CS 1110 Review: Final Worksheet Solutions

This worksheet contains various problems for you to practice. We will go over only some problems during the final review session and will let you take this worksheet home to practice more.

List and sequences

def find_in_list(lst, v):
    """ Returns: the position of the first occurrence of v in lst or -1 if not found
    Parameters:
    lst - a possibly empty list
    v - a value that may or may not be in the list"

    for i in range(len(lst)):
        if v == lst[i]:
            return i
    return -1

def sum_nums(s):
    """ Returns a string representing the sum of the numbers separated by spaces in the string s (Example: '100 2 34' will return '136')
    Parameter: s is a string with spaces and/or digits"

    lst = s.split() #if you do s.split(' '), lst may have empty strings!
    acc = 0
    for num in lst:
        acc += int(num)
    return acc
def transpose(x):
    """Returns: a nested list representing the transposed matrix x.

Example: transpose([[1,2,3],[4,5,6]]) returns [[1,4],[2,5],[3,6]]

Parameter: x - a nested list representing a rectangular matrix (the length of each row is the same)"

    newrow = len(x[0])
    newcol = len(x)
    newx = []
    for i in range(newrow):
        row = []
        for j in range(newcol):
            row.append(x[j][i])
        newx.append(row)
    return newx

# A matrix can be represented as a 2D list. In this case, # transpose would take a matrix and change it like so:
#  
#  [[1, 2, 3],   turns to   [[1,4],
#  [4, 5, 6]]   -------->  [2,5],
#                         [3,6]]

def creditsToClasses(classes):
    """Returns: a dict with number of credits (ints) as keys and the corresponding classes (a list of str) as values. The order of the classes within the lists does NOT matter.

Example: given classes {'CS 1110': 4, 'CS 7090': 1, 'CS 1112': 4},
you would return {4: ['CS 1110', 'CS 1112'], 1: ['CS 7090']}

Parameter: classes is a dict with the keys being course names as strs (e.g. 'CS 1110') and the values being the number of credits as ints (e.g., classes['CS 1110'] = 4) ""

    numToClass = {}
    for c in classes:
        if classes[c] in numToClass:
            numToClass[classes[c]].append(c)
        else:
            numToClass[classes[c]] = [c]
    return numToClass
The following function is supposed to take in a list `lst` and a value `v` that occurs more than once in `lst`, and return the index of the second occurrence of `v`. However, this is not what happens. Instead, the function always returns the index of the first occurrence of `v`. For example, the output of `secondInd([1,2,3,1], 1)` is equal to 0. What is the issue?

```python
def secondInd(lst, v):
    """ Given a list lst and value v that occurs more than once in lst, 
    return the index of the second position where v occurs in lst. 
    ""
    seen_once = False
    for i in lst:
        if i == v:
            if not seen_once:
                seen_once = True
            else:
                return lst.index(v)
```

The issue is that this implementation returns the index of the first instance of the value rather than the second. An easy fix is to reimplement this function using a for loop with `range(len(lst))` so we are keeping track of the indices. A possible implementation is as so:

```python
def secondInd(lst, v):
    seen_once = False
    for i in range(len(lst)):
        if lst[i] == v:
            if not seen_once:
                seen_once = True
            else:
                return i
```

The following function is supposed to remove all values of `lst` that are even. However, the function does not behave as expected. Why did this happen?

```python
def removeEvens(lst):
    """ Given a list lst, remove all even elements from lst in-place 
    (does not return) 
    ""
    for i in lst:
        if i % 2 == 0:
            lst.remove(i)
```

```
>>> a = [1,1,2,2,2,3,3,4,4,4]
>>> removeEvens(a)
>>> a
[1, 1, 2, 3, 3, 4]
```

In this function, the list is modified as it is iterated through. Thus, this code would skip an element if the element before it was removed. A possible fix for this code is to use a while loop. Challenge yourself to fix it!
Loop invariants and sequence algorithms

Draw boxes for the preconditions, postconditions, and invariant for the function smallest_index. Then, write the actual code for the loops. This loop should take a list of ints s and returns the index x of the smallest int (s[x]). k has been filled out for you.

```python
def smallest_index(s):
    """ Returns: an int representing the index of the smallest number in s.
    Precondition: s is a non-empty list of integers.
    ""
    x = 0
    m = 0
    k = len(s) - 1
    while m < k:
        if s[x] > s[m+1]:
            x = m + 1
            m += 1
    return x
```

A palindrome is a sequence of characters which reads the same backward or forward. For example, madamimadam is a palindrome. Given the below function and invariant, fill in the missing lines of the function.

```python
def is_palindrome(s):
    """ Returns: True if the string is a palindrome, false otherwise
    Prec: s is a string"
    # initialize loop variable here
    h = 0
    k = len(s)-1
    # invariant: s[0..h-1] is the reverse of s[k+1..len(s)-1]
    while h<=k:
        if s[h]!=s[k]:
            return False
        h += 1
        k -= 1
    return True
```
Call Frames

Draw the entire call stack for skip('abc')

```python
def skip(s):
    """ Returns: copy of s odd (from end) skipped """
    result = ""
    if (len(s) % 2 == 1):
        result = skip(s[1:])
    elif len(s) > 0:
        result = s[0]+skip(s[1:])
    return result
```

