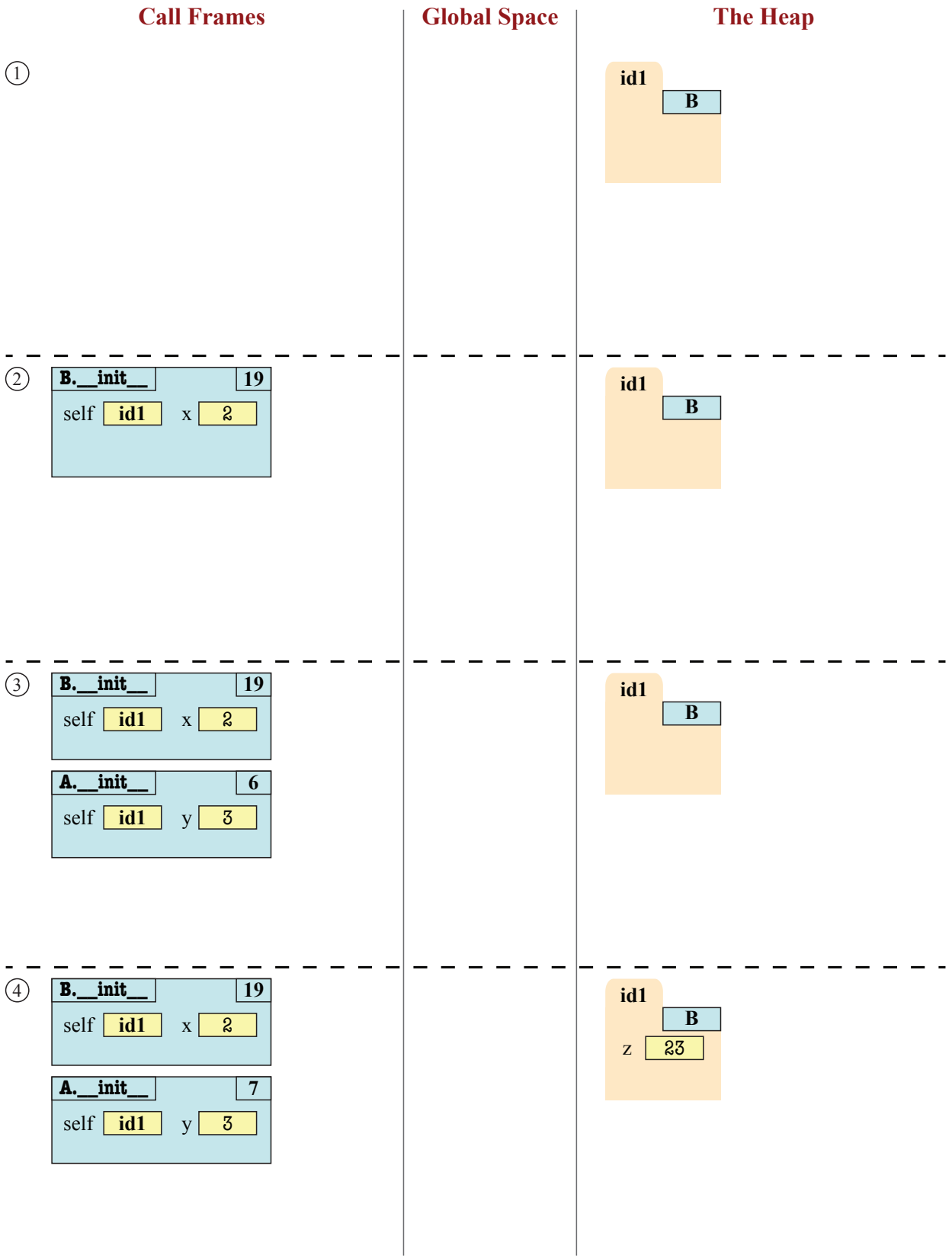
(b) [19 points] On the next three pages, diagram the call

```
>>> b = B(2)
```

