Last Name: _____ First: _____ Netid: _____

CS 1110 Final, December 17th, 2019

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	12	
3	19	
4	14	
5	12	
6	14	
7	12	
8	15	
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: _____

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`.

Last Name: _____ First: _____ Netid: _____

2. [12 points total] **Testing and Exceptions**

(a) [6 points] Below is the specification of a function demonstrated in class. Do not implement it. In the space below, provide **at least six 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 loose_palindrome(s):
    """Returns True if s is a palindrome considering only the letters.
    Case and any non-letter characters are ignored.
    Example: loose_palindrome('A man, a plan, a canal. Panama!') returns True.
    Precondition: s is a string"""
```

There are several possible answers, but they must all be *significantly* different. Here are some examples of some significantly different tests:

Input	Output	Reason
`'abc'`	False	Not a palindrome
`'aba'`	True	Simple palindrome
`'abA'`	True	Palindrome, ignoring case
`'a,ba'`	True	Palindrome, ignoring punctuation
`'ab,A'`	True	Palindrome, ignoring both
`''`	True	Empty string is a special palindrome

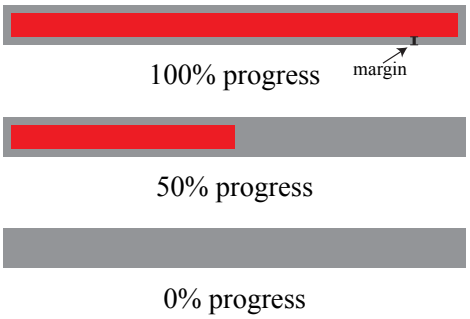
(b) [6 points] Implement the function specified below. You must use a try-except statement. **Answers using if statements will receive no credit.**

```
def next_int(s):
    """Returns the string representation of next int after string s.
    If s does not represent an int, it returns None.
    Example: next_int('10') returns '11'; next_int('a') returns None.
    Precondition: s is a string"""
    # Try block is if no errors occurred.
    try:
        n = int(s)
        n = n+1
        return str(n)
    except:
        pass # Returning here is okay, too

    return None
```

Last Name: _____ First: _____ Netid: _____

3. [19 points] **Classes and Subclasses**

For this question, you will use the classes of Assignment 7 to make a `GProgressBar`. Shown to the right, this is one rectangle nested inside of another to indicate progress towards a goal. The progress is a value between 0 and 1. The size of the progress bar is determined by the outer (grey) rectangle. The inner (red) rectangle, when progress is 1 or 100%, is smaller than the outer one by `margin` on all sides. If the progress is less than one, the width of the inner bar is reduced to that percentage (and invisible if progress is 0).



One easy way to make a GUI element composed of two different objects is to make the class a subclass on one and have the other as an attribute. That is what we have done on the next page. `GProgressBar` is a subclass of `GRectangle` but it has a `_bar` attribute for the interior `GRectangle`.

As a reminder, the `GRectangle` class comes with the following attributes.

Attribute	Invariant	Description
`left`	`float`	x-coordinate of the left edge of the rectangle.
`bottom`	`float`	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'`).
`linecolor`	`str`	The border color (represented as the name, e.g. `'green'`).

There is also an attribute for the thickness of the rectangle border, but you can ignore that for this question.

With this in mind, implement this class on the next page. We have provided the specifications for the methods `__init__` 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`.

One of the attributes – `_bar` – is **inaccessible**. This is a stronger restriction than immutable. It means the user can *never* access the attribute, even with a getter. If user could access the attribute with a getter, that user could then change the dimensions of the interior rectangle. This should never happen. The dimensions of the interior rectangle should always be a function of (1) the outer rectangle, (2) the progress amount, and (3) the margin.

Instead, you have the pseudo setter/getter `getBarColor` and `setBarColor`, which are specified for you. These allow you to change the color of the interior rectangle without accessing the interior rectangle directly.

Hint: The attributes in `GRectangle` work like they do in Assignment 7, and have invisible setters and getters. Therefore, you never have to enforce the invariants for these attributes. You only need to worry about your new attributes: `_progress`, `_margin`, and `_bar`. Also, remember that their constructors use keyword arguments (e.g. `GRectangle(fillcolor='red')`).

Finally, note that the width of a `GRectangle` can never be 0. But it can be less than 1.

Last Name: _____ First: _____ Netid: _____

