Last Name: ________________ First: ________________ Netid: ________________

CS 1110 Final, December 12th, 2021

This 150-minute exam has 8 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
```

Unless you are explicitly directed otherwise, you may use anything you have learned in this course. You may use the backside of each page for extra room for your answers. However, if you do this, **please indicate clearly** on the page of the associated problem.

Question	Points	Score
1	2	
2	14	
3	9	
4	14	
5	10	
6	22	
7	12	
8	17	
Total:	100	

The Important First Question:

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

References

String Operations

Expression	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.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 upper case; it returns False otherwise (e.g. `'A123'` is True but `'123'` is False).
`s.lower()`	**Returns**: A copy of `s` but with all letters converted to lower case (so `'A1b'` becomes `'a1b'`).
`s.upper()`	**Returns**: A copy of `s` but with all letters converted to upper case (so `'A1b'` becomes `'A1B'`).
`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 numbers; it returns False otherwise.
`s.isalnum()`	**Returns**: True if `s` is *not empty* and its elements are all letters or numbers; it returns False otherwise.

List Operations

Expression	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

Expression	Description
`len(d)`	**Returns**: number of keys in dictionary `d`; it can be 0.
`y in d`	**Returns**: True if `y` is a key in dictionary `d`; False otherwise.
`d[k] = v`	Assigns value `v` to the key `k` in dictionary `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. [14 points total] **Testing and Exceptions**

(a) [5 points] Below is the specification of a function. Do not implement it. In the space below, provide **at least five different test cases** to verify that this function is working correctly. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

```
def repeat(s,n):
    """Returns a copy of s concatentated together n times.
    Example: repeat('a',4) returns 'aaaa'
    Precondition: s is a string, n > 0 an int"""
```

This is a classic "rule of numbers" problem. You want to look at strings of length 0 (empty string), one, and two. And you want n to be either 1 or 2 (0 is not permitted by the precondition). This gives 6 possibilities. The contents of the string **do not matter** as we are only duplicating them, not checking them. However, we did make a distinction between a string with repeated characters and one without. Here were our canonical tests:

Input	**Output**	**Reason**
s = '', n = 2	''	Empty string
s = 'a', n = 1	'a'	One character, no repeat
s = 'a', n = 2	'aa'	One character, repeated
s = 'ab', n = 1	'ab'	Multiple characters, no repeat
s = 'ab', n = 2	'abab'	Multiple characters, repeated

(b) [9 points] Consider the functions listed below. Note that `ValueError` and `AssertionError` are both subclasses of `Exception` but neither is a subclass of the other. On the next page, we want you to write down the text printed out (e.g. the traces) for each function call.

```
def first(n):
    x=0
    try:
        print('Try first')
        x = second(n)
    except Exception:
        print('Except first')
        x = x+2
    print('End first at '+str(x))
    return x
```

```
def second(n):
    y=5
    try:
        print('Try second')
        y = third(n)
    except ValueError:
        print('Except second')
        y = y-2
    print('End second at '+str(y))
    return y
```

```
def third(n):
    print('Start third')
    if n == 0:
        print('Third at n == 0')
        raise ValueError('Value error')
    elif n == 1:
        print('Third at n == 1')
        assert False, 'Assertion error'
    print('End third at '+str(n))
    return n
```

i. first(0)

```
Try first
Try second
Start third
Third at n == 0
Except second
End second at 3
End first at 3
```

ii. first(1)

```
Try first
Try second
Start third
Third at n == 1
Except first
End first at 2
```

iii. first(2)

