CS 1110 Final Exam Solutions May 2018

1. **Object Diagramming and Terminology.**

   (a) [10 points] The questions on the right pertain to the code on the left. Some questions may have multiple correct answers. Write down **only one** answer.

```
 1  class A():
 2      x = 1
 3
 4      def __init__(self, n):
 5          self.y = n
 6          A.x += 1
 7
 8      def p(self):
 9          print(self.y)
10          self.y += 3
11          self.r()
12
13      def r(self):
14          self.y += 2
15          print(self.y)
16
17  class B(A):
18      x = 10
19
20      def __init__(self, n):
21          super().__init__(n)
22          sum = self.y + B.x
23          self.m = sum
24
25      def r(self):
26          self.y += self.x
27          print(self.m)
28
29  a = A(1)
30  b = B(2)
```

an **object folder** is created when Python executes line ______

a **class folder** is created when Python executes line ______

an **object attribute** is created on line ______

a **class attribute** is created on line ______

a **superclass** definition begins on line ______

a **class method** definition begins on line ______

an attribute definition **that overrides another** begins on line ______

a method definition **that overrides another** begins on line ______

a **local variable** is created on line ______

a **global variable** is created on line ______

Solution:
object folder created: 29 or 30
class folder created: 1 or 17
object attribute created: 5 or 23
class attribute created: 2 or 18
superclass begins: 1
class method begins: 4, 8, 13, 20, 25
attribute override: 18

method override: 20 or 25
local variable: 22
global variable: 1, 17, 29, or 30

This code is copied from the previous page with **two additional lines of code**. It runs error-free.

```
 1  class A():
 2      x = 1
 3  
 4      def __init__(self, n):
 5          self.y = n
 6          A.x += 1
 7  
 8      def p(self):
 9          print(self.y)
10          self.y += 3
11          self.r()
12  
13      def r(self):
14          self.y += 2
15          print(self.y)
16  
17  class B(A):
18      x = 10
19  
20      def __init__(self, n):
21          super().__init__(n)
22          sum = self.y + B.x
23          self.m = sum
24  
25      def r(self):
26          self.y += self.x
27          print(self.m)
28  
29  a = A(1)
30  b = B(2)
31  a.p()
32  b.p()
```

(b) [4 points] What will be printed when Python executes **line 31**?

Solution:
1
6

(c) [4 points] What will be printed when Python executes **line 32**?

Solution:
2
12

2. **Object Creation and Foor Loops.** Consider the following code:

```
class MenuItem():
    """An instance represents an item on a menu."""
    def __init__(self, name, is_veggie, price):
        """A new menu item called name with 3 attributes:
        name:      a non-empty str, e.g. 'Chicken Noodle Soup'
        is_veggie: a Bool indicating vegetarian or not
        price:     an int > 0 """
        self.name = name
        self.is_veggie = is_veggie
        assert price > 0
        self.price = price

class LunchItem(MenuItem):
    """An instance represents an item that can also be served at lunch"""
    def __init__(self, name, is_veggie, price, lunch_price):
        """A menu item with one additional attribute:
        lunch_price:  an  int > 0 and <= 10"""
        super().__init__(name, is_veggie, price)
        assert lunch_price > 0
        assert lunch_price <= 10
        self.lunch_price = lunch_price
```

(a) [2 points] Write a python assignment statement that stores in variable `item1` the ID of a new `MenuItem` object whose name is “Tofu Curry”, a vegetarian dish costing 24 dollars.

Solution:
item1 = MenuItem("Tofu Curry", True, 24)

(b) [2 points] Write a python assignment statement that stores in variable `item2` the ID of a new `LunchItem` object whose name is “Hamburger”, a non-vegetarian dish that costs 12 dollars, but only 8 dollars at lunch.

Solution:
item2 = LunchItem("Hamburger", False, 12, 8)

(c) [2 points] **Class Invariants.** Lunch should never cost more than 10 dollars. The `init` method prevents this. Write a line of python that shows how this invariant can still be broken. You may use any of the global variables you created from the previous parts.

