Last Name: _____ First: _____ Netid: _____

CS 1110 Prelim 1 October 18th, 2020

This 90-minute exam has 5 questions worth a total of 100 points. Read over 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, to look at any other reference material, or to 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 loops or recursion on this exam. Beyond that, you may use any Python feature that you have learned in class (if-statements, try-except, lists), **unless directed otherwise**.

Question	Points	Score
1	2	
2	23	
3	26	
4	24	
5	25	
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

Throughout this exam you will be asked questions about strings and lists. You are expected to understand how slicing works. In addition, the following functions and methods may be useful.

String Functions and Methods

Expression or Method	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.count(s1)`	**Returns**: the number of times `s1` occurs in `s`
`s.find(s1)`	**Returns**: index of the first character of the FIRST occurrence of `s1` in `s` (-1 if `s1` does not occur in s).
`s.find(s1,n)`	**Returns**: index of the first character of the first occurrence of `s1` in `s` STARTING at position n. (-1 if `s1` does not occur in s from this position).
`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.
`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).

List Functions and Methods

Expression or 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.count(y)`	**Returns**: the number of times `y` occurs in `x`
`x.index(y)`	**Returns**: index of the FIRST occurrence of `y` in `x` (an error occurs if `y` does not occur in `x`).
`x.index(y,n)`	**Returns**: index of the first occurrence of `y` in `x` STARTING at position n (an error occurs if `y` does not occur in `x`).
`x.append(y)`	Adds `y` to the end of list `x`.
`x.insert(i,y)`	Inserts `y` at position `i` in list `x`, shifting later elements to the right.
`x.remove(y)`	Removes the first item from the list whose value is `y` (an error occurs if `y` does not occur in `x`).

The last three list methods are all procedures. They return the value `None`.

Last Name: _____ First: _____ Netid: _____

2. [23 points total] **Objects and Functions**.

   Remember the class `RGB` from Assignment 3. Objects of this class have three attributes: `red`, `green`, and `blue` (we will ignore `alpha` for this question). These values must be integers between 0 and 255; assigning any other value to them will result in an error.

   (a) [9 points] When we multiply two colors together, we do it attribute by attribute. That is, we first convert the attributes to the range 0..1 (as in the assignment) and then multiply each attribute like this

   $$C' = C_1 * C_2 \qquad \textbf{Multiply the color attributes } C_1 \textbf{ and } C_2$$

   where C_i is **each one of** `red`, `green`,or `blue`. We then convert each value to the range 0..255 when done (remembering to round the result). With that in mind, implement the function below.

   NOTE: You do not need to worry about importing `introcs`. Assume it is available.

```
def multiply(color1,color2):
    """Returns a new RGB object that is the product of color1 and color2

    The original colors should not be modified

    Preconditions: color1 and color2 are RGB objects"""

    # Compute each color in range 0..1
    red = color1.red/255.0*color2.red/255.0
    green = color1.green/255.0*color2.green/255.0
    blue = color1.blue/255.0*color2.blue/255.0

    # Convert back
    red = int(round(red*255))
    green = int(round(green*255))
    blue = int(round(blue*255))

    # Make a new object
    return RGB(red, green, blue)
```

(b) [14 points] The opposite of multiplication is division. To divide two colors, we again convert the attributes to the range 0..1 (as in the assignment) and then use the formula

$$C' = \begin{cases} 0 & \text{if } C_1 = 0 \\ C_1/C_2 & \text{if } C_1/C_2 \leq 1 \\ 1 & \text{otherwise} \end{cases}$$

where C_i is **each one of** `red`, `green`,or `blue`, converting back to the range 0..255 when done (remembering to round the result). Using this formula, implement the function below.

Note: In mathematics, $a/0$ is ∞ whenever $a > 0$. That is implicit in the formula above.

```
def divide(color1,color2):
    """MODIFIES color1 to store the result of division.

    This function divides color1 by color2 and stores the result in color1.

    Preconditions: color1 and color2 are RGB objects"""

    # Compute each color in range 0..1
    r1 = color1.red/255.0
    r2 = color2.red/255.0
    g1 = color1.green/255.0
    g2 = color2.green/255.0
    b1 = color1.blue/255.0
    b2 = color2.blue/255.0

    # Compute red, handling divide by 0
    if r2 != 0 and r1/r2 <=1:
        r1 = r1/r2
    elif r1 > 0:
        r1 = 1

    # Compute green, handling divide by 0
    if g2 != 0 and g1/g2 <=1:
        g1 = g1/g2
    elif g1 > 0:
        g1 = 1

    # Compute blue, handling divide by 0
    if b2 != 0 and b1/b2 <=1:
        b1 = b1/b2
    elif b1 > 0:
        b1 = 1

    # Convert and assign to color1
    color1.red = int(round(r1*255))
    color1.green = int(round(g1*255))
    color1.blue = int(round(b1*255))

    # NO return statement
```

Last Name: _____ First: _____ Netid: _____

3. [26 points total] **Testing and Debugging**.