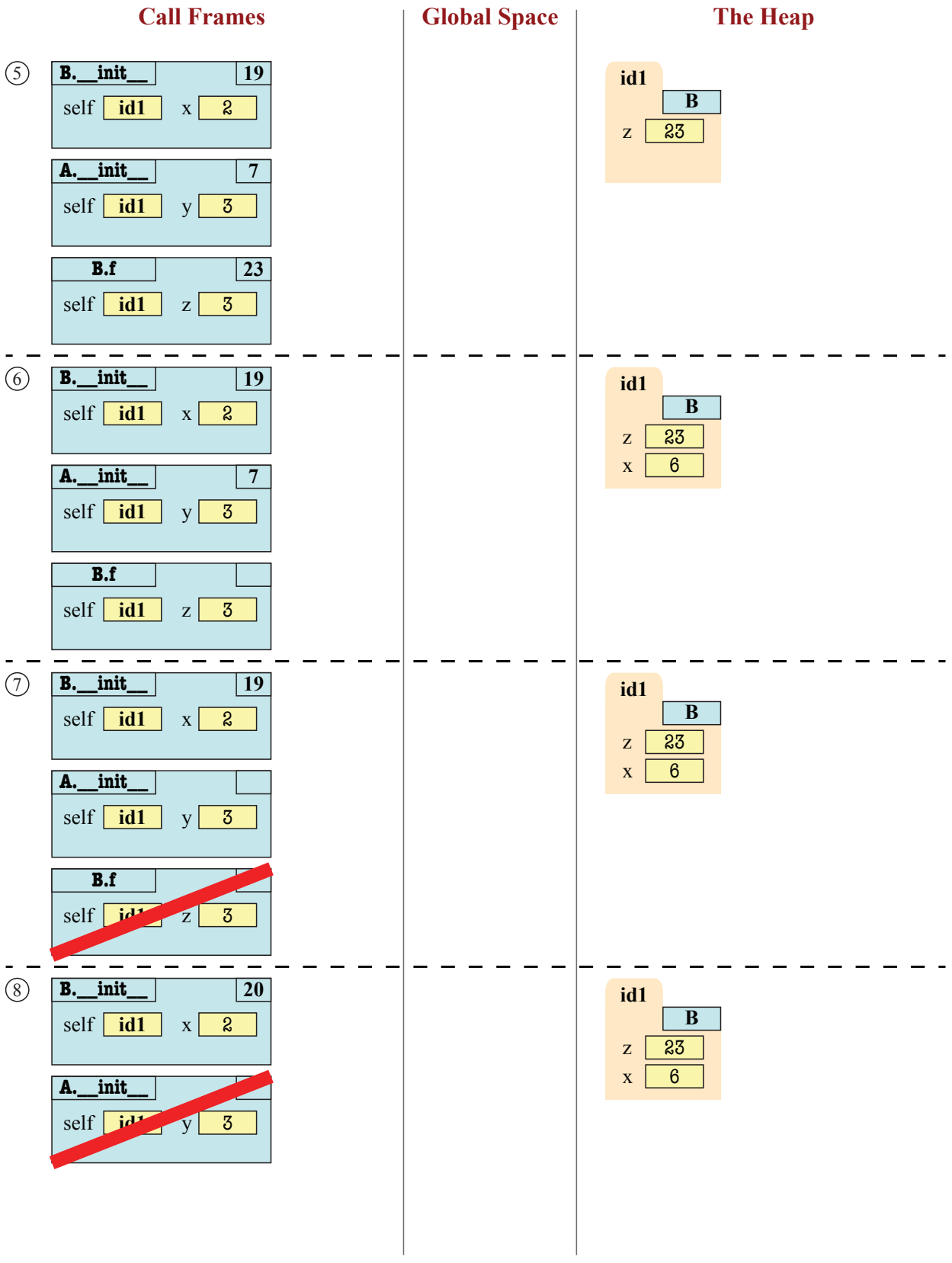
You will need **ten diagrams**. Draw the call stack, global space and the heap. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You **you do not need to draw the folders from part (a)**. Your heap should only contain the folders that are newly created by this part.

