Last Name: ______________ First: ______________ Netid: ______________

CS 1110 Prelim 2 November 21st, 2023

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	22	
3	24	
4	28	
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.
`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. [22 points total] **Iteration**. Implement the functions on the next two pages, according to their specification, using for-loops. You **do not** need to enforce preconditions.

   (a) [12 points]

```
def lowercount(lst):
    ""Returns the number of lowercase letters in each element of lst

    Example: lowercount(['abc','Hello','OUT!']) returns [3,4,0]
    Example: lowercount(['aBc', '']) returns [2,0]
    Precond: lst a nonempty list of strings (the strings can be empty)"""

    # Create list accumulator
    result = []

    for s in lst:
        # Create accumulator for this string
        count = 0:

        for char in s:
            if char.islower():
                count += 1

        # Append count to list accumulator
        result.append(count)

    return result
```

Last Name: ______________ First: ______________ Netid: ______________

(b) [10 points]

```
def replace(text,subst):
    """Returns a COPY of text using subst to replace letters.

    The dictionary subst has lowercase letters both as keys and values. This
    function takes the string text and replaces any key of subst with the
    associated value.

    Example: replace('cat',{ 'a':'o' }) returns 'cot'
    Example: replace('pet',{ 'a':'o' }) returns 'pet'
    Example: replace('razzle',{ 'a':'o', 'z':'b' }) returns 'robble'

    Precond: text is a (possibly empty) string of lowercase letters
    Precond: subst is a dict with lowercase letters as keys and values"""
    # HINT: Only loop over ONE of the parameters. One is easier than the other.

    # Create string accumulator
    result = ''

    for letter in text:
        # Check if letter in subst
        if letter in subst:
            # Replace letter if there
            result = result+subst[letter]
        else:
            result = result+letter

    return result

   # NOTE: The following DOES NOT work.
   # Fails on subst = { 'a':'b', 'b':'a' }
   # result = text
   # for x in subst:
   #     result = result.replace(x,subst[x])
   # return result
```

Last Name: __________________ First: __________________ Netid: __________________

3. [24 points total] **Recursion**.

   Use recursion to implement the functions on the next two pages. **Solutions using loops will receive no credit.**

   HINT: To maximize partial credit, do not take shortcuts. Follow the three steps.

   (a) [10 points]

```
def swapcase(s):
    """Returns 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. Nonletters are unaffected.

    Example: swapcase('Hello World!')  returns 'hELLO wORLD!'
    Precond: s is a string (possibly empty)."""

    # Small data
    if s == '':
        return ''
    elif len(s) == 1:
        if s.isupper():
            return s.lower()
        else:
            return s.upper()

    # Break up the string
    left  = swapcase(s[:1])
    right = swapcase(s[1:])

    # Combine the answers
    return left + right
```

(b) [14 points]

```
def separate(nums):
    """Returns: A tuple separating nums into negative and non-negative portions

    This function returns a tuple (neg,pos). The value neg is a list of all the
    negative elements of nums (in their order from nums), while pos is a list
     of all the non-negative elements of nums (in their order from nums).

    Example: separate([1, -1, 2, -5, -3, 0]) returns ([-1, -5, -3], [1, 2, 0])
    Example: separate([-1, -5, -3]) returns ([-1, -5, -3],[])
    Example: separate([1, 2, 0]) returns ([],[1, 2, 0])
    Precond: nums is a (possibly empty) list of integers"""

    # Small data
    if len(nums) == 0:
        return ([],[])
    elif len(nums) == 1:
        if nums[0] < 0:
            return (nums[:],[])
        else:
            return ([],nums[:])

    # Break up the list
    left = separate(nums[:1])
    rght = separate(nums[1:])

    # Combine the answers
    neg = left[0]+right[0]
    pos = left[1]+right[1]
    return (neg,pos)
```

4. [28 points total] **Classes and Subclasses**

In this problem, you will create a class representing a license plate in a small state. License plates in this state are a number 0..999 followed by three (upper case) letters. When converted to a string, the number is padded with leading 0s to make it three digits. Examples of licenses are `001-ABC` or `093-XYZ`.

