Last Name: UTION First Name: SOL Cornell NetID: CS1110

Circle your lab: Tu 12:20 red/orange Tu 12:20 blue Tu 1:25 Tu 2:30 Tu 3:35

W 12:20 W 1:25 red/orange W 1:25 blue W 2:30 W 3:35

Solution: CS 1110 Prelim 1 March 11th, 2014

1. [2 points] When allowed to begin, write your last name, first name, and Cornell NetID at the top of *each* page, and circle your lab time on the top of the first page of the exam.

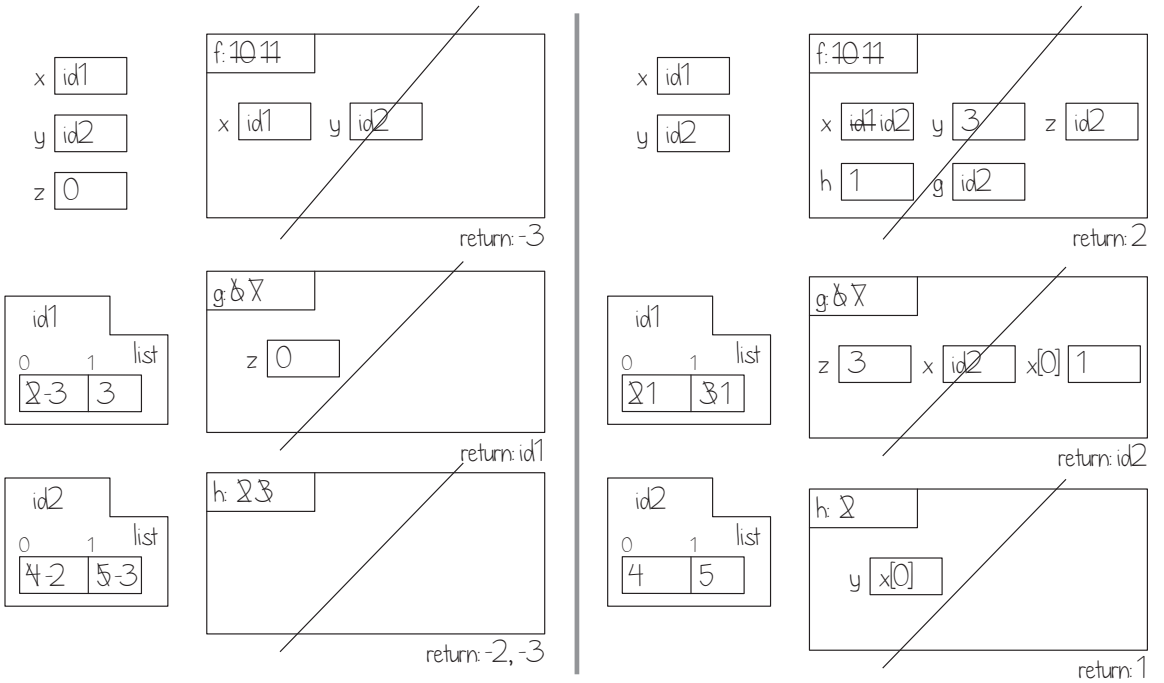
   Solution: Every time a student doesn't do this, somewhere, a kitten weeps.

   More seriously, we sometimes have exams come apart during grading, so it is actually important to write your name on each page. Also, remember that if we need to figure out your lab section at the end of the grading session, our chances of putting it the wrong pile, and thus you not being able to find it when you get to lab, grow high.

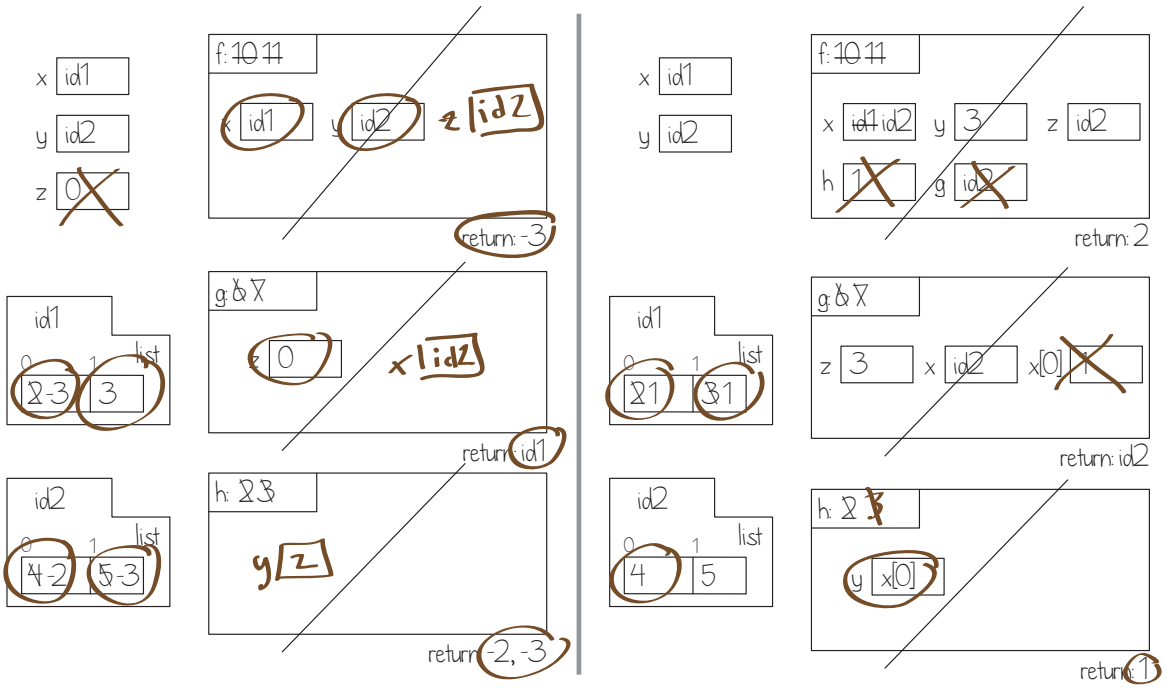
2. [12 points] Two students were assigned to diagram the execution of the following code.

```
1   def h(y):
2       y = x[1] - y
3       return y
4
5   def g(z, x):
6       x[0] = x[0] - z
7       return x
8
9   def f(x, y, z):
10      x = g(y, z)
11      return h(x[0])
12
13  x = [2,3]
14  y = [4,5]
15  x[1] = f(x, 3, y)
```