   (a) [10 points] The function `romanize` takes a integer 1..99 and converts it into a Roman numeral such as I, VII, or XIX. Roman numerals are represented as strings with the letters `'I'`, `'V'`, `'L'`, `'X'`, and `'C'`.

   There are ***at least* three bugs** in the code below. These bugs are potentially spread across multiple functions. To help find the bugs, we have added several print statements throughout the code, and show the results on the next page. Using this information as a guide, identify and fix the three bugs on the next page. Your fixes may include more than one line of code. You should explain your fixes.

```
1  def romanize(n):
2      """Returns the Roman numeral for n
3
4      Precond: 0 < n < 100 is an int"""
5      tens = ''
6      ones = ''
7      if n >= 50:
8          print('More than 50')      # TRACE
9          tens = numeralL(n//10)
10     elif n >= 10:
11         print('More than 10')      # TRACE
12         tems = numeralX(n//10)
13     print('tens = '+repr(tens))    # WATCH
14
15     ones = romanize1to9(n % 10)
16     print('ones = '+repr(ones))    # WATCH
17
18 def romanize1to9(n):
19     """Returns the Roman numeral for n
20
21     Precond: 0 < n < 10 is an int"""
22     # Combined TRACE and WATCH
23     print('romanize1to9: n = '+repr(n))
24     if n < 5:
25         print('Less than 5')       # TRACE
26         return romanize1to4(n)
27     elif n < 9:
28         print('Between 5 and 8')   # TRACE
29         return 'V'+romanize1to4(n-5)
30     else:
31         print('Equal to 9')        # TRACE
32     return 'IX'
33
34 def romanize1to4(n):
35     """Returns the Roman numeral for n
36
37     Precond: 0 < n < 5 is an int"""
38     # Combined TRACE and WATCH
39     print('romanize1to4: n = '+repr(n))
40     values = ['I','II','III','IV']
41     choiceI = values[n-1]
42     # WATCH
43     print('choiceI = '+repr(choiceI))
44     return choiceI
45
46
47
48 def numeralL(n):
49     """Returns Roman numeral for tens value
50
51     The value n is the tens DIGIT of the
52     number. So numeralL(5) is 'L'.
53
54     Precond: 5 <= n < 10 is an int"""
55     # Combined TRACE and WATCH
56     print('numeralL: n = '+repr(n))
57     if n < 9:
58         print('Less than 90')      # TRACE
59         return 'L'+numeralX(n-5)
60     else:
61         print('Equals to 90')      # TRACE
62         return 'XC'
63
64
65
66
67
68 def numeralX(n):
69     """Returns Roman numeral for tens value
70
71     The value n is the tens DIGIT of the
72     number. So numeralL(3) is 'XXX'.
73
74     When n is 0, it returns the empty
75     string (to be compatible w/ numeralL)
76
77     Precond: 0 <= n < 5 is an int"""
78     # Combined TRACE and WATCH
79     print('numeralX: n = '+repr(n))
80     values = ['','X','XX','XXX','XL']
81
82     choiceX = values[n]
83     # WATCH
84     print('choiceX = '+repr(choiceX))
85     return choiceX
86
87
88
```

Last Name: _____ First: _____ Netid: _____

Hint: Some bugs cannot be fixed with just one line. You might need to add a conditional.

Tests:

```
>>> romanize(14) # Expected: 'XIV'
More than 10
numeralX: n = 1
choiceX = 'X'
tens = ''
romanize1to9: n = 4
Less than 5
romanize1to4: n = 4
choiceI = 'IV'
ones = 'IV'
IV
```

```
>>> romanize(75) # Expected: 'LXXV'
More than 50
numeralL: n = 7
Less than 90
numeralX: n = 2
choiceX = 'XX'
tens = 'LXX'
romanize1to9: n = 5
Between 5 and 8
romanize1to4: n = 0
choiceI = 'IV'
ones = 'VIV'
LXXVIV
```

```
>>> romanize(60) # Expected: 'LX'
More than 50
numeralL: n = 6
Less than 90
numeralX: n = 1
choiceX = 'X'
tens = 'LX'
romanize1to9: n = 0
Less than 5
romanize1to4: n = 0
choiceI = 'IV'
ones = 'IV'
LXIV
```