One of the most important properties of a license plate is that there can only be one of them with a given value. So we cannot have two different objects for the same license `001-ABC`. To model this propery, the class `License` has a class attribute list named `USED`. Every time a new license plate is created, the value is added to this list so that it cannot be used again. In addition, the license plate value is immutable (since allowing a user to change it would mean that the user could create two plates with the same value).

In addition to normal license plates, some people like to have vanity plates. A common vanity plate is one that is attached to a specific university, showing that the owner is an alum. Again, we cannot have a vanity plate with the same number as an existing plate. But since `Vanity` is a subclass of `License`, this should not be a problem if we initialize it properly.

On the next four 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.
6. Use `isinstance` when enforcing type-based preconditions.

We have not added headers for any of the getters and setters. You are to write these from scratch. However, **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: `Vanity` is not allowed to access any hidden attributes of `License`. We are also adding the additional restriction that `Vanity` may not access any getters and setters in `License`.

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

```
class License(object):                                          # Fill in missing part
    """A class representing a license plate

    CLASS ATTRIBUTES
    Attribute USED: All of the license plates used so far
    Invariant: USED is a list of tuples (prefix,suffix), initially empty"""
    # MUTABLE ATTRIBUTES
    # Attribute _owner: The name of the owner
    # Invariant: _owner is a NONEMPTY string, or None
    # IMMUTABLE ATTRIBUTES
    # Attribute _prefix: The first half of the licence
    # Invariant: _prefix is an int 0..999, inclusive
    #
    # Attribute _suffix: The second half of the licence
    # Invariant _suffix is a string of 3 upper case letters

    # CLASS ATTRIBUTES
    USED = []


    # DEFINE GETTERS/SETTERS/HELPERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getOwner(self):
        """
        Returns the owner of this license plate
        """
        return self._owner

    def setOwner(self,value):
        """Sets the owner of this license plate

        Parameter value: The owner's name
        Precondition: value a nonempty string or None"""
        assert value is None or (isinstance(value, str) and value != '')
        self._owner = value

    def getPrefix(self):
        """
        Returns the prefix of this license plate
        """
        return self._prefix

    def getSuffix(self):
        """
        Returns the suffix of this license plate
        """
        return self._suffix
```

Last Name: ______________ First: ______________ Netid: ______________

```
# Class License (CONTINUED).
def __init__(self,    prefix,    suffix,    owner = None):   # Fill in missing part
    """Initializes a license plate with the given prefix and suffix.

    No license plate can be created if it has the same prefix and suffix as an
    existing plate. On creation, the pair (prefix,suffix) is added to the class
    attribute USED to ensure that they cannot be reused.

    Precond: prefix is an int in 0..999, inclusive
    Precond: suffix is a string of 3 upper case letters
    Precond: owner is a nonempty string or None (Optional; default None)
    Additional precondition: No other  plate has this prefix,suffix"""
    assert isinstance(prefix, int)
    assert 0 <= prefix and prefix <= 999
    assert isinstance(suffix,str) and len(suffix) == 3
    assert suffix.isupper() and suffix.isalpha()
    assert not [prefix,suffix] in License.USED
    self.setOwner(owner)
    self._prefix = prefix
    self._suffix = suffix
    License.USED.append((prefix,suffix))




def __str__(self):                                          # Fill in missing part
    """Returns a string representation of this license plate.

    The string is of the form prefix-suffix. The prefix is padded with leading 0s
    to have three characters. If the plate has an owner, the owner follows the
    string in parentheses. Otherwise, nothing is added to the string.
    Example: '001-ABC' if no owner, or '093-XYZ (Bob)' """
    prefix = str(self._prefix)
    prefix = '0'*(3-len(prefix))+prefix
    result = prefix + '-' + self._suffix
    if not self._owner is None:
        result = result+' ('+self._owner+')'
    return result
```