   After executing the whole program they handed in the diagrams below. You are their grader; please mark up each student's solution as follows: (1) Draw an X through anything that is present that should not be; (2) circle anything that is present and should be, but has the wrong value or name; and (3) write in anything that is missing. You may wish to do this question by first drawing the relevant frames and objects yourself.



Solution: -.5 for each incorrect thing For items where something needed to be added, for example a variable, one gets the point even if the value inside is incorrect. (This way, there is even weight to adding something or crossing something off.)



3. [13 points] Implement `reshmuplify` and `lower_one_char` so that they meet their specifications. Your solution can call the function `find_first_vowel` below; you may assume it's already been implemented according to its specification.

```
def find_first_vowel(s):
    """Return: the index of the first vowel in s, or -1 if s has no vowels."""
    # Assume this function has already been implemented correctly, so you can call it.

def reshmuplify(s):
    """Return: the string s followed by a ', ' and then the same string with the substring
    before the first vowel replaced by 'shm'.
    Precondition:
        s contains only letters
        s contains at least one vowel.
    Examples:
        'apple' -> 'apple, shmapple'
        'banana' -> 'banana, shmanana'
        'prelim' -> 'prelim, shmelim'
    """
```

Solution:

```
    i = find_first_vowel(s)
    return s + ', shm' + s[i:]
```

+1 find_first_vowel on `s`

+1 `s` first

+1 includes the ', shm'

+2 string slice starting at first vowel, ending at string's end

+1 concatenation

"Reshmuplification" actually has been discussed the linguistics literature. As Geoffrey Pullum wrote in an article called "Footloose and context-free", collected in the wonderful book *The Great Eskimo Vocabulary Hoax and Other Irreverent Essays on the Study of Language*, Alexis Manaster-Ramer posited that "one could develop [an argument that natural language lies outside the class of languages known as "context-free"] using the contemptuous reduplication of Yiddish-influenced English." According to Pullum, Manaster-Ramer published this in the Proceedings of the Chicago Linguistics Society 19, 256-262 in a paper "The soft formal underbelly of theoretical syntax."

```
def lower_one_char(s, n):
    """Return: the string s with the character at index n
    converted to lowercase.
    Precondition:
        s is a string containing only letters, and has at least one character in it.
        n is a valid index into s.
        n > 0 and n < len(s) - 1.
    Examples:
        lower_one_char('FOO', 1) -> 'FoO'
        lower_one_char('HELLo', 2) -> 'HElLo'
        lower_one_char('HELLo', 4) -> 'HELLo'
        lower_one_char('TEST', 0) violates the precondition
        lower_one_char('TEST', 3) violates the precondition
    """
```

Solution:

```
return s[:n] + s[n].lower() + s[n+1:]
```

The third example we gave was actually invalid because it violated the precondition, as was announced during the exam.

+2 string slice starting at 0, last index n

+1 correct reference to s[n]

+1 correct format for call to lower

+2 string slice starting at n+1, going to end of string

+1 concatenation

4. [13 points] Assume that inside a module named `transcript2` is the definition of class `Titem2`, which is an extension of class `Titem` from Lab 3. Titem2s have the following attributes:

`name`	non-empty string of lowercase letters followed by numbers
`credits`	positive `int`
`gradeval`	positive `float`

and are created by calls like `transcript2.Titem2('this should have said cs1CS1',4,'A+')` (if `transcript2` has been imported).

We define the *GPA contribution* of a Titem2 as its number of credits times its gradeval. For example, for the Titem2 created by the call above, the GPA contribution is $4 \times 4.3 = 17.2$.