```
Try first
Try second
Start third
End third at 2
End second at 2
End first at 2
```

3. [9 points total] **Short Answer**

(a) [3 points] Name the four categories of Python variables seen in class.

The variable categories are *global variable*, *local variable*, *parameter* and *attribute*. Accumulator is not a category recognized by Python.

(b) [2 points] What is the difference between a function and a method?

A *method* is a function defined inside of the body of a class. Its name is stored in the class folder, and not inside a module or global space. A method is called differently than a function, with the first argument before the method name, rather than inside of the parentheses.

(c) [4 points] Name four sorting algorithms we covered in class, and the running time of each (e.g. n, $n \log n$, or n^2).

The algorithms *insertion sort* and *selection sort* take n^2 steps. The algorithms *quick sort* and *merge sort* take $n \log n$ steps average time. Though we would also accept n^2 worst time for *quick sort*.

4. [14 points] **Iteration**

The Python package `numpy` has a lot of tools for manipulating matrices (what you may call a table), which are 2-dimensional lists of numbers. One of the things that `numpy` can do is to *reshape* a matrix. When you reshape a matrix with m rows and n columns, you turn it into a matrix with s rows and t columns, where $mn = st$. For example, we can call reshape twice to turn a 2x4 matrix into a 4x2 matrix, and then into a 1x8 matrix as follows:

$$\begin{bmatrix} 1 & 2 & 3 & 4 \\ 5 & 6 & 7 & 8 \end{bmatrix} \underset{\texttt{reshape(a,4,2)}}{\Longrightarrow} \begin{bmatrix} 1 & 2 \\ 3 & 4 \\ 5 & 6 \\ 7 & 8 \end{bmatrix} \underset{\texttt{reshape(a,1,8)}}{\Longrightarrow} \begin{bmatrix} 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 \end{bmatrix}$$

Note that 1x8 matrix **is not a 1-dimensional list**. It is a nested list containing one list (the row) of eight elements. Similarly, an 8x1 matrix is a nested list of eight one-element lists.

When you reshape the matrix, you process the elements in normal row-major order. So top row first, left-to-right. With this in mind, implement the function below according to its specification. You are allowed to use any form of iteration (for-loop or while-loop) that you want.

Hint: This function is a lot simpler if you first create a 1-dimensional list with the elements in the proper order. Then use this new list to make your matrix.

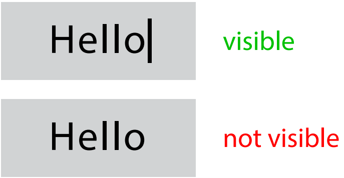
```
def reshape(table,r,c):
    """Returns a copy of the table reshaped to have r rows and c cols

    Example: reshape([[0,1,2],[3,4,5]],3,2) returns [[0,1],[2,3],[4,5]]

    Precondition: table is a nonempty 2d list of numbers. r and c are ints with
    r*c equal to the total number of elements in the table"""

    # Make the 1-d list
    flat = []
    for row in table:
        for col in row:
            flat.append(col)

    # Accumulate the reshaped copy
    result = []
    pos = 0
    for x in range(r):
        row = []    # Accumulator for one row
        for y in range(c):
            row.append(flat[pos])
            pos = pos+1
        result.append(row)

    return result
```

5. [10 points] **Recursion**

Implement the function below using recursion according to its specification. You may not use any loops in your answer (for or while). You also **may not use any list methods** (though slicing and adding lists is okay). Answers that violate this rule **will receive no credit**.

```
def insert(nums,x):
    """Returns a copy of nums, putting x into the correct, ordered position.

    List nums is sorted (e.g. the elements are in order). The function puts
    puts x into nums at the right position so it is still ordered. If x is
    already in nums, this function still inserts a copy of x.

    Example: insert([0,2,4,5],3)  returns [0,2,3,4,5]
             insert([1,2,3,7],-1) returns [-1,1,2,3,7]
             insert([1,2,2,7],2) returns [1,2,2,2,7]
             insert([],4) returns [4]

    Precondition: nums is a sorted (possibly empty) list of ints; x is an int"""