Solution:
item2.lunch_price = 16

The same code has been copied to this page for your convenience:

```
class MenuItem():
    """An instance represents an item on a menu."""
    def __init__(self, name, is_veggie, price):
        """A new menu item called name with 3 attributes:
        name:      a non-empty str, e.g. 'Chicken Noodle Soup'
        is_veggie: a Bool indicating vegetarian or not
        price:     an int > 0 """
        self.name = name
        self.is_veggie = is_veggie
        assert price > 0
        self.price = price

class LunchItem(MenuItem):
    """An instance represents an item that can also be served at lunch"""
    def __init__(self, name, is_veggie, price, lunch_price):
        """A menu item with one additional attribute:
        lunch_price:  an  int > 0 and <= 10"""
        super().__init__(name, is_veggie, price)
        assert lunch_price > 0
        assert lunch_price <= 10
        self.lunch_price = lunch_price
```

(d) [8 points] **For Loops.** Make effective use of a for-loop to write the body of the function audit_menu according to its specification.

```
def audit_menu(the_menu):
    """Performs an audit of each LunchItem on the_menu, making sure that each
    lunch_price is never more than 10 dollars. A lunch_price of 11 dollars is
    changed to 9. An item whose lunch_price is more than 11 is too expensive to
    be offered at lunch; it must be replaced with a new, equivalent MenuItem
    (that has no lunch price). Items that are not LunchItems are unchanged.
    Modifies the_menu; does not create or return a new menu/list
    the_menu: possibly empty list of MenuItem """
```

Solution:

```
    for i in list(range(len(the_menu))):
        item = the_menu[i]
        if isinstance(item, LunchItem):
            if item.lunch_price == 11:
                item.lunch_price = 9
            elif item.lunch_price > 11:
                newItem = MenuItem(item.name, item.is_veggie, item.price)
                the_menu[i] = newItem
```

3. [13 points] **String processing.** Complete the function below so that it obeys its specification. Sample inputs, the value of an important local variable, and desired outputs are given at the bottom of the page.

```
def after_at(s):
    """Returns a list of every non-empty sequence of non-space characters
    that follows an @ in s.

    The elements should be ordered by occurrence in s, and there should be no repeats.

    Pre: s is a string, possible empty.
    """
    temp = s.split('@')
    if len(temp) == 1:  # There were no @s in s.
        return []
    afters_list = temp[1:]  # Drop stuff before 1st @. See table at bottom of page.

    # DON'T use split. (It sometimes calls strip(), which you don't want here.)
    # Hint: for each item in afters_list, find where its first space is.
```

Solution:
Code for testing your own implementation:

http://www.cs.cornell.edu/courses/cs1110/2018sp/exams/final/2018_spring_string_processing.py

```
# BEGIN REMOVE
    outlist = []
    for item in afters_list:
        space_pos = item.find(' ')
        if space_pos == -1:
            if item != '' and item not in outlist:
                outlist.append(item)
        elif space_pos > 0:
            follower = item[:space_pos]
            if follower not in outlist:
                outlist.append(follower)
        # If space_pos == 0, don't append anything
```

Alternate solution by Kevin Cook, with some variable-name changes to match the above:

```
    outlist = []
    for item in afters_list:
        i = 0
        follower = ''
        while i < len(item) and item[i] != ' ':
            follower += item[i]
            i += 1
        if follower not in outlist and len(follower) > 0:
            outlist.append(follower)
    return outlist
```

Some further notes on the string processing question by TA Nancy Gu (with minor edits from Prof. Lee):

The correct implementation for this problem involves creating an output_list and adding elements to the output_list. Each element x in after_list is a string that can have one of 4 possible formats:

1. x is the empty string, “”. In this case, you should not append anything to the output_list.
2. x starts with at least one space, e.g., “   Hello World.” In this case, you should not append anything to the output_list.
3. x contains at least one empty spacebut the first one isn’t at position 0, e.g., “Hello World.” In this case you should append “Hello” to the output_list.
4. x is neither empty nor contains any space(s), e.g., “HelloWorld.” In this case, you should append “HelloWorld” to the output_list.