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.

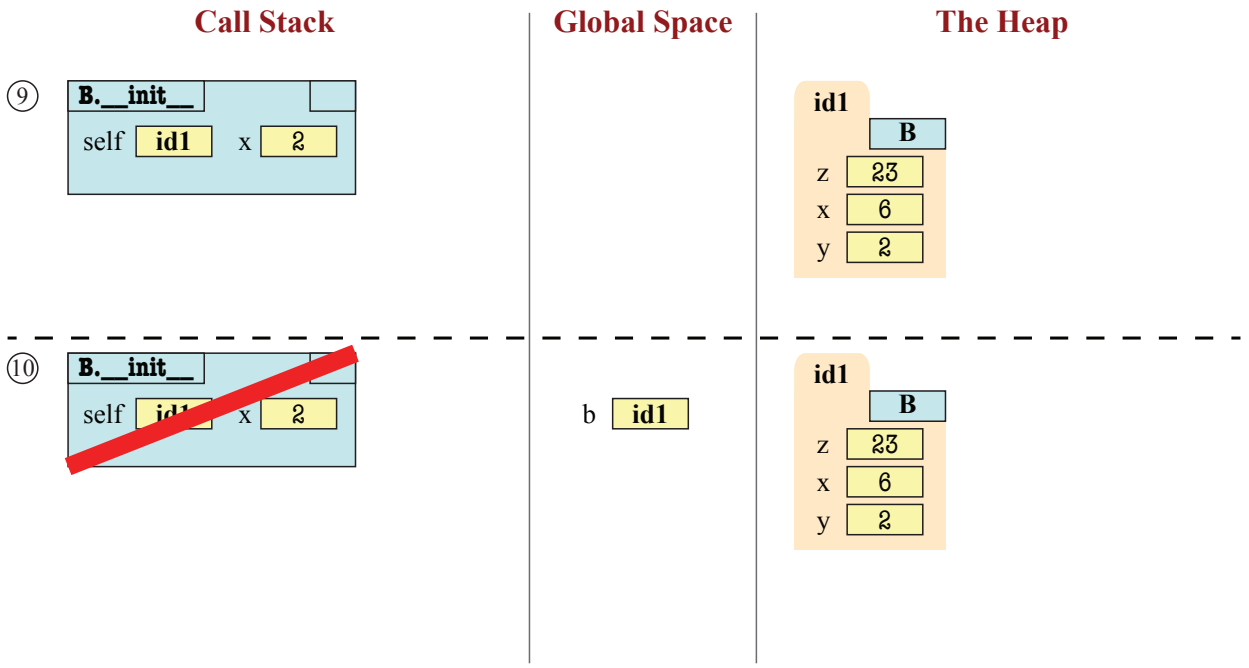
Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



3. [22 points total] **Iteration**. Implement the functions on the next two pages, according to their specification, using loops (either for-loops or while loops). For both of these questions, you **do not** need to enforce preconditions.

   (a) [10 points]

```
def tablify(alist):
    """MODIFIES alist to make it into a table.

    This function takes alist, which can be a ragged 2d-list. It converts alist
    into a table with a number of columns equal to the length of the shortest row.
    Any row with more columns is sliced to remove the extra columns at the end.
    Example: if a = [[1,2,3],[4,5],[6,7,8,9]] then tablify(a) changes a into the
    table [[1,2],[4,5],[6,7]]

    Precond: alist is a nonempty 2d list"""
    # HINT: Compute the number of columns first
    cols = len(alist[0])
    for row in alist:
        if len(row) < cols:
            cols = len(row)

    # Reduce all the rows
    for pos in range(len(alist)):
        alist[pos] = alist[pos][:cols]

    # Procedure: no return value
```

