Last Name: _____ First: _____ Netid: _____

CS 1110 Final, December 10th, 2018

This 150-minute exam has 7 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	19	
3	19	
4	12	
5	12	
6	22	
7	14	
Total:	100	

Last Name: _____ First: _____ Netid: _____

References

String Functions and Methods

Function/Method	Description
`len(s)`	**Returns**: Number of characters in `s`; it can be 0.
`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.replace(a,b)`	**Returns**: A *copy* of `s` where all instances of `a` are replaced with `b`.
`s.upper()`	**Returns**: A *copy* of `s`, all letters converted to upper case.
`s.lower()`	**Returns**: A *copy* of `s`, all letters converted to lower case.
`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.

List Functions and Methods

Function/Method	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.pop()`	**Returns**: The last element of `x`, removing it from the list.
`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`).
`x.sort()`	Rearranges the elements of `x` to be in ascending order.

Other Functions

Function	Description
`range(a)`	**Returns**: An iterable for processing elements (0, 1, ..., `a`-1).
`range(a,b)`	**Returns**: An iterable for processing elements (`a`, `a`+1, ..., `b`-1).
`range(a,b,n)`	**Returns**: An iterable for elements (`a`, `a`+`n`, ..., `a`+`n`*((`b`-`a`-1)//`n`).
`isinstance(o,c)`	**Returns**: True if `o` is an instance of the class `c`; False otherwise.

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: _____

2. [19 points total] **Testing and Exceptions**

(a) [11 points] Consider the following function specification.

```
def is_matrix(value):
    """Returns True if value is a table of floats; False otherwise.

    A table is a list containing non-empty lists of the same length.
    Precondition: value is a non-empty list"""
```

Do not implement this function. Instead, provide a list of at least **six test cases** to test this function. For each test case provide: (1) the function input, (2) the expected output, and (3) an explanation of what makes this test *significantly* different.

There are many possible answers. However, these were the main tests that we had in mind.

Input	Output	Reason
`[1.0]`	`False`	A list that does not contain lists.
`[[1.0],1.0]`	`False`	A list that combines lists with non-lists.
`[[1.0],[1.0,2.0]]`	`False`	A list with elements of different lengths.
`[[],[]]`	`False`	A list containing only empty list.
`[['a',2.0]]`	`False`	A lists of lists with non-floats in it.
`[[1.0,2.0]]`	`True`	A table with a single row.
`[[1.0,2.0],[3.0,4.0]]`	`True`	A table with mutliple rows.

(b) [4 points] Given the function below, enforce its preconditions (but do **not** implement it) according to the specification. You should **not use assert statements**. Instead, write code that produces the errors specified. You may use the function above as a helper.

```
def transpose(value):
    """Swaps the columns and rows of the given matrix..
    Precondition: value is a valid matrix (2d table of floats)
    This function produces a TypeError if value is not a list,
    and a ValueError if it is a list but not a matrix."""

    if type(value) != list:
        raise TypeError()
    if value == [] or not is_matrix(value):
        raise ValueError()
```

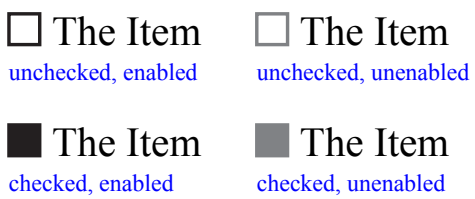
(c) [4 points] Implement the function below. You may use any of the previous two function as helpers. However, you **may not use if-statements**; you must use **try-except**.

```
def transpose_if_safe(value):
    """Transposes value IF it is a matrix; leaves it untouched otherwise.
    Precondition: value is a list (produces a TypeError if not a list)"""

    try:
        transpose(value)
    except ValueError:
        pass
```

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

For this question, you will use the classes of Assignment 7 to make a `GCheckbox`. Shown to the right, this is a label with a button that you can turn on and off. You can also set the checkbox as "unenabled" meaning that it is locked in position and the user can no longer click on it. A checkbox is built up of two graphical objects, a `GLabel` (the text) and a `GRectangle` (the button).