```
class GProgessBar(GRectangle):
    """A class representing a simple progress bar

    The progress bar is drawn as two rectangles, one inside of the other.
    The width of the interior rectangle is proportional to the progress.
    The interior rectangle has a single color for both fill and line."""
    # MUTABLE ATTRIBUTE:
    # _progress: the amount of progress; an int or float between 0 and 1 (inclusive)
    # IMMUTABLE ATTRIBUTE:
    # _margin: the distance between the inner and outer bar; an int or float >= 0
    # INACCESSIBLE ATTRIBUTE:
    # _bar: the interior progress bar; a GRectangle with one color for fill and line
    # Because the outer bar can be resized, _bar must be rescaled each time it is drawn

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

    def getProgress(self):
        """Returns amount of progress""
        return self._progress

    def setProgress(self,value):
        """Sets amount of progress""
        assert type(value) in [int,float]
        assert 0 <= value <= 1
        self._progress        = value

    def getMargin(self):
        """Returns the margin spacing"""
        return self._margin



    def getBarColor( self ):                                          # Fill in
        """Returns the color (fill and line) of the interior bar"""
        return self._bar.fillcolor




    def setBarColor( self,  value ):                                  # Fill in
        """Sets the color (fill and line) of the interior bar

        Parameter value: a string, and not the fillcolor of the outer bar"""
        assert type(value) == str
        assert value != self.fillcolor
        self._bar.fillcolor = value
        self._bar.linecolor = value
```

```
# Class GProgessBar (CONTINUED).

def __init__( self,  left,  bottom,  width,  height,  bcolor, margin = 0): # Fill in
    """Initializes a progress bar with the given parameters.

    The values left, bottom, width, and height define the size of the
    outer rectangle. Both the fill color and line color of the outer
    rectangle is 'grey'. The initial progress value is 0.

    Parameter left:   left edge of outer rectangle; a float
    Parameter bottom: bottom edge of outer rectangle; a float
    Parameter width:  width of outer rectangle; a float > 0
    Parameter height: height of outer rectangle; a float > 0
    Parameter bcolor: color of interior bar; a string and NOT 'grey'
    Parameter margin: margin of interior bar; an int or float >= 0
    The argument margin is OPTIONAL with default value 0."""
    assert type(bcolor) == str and bcolor != 'grey'
    assert type(margin) in [int,float]
    assert margin >= 0
    super().__init__(left=left,bottom=bottom,width=width,height=height)
    self.linecolor = 'grey'
    self.fillcolor = 'grey'
    self._margin   = margin
    self._progress = 0
    self._bar = GRectangle(left=left+margin,bottom=bottom+margin,
                           width=1,height=height-2*margin)
    self._bar.linecolor = bcolor
    self._bar.fillcolor = bcolor



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

    The interior bar is scaled to the progress, and hidden if progress is 0.
    Parameter view: the view to draw to, which is a GView object"""
    # HINT: The user may have altered the position and dimensions of the outer bar.
    # You must recompute the position and dimensions of the inner bar.

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

    if self._progress > 0:
        w = (self.width-2*self._margin)*self._progress
        self._bar.left = self.left+self._margin
        self._bar.bottom = self.bottom+self._margin
        self._bar.width  = w
        self._bar.height = self.height-2*self._margin
        self._bar.draw(view)
```

Last Name: _____ First: _____ Netid: _____

4. [14 points] **Iteration**

A matrix is a 2-dimensional list of numbers. The diagonal of a square matrix are all elements with the same row and column. We say that a matrix is *pure diagonal* if any element that is not on the diagonal is 0. We say that it is *upper triangular* if any element below the diagonal is 0, and *lower triangular* if any element above the diagonal is 0. These are illustrated below.

1	0	0	0
0	6	0	0
0	0	11	0
0	0	0	16

Fig 1: Pure Diagonal

1	2	3	4
0	6	7	8
0	0	11	12
0	0	0	16

Fig 2: Upper Triangular

1	0	0	0
5	6	0	0
9	10	11	0
13	14	15	16

Fig 3: Lower Triangular

Note that the other elements are always allowed to be 0 as well. So technically, a pure diagonal matrix is also upper (and lower) triangular. Use this to implement the function according to its specfication.