A common mistake we have seen is removing items as you go through the list, for example:

```
for x in after_list:
      if x == “”:
            after_list.remove(x)
```

This would not work because when you remove items from a list you shift it an element ahead, which will cause the iteration to skip an element. For example, the following code

```
x = [1, 2, 2, 3, 4]
for i in x:
      if i == 2:
            x.remove(i)
print(x)
```

prints [1, 2, 3, 4] even though the intention is to print [1, 3, 4].

Another common mistake is assigning values to a variable in the loop and expecting it to modify the values in the underlying list. For example:

```
after_list = [“hello”, “world”]
for x in after_list:
      x = “new value”
print(after_list)
```

This would still print [“hello”, “world”] because the assignment to `x` of the string “`new value`” does not change anything in after_list. In the for-loop, x is a variable that is assigned a value taken from after_list. The assignment to x of “new value” only changes the value of that variable within that iteration of the loop, because at the next iteration, x is reset to be the next item in after_list.

A finer-grained explanation of the rubric items

1. (+1.0) initializes accumulator list
    a. If you created an output_list you will get this point.

    b. If you made the mistake of removing from after_list in a for loop, you used after_list as your accumulator, so you will still get this point.

2. (+1.0) Loops through afters_list (index or element OK) and refer to the element in the loop correctly (e.g. would not get point if do "for x in after_list" and "after_list[x]").
    a. If you did “for i in range(len(after_list))” and refer to elements of after_list as “afterlist[i]”, you will get this point.
    b. If you did “for x in after_list” and refer to elements of after_list as “x”, you will get this point.
3. (+1.0) Finds location of first space (if use index() without guard do not get point)
    a. If you used string.find(“ “) you will get this point.
    b. If you put string.index(“ “) in a try catch block, you will get this point.
    c. If you put string.index(“ “) in a “if ‘ ‘ in string”, you will get this point.
    d. If you simply did string.index(“ “) you will lose this point because your code will throw an not found error.
4. (+2.0) Correctly processes string to obtain new string up to (and not including) the first space.
    a. If the position you got from 3 is named “pos” (e.g. pos = string.find(“ “), then you will get 2 points if you did string[:pos].
    b. If you did string[:pos+1] or string[:pos-1] you will get 2 points for this rubric item but you will also get 1 point deduction for off by 1.
5. (+2.0) Correctly adds processed string to accumulator (in the case that there is a space AND it’s not at position 0)
    a. This corresponds to correct implementation rule 3. If string = “Hello   World”, and your implementation appended “Hello” to the output_list, you will get 2 points.
    b. If string = “Hello   World”, and your implementation removed “Hello   World” from the after_list and added “Hello” back to the after_list, you will get these 2 points, but will get a 3 point deduction (which will only be applied one time even though you remove something else for later rubric items) for removing item from after_list while going through it.
6. (+2.0) Correctly adds string to accumulator (in the case that there is no space AND it’s not empty
    a. This corresponds to correct implementation rule 4. If string = “HelloWorld”, and your implementation appended “HelloWorld” to the output_list, you will get 2 points.
    b. If string = “HelloWorld”, and your implementation used after_list as accumulator and you chose to do nothing in this case, you will get these 2 points.
7. (+1.0) Does NOT add processed string if it’s empty.
    a. This corresponds to correct implementation rule 1 and 2. If you added any empty string to the output_list you will lose this point.
    b. If you used remove() on empty string from after_list, you will get this point, but will get a 3 point deduction if you haven’t got it from previous rubric item for removing item from after_list while going through it.