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. `GCheckbox` is a subclass of `GLabel` but it has a `_box` attribute for the `GRectangle`.

Since `GLabel` is a subclass of `GRectangle`, they have a lot of attributes in common.

Attribute	Invariant	Description
`x`	`float`	x-coordinate of the rectangle center.
`y`	`float`	y-coordinate of the rectangle center.
`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. Finally, `GLabel` has an additional attribute `text` which is the string displayed.

With this in mind, implement this class on the next page. We have provided the specifications for the methods `__init__`, `draw`, and `toggle`. You should fill in the missing details to meet these specifications. In addition, you must add the getters and setters (where appropriate) for the new attributes. Those setters must have preconditions to enforce the attribute invariants.

Hint: The attributes in `GLabel` and `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: `_box`, `_checked`, and `_enabled`. Also, remember that their constructors use keyword arguments (e.g. `GLabel(text='Hello')`).

```
class GCheckbox(GLabel):
    """A class representing a check-box style button

    MUTABLE ATTRIBUTES:
        _enabled: whether the check box is enabled  [bool]

    IMMUTABLE ATTRIBUTES:
        _checked: whether the check box is checked [bool]
        _box:     the check box to draw [GRectangle]

    There are two additional invariants:
    1. If _checked is true, then _box fillcolor is black (if _enabled is true) or grey
    (if _enabled is not). Also, if _checked is not true, then _box fillcolor is white.
    2. If _enabled is true, then _box linecolor is black. If not, the line color is grey."""

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

    def getBox(self):
        """Returns the check-box button"""
        return self._box

    def getChecked(self):
        """Returns whether the box is checked"""
        return self._checked

    def getEnabled(self):
        """Returns whether the box is enabled"""
        return self._enabled

    def setEnabled(self,value):
        """Sets whether the box is enabled"""
        assert type(value) == bool
        self._enabled = value
        if value:
            self._box.linecolor = 'black'
            if self._checked:
                self._box.fillcolor = 'black'
        else:
            self._box.linecolor = 'grey'
            if self._checked:
                self._box.fillcolor = 'grey'
```

Last Name: _____ First: _____ Netid: _____

```
# Class GCheckbox (CONTINUED).

def __init__( self, text, x, y, width, height, check = True ):      # Fill in
    """Initializes a check box with the given parameters.

    The values text, x, y, width, and height define the label of the check box.
    The button is a GRectangle to the left of the label. It is square whose width and
    height are the height of the label. Its RIGHT EDGE touches the LEFT EDGE of the
    label. The checkbox starts off enabled.

    Parameter text:   The check box text                        [str]
    Parameter x:      The x-coordinate of the text center       [int]
    Parameter y:      The y-coordinate of the text center       [int]
    Parameter width:  The width of the text label               [int]
    Parameter height: The height of the text label              [int]
    Parameter check:  Whether the button is initially checked   [bool]
    The argument check is OPTIONAL with default value False."""
    assert type(width) == int and type(height) == int  # Tech. needed but not counted
    assert type(x) == int and type(y) == int           # Tech. needed but not counted
    assert type(check) == bool
    super().__init__(text=text,x=x,y=y,width=width,height=height)
    bx = x-width/2-height/2
    self._box = GRectangle(x=bx,y=y,width=height,height=height)
    self._box.linecolor = 'black'
    self._box.fillcolor = 'black' if check else 'white'
    self._checked = check
    self._enabled = True


def draw( self, view ):                                             # Fill in
    """Draws this object to the given view.  Both the box and label are shown

    Parameter view: the view to draw to [GView]"""
    super().draw(view)
    self._box.draw(view)


def toggle( self ):                                                 # Fill in
    """Switches this checkbox from checked to not checked, or vice-versa.

    This method does NOTHING if the checkbox is not enabled."""
    if self._enabled:
        self._checked = not self._checked
        if self._checked:
            self._box.fillcolor = 'black'
        else:
            self._box.fillcolor = 'white'
```

Last Name: _____ First: _____ Netid: _____

4. [12 points] **Iteration**

Remember that a table is a 2-dimensional list of numbers. A table is square if the number of rows and columns are the same. You can create a *subtable* of a square table by drawing a square from one element to another on the diagonal. For example, the left side of **Figure 1** shows the subtable that starts at row 1 and has two rows and two columns.