Last Name: ______________ First: ______________ Netid: ______________

(b) [10 points] The class `Vanity`.

```
class Vanity(License):                                    # Fill in missing part
    """A class representing a vanity license plate"""
    # MUTABLE ATTIBUTE (In addition to those from License):
    # Attribute _university: The university displayed on the plate
    # Invariant: _university is a a nonempty string

    # DEFINE GETTERS/SETTERS AS APPROPRIATE. SPECIFICATIONS NOT NEEDED.
    def getUniversity(self):
        """
        Returns the university displayed on the plate
        """
        return self._university

    def setUniversity(self,value):
        """Sets the university displayed on the plate

        Parameter value: the university name
        Precondition: value a nonempty string"""
        assert isinstance(value, str) and value != ''
        self._university = value






    def __init__(self, prefix, suffix, owner, university):    # Fill in missing part
        """Initializes a vanity license plate with the given values.

        Vanity plates must have an (initial) owner. NO arguments are optional.
        Precondition: prefix is an int in 0..999, inclusive
        Precondition: suffix is a string of 3 upper case letters
        Precondition: owner is a nonempty string, NOT OPTIONAL
        Precondition: university is a nonempty string
        Additional precondition: No other plate has this prefix,suffix"""
        assert not owner is None
        # This contains asserts and must go first
        self.setUniversity(university)
        super().__init___(prefix,suffix,owner)
```

Last Name: ____________ First: ____________ Netid: ____________

```
# Class License (CONTINUED).
def __str__(self):                                          # Fill in missing part
    """Returns a string representation of this vanity plate

    The format is 'prefix-suffix (Owner, University)'. If owner is None (the
    setter allows this to happen), the format is 'prefix-suffix (University)'.
    Example: '001-ABC (Cornell)' if no owner, or '093-XYZ (Bob, Syracuse)'"""
    result = super().__str__()
    if result[-1] == ')':
        result = result[:-1]+', '+self._university+')'
    else:
        result = result + '('+self._university+')'
    return result
```

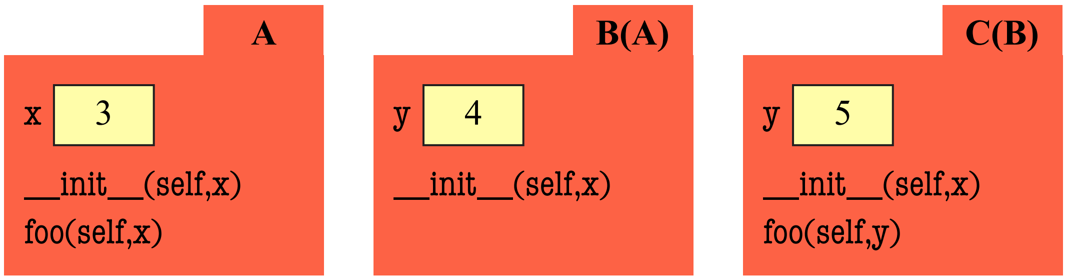
5. [24 points total] **Call Frames and Name Resolution**

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

```
1  class A(object):
2      x = 3
3
4      def __init__(self,x):
5          self.x = x+3
6
7      def foo(self,x):
8          self.y = self.x
9          self.z = x
10
11 class B(A):
12     y = 4
13
14     def __init__(self,x):
15         self.foo(x-1)
16
```

```
17 class C(B):
18     y = 5
19
20     def __init__(self,x):
21         super().__init__(x+1)
22
23     def foo(self,y):
24         self.y = 2*self.x
25         self.z = 3*y
26
27
28
29
30
31
32
```

(a) [6 points] Draw the class folders in the heap for these three classes.

