Last Name: ____________________ First Name: ____________________ Cornell NetID, all caps: ______

CS 1110 Final Exam, May 2018

This 150-minute exam has 7 questions worth a total of 79 points. You may tear the pages apart; we have staplers.

You may use any Python introduced so far in this course, but not Python constructs that we have not introduced.

When a question asks for a particular technique to be used, do not expect to receive credit for questions based on a different technique. **For example, if we ask you to make effective use of recursion, your solution must be fundamentally based on recursion, even if a recursion-less for-loop would also solve the problem.**

The second page of this exam gives you the specifications for some useful functions and methods.

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
```

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.
We also ask that you not discuss this exam with students who are scheduled to take a later makeup.
Academic Integrity is expected of all students of Cornell University at all times, whether in the presence or absence of members of the faculty. Understanding this, I declare I shall not give, use or receive unauthorized aid in this examination.

Signature: ________________________________ Date ____________

`s.find(substr)`	Returns: index of first occurrence of string `substr` in string `s` (-1 if not found)
`s.strip()`	Returns: copy of string `s` where all whitespace has been removed from the beginning and the end of `s`. Whitespace not at the ends is preserved.
`s.split(sep)`	Returns: a list of the "words" in string `s`, using `sep` as the word delimiter (whitespace if `sep` not given)
`s.join(slist)`	Returns: a string that is the concatenation of the strings in list `slist` separated by string `s`
`s[i:j]`	Returns: if `i` and `j` are non-negative indices and `i` $\leq$ `j-1`, a new string containing the characters in `s` from index `i` to index `j-1`, or the substring of `s` starting at `i` if `j` $\geq$ `len(s)`

`lt.insert(i,item)`	Insert `item` into list `lt` at position `i`
`lt.append(item)`	Adds `item` to the end of list `lt`
`lt.count(item)`	Returns: count of how many times item occurs in list
`list(range(n))`	Returns: the list `[0 .. n-1]`
`lt.remove(item)`	Removes the first occurrence of `item` from list `lt`; raises an error if `item` not found.
`lt.index(item)`	Returns: index of first occurrence of `item` in list `lt`; raises an error if `item` is not found. (There's no "find" for lists.)
`lt[i:j]`	Returns: A new list `[lt[i], lt[i+1], ..., lt[j-1]]` under ordinary circumstances. Returns `[]` if `i` and `j` are not both sensible indices.
`lt.pop(i)`	Returns: element of list `lt` at index `i` **and also removes that element from the list** `lt`. Raises an error if `i` is an invalid index.

`list(map(func, lt))`	Returns: A list obtained by applying function `func` to each element in list `lt` and concatenating the results of each application.
`isinstance(o, c)`	Returns: True if `o` is an instance of class `c`, False otherwise.

Question	Points	Score
1	18	
2	14	
3	13	
4	5	
5	12	
6	6	
7	11	
Total:	79	

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 ______

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**?

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

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.

(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.

(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.

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 """
```

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.
```

s	afters_list (local variable)	return value
'@bill @ted meet @ the Circle K'	['bill ', 'ted meet ', ' the Circle K']	['bill', 'ted']
'meet @ the Circle K @Bill@Ted'	[' the Circle K ', 'Bill', 'Ted']	['Bill', 'Ted']
'hi'	[]	[]
'@martha @Martha @martha'	['martha ', 'Martha ', 'martha']	['martha', 'Martha']
'The symbol du jour is an @'	['']	[]
'The symbol du jour is an @!'	['!']	['!']

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

    """
```

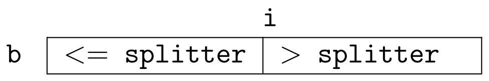
5. [12 points] **Call Frames.** On the right, **draw the full call stack** as it would look after all of the code on the left has executed. Include crossed-out frames. Do not worry about drawing any variables outside the call frames.

```python
def fib(n):
    sum = 0
    if n == 0 or n == 1:
        return 1

    sum += fib(n-1)
    sum += fib(n-2)
    return sum

x = fib(2)
```

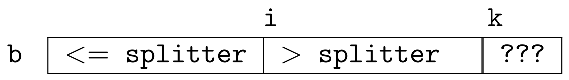
6. **Invariants.** Let `b` be a non-empty list of ints, and `splitter` be an int. We want a for-loop that swaps elements of `b` and sets the variable `i` so that the following postcondition holds:



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.



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.

(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.

(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.

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.

```
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
```

Did you re-read all specs, and check your code works against test cases?
Then, HAVE A GREAT SUMMER!