8. (+1.0) Checks for duplicates at ANY place before adding to accumulator (if string is already in accumulator list)
    a. If you successfully removed at least 1 duplicate, you will get this point.
    b. If you tried to remove duplicates from the after_list while going through after_list in a for loop, you will still get this point, but will get a 3 point deduction if you haven't got it from previous rubric item for removing item from after_list while going through it.
    c. If you did not check for duplicates when you loop through after_list and appended everything to a temporary_list, and correctly removed duplicates in temporary_list, you will get this point.
    d. If you did not check for duplicates when you loop through after_list and appended everything to a temporary_list, but attempted to remove items from temporarily_list in a for loop, you are still getting this point.
    e. If you did not check for duplicate at all, you are not getting this point.
9. (+1.0) Checks for duplicates at ALL places before adding to accumulator (if string is already in accumulator list)
    a. If you successfully removed all duplicates by checking if element is in the output_list, you will get this point.
    b. If you did not check for duplicates when you loop through after_list and appended everything to a temporary_list, and correctly removed duplicates in temporary_list, you will get this point.
    c. If you tried to remove duplicates from the after_list while going through after_list in a for loop, you will not get this point.
    d. If you did not check for duplicates when you loop through after_list and appended everything to a temporary_list, but attempted to remove items from temporarily_list in a for loop, you are not getting this point.
    e. If you did not check for duplicate at all, you are not getting this point.
10. (+1.0) Returns accumulator list.
    a. If you returned output_list, you will get this point.
    b. If you used after_list as accumulator by removing elements from it, and you returned after_list, you will get this point.

The other mistake we saw was using strip(). Strip() would get rid of the empty spaces, but also "\n", "\t", etc, which we would like to keep in this case. If you are appending string.strip() to the output_list, you are possibly adding an empty string to the output_list, and will lose the point for rubric item 7.

4. [5 points] **While Loops.** Make effective use of while loops to implement `countdown_by_n`. Your solution *must use a while loop* to receive points.

```
def countdown_by_n(count_from, count_by):
    """Prints a count down from count_from by count_by.
    Stops printing before the result goes negative.
    Note: this function does not return anything.

    count_from: the number you're counting down from [int]
    count_by: the amount you're counting down by [int > 0]

    Examples:

    coundown_by_n(16, 5) should print:
       16
       11
       6
       1

    coundown_by_n(21, 7) should print:
       21
       14
       7
       0

    """
```

Solution:

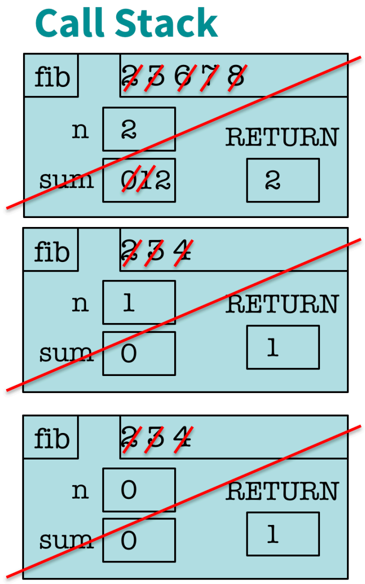
Note that count_from can start negative.

```
    while (count_from >= 0):
        print(count_from)
        count_from -= count_by
```

5. [12 points] **Call Frames.** On the right, **draw the full call stack** as it would look after all of the code on the left has executed. Include crossed-out frames. Do not worry about drawing any variables outside the call frames.

```
1   def fib(n):
2       sum = 0
3       if n == 0 or n == 1:
4           return 1
5
6       sum += fib(n-1)
7       sum += fib(n-2)
8       return sum
9
10  x = fib(2)
```

Solution:



6. **Invariants.** Let `b` be a non-empty list of ints, and `splitter` be an int. We want a for-loop that swaps elements of `b` and sets the variable `i` so that the following postcondition holds:

		i
b	<= splitter	> splitter

Examples:

Before		After	
splitter	b	i	b
0	[16, -4, 22]	1	[-4, 22, 16] or [-4, 16, 22]
0	[-10, -20, 15]	2	[-10, -20, 15] or [-20, -10, 15]
0	[-30, -50, -60]	3	any ordering of `b` works
-4	[10, 20, 30]	0	any ordering of `b` works

The code must maintain the following invariant.

		i	k
b	<= splitter	> splitter	???