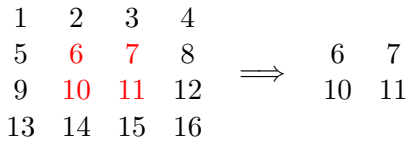


Figure 1: Subtable of size 2 at row 1

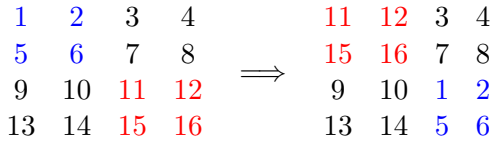


Figure 2: `table_swap(table,2,0,2)`

In the function below, you are asked to swap the contents of two subtables. For example, if `table` is shown on the left in **Figure 2**, then `table_swap(table,2,0,2)` produces the table on the right. Implement this function. **You do not need to assert preconditions.**

Hint: The subtables cannot overlap, so you can safely loop over the subtables.

```
def table_swap(table,n,k,h):
    """MODIFIES the list to swap the subtable (of n rows) at k with the one at h

    The subtable of size n at k is the table that starts at row k and column k,
    and continues for n columns to the right and n rows to the down.  See the
    picture above for a diagram of how this function works.

    Precondition: table a square 2d list of numbers. Either k+n <= h or h+n <= k
    (so the subtables do not overlap), 0 <= h, k and h+n, k+n < len(table)."""

    # Loop over an n x n subtable
    for x in range(n):
        for y in range(n):
            # Swap k+x,k+y with h+x,h+y
            tmp = table[k+x][k+y]
            table[k+x][k+y] = table[h+x][h+y]
            table[h+x][h+y] = tmp
```

Last Name: _____ First: _____ Netid: _____

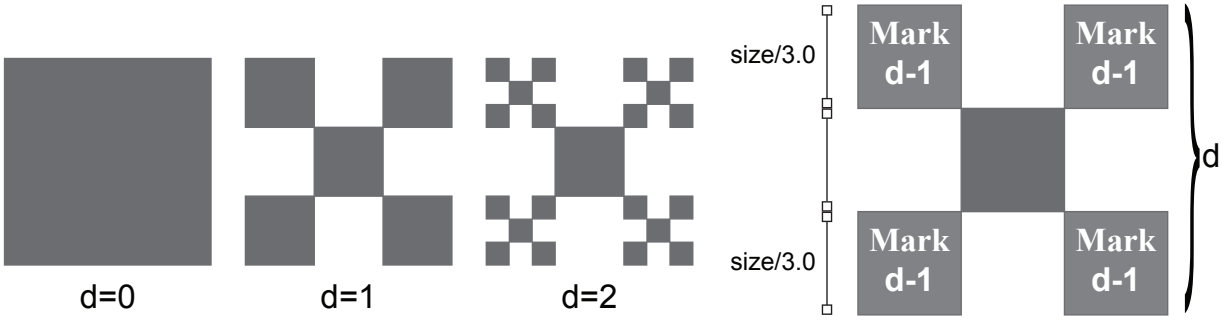
5. [12 points] **Recursion**

In Assignment 4, you had a helper function similar to the one below.

```
def fillsquare(w,x,y,size):
    """Draws a solid square with bottom left corner (x,y) and length size.

    Preconditions: w is a window, and x, y, and size are positive floats"""
```

You are going to use this helper to recursively draw the *cantor mark*. This shape is a square `X` where the corners get recursively fancy. The shape is defined as follows:



Define the function below. You do not need to create a `Window`, or worry about color. Simiply use the function `fillsquare` above to defined the shape. **Do not assert preconditions.**

```
def mark(w, x, y, size, d):
    """Draws a cantor mark of depth d with bottom left at (x,y) and length size.

    Preconditions: w a window; x, y, and size are floats > 0; d >= 0 an int"""
    if d == 0:
        fillsquare(w,x,y,size)
        return

    nsize = size/3
    fillsquare(w,x+nsize,y+nsize,nsize)
    mark(w,x,y,nsize,d-1)
    mark(w,x,y+2*nsize,nsize,d-1)
    mark(w,x+2*nsize,y,nsize,d-1)
    mark(w,x+2*nsize,y+2*nsize,nsize,d-1)
```

Last Name: _____ First: _____ Netid: _____

6. [22 points] **Call Frames**

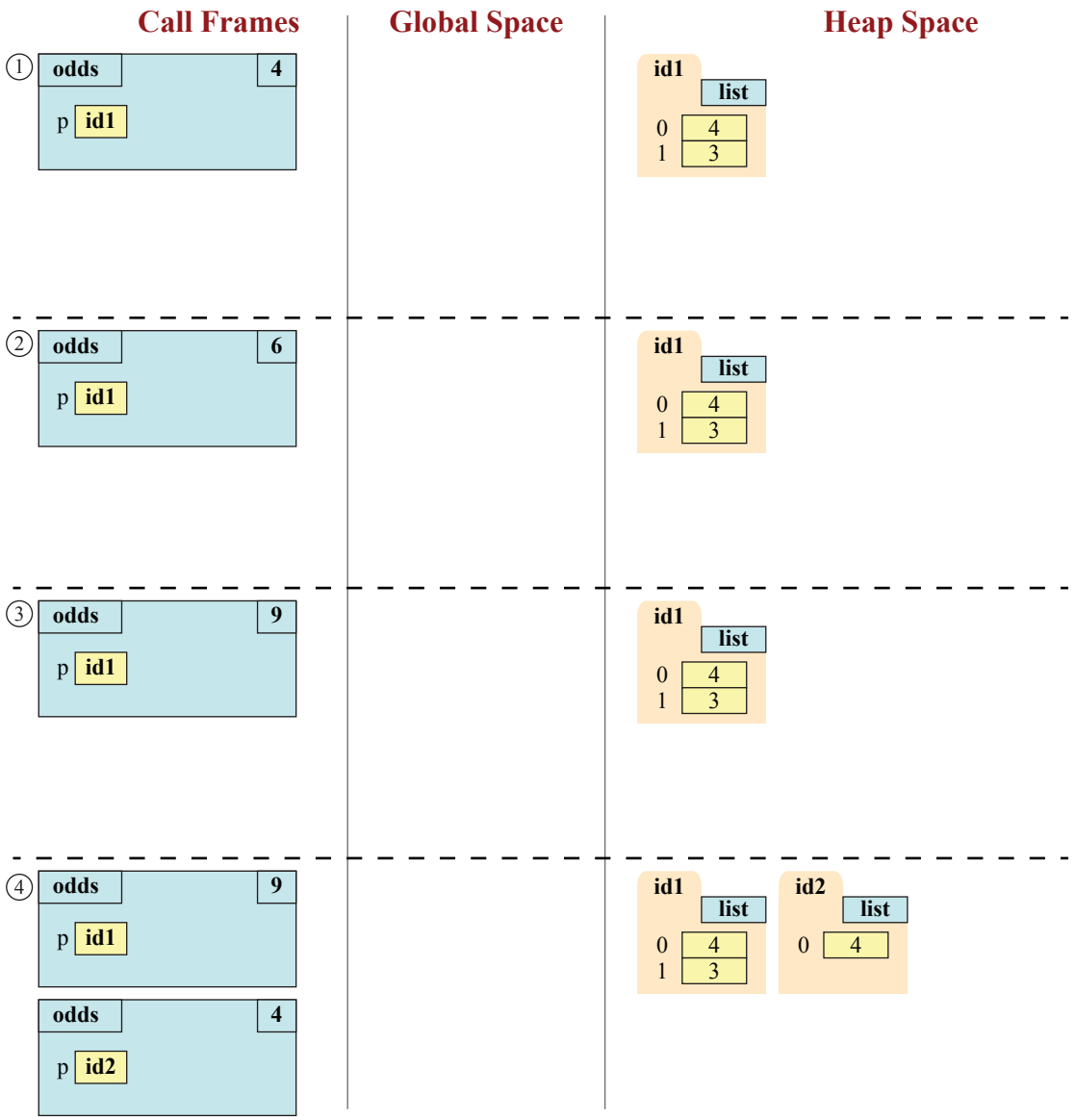
Consider the recursive function `odds`, shown on the right. On this page (and the next two) diagram the execution of the assignment

```
>>> p = odds([4,3])
```