```
NORMAL_MATRIX    = -1    # Non-zeroes in both upper and lower triangle
UPPER_TRIANGULAR = 0     # Zeroes only in lower triangle
LOWER_TRIANGULAR = 1     # Zeroes only in upper triangle
PURE_DIAGONAL    = 2     # Zeroes only off the diagonal

def checkmatrix(matrix):
    """Returns the value BEST describing the given matrix.

    Value returned is one of NORMAL_MATRIX, UPPER_TRIANGULAR, LOWER_TRIANGULAR,
    or PURE_DIAGONAL. It should return the highest value describing the matrix.

    Precondition: matrix is a non-empty square 2d list of numbers."""

    upper = True
    lower = True
    for r in range(len(matrix)):
        for c in range(len(matrix[0])):
            if matrix[r][c] != 0:
                if r < c:
                    lower = False
                elif r > c:
                    upper = False

    if lower and upper:
        return PURE_DIAGONAL
    elif lower:
        return LOWER_TRIANGULAR
    elif upper:
        return UPPER_TRIANGULAR

    return NORMAL_MATRIX
```

Last Name: _____ First: _____ Netid: _____

5. [12 points] **Recursion**

If you have ever downloaded DLC for a game, or redeemed a coupon online, you know that they are often defined as groups letters and numbers separated by dashes. The simplest variation has just one dash and groups its letters in numbers in blocks of four, like this: `K97J-FTRE`.

The function below takes a string of (potentially lower case) letters and numbers and returns it as a string of (upper case) letters and numbers separateds into groups of 4. If the length of the string is not divisible by 4, the leftovers at the end are dropped.

Implement this function using recursion. **Answers using loops will receive no credit**.

```
def couponify(code):
    """Returns code properly formatted as a coupon code in blocks of 4.

    Each block of a coupon code is composed of upper case letters and numbers.
    Dashes are used to separate the blocks from each other.

    Example: couponify('k97J') returns 'K97J'
             couponify('K97j9876') returns 'K97J-9876'
             couponify('K97JfTRe9876') returns 'K97J-FTRE-9876'
             couponify('K97') returns ''
             couponify('K97J8') returns 'K97J'
             couponify('k97JfTRe9') returns 'K97J-FTRE'

    Preconditions: code is a (possibly empty) string of letters and numbers"""
    # Base case is anything 4 elements or less
    if len(code) < 4:
        return ''        # Drop anything too small
    elif len(code) == 4:
        return code.upper()  # Convert lower case letters

    # Break up into groups of 4
    left = code[:4]
    right = couponify(code[4:])
    left = left.upper()  # Convert lower case letters

