Spring 2019 CS 1110 Prelim 2 Solutions

Please turn off and stow away all electronic devices. You may not use them for any reason during the exam. Do not bring them with you if you leave the room temporarily.

This is a closed book and notes examination. You may use **the reference sheet on the last page of the exam.**

There are **5 problems**. Make sure you have the whole exam.

You have **90 minutes** to complete 90 points. Use your time accordingly.

Question	Points	Score
1	9	
2	20	
3	12	
4	20	
5	29	
Total:	90	

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.

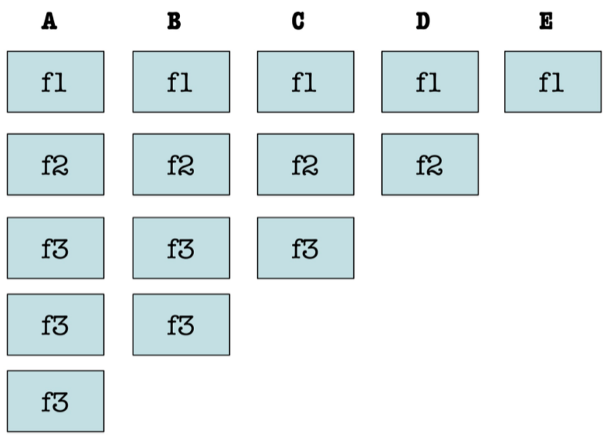
Signature: ____________________ Date ________

Name: ____________________ NetID ________

1. **9 Minute Warm-up**

(a) [3 points] If Python has printed `"f3"` **5 times**, what does the call stack look like?

```
def f3():
    print("f3")
    print("f3")

def f2():
    print("f2")
    f3()
    f3()
    f3()

def f1():
    print("f1")
    f2()

f1()
```



Correct Answer: C

(b) [3 points] What is the difference between an instance attribute and a class attribute?

List all that apply: ____________ E

A. An instance attribute lives in Global Space.
B. Instance attributes can be modified, but class attributes cannot.
C. Class attributes cannot be accessed by an instance of the class, but instance attributes can be.
D. Within a given class, there can be one instance attribute named `x` but possibly many class attributes named `x`.
E. Within a given class, there can be one class attribute named `x` but possibly many instance attributes named `x`.

(c) [3 points] Suppose you have a function `fun1` with the following line of code in it:

```
a1 = b1 * 2
```

Where might `a1` be located? List all that apply: ____________ B

A. the global space
B. the call frame for `fun1`
C. the call frame of the function that called `fun1`
D. (if `fun1` is a class method) an instance attribute
E. (if `fun1` is a class method) a class attribute

2. **Waldo's Brother Max**

In this question you will be asked to implement the definition of the following function in two different ways:

```
def find_max(my_list):
    """
    Returns the maximum integer in the integer list my_list.
    Note: my_list remains unchanged.

    my_list: a list of integers with at least 1 element

    Examples:
    find_max([0]) Returns 0
    find_max([4,0,12]) Returns 12
    find_max([-4,-10,-2]) Returns -2
    """
```

In *neither* part are you responsible for asserting/enforcing preconditions.

(a) [10 points] **For Loops.** Make effective use of a **for loop** to implement find_max. Your solution *must use a for loop* to receive points. When you have finished, step through your code to make sure it works on the given examples.

```
def find_max(my_list):

    max = my_list[0]
    for x in my_list:
        if x > max:
            max = x
    return max
```

(b) [10 points] **Recursion.** Make effective use of a **recursion** to implement `find_max`. Not only this, we want you to use a very specific approach to the divide-and-conquer. Your recursion should **split the input list into two halves of equal length** (or off by 1), find the maximum of both halves, and then return the maximum of the two. Your solution *must use this approach* in order to receive points. When you have finished, step through your code to make sure it works on the given examples.

The spec has been copied for your convenience.

```
def find_max(my_list):
    """
    Returns the maximum integer in the integer list my_list.
    Note: my_list remains unchanged.

    my_list: a list of integers with at least 1 element

    Examples:
    find_max([0]) Returns 0
    find_max([4,0,12]) Returns 12
    find_max([-4,-10,-2]) Returns -2
    """

    n = len(my_list)
    if n == 1:
        return my_list[0]
    i = n//2
    leftmax = find_max(my_list[:i])
    rightmax = find_max(my_list[i:])
    if leftmax > rightmax:
        return leftmax
    return rightmax
```

3. [12 points] **'Redacted' is the Word of the Day**
Implement the function `redact_emails` according to its specification. You do *not* need to assert/enforce any preconditions. Do **not** use recursion in your solution.