Note: The counter numbers would be all crossed out, but we have not done that for readability reasons.
Draw the call stack for `sumStringList([' 8', '5', ' 70 '])`

```python
def sumStringList(li):
    '''Returns: the sum of a list of strings.
    Precondition: li is list of strings of digits with
    possible white space before or after the digits.
    ex: li = [' 8 ', '32', ' 1'] returns 41'''
    counter = 0
    for x in li:
        x = x.strip()
        counter += int(x)
    return counter
```

```
sumNestedList  6 7 8 9 7 8 9 7 10
               |   |
     li         id1 x 7 0
               counter 8 3 3
```

```
  8
  5
  70
```
Draw the diagrams for the 2 object folders and the class folder when you execute:

```python
>>> a = Cornellian("Alice")
>>> b = Cornellian("Bob")

class Cornellian(object):
    """Instance attributes:
    _cuid: Cornell id [int > 0]
    _name: full name [nonempty str]"
    NEXT = 1 # Class Attribute

def _assignCUID(self):
    """Assigns _cuid to next Cornell id""
    self._cuid = Cornellian.NEXT
    Cornellian.NEXT = Cornellian.NEXT+1

def __init__(self, n):
    """Initializer: Cornellian with name n.""
    self._name = n
    self._assignCUID()

Note: self should be included in _assignCUID and __init__."
```

Execute:

```python
>>> a = Cornellian('Alice')
>>> b = Cornellian('Bob')
```

![Diagram showing object folders and class folder with instance attributes and class attribute NEXT]
Diagram the class folder and the constructor call frames for the following two classes when you execute the following code:

```python
class X():
    b = 1110
    a = 5
    def __init__(self, d):
        self.a = self.b
        self.c = d

class Y(X):
    b = 10
    def __init__(self, d, a):
        super().__init__(a)
        self.b = d

>>> y = X(5)
>>> x = Y(1110,5)
```

Solution: except the class labels on id1 and id2 should be swapped
Classes

class Book():
    """Instance is a book that is currently being read
    Attributes:
        title [str]: title of the book
        sequel [Book]: sequel to the book, None if nonexistent
        pages_left [int]: number of pages still unread"
"
    def __init__(self, t, s=None, p=0):
        """Initializer for class Book
        Default value for sequel is None
        Default value for pages_left is 0 """
        self.title = t
        self.sequel = s
        self.pages_left = p

def __eq__(self, b):
    return ((self.title == b.title) and (self.sequel == b.sequel)
            and (self.pages_left == b.pages_left))

def __str__(self):
    """Format: title (sequel, pages_left)"
    sequel_str = ''
    if self.sequel != None:
        sequel_str = str(self.sequel)
        return (self.title + ' (' + sequel_str + ', ' +
                str(self.pages_left) + '))
    return (self.title + ' ()')

def readPages(self, n):
    """Read n number of pages in the book until the end"
    self.pages_left = max(self.pages_left - n, 0)

What are the outputs when running the following pieces of code?

>>> b1 = Book("Eldest")
>>> b2 = Book("Eragon", b1, 30)
>>> b3 = Book("Eldest")
>>> b4 = b1
>>>
>>> print(b1 == b2)
False
>>> print(b1 is b2)
False
>>> print(b1 == b3)
False
>>> print(b1 is b3)
False
>>> print(b1 == b4)
True
>>> print(b1 is b4)
True
>>> print(b1.sequel)

False
False
True
False
True
True

None
0
Eldest (, 0)
30
Eragon (Eldest (, 0), 30)
# class Complex with the following specification

class Complex:
    '''Instance is a complex number, with real and imaginary parts
    Attributes:
    real: real portion of the number    [float]
    imag: imaginary portion of number    [float]'''

    # initializer
def __init__(self, real, imag):
        '''Initializes attributes with floats
        Precondition: real, imag can be ints or floats'''
        self.real = float(real)
        self.imag = float(imag)

    # implement +
def __add__(self, other):
        return Complex(self.real + other.real, self.imag + other.imag)

    # implement *
def __mul__(self, other):
        '''Note: (a + bi) * (c + di) = (ac - bd) + (ad + bc)i'''
        a = self.real
        b = self.imag
        c = other.real
        d = other.imag
        return Complex(a * c - b * d, a * d + b * c)

    # implement str: Complex(1, -2) looks like "1.0 + -2.0i"
def __str__(self):
        return str(self.real) + ' + ' + str(self.imag) + 'i'

    # implement ==
def __eq__(self, other):
        return self.real == other.real and self.imag == other.imag

# subclass Real, instance is a real number
class Real(Complex):
    '''Instance is a real number'''

    # initializer for Real
def __init__(self, num):
        '''Init for Real calls Complex __init__ method'''
        super().__init__(num, 0.0)
What is printed when the following code is executed?

```python
>>> a = Complex(1, 2)
>>> print (a.getReal())
>>> print (a.getImag())
>>> print (a)
>>> b = Complex(-1, -2)
>>> print (b.getReal())
>>> print (b.getImag())
>>> print (b)
>>> c = Real(3)
>>> print (c.getReal())
>>> print (c.getImag())
>>> print (c)
>>> print ()
>>> print ("Operation results:")
>>> print (a + b)
>>> print (a - b)
>>> print (a * b)
>>> print (a == b)
>>> print (a == Complex(1, 2))
```

```
1.0
2.0
1.0 + 2.0i
-1.0
-2.0
-1.0 + -2.0i
3.0
0.0
3.0 + 0.0i

Operation results:
0.0 + 0.0i
2.0 + 4.0i
3.0 + -4.0i
False
True
```