    # Combine the result
    if len(right) > 0:
        return left+'-'+right
    else:
        return left
```

Last Name: _____ First: _____ Netid: _____

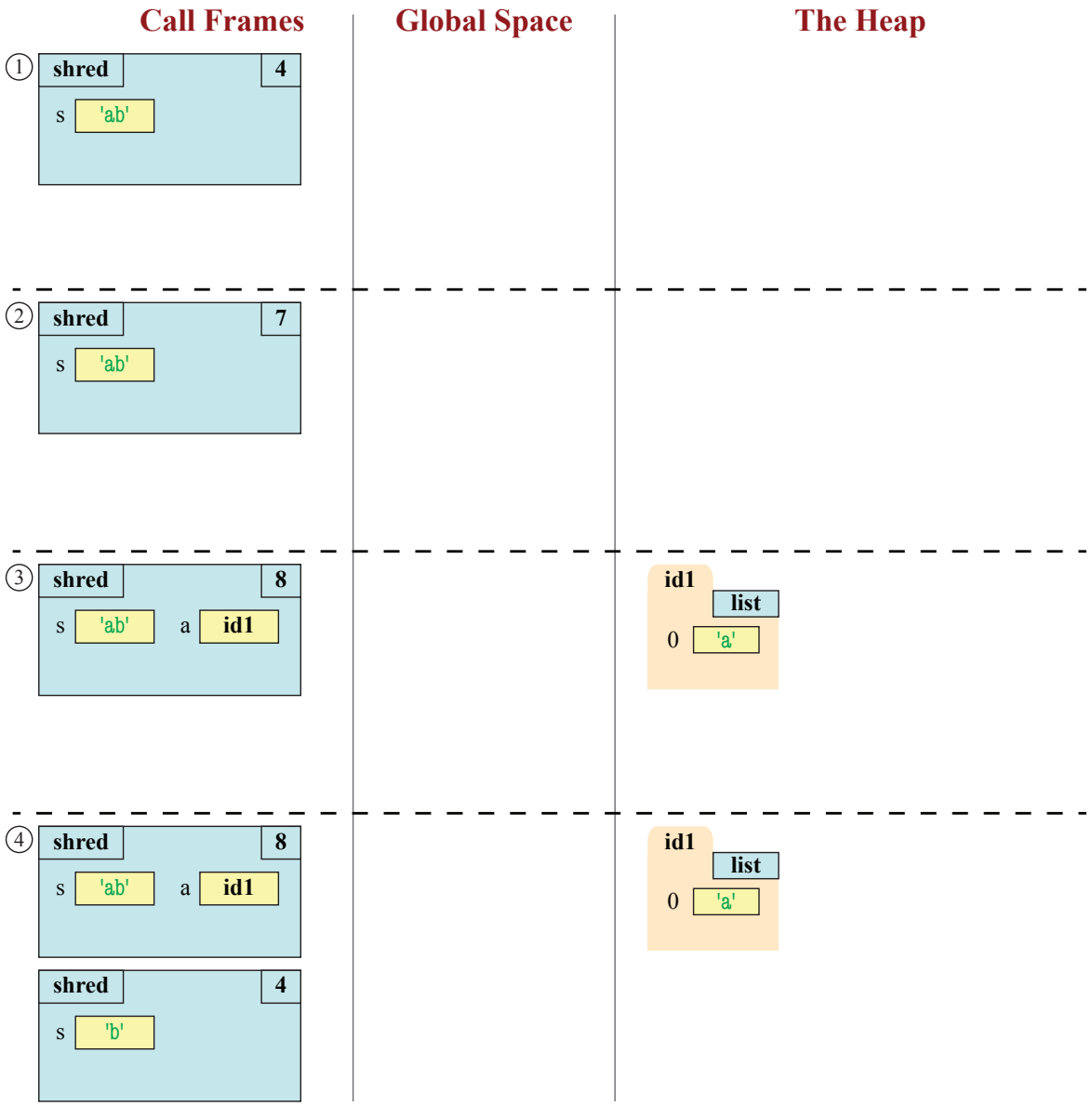
6. [14 points] **Call Frames**

Consider the recursive function `shred`, shown on the right. On this page