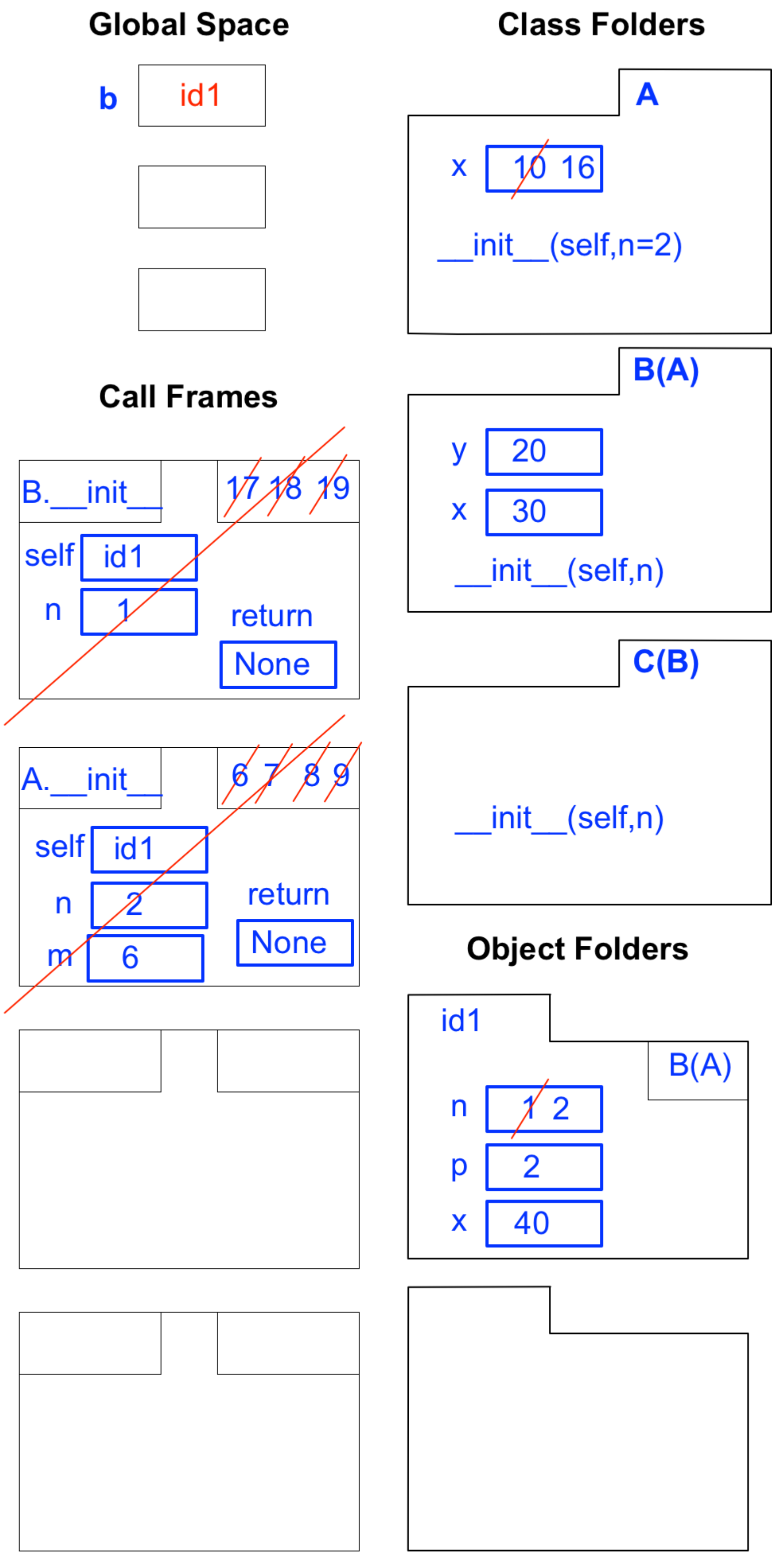
```
def redact_emails(email_list, token):
    """ Inputs:
    email_list: a list of emails. each email is a list of strings.
    token: a string of length 1 or more

    Modifies email_list as follows: each string containing the token is
      replaced by x's. the string of x's is as long as the original string.

    email_list is modified but not returned

    Examples:
    email_list: [["call", "me", "ASAP"],["meet", "me", "at", "3", "PM"]]
    token:  "me"
    redacted:  [["call", "xx", "ASAP"],["xxxx", "xx", "at", "3", "PM"]]

    email_list: [["call", "me", "ASAP"],["meet", "me", "at", "3", "PM"]]
    token:  "a"
    redacted:  [["xxxx", "me", "ASAP"],["meet", "me", "xx", "3", "PM"]]

    email_list: [[],["", "", "i", "miss", "you"], ["Hi", "please", "email", "back"]]
    token:  "i"
    redacted:  [[],["", "", "x", "xxxx", "you"], ["xx", "please", "xxxxx", "back"]]

    email_list: [[],["", "", "i", "miss", "you"], ["Hi", "please", "email", "back"]]
    token:  "x"
    redacted:  [[],["", "", "i", "miss", "you"], ["Hi", "please", "email", "back"]]
               ...unchanged...
    """

    for email in email_list:
        k = 0
        while k < len(email):
            word = email[k]
            if token in word:
                l = len(word)
                email[k] = l*'x'
            k += 1
```

There are many correct ways to do this. This is just one possible answer.