    # Empty list is easy
    if nums == []:
        return [x]

    # If x is before everything, put at front
    if x <= nums[0]:
        return [x]+nums

    # Otherwise divide; x MUST go with right
    left  = nums[:1]
    right = insert(nums[1:],x)
    return left+right
```

6. [22 points] **Classes and Subclasses**



For this question, you will use the classes of Assignment 7 to make a `GTextField`. Shown to the right, this is a text label with a blinking cursor. The idea is that the cursor represents where text will appear when typed. However, you will not worry about typing or adding text. You will **only** focus on the blinking cursor.

The cursor itself will be a solid black `GRectangle`. As a reminder, the `GRectangle` class comes with the following (mutable) attributes.

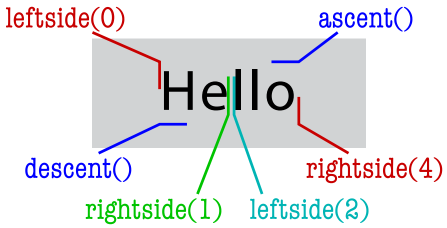
Attribute	Invariant	Description
`x`	`float`	x-coordinate of the center of the rectangle.
`y`	`float`	y-coordinate of the center of the rectangle.
`left`	`float < x`	x-coordinate of the left edge of the rectangle.
`right`	`float > x`	x-coordinate of the right edge of the rectangle.
`top`	`float > y`	y-coordinate of the top edge of the rectangle.
`bottom`	`float < y`	y-coordinate of the bottom edge of the rectangle.
`width`	`float > 0`	The width along the horizontal axis.
`height`	`float > 0`	The height along the vertical axis.
`fillcolor`	`str`	The interior color (represented as the name, e.g. `'blue'`).

There are other attributes, but they can be ignored for this problem. Recall that when you create a `GRectangle`, you use keyword arguments to specify the attributes, and all arguments are optional (e.g. `GRectangle(fillcolor='red')`).

The class `GLabel` is a subclass of `GRectangle` and has additional attributes. **You can ignore all the font attributes for this problem**. We will assume the default font. Instead, you only need to use the attribute `text`, which is a string. In addition, the version of `GLabel` we are using has the following methods.

Method	Precondition	Description
`l.descent()`	None	**Returns**: The y-coord of the lowest character position.
`l.ascent()`	None	**Returns**: The y-coord of the highest character position.
`l.leftside(p)`	an index of `l.text`	**Returns**: The x-coord left of character at position `p`.
`l.rightside(p)`	an index of `l.text`	**Returns**: The x-coord right of character at position `p`.

To greatly simplify this problem, we are going to assume that `text` is an *immutable* attribute of the `GLabel`, meaning it does not change after you set it in the constructor call. Otherwise, it will be very hard to satisfy all of the invariants (because they depend on the length of `text`). We will also assume all the text fits on one line (e.g. newlines are ignored). Once the attribute `text` is set, you can call the methods above to get the position of the characters in the label.



You will note that this diagram includes some spacing between the actual characters and the values returned, such as with `l.descent()`. This is natural – it is the way that fonts work.

With this in mind, implement `GTextField` as outlined on the next two pages. This is a subclass of `GLabel` that has a single `GRectangle` as an attribute (for the cursor). We have provided the specifications for the methods `__init__`, `update`, and `draw`. You should also add getters and setters (where appropriate) for the new attributes. Those setters must have preconditions to enforce the attribute invariants. Furthermore, all methods (not just the setters) must enforce the preconditions for any value other than `self`. Finally, all type-based preconditions should be enforced with `isinstance` and not `type`.

You **do not** need setters and/or getters for the attributes inherited from `GRectangle` or `GLabel`. Those can be left as is. Remember that attribute `text` is immutable.

```
class GTextField(GLabel):
    """A class representing a label with a flashing cursor

    The cursor is drawn as a rectangle that is 2 pixels wide, and whose top
    is ascent() and bottom is descent(). If p is the current position, the
    horizontal coordinates of the cursor are defined as follows:
     * if p is 0, the right edge of the cursor is leftside(0)
     * if p is len(text), the left edge of the cursor is rightside(p-1)
     * otherwise the cursor is centered between leftside(p) and rightside(p-1)
    The cursor is only drawn if the visible attribute is True. The visible
    switches between True and False every BLINK_RATE animation frames.

    Attribute BLINK_RATE: A CLASS ATTRIBUTE set to the integer 20"""
    # MUTABLE ATTRIBUTE:
    # _position: the cursor position; an int between 0 and len(text) (inclusive)
    # NOTE: Changing this value moves the cursor (see above)
    # IMMUTABLE ATTRIBUTE:
    # _visible: whether to show the cursor; a boolean
    # INACCESSIBLE ATTRIBUTES (NO GETTERS OR SETTERS):
    # _cursor: the cursor; a GRectangle whose position/size is as explained above
    # _blinkdown: frame counter for blinking; an int >= 0 and <= BLINK_RATE

    # CLASS ATTRIBUTE. NO GETTERS OR SETTERS.
    BLINK_RATE = 20

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

    def getPosition(self):
        """Returns the cursor psition"""
        return self._position

    def setPosition(self,value):
        """Sets the cursor position"""
        isinstance(value,int)
        assert value >= 0 and value <= len(self.text)
        self._position = value

        # Update the cursor position
        if value == 0:
            self._cursor.right = self.leftedge(0)
        elif value == len(self.text):
            self._cursor.left = self.rightedge(self._position-1)
        else:
            left = self.leftedge(self._position)
            rght = self.rightedge(self._position-1)
            self._cursor.x = (left+rght)/2

    def getVisible(self):
        """Returns the cursor visibility"""
        return self._visible
```

```
# Class GTextField (CONTINUED).

def __init__( self,  left,  bottom,  width,  height,  text = 'Hello'):     # Fill in
    """Initializes a text field with the given parameters.