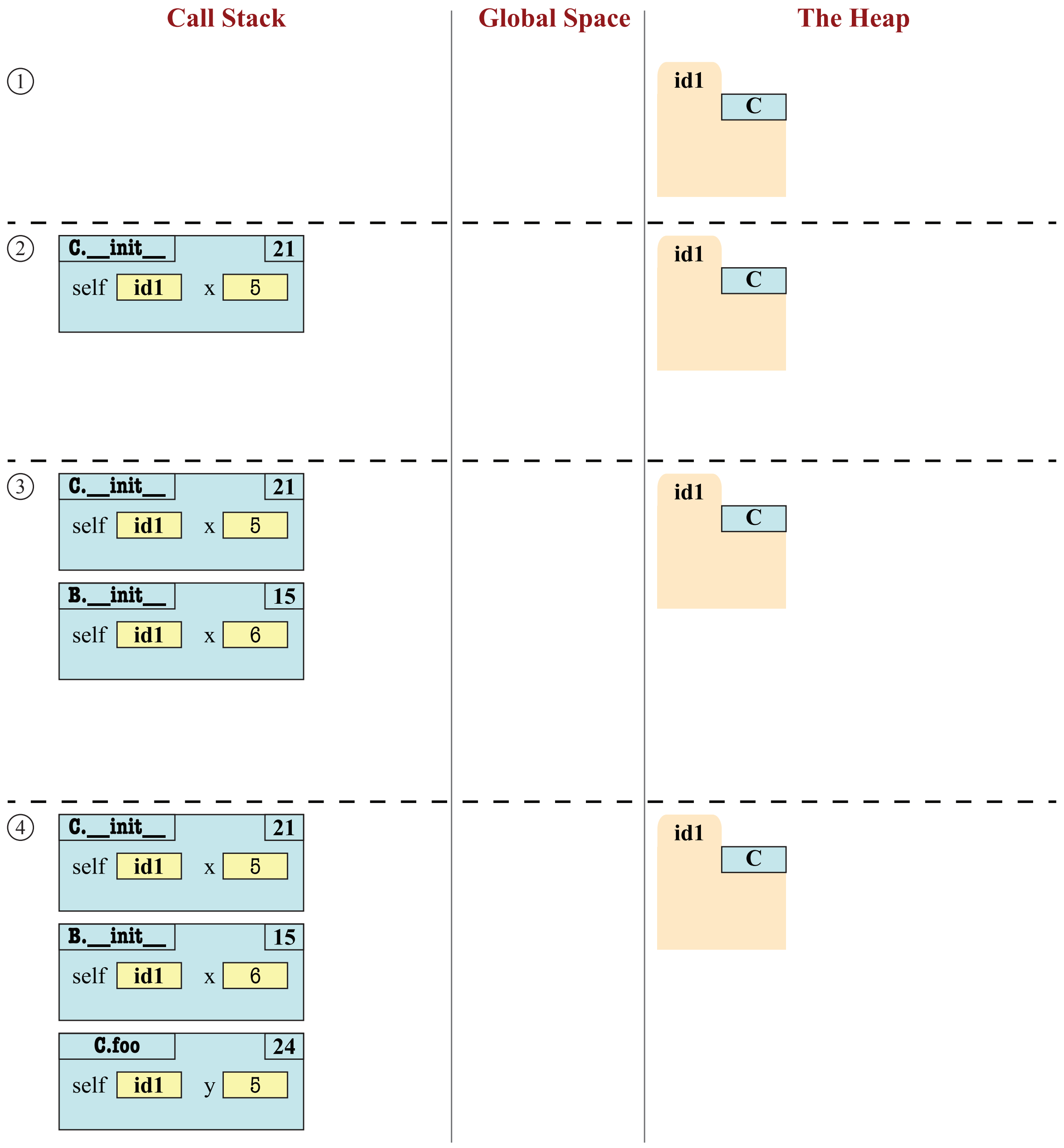


(b) [18 points] Below and on the two page, diagram the call

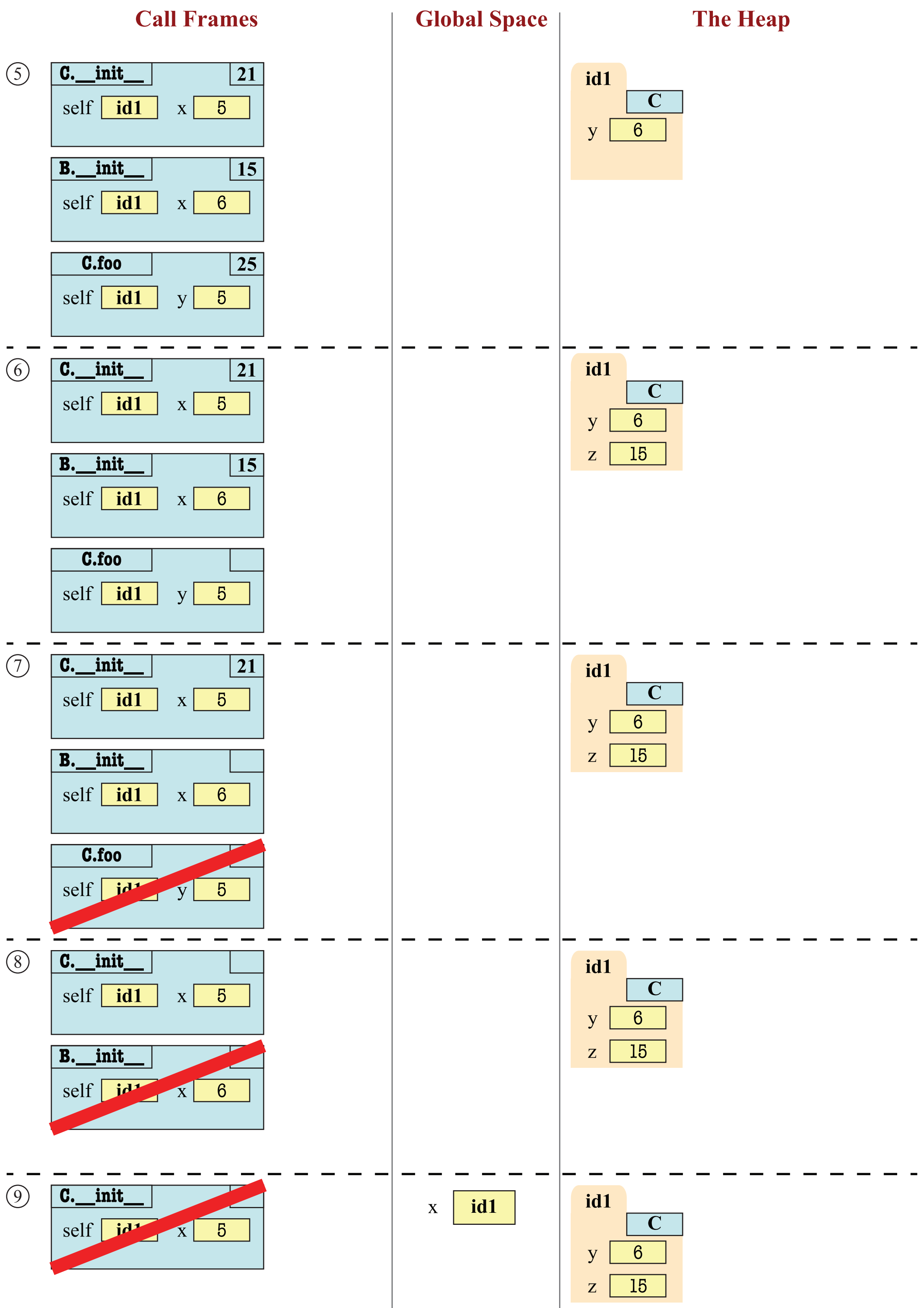
```
>>> x = C(5)
```

You will need **nine diagrams**. Draw the call stack, global space and heap space. If the contents of any space are unchanged between diagrams, you may write *unchanged*. You do not need to draw the class folders from 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 Frames
Global Space
The Heap
5
C.__init__ 21 self id1 x 5
B.__init__ 15 self id1 x 6
C.foo 25 self id1 y 5
id1 C y 6
6
C.__init__ 21 self id1 x 5
B.__init__ 15 self id1 x 6
C.foo self id1 y 5
id1 C y 6 z 15
7
C.__init__ 21 self id1 x 5
B.__init__ self id1 x 6
C.foo self id1 y 5
id1 C y 6 z 15
8
C.__init__ self id1 x 5
B.__init__ self id1 x 6
id1 C y 6 z 15
9
C.__init__ self id1 x 5
x id1
id1 C y 6 z 15