First Bug:

This is a straight-forward misspelling error. The variable `tens` is mispelled as `tems` on **Line 12**, causing `romanize` to fall back to the original assignment on Line 5. To fix it, just rewrite Line 12 as

```
tens = numeralX(n//10)
```

Second Bug:

The function `romanize1to4` is being called in such a way that the precondition is violated. That is because `romanize1to9` does not handle the case `n == 5` properly. We need to add the following code to **Line 27** of `romanize1to9` (just before the **elif**

```
elif n == 5:
    return 'V'
```

We realized that this fix requires some familiarity with Roman numerals (though the bug can be found from the preconditions alone). Therefore, we gave students full credit just for realizing it needed a special case for `n == 5` even if the fix was not correct.

Third Bug:

The is *another* precondition violation of the function `romanize1to4`. But this time we see that the precondition of `romanize1to9` is also violated. So the problem is in the top level `romanize`. The problem is that we do not need to call `romanize1to9` at all if there is nothing in the ones position. So we should change **Line 15** to

```
if n % 10 > 0:
    ones = romanize1to9(n % 10)
```

Fourth Bug:

We (unintentionally) forgot to add a return statement to `romanize`. Discovering this could be used to replace one of the bugs above.

(b) [10 points] Consider the specification below

```
def count_adjacent(a,b):
    """Returns the number of adjacent pairs of b inside of the string a.

    An adjacent pair just means that two copies of b appear next to each other
    in a. For example, if b = 'b' it appears twice as an adjacent pair in the
    string 'abbcbbdb'.

    Preconditions: a and b are both nonempty strings of lowercase letters.
```

Do not implement this function. Instead, we want you to write at least **six test cases** below. By a test case, we just mean an input and an expected output; you do not need to write an `assert_equals` statement. For each test case, you should explain why it is substantially different from the others.

There are many answers to this question. Here are some of the ones we had in mind

Input	Output	Reason
a='abbc', b='x'	0	b is not in a at all
a='abbc', b='c'	0	b is not an adjacent pair of a
a='abbc', b='b'	1	b is appears once as an adjacent pair
a='abbcbb', b='b'	2	b appears as multiple, non-overlapping pairs
a='abbbc', b='b'	2	b appears as overlapping pairs
a='abcbccbc', b='bc'	1	b is a multi-letter adjacent pair

(c) [6 points] **Do not implement the function specified below.** Instead, use assert statements to enforce the precondition. Furthermore, each the assert statement should produce one of the three error messages shown below

```
>>> after_space(13)
AssertionError: 13 is not a string.
>>> after_space('abc')
AssertionError: 'abc' has no spaces.
>>> after_space(' abc')
AssertionError: ' abc' has an illegal space.
```

```
def after_space(s):
    """Returns the part of the string after the first space in s

    Precond: s a string with at least one space.
    Furthermore, s does not start or end with a space."""

    assert type(s) == str, repr(s)+' is not a string.'
    assert ' ' in s, repr(s)+' has no spaces.'
    assert s[0] != ' ' and s[-1] != ' ', repr(s)+' has an illegal space.'
```

4. [24 points] **Call Frames.**

Consider the following function definitions.

```
1  def fold_front(p):
2      """Returns sum of first 2 items
3      This function modifies the list
4      Pre: p is a list, len(p) >= 2"""
5      p[0]= p[0]+p[1]
6      return p[0]
7
```

```
8  def sum_back(q):
9      """Returns sum of last 2 items
10     Pre: q is a list, len(q) >= 2"""
11      r = q[-2:]
12      result = fold_front(r)
13      return result
14
```