    The cursor starts off visible and at position len(text) (after the last
    character). This method sets the frame counter for blinking to BLINK_RATE.

    Parameter left:   left edge of the field; a float
    Parameter bottom: bottom edge of the field; a float
    Parameter width:  width of the field; a float > 0
    Parameter height: height of the field; a float > 0
    Parameter text:   the text to display; a string
    The argument text is OPTIONAL with default value 'Hello'."""

    super().__init__(left=left,bottom=bottom,width=width,height=height,text=text)

    height = self.texttop()-self.textbot()
    self._cursor = GRectangle(width=2,height=height,fillcolor= 'black')
    self._visible   = True
    self._blinkdown = self.BLINK_RATE
    self.setPosition(len(text))    # Puts cursor object in right place


def update( self ):                                                        # Fill in
    """Updates the blinking cursor

    Subtracts one from the blinking frame counter. If the result is 0, it
    it resets the counter to BLINK_RATE and swaps the visibility attribute."""

    self._blinkdown = self._blinkdown-1

    if self._blinkdown == 0:
        self._blinkdown = self.BLINK_RATE
        self._visible = not self._visible


def draw( self,  view ):                                                   # Fill in
    """Draws this object to the given view.

    The cursor is only drawn when it is visible"""
    Parameter view: the view to draw to, which is a GView object"""

    super().draw(view)              # Enforces precondition for us

    if self._visible:
        self._cursor.draw(view)
```

7. [12 points] **Generators**

Implement the generator shown below. You can use any Python you have learned in class. However, note the precondition. The value `input` is iterable, but it is **not** necessarily a sequence. So it cannot be sliced and it has no length.

```
def pairify(input):
    """Generates a sequence of adjacent pairs (as tuples) from input

    If there is one element left over when the generator is finished,
    then it yields a pair with second element None. If input is empty
    this generator outputs nothing (and hence crashes).

    Example: pairify([1,2,3,4]) yields (1,2), and then (3,4)
             pairify([1,2,3]) yields (1,2), and then (3,None)

    Precondition: input is any iterable of numbers"""

    # Keep track of last one seen
    last = None

    for x in input:
        if last is None:
            last = x
        else:
            yield (last,x)
            last = None