amd the next diagram the execution of the assignment

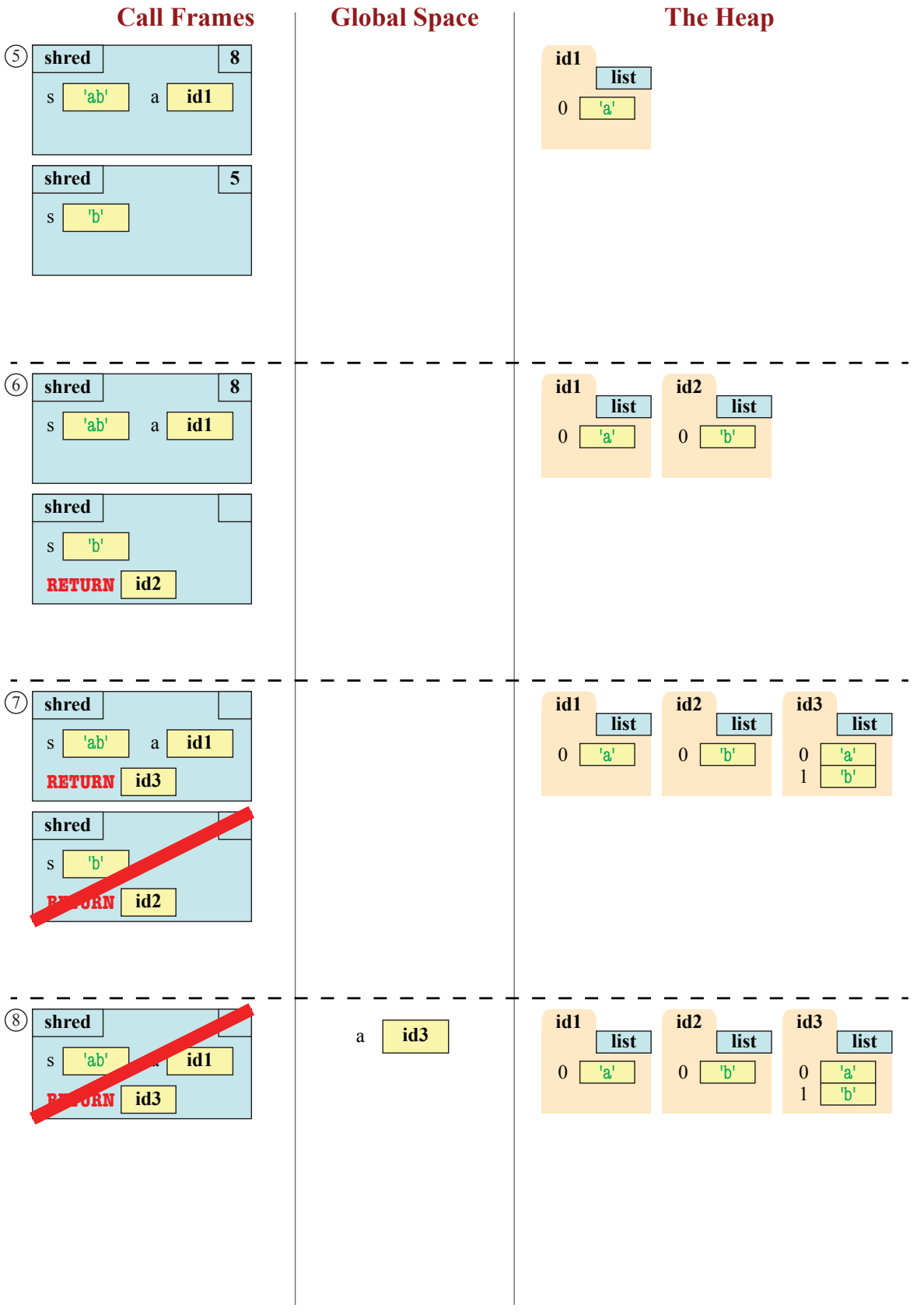
```
>>> a = shred('ab')
```

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 **eight** diagrams to draw. You may write *unchanged* in any of the three spaces if the space does not change at that step.