(b) [12 points]

```
def zipit(alist,blist):
    """Returns a dictionary mapping alist to blist.

    This function takes each element in alist and makes it a key in the returned
    dictionary. Values are taken from blist and are assigned in the order that
    they appear in blist. If a key appears more than once in alist, it is skipped
    over and its value is assigned to the next key. If blist does not have enough
    values for all the keys, later keys have a value of None.

    Examples:
        zipit(['a','b'],[1,2]) returns { 'a':1, 'b': 2 }
        zipit(['a','b'],[1,2,3]) returns { 'a':1, 'b': 2 }
        zipit(['a','b','c'],[1,2]) returns  { 'a':1, 'b': 2, 'c':None }
        zipit(['a','b','a','b','c'],[1,2,3]) returns { 'a':1, 'b': 2, 'c':3 }

    Precond: alist and blist are nonempty lists"""
    # HINT: Only loop over one of the two lists.
    # Use another variable for position in the second list

    # Create the accumulator
    result = {}
    # Track the position in blist
    pos = 0

    for key in alist:
        # Get the value to assign
        if pos < len(blist):
            value = blist[pos]
        else:
            value = None

        # Match up key if new
        if not key in result:
            result[key] = value
            # Increment the blist position
            pos = pos+1

    return result
```

4. [29 points] **Classes and Subclasses**

In this problem, you will create `Radditor`, a class for a user on the social media platform Raddit. You will also create the subclass `Moderator`, which is a moderator for a subraddit.

Each `Radditor` must have a unique user name. The class attribute `TAKEN` is used to keep track of which user names are taken, and therefore not available. In addition, Raddit is a small social media platform, and so each subraddit can only have one moderator. The class attribute `MODS` is used to track the moderators of each subraddit.

Note that user names and subraddits must be **identifiers**. An identifier is a string that follows the rules of variables. That is, they can only have letters, digits, and underscores, and they cannot start with a digit nor can they be empty.

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

`Radditor`

Attribute	**Invariant**	**Category**
`TAKEN`	list of strings	Class attribute
`_unname`	identifier string	**Immutable** instance attribute
`_display`	nonempty string or None	**Mutable** instance attribute

`Moderator`

Attribute	**Invariant**	**Category**
`MODS`	dict with strings for keys and values	Class attribute
`_subraddit`	identifier string	**Immutable** 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 (unless otherwise directed)

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

```
class Radditor(object):                                    # Fill in missing part
    """A class representing a user on Raddit

    Attribute TAKEN: A CLASS ATTRIBUTE that is a list of usernames; initially empty."""
    # Attribute _uname:   The user name. An idenfifier string (IMMUTABLE)
    # Attribute _display: The display name. A non-empty string or None (MUTABLE)
    # INITIALIZE THE CLASS ATTRIBUTE
    TAKEN = []

    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getUserName(self):
        """
        Returns the user name
        """
        return self._uname

    def getDisplay(self):
        """
        Returns the display name
        """
        return self._display

    def setDisplay(self,value):
        """
        Sets the display name
        Precondition: value a nonempty string
        """
        assert value is None or isinstance(value, str) and value != ''
        self._display = value

    def __init__(self,    uname,    display=None):              # Fill in missing part
        """Initializes a Radditor with the given user name and display name.

        No object can be created if uname is in TAKEN. That will cause a ValueError with
        message "uname is already taken" (replace uname with the actual value). If the
        object is successfully created, the user name is added to TAKEN.

        Precondition: uname is an identifier string
        Precondition: display is a nonempty string or None (optional; default None)
        Additional precondition: uname not in TAKEN (enforced by ValueError, not assert)"""

        assert isinstance(uname,str) and uname != ''
        assert uname.isidentifier()
        self.setDisplay(display)
        if uname in Radditor.TAKEN:
            raise ValueError(uname+' is already taken')

        self._uname = uname
        Radditor.TAKEN.append(uname)
```

```
    # Class Radditor (CONTINUED).
    def __str__(self):                                          # Fill in missing part
        """Returns a description of this Radditor

        If the account has a display name, the form is 'display (uname)'.
        If the display name is None, the form is just 'uname'."""

        if self._display is None:
            return self._uname
        else:
            return self._display + ' (' + self._uname + ')'


    def __eq__(self,   other):                                  # Fill in missing part
        """Returns Truem if other is a Radditor object with the same uname.

        Precondition: NONE (other can be anything)"""
        if isinstance(other,Radditor):
            return other._uname == self._uname

        return False


class Moderator(Radditor):                                      # Fill in missing part
    """A class representing a moderator account on Raddit

    Attribute MODS: A CLASS ATTRIBUTE that is a dictionary with subraddits as keys and
    user names as values; initially empty"""
    # Attribute _subraddit: The moderator subraddit. An identifier string (IMMUTABLE)
    # INITIALIZE THE CLASS ATTRIBUTE
    MODS = {}

    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getSubraddit(self):
        """
        Returns the moderators subraddit
        """
        return self._subraddit


    #  Moved before __init__ to make space on exam
    def __str__(self):                                          # Fill in missing part
        """Returns a description of this Moderator

        The description is the same as Radditor, except that it adds '[mod subraddit]'.
        Examples: 'Alex (alex42) [mod Cornhell]' or 'wmw [mod CS1110]' """
        result = super().__str__()
        return result + '[mod ' + self._subraddit + ']'
```