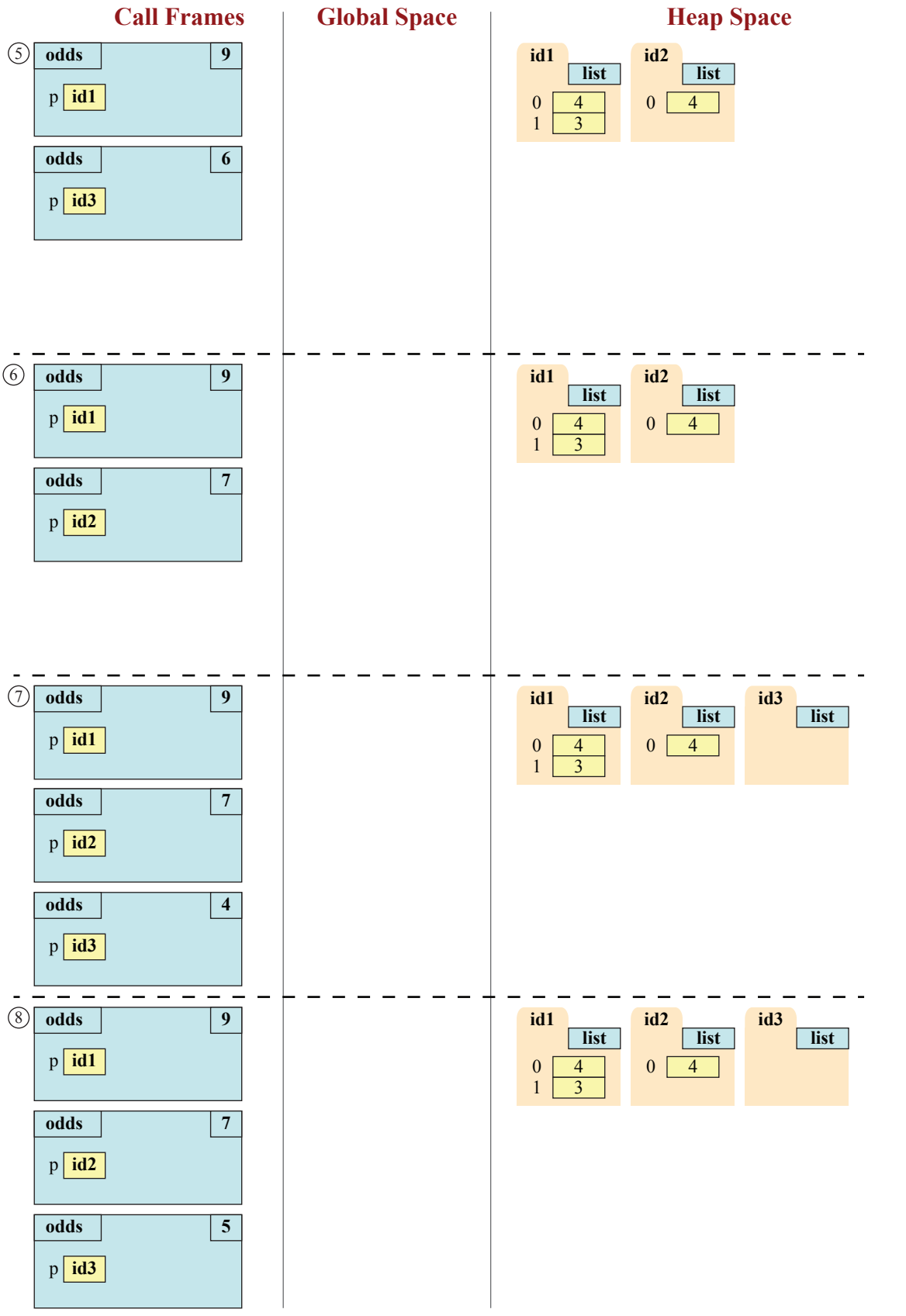
You should draw a new diagram every time a call frame is added or erased, or an instruction counter changes. There are a total of **twelve** diagrams to draw. You may write *unchanged* in any of the three spaces if the space does not change at that step.

```
1 def odds(p):
2     """Returns a copy of p with only odds
3     Precondition: p a list of ints"""
4     if len(p) == 0:
5         return []
6     elif p[-1] % 2 == 0:
7         return odds(p[:-1])
8     else:
9         return odds(p[:-1])+[p[-1]]
```