Complete the function definition below so that it meets its specification. *Note that we already wrote a line of code for you.*

```
def separate(sourcelist, threshold, highlist, lowlist):
    """Appends to list highlist the Titem2s from sourcelist whose GPA
    contribution [see definition above] is greater than or equal to threshold,
    and appends to list lowlist the other items in sourcelist.
    Precondition:
      sourcelist is a (possibly empty) list of Titem2s
      threshold is a float or int
      highlist and lowlist are (possibly non-empty) lists."""
    # For an example, see text at the bottom of this page.

    # **MAKE SURE YOU SEE THIS LINE, AND INDENT RELATIVE TO IT**
    for item in sourcelist:
```

Solution

```
        if item.credits*item.gradeval >= threshold:
        ## +1 use of if
        ## +1 correct inequality with threshold
        ## +1 access of attribute credits
        ## +1 used "item." for credits
        ## +1 access of attribute gradeval
        ## +1 used "item." for gradeval
            highlist.append(item)
            ## +1 use of append
            ## +1 mention of highlist
            ## +1 list name goes in front of the dot
            ## +1 mention of item
            ## +1 thing being appended goes in parens
        else:
        ## + 1 use of an else
            lowlist.append(item)
            ## If something here is inconsistent with the
            ## understanding in the previous list access line,
            ## the corresponding point granted above should be deducted
        ## +1 no return statement anywhere
```

Illustrative example: let id1 be the ID of a Titem2 with GPA contribution 8, and id2 be the ID of a Titem2 with GPA contribution 16. Suppose x is a 2-item list holding id1 and id2, y is an empty list, and z is the list `[7]`. Then, the result of the call `separate(x, 10, y, z)` is that:

`x` remains the same;
The list that `y` refers to is modified to be a 1-item list containing id2;
The list that `z` refers to is modified to be a 2-item list containing the number 7 and id1.

5. [5 points] Assume that `separate` from the previous question is (correctly) defined in module `prelim1`. Now, suppose the following sequence of statements is executed.

```
import transcript2
import prelim1
nextlist = [transcript2.Titem2('class1', 12, "A+"),
            transcript2.Titem2('class2', 1, "B-"),
            transcript2.Titem2('class3', 1, "B")]
high = []
low = [transcript2.Titem2('engl9999', 2, "B-"),
       transcript2.Titem2('is666', 2, "B-")]
prelim1.separate(nextlist, 12, high, low)
high[0].name = 'FAKE'
print nextlist[0].name
```

Write down what the resulting printout(s) or error(s) are. Then explain your answer in one to three sentences.

Solution: The word 'FAKE' gets printed out. Function `separate` will append nextlist[0] in high at index 0, so nextlist[0] and high[0] are references to the same object. Hence a change to high[0] is a change to nextlist[0].

+1 for understanding that nextlist[0] will be appended to the end of high
+3 explaining that this means high[0] and nextlist[0] are the same thing. (This is probably all or none)
+1 for noting that the change to high[0] therefore is also a change to nextlist[0]
-1 for each clearly false statement
So, *no credit* for guessing that the output will be 'FAKE' if the justification is not correct.

Note: we gave a point for noting that assigning 'FAKE' as the name of a Titem2 violates the precondition, and so is a conceptual error. However, it does *not* cause Python to halt, so the last line is executed.

6. [5 points] Complete this function definition according to its specification. One or two lines of code suffices.

```
def avg(inlist):
   """Returns: float value of the average of the values in list inlist.

   Pre: inlist a non-empty list, each item either an int or a float."""
   # The average of a list of numbers is the sum of the values in the list
   # divided by the length of the list.
```

Solution:

```
    return sum(inlist)/float(len(inlist))
    ## +1 sum
    ## +1 len
    ## +1 inlist is the argument for sum
    ## +1 inlist is the argument for len
    ## +1 correct type conversion
```

7. [5 points] Consider the following function definition, which makes use of the `avg` function from the previous question.

```
def string_avg(nums_as_str):
    """Returns: float value of the average of the values represented
    by nums_as_str.

    Examples:  input ' 1.   3.5 6 ' -> output 3.5
              input '2 17.6' -> output 9.8

    Pre: nums_as_str is a string representing a non-empty sequence of numbers
    separated by whitespace.
    """
    return avg(nums_as_str.split())
```

Unfortunately, even if function `avg` is implemented correctly, `string_avg` is not correct, because `nums_as_str.split()` is a list of strings, which `avg` is not expecting as input.

Rewrite the last line of `string_avg`, using a call to `map`, to fix this error. One to three lines of code suffices.

Solution: OK if they use iteration/for loop instead

```
return avg(map(float, nums_as_str.split()))
# +1 call to map has two arguments
# +1 its second argument is (equivalent to) nums_as_str.split()
# +1 refers to the float function
# +1 did not include parenthesis after function name (hopefully "float")
# +1 applies avg to the result of map, and everything else checks
#    out syntactically/semantically
```

Did you write your name & netID on each page, circle your lab on the front, and carefully re-read all instructions and specifications?