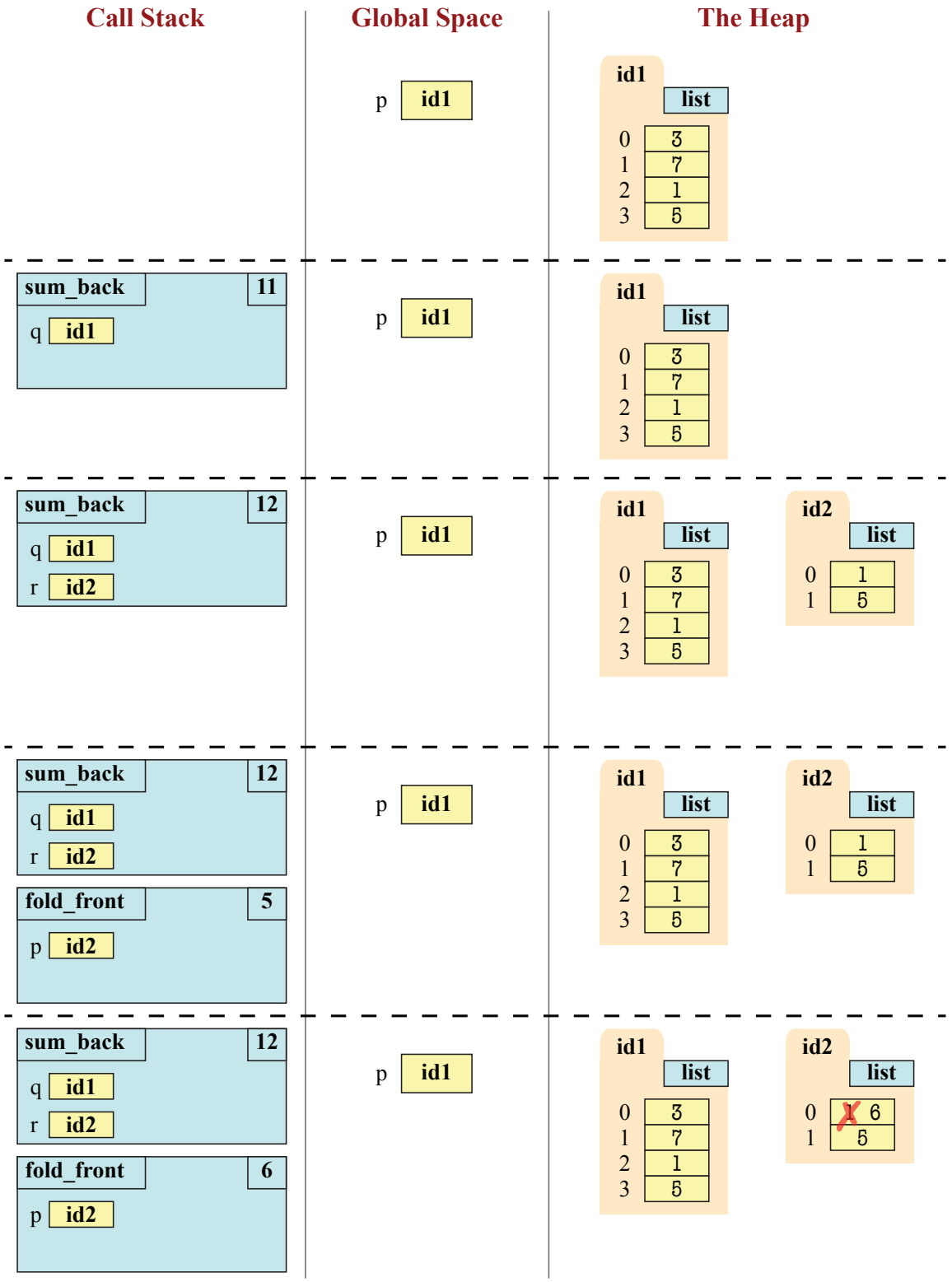
Assume `p = [3, 7, 1, 5]` is a global variable referencing a list in the heap, as shown on the next page. On the next two pages, diagram the evolution of the call

```
r = sum_back(p)
```