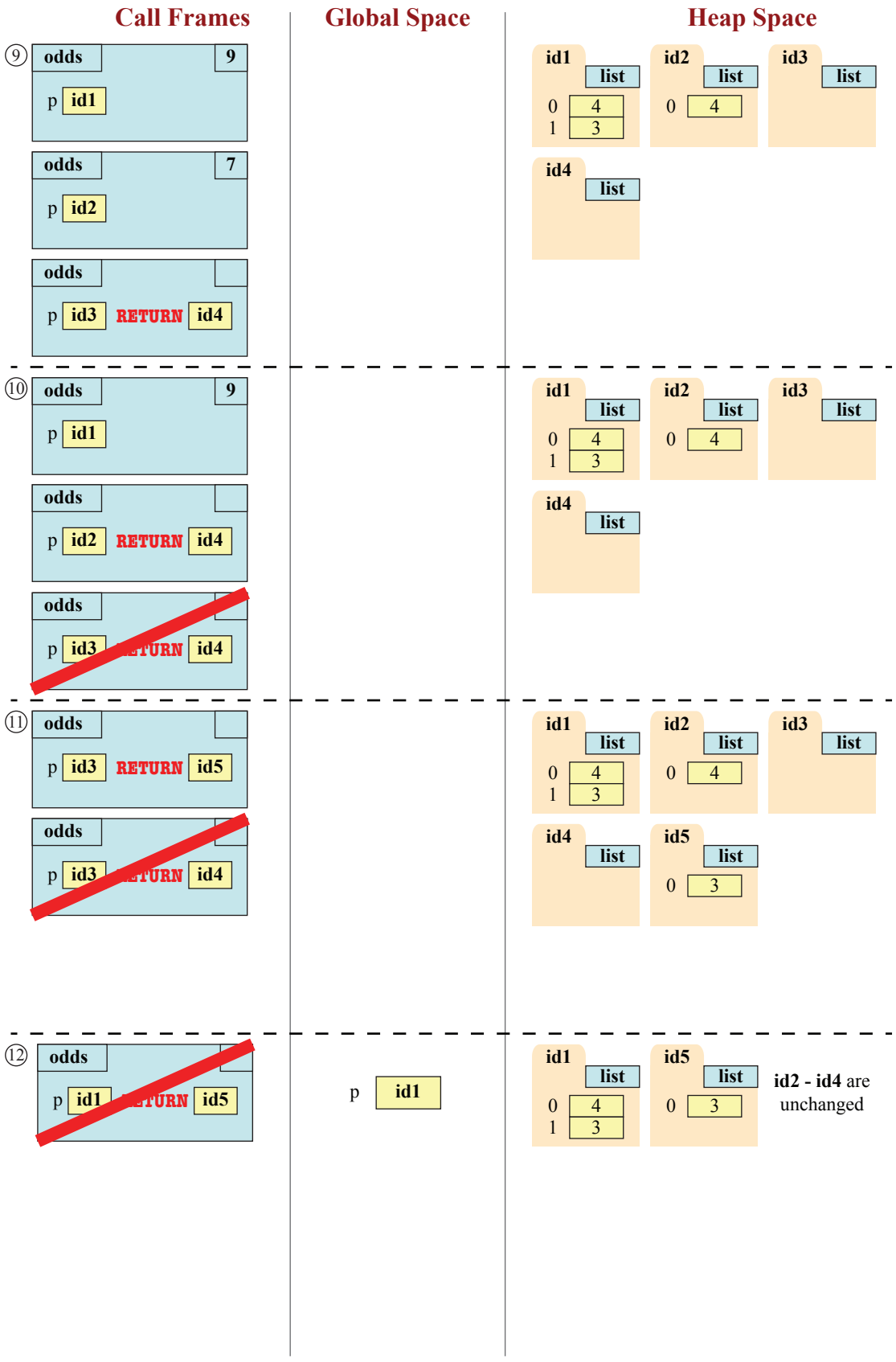
Hint: Remember that slicing a list makes a copy. Keep track of your folders.