    # Check if we have one left over
    if not last is None:
        yield (last,None)
```

8. [17 points] **Call Frames**

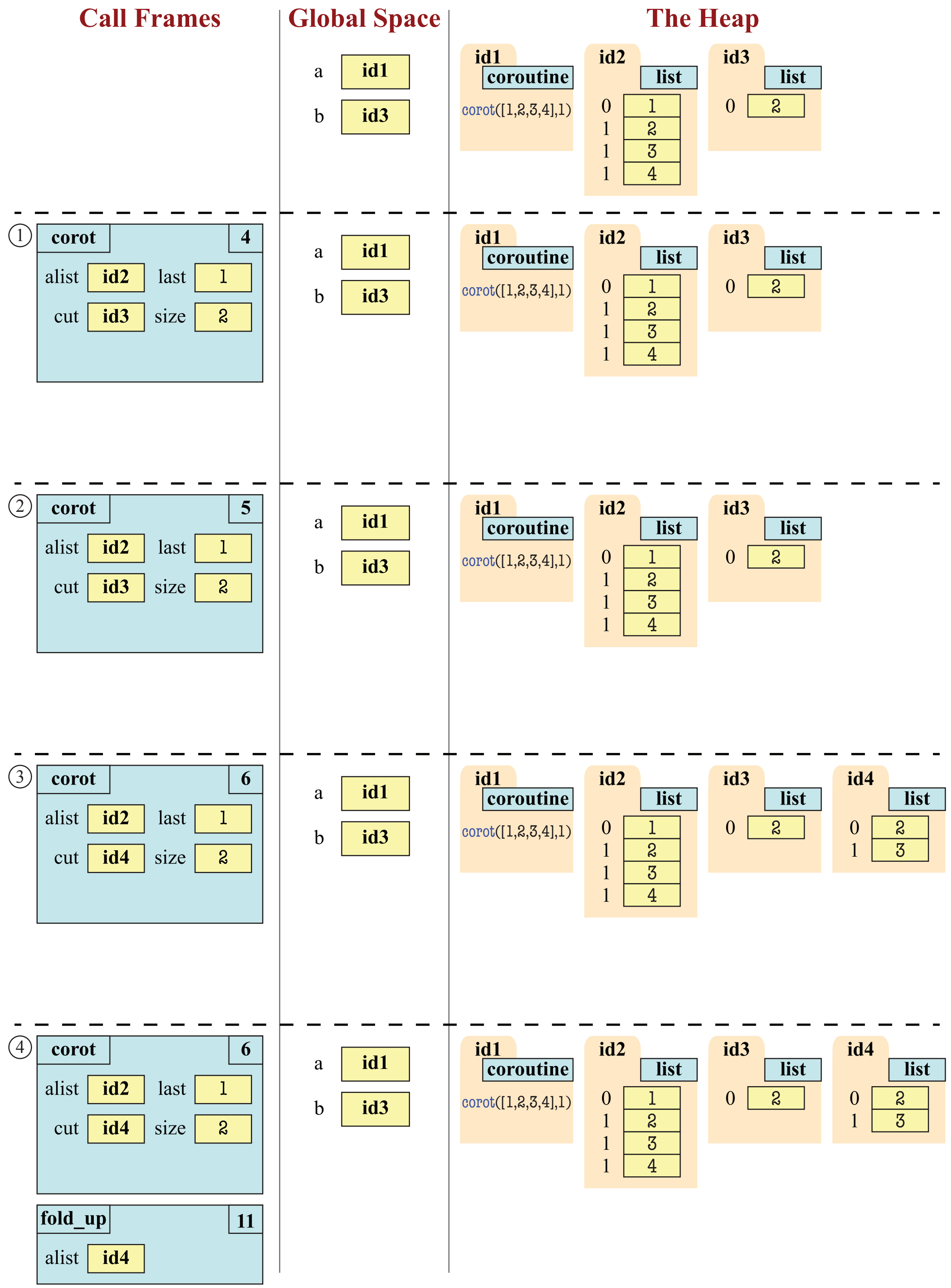
Consider the coroutine `corot` and the function `fold_up` shown below

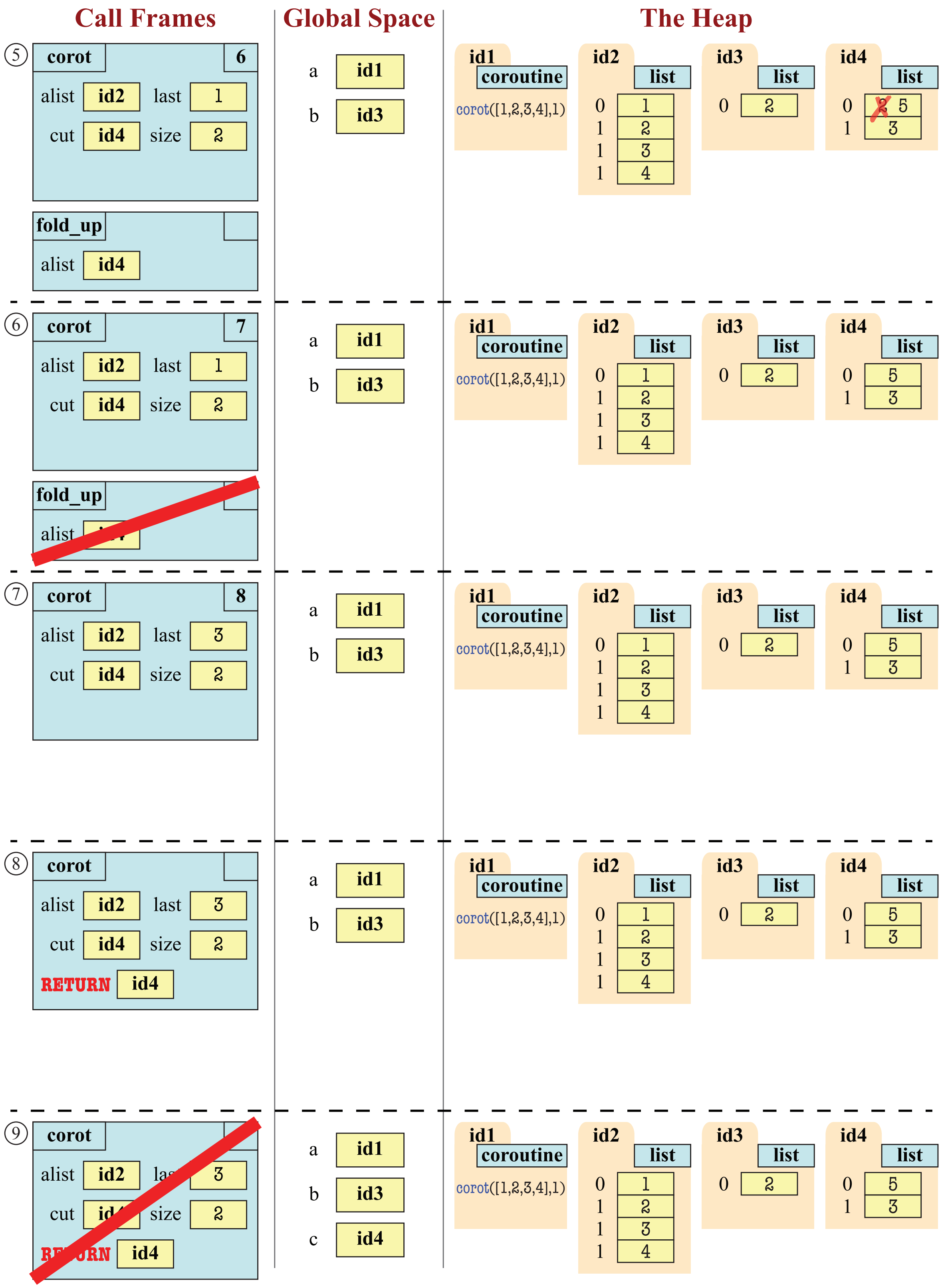
```
1  def corot(alist,size):
2      """Generates folded segs of alist"""
3      last = 0
4      while last < len(alist):
5          cut = alist[last:last+size]
6          fold_up(cut)
7          last = last+size
8          size = (yield cut)
```