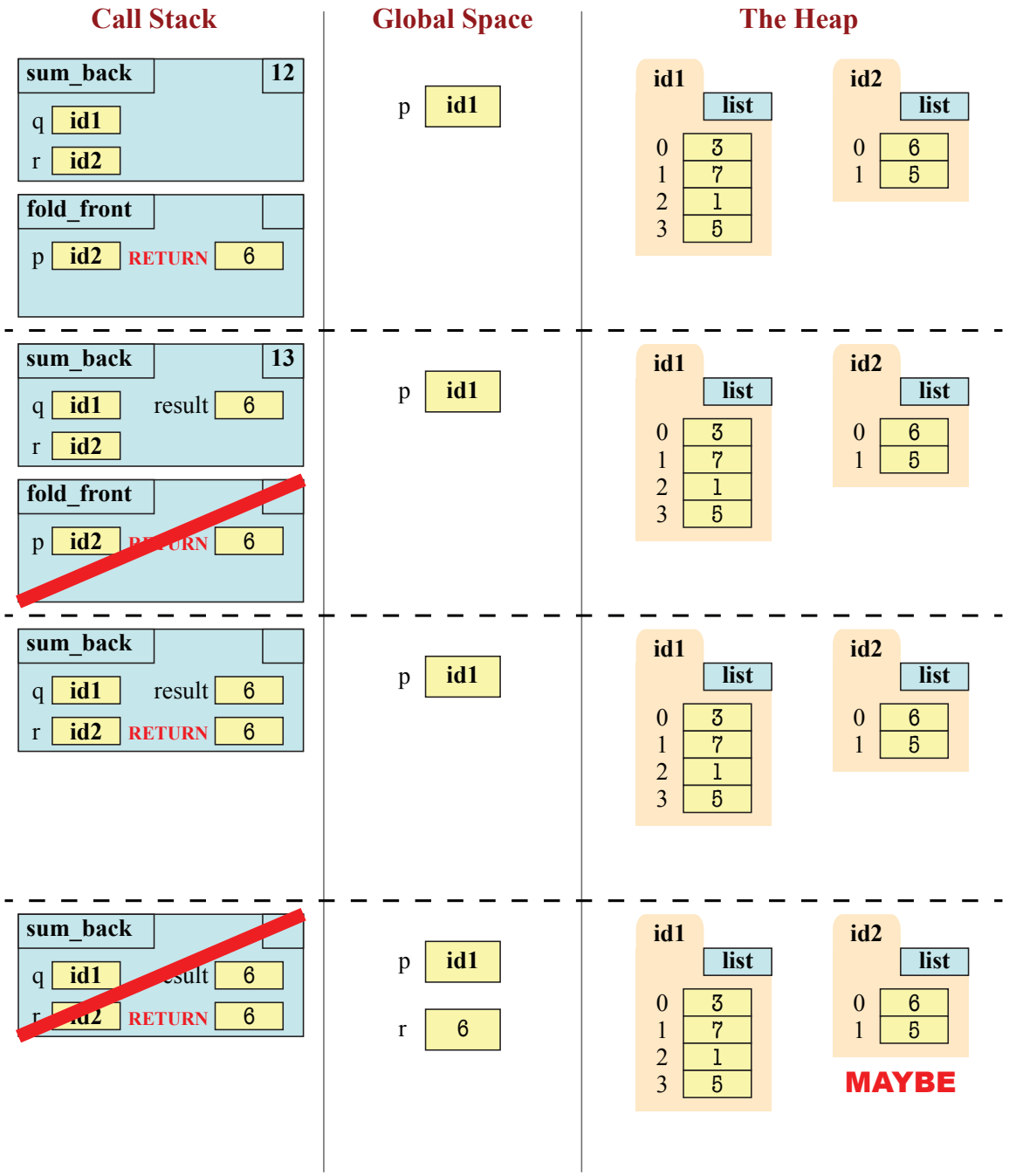
Diagram the state of the *entire call stack* for the function `sum_back` when it starts, for each line executed, and when the frame is erased. If any other functions are called, you should do this for them as well (at the appropriate time). This will require a total of **eight** diagrams, not including the (pre-call) diagram shown.

You should draw also the state of global space and the heap at each step. You can ignore the folders for the function definitions. Only draw folders for lists or objects. You are also allowed to write "unchanged" if no changes were made to either global space or the heap.

Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____



Last Name: _____ First: _____ Netid: _____

5. [25 points] **String Slicing**.

   Implement the function below. You may need to use several of the functions and methods on the reference page. Pay close attention to the examples to better understand the function.

```
def swap_first(word,a,b):
    """Returns a copy of word with first instance each of a and b swapped

    If either a or b is not a substring of word, then word is unchanged.
    If a and b overlap inside of word, then the word is also unchanged.

    Examples:
        swap_first('aBcD','B','c') returns 'acBD'
        swap_first('aBcD','c','B') returns 'acBD'
        swap_first('aBcD','x','c') returns 'aBcD'
        swap_first('aBCdeF','BC','de') returns 'adeBCF'
        swap_first('aBCdeF','BC','Cd') returns 'aBCdeF'

    Preconditions: word, a, and b are all non-empty strings"""
    firsta = word.find(a)
    firstb = word.find(b)

    # Do nothing if missing
    if firsta == -1 or firstb == -1:
        return word

    lasta = firsta+len(a)
    lastb = firstb+len(b)

    if firsta < firstb and lasta <= firstb:
        # a comes first, no overlap
        head = word[:firsta]
        tail = word[lastb:]
        mids = word[lasta:firstb]
        return head+b+mids+a+tail
    elif firstb < firsta and lastb <= firsta:
        # b comes first, no overlap
        head = word[:firstb]
        tail = word[lasta:]
        mids = word[lastb:firsta]
        return head+a+mids+b+tail
    else:
        # overlap
        return word
```