Last Name: _____ First: _____ Netid: _____

```
# Class Moderator (CONTINUED).
def __init__(self,   uname,   subraddit,   display=None):       # Fill in missing part
    """Initializes a Moderator with the given uname, subraddit, and display name.

    The subraddit cannot be a key in MODS. If it is, there will be a PermissionError
    with the message "subraddit has a moderator" (replace subraddit with the value). If
    the object is successfully created, subraddit is added to MODS with value uname.

    Precondition: uname, display are same as for Radditor
    Precondition: subraddit is an identifier string
    Additional precondition: subraddit not in MODS (enforced by PermissionError)"""

    assert isinstance(subraddit,str) and subraddit != ''
    assert subraddit.isidentifier()
    if subraddit in Moderator.MODS:
        raise PermissionError(subraddit+' has moderator')

    super().__init__(uname, display)
    self._subraddit = subraddit
    Moderator.MODS[subraddit] = uname
```

5. [23 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 devowel(s):
    """Returns a copy of s with all vowels removed

    Vowels are limited to 'a', 'e', 'i', 'o', and 'u' (both lower and upper case).
    Example: devowel('Apple') returns 'ppl'

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

    # Small data
    if len(s) == 0:
        return s
    elif len(s) == 1:
        if s in 'aeiouAEIOU':
            return ''
        else:
            return s

    # Break up and recurse
    left = devowel(s[0])
    right = devowel(s[1:])

    # Combine the result
    return left+right
```

(b) [13 points]

```
def find_runs(nlist):
    """Returns a list of all the runs in nlist

    A run is a list of adjacent numbers in ascending order. For example in the list
    [1,2,0,3,5,1,4], the runs are [1,2], [0,3,5], and [1,4], in that order. This
    function takes a list and extracts all of the individual runs and puts them in
    a new list (in order).

    Examples: find_runs([1,2,0,3,5,1,4]) returns [[1,2], [0,3,5], [1,4]]
              find_runs([5,4,3,2,1]) returns [[5],[4],[3],[2],[1]]
              find_runs([1,2,3,4,5]) returns [[1,2,3,4,5]]
              find_runs([1]) returns [[1]]

    Precond: nlist is a nonempty list of numbers"""
    # HINT: The hard part of this is the combination step.
    # Keep the small data and the division part simple.

    # Small data
    if len(nlist) == 1:
        return [nlist[:]]

    # Break up and recurse
    left  = find_runs(nlist[:1])
    right = find_runs(nlist[1:])

    # Look at last element of left and first of right
    last  = left[-1][-1]
    first = right[0][0]

    if last < first:
        # In this case, merge left and right
        right[0] = left[-1]+right[0]
        answer = right
    else:
        answer = left+right

    return answer
```