In words, `b[0..i-1]` are all less than or equal to `splitter`;
`b[i..k-1]` are all greater than `splitter`; `b[k..len(b)-1]` have not yet been processed.

Solution:
This invariant was inspired by Section 2.2 of Kernighan and Pike, The Practice of Programming (1999).

(a) [2 points] According to the invariant, should the initialization be `i=1`?

- If yes, explain why — no credit without correct explanation.
- If no, give the correct initialization; omit explanation in this case.

Solution:
No, should be $i = 0$. Optional explanation: we don't know if b[0..i-1] == b[0] is <= splitter. See 4th example.

(b) [4 points] Here is the for-loop header:
`for k in list(range(len(b))):`
According to the invariant, is the following the correct and complete for-loop body? (Assume the helper does what the comment says.)

```
if b[k] <= splitter:
    swap(b, i, k)  # Helper that swaps items at position i and k in b
i += 1
```

- If yes, explain why — no credit without correct explanation.
- If no, give the correct and complete for-loop body; omit explanation in this case.

Solution:
The update of `i` should happen in the if-statement:

```
if b[k] <= splitter:
    swap(b, i, k)  # Helper that swaps items at position i and k in b
    i += 1
```

7. [11 points] **Recursion.** Assume that objects of class Course have two attributes:

- label [non-empty str]: unique identifying string, e.g., 'CS1110'
- prereqs [list of Course, maybe empty]: Courses that one must complete before this one.

Consider the following header and specification of a **non**-method function.

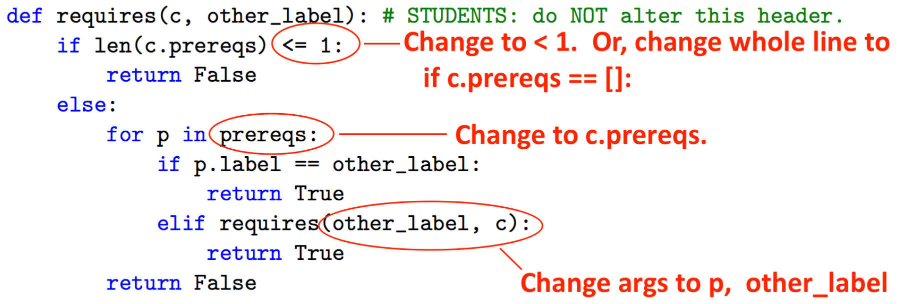
```
def requires(c, other_label):
    """Returns True if Course with label other_label must be taken before c,
    False otherwise.

    Pre: c is a Course.  other_label is a non-empty string."""
```

Example intended operation: suppose c1 is a Course with label 'CS1110' and empty prereqs list;
c2 is a Course with label 'CS2110' and prereqs list [c1];
c3 is a Course with label 'CS2800' and prereqs list [c1];
c4 is a Course with label 'CS3110' and prereqs list [c2, c3]

Then, all of the following should evaluate to True:	And all of the following should evaluate to False:
requires(c4, 'CS2800')	requires(c1, 'CS2800')
requires(c4, 'CS2110')	requires(c3, 'CS2800')
requires(c4, 'CS1110')	requires(c4, 'randomstring')

While a majority of the lines are correct, there is at least one error in the proposed implementation below. **For <u>each</u> error, circle it, write down/explain the correct version, and draw a line between the circle and the corresponding correction.** Responses where the correction is wrong may not receive any credit.

Solution:



```
def requires(c, other_label): # STUDENTS: do NOT alter this header.
    if len(c.prereqs) <= 1:
        return False
    else:
        for p in prereqs:
            if p.label == other_label:
                return True
            elif requires(other_label, c):
                return True
        return False
```

Change to < 1. Or, change whole line to if c.prereqs == []:

Change to c.prereqs.

Change args to p, other_label