```
1 def shred(s):
2     """Splits a string up into a list
3     Precondition: s a nonempty string"""
4     if s[0] == s:
5         return [s[0]]
6     else:
7         a = [s[0]]
8         return a+shred(s[1:])
```



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

7. [12 points total] **Short Answer**

   (a) [3 points] What is a watch? What is a trace? What purpose do they serve?

   A *watch* is a print statement used to display the contents of a variable.
   A *trace* is a print statement used to visualize program flow.
   Both are useful in debugging code.

   (b) [3 points] What is the difference between `==` and `is`?

   The operator `is` compares the identifiers of the two objects/values. The operator `==` uses the `__eq__` of the object on the left to compare them (by default it is the same as `is`).

   (c) [3 points] Consider the function foo defined below, as well as the assignments to the right.

```
def foo():
    return 5
```

```
>>> x = foo()
>>> y = foo
```

   What are the values of the variables x and y and why?

   `x` contains 5; `foo()` is a function call.
   `y` contains the name of the function definition for `foo` in the heap.

   (d) [3 points] In the code below, you are given variables `x` and `n`. You can assume that they have already been initialized so that the initial assertion is true. Add code so that the later assertions are true (you may **not** reassign the variable `n`).

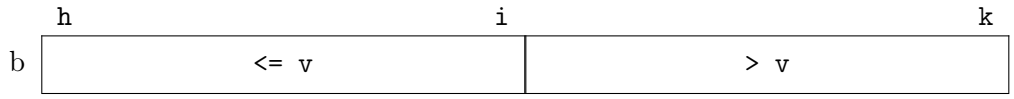
```
# Assertion: x is the product of all numbers 1..n
n = n + 1
x = x * n              # Insert code here

# Assertion: x is the product of all numbers 1..n

x = x * (n+1)          # Insert code here
n = n + 1
# Assertion: x is the product of all numbers 1..n
```

8. [15 points total] **Loop Invariants**

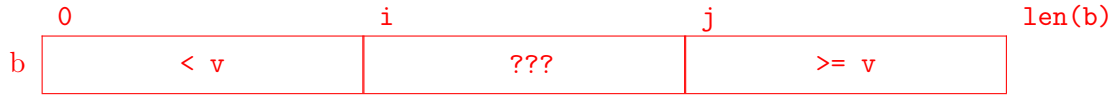
   On the next page are two versions of binary search. The one on the left is the tradition version. The one on the right (which is unfinished) is a version that looks for the **last occurrence of a value in the range** `b[h..k]`. Note that this function has a different precondition, postcondition and invariant.

Last Name: _____ First: _____ Netid: _____

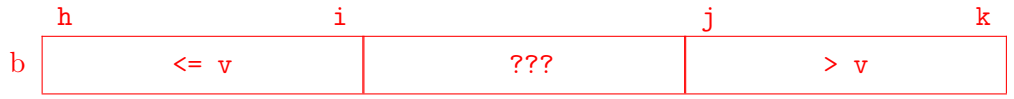
In particular, the postcondition of the function on the right is



(a) [2 points] Draw the horizontal notation representation for the loop invariant on the left.



(b) [2 points] Draw the horizontal notation representation for the loop invariant on the right.



(c) [11 points] Add the missing code to the function on the right. Note the important variable `isfound` at the end. **Solutions violating the loop invariant will not receive credit.**

```
def bin_search1(b,v):
    """Returns first pos of v in b
    Returns -1 if not found."""
    # pre: b[0..len(b)-1] sorted
    # Make invariant true at start
    i = 0
    j = len(b)

    # inv: b[0..i-1] < v, b[i..j-1] ???,
    #      b[j..] >= v

    mid = (i+j)//2

    while i < j:

        if b[mid] < v:
            i = mid+1

        else:             #  b[mid] >= v
            j = mid

        # Compute a new middle.
        mid = (i+j)//2

    # post: b[0..i-1] < v, b[i..] >= v

    isfound = (i < len(b) and b[i] == v)
    return i if isfound else -1
```

```
def bin_search2(b,v,h,k):
    """Returns last pos of v in b[h..k]
    Returns -1 if not found."""
    # pre: b[h..k] sorted
    # Make invariant true at start
    i = h-1
    j = k+1

    # inv: b[h..i] <= v, b[i+1..j-1] ???,
    #      b[j..k] > v

    mid = (i+j)//2

    while i < j-1                    :

        if b[mid] <= v:
            i = mid

        else:             #  b[mid] > v
            j = mid

        # Compute a new middle.
        mid = (i+j)//2

    # post: b[0..i] <= v, b[i+1..] > v

    isfound = (i >= h and b[i] == v)
    return i if isfound else -1
```