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

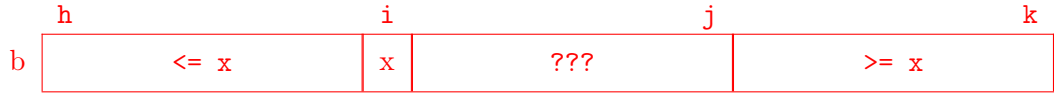


Last Name: _____ First: _____ Netid: _____

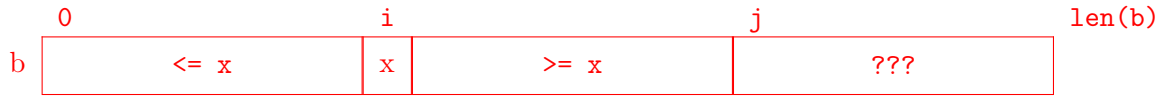
7. [14 points total] **Loop Invariants**

Below are two variations of the partition algorithm. The version on the left is completed for you. The version on the right has a different precondition, postcondition, and loop invariant. It is also missing the code for initialization, the loop condition, and the loop body.

(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. **Clearly indicate the positions of the ends.**



(c) [10 points] Add the missing code to the function on the right. Like the function on the left, you may use the helper function `swap(b,n,m)` to swap two positions in the list. **Solutions that violate the loop invariant will not receive credit.**

```
def partition1(b,h,k):
    """Partitions list b[h..k] at b[h]"""
    # pre: b[h..k] ???
    x = b[h]

    # Make invariant true at start

    i = h
    j = k

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

    while i < j:

        if b[i+1] >= x:
            swap(b,i+1,j)
            j = j - 1
        else:
            swap(b,i,i+1)
            i = i + 1



    # post: b[h..i-1] < x, b[i] is x,
    #       b[i+1..k] >= x
```

```
def partition2(b):
    """Partitions list b at b[0]"""
    # pre: b[0..] ???
    x = b[0]

    # Make invariant true at start

    i = 0
    j = 1

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

    while j < len(b)          :

        if b[j] >= x:
            j = j + 1
        else:
            swap(b,j,i)
            swap(b,j,i+1)
            i = i + 1
            j = j + 1



    # post: b[0..i-1] < x, b[i] is x,
    #       b[i+1..] >= x
```