4. [20 points] **Inheritance**.

Use the templates provided to draw the complete history of memory after the *error-free* code below runs to completion. Include method names and class variables in the class folders. Add id #s and attributes to the object folders. If any values change as the code is executed, show *all* values by crossing out old values and putting new values next to the crossed out ones. You do not need to put global variables associated with the class folders in the global space; in fact you don't have enough boxes in Global Space to do so. You may not need all the templates provided. Only use as many as are needed.

```
 1  class A(object):
 2
 3      x = 10
 4
 5      def __init__(self, n=2):
 6          self.n = n
 7          m = 3 * n
 8          A.x += m
 9          self.x = 40
10
11  class B(A):
12
13      y = 20
14      x = 30
15
16      def __init__(self, n):
17          self.n = n
18          super().__init__()
19          self.p = 2
20
21  class C(B):
22
23      def __init__(self, n):
24          z = 8
25          self.x = n
26
27  b = B(1)
```



5. **We Are Family!**

   Familiarize yourself with the `Person` class below, including helper methods `add_parents` and `add_children`. (The question begins on the next page.)

   *Do **not** assert preconditions for any part of this question.*

```
class Person(object):
    """ A class to represent a person in a genealogical tree.

    CLASS ATTRIBUTE:
        population: tracks how many total Persons there are [int]

    INSTANCE ATTRIBUTES:
        first:   First Name [str]
        last:    Last Name [str]
        parents: (possibly empty) list of Person
        children: (possibly empty) list of Person
    """

    population = 0

    def add_parents(self, parents):
        """
        adds new parents to existing parents list
        parents: (possibly empty) list of Person
        """
        for p in parents:
            self.parents.append(p)

    def add_children(self, children):
        """
        adds new children to existing children list
        children: (possibly empty) list of Person
        """
        for c in children:
            self.children.append(c)
```

(a) [10 points] **Complete the class method `__init__` below:**

```
def __init__(self,first,last,parents=[]):
    """
    Creates a new Person with 4 instance attributes.
    Updates population accordingly.
    The optional parameter parents contains an initial list of
    parents that should be added to a NEW parent list maintained
    exclusively by this new Person. All parents need to update
    their children list in response to the creation of this child.

    first: first name [str]
    last:  last name [str]
    parents: list of Person
    """

    self.first = first
    self.last = last
    self.parents = []
    self.add_parents(parents)
    for p in parents:
        p.add_children([self])
    self.children = []
    Person.population += 1
```

(b) [10 points] Make effective use of **recursion** to complete the class method `count_the_kidless` below. You may assume that the family tree does not have any cycles (traveling up or down the family tree, you'll never encounter the same Person twice).

```
def count_the_kidless(self):
    """
    Counts the number of descendents (self, child, grandchild, etc.)
    with no children. Includes self as a descendent.

    Returns: an integer

    Examples:
    - if p1 has no children, p1.count_the_kidless() returns 1
    - if p1 has two children each with exactly 2 children (who
      in turn have no children), p1.count_the_kidless() returns 4

    """

    if self.children == []:
        return 1
    sum = 0
    for c in self.children:
        sum += c.count_the_kidless()
    return sum
```

(c) [9 points] **Complete the function blend below.** Do **not** worry about preventing:
- duplicates (a Person appearing multiple times on a parent or child list)
- cycles (a Person being both someone's descendant AND ancestor)

```python
def blend(p1, p2):
    """
    Blends the families of p1 and p2

    p1: parent 1 [Person]
    p2: parent 2 [Person]

    p1 puts p2's children on their (p1's) children list
       (p2's children get p1 as an additional parent)
    p2 puts p1's children on their (p2's) children list
       (p1's children get p2 as an additional parent)

    Example:
    - if p1 has child c1 and p2 has no children, blend(p1,p2) results
      in p1 and p2 both having child c1; c1 has parents p1 and p2
    """

    p1.add_children(p2.children)
    p2.add_children(p1.children)
    for c in p1.children:
        c.add_parents([p2])
    for c in p2.children:
        c.add_parents([p1])
```