```
9  def fold_up(alist):
10     """Modifies alist to add first, last"""
11     alist[0] = alist[0]+alist[-1]
12
13 # Code to be executed
14 a = corot([1,2,3,4],1)
15 b = next(a)
16 c = a.send(2)
```

At the top of the next page, we have provided the contents of global space and the heap after executing the assignment statement to `b` on line 15 above. We want you to diagram the assignment statement on the next line, **line 16**. You should draw a new diagram every time a call frame is added or erased, or an instruction counter changes. There are a total of **nine** diagrams to draw, not including the initial diagram we have provided. You may write *unchanged* in any of the three spaces if the space does not change at that step.

Last Name: ____________________ First: ____________________ Netid: ____________________




Call Frames
Global Space
The Heap
5
corot 6
alist id2
last 1
cut id4
size 2
fold_up
alist id4
a id1
b id3
id1 coroutine
corot([1,2,3,4],1)
id2 list
0 1
1 2
1 3
1 4
id3 list
0 2
id4 list
0 2 5
1 3
6
corot 7
alist id2
last 1
cut id4
size 2
fold_up
alist
a id1
b id3
id1 coroutine
corot([1,2,3,4],1)
id2 list
0 1
1 2
1 3
1 4
id3 list
0 2
id4 list
0 5
1 3
7
corot 8
alist id2
last 3
cut id4
size 2
a id1
b id3
id1 coroutine
corot([1,2,3,4],1)
id2 list
0 1
1 2
1 3
1 4
id3 list
0 2
id4 list
0 5
1 3
8
corot
alist id2
last 3
cut id4
size 2
RETURN id4
a id1
b id3
id1 coroutine
corot([1,2,3,4],1)
id2 list
0 1
1 2
1 3
1 4
id3 list
0 2
id4 list
0 5
1 3
9
corot
alist id2
last 3
cut id4
size 2
RETURN id4
a id1
b id3
c id4
id1 coroutine
corot([1,2,3,4],1)
id2 list
0 1
1 2
1 3
1 4
id3 list
0 2
id4 list
0 5
1 3