MIT-6.00.1x-Introduction-to-Computer-Science-and-Programming-Using-Python-FINAL EXAM

Problem 1:

a) **In the statement L = [1,2,3], L is a class.**
   TRUE
   FALSE
b) **The orders of growth of and are both polynomial.**
   TRUE
   FALSE
c) **A bisection search algorithm always returns the correct answer when searching for an element in a sorted list.**
   TRUE
   FALSE
d) **Performing binary search on an unsorted list will always return the correct answer in time where is the length of the list.**
   TRUE
   FALSE

Problem 2:

a)You have the following class hierarchy:

```
class A(object):

   def foo(self):

      print('hi')

class B(A):

   def foo(self):

      print('bye')
```

b)Which of the following is correct?

# When a = A() we say that a is an instance of A

# When b = B() we say that b is a subclass of A

# Both of the above

# Neither of the above

c) Consider the function f below. What is its Big O complexity?

```
def f(n):
    def g(m):
        m = 0
        for i in range(m):
            print(m)
    for i in range(n):
        g(n)
```

d) A dictionary is an immutable object because its keys are immutable.

True

False because its keys can be mutable

False because a dictionary is mutable

e) Consider the following two functions and select the correct choice below:

```
def foo_one(n):
    """ Assume n is an int >= 0 """
    answer = 1.0
    while n > 1:
        answer *= n
        n -= 1
    return answer

def foo_two(n):
    """ Assume n is an int >= 0 """
    if n <= 1:
        return 1.0
    else:
        return n*foo_two(n-1)
```

The worst case Big Oh time complexity of foo_one is worse than the worst case Big Oh time complexity of foo_two.

The worst case Big Oh time complexity of foo_two is worse than the worst case Big Oh time complexity of foo_one.

The worst case Big Oh time complexity of foo_one and foo_two are the same.

Impossible to compare the worst case Big Oh time complexities of the two functions.

f)The complexity of 1^n + n^4 + 4n + 4 is

constant

logarithmic

linear

polynomial

exponential

Problem 3:

```
# Implement a function that meets the specifications below.


# def sum_digits(s):
#     """ assumes s a string
#         Returns an int that is the sum of all of the digits in s.
#           If there are no digits in s it raises a ValueError exception. """
#     # Your code here
# For example, sum_digits("a;35d4") returns 12.


# Paste your entire function, including the definition, in the box below.
# Do not leave any debugging print statements.
```

Problem 4:

Implement a function that meets the specifications below.

```
def max_val(t):
    """ t, tuple or list
      Each element of t is either an int, a tuple, or a list
       No tuple or list is empty
       Returns the maximum int in t or (recursively) in an element of t """
    # Your code here
```

For example,

max_val((5, (1,2), [[1],[2]])) returns 5.

max_val((5, (1,2), [[1],[9]])) returns 9.

Paste your entire function, including the definition, in the box below. Do not leave any debugging print statements.

Problem Set 5:

Implement a function that meets the specifications below.

```
def cipher(map_from, map_to, code):

    """ map_from, map_to: strings where each contain

                    N unique lowercase letters.

        code: string (assume it only contains letters also in map_from)

        Returns a tuple of (key_code, decoded).

        key_code is a dictionary with N keys mapping str to str where

        each key is a letter in map_from at index i and the corresponding

        value is the letter in map_to at index i.

        decoded is a string that contains the decoded version

       of code using the key_code mapping. """

    Your code here For example,
```

cipher("abcd", "dcba", "dab") returns (order of entries in dictionary may not be the same) ({'a':'d', 'b': 'c', 'd': 'a', 'c': 'b'}, 'adc') Paste your entire function, including the definition, in the box below.

Do not leave any debugging print statements.

Problem Set 6:

You are given the following superclass. Do not modify this.

```
class Container(object):
    """ Holds hashable objects. Objects may occur 0 or more times """
    def __init__(self):
        """ Creates a new container with no objects in it. I.e., any object
            occurs 0 times in self. """
        self.vals = {}
    def insert(self, e):
        """ assumes e is hashable
            Increases the number times e occurs in self by 1. """
        try:
            self.vals[e] += 1
        except:
            self.vals[e] = 1
    def __str__(self):
        s = ""
        for i in sorted(self.vals.keys()):
            if self.vals[i] != 0:
                s += str(i)+":"+str(self.vals[i])+"\n"
        return s
```

Write a class that implements the specifications below. Do not override any methods of Container.

```
class Bag(Container):
    def remove(self, e):
        """ assumes e is hashable
            If e occurs in self, reduces the number of
            times it occurs in self by 1. Otherwise does nothing. """
        # write code here
    def count(self, e):
        """ assumes e is hashable
            Returns the number of times e occurs in self. """
        # write code here
```

For example,

```
d1 = Bag()
d1.insert(4)
d1.insert(4)
print(d1)
d1.remove(2)
print(d1)
```

prints

```
4:2
4:2
```

For example,

```
d1 = Bag()
d1.insert(4)
d1.insert(4)
d1.insert(4)
print(d1.count(2))
print(d1.count(4))
```

prints

```
0
3
```

Paste your entire class, including the definition, in the box below.

Do not leave any debugging print statements.

b)Write a method in Bag such that if b1 and b2 were bags then b1+b2 gives a new bag representing the union of the two bags.

For example,

```
 a = Bag()
a.insert(4)
a.insert(3)
 b = Bag()
 b.insert(4)
 print(a+b)
```

prints 3:1 and 4:2

Paste your entire class for Bag with the new method, including the definition, in the box below. Do not leave any debugging print statements.

c)Write a class that implements the specifications below. Do not override any methods of Container.

```
class ASet(Container):
    def remove(self, e):
        """assumes e is hashable
           removes e from self"""
        # write code here


    def is_in(self, e):
        """assumes e is hashable
           returns True if e has been inserted in self and
           not subsequently removed, and False otherwise."""
        # write code here
```

For example,

```
d1 = ASet()
d1.insert(4)
d1.insert(4)
d1.remove(2)
print(d1)
d1.remove(4)
print(d1)
```

prints 4:2 # from d1.remove(2) print

# (empty) from d1.remove(4) print

For example,

```
d1 = ASet()
d1.insert(4)
print(d1.is_in(4))
d1.insert(5)
print(d1.is_in(5))
d1.remove(5)
print(d1.is_in(5))
```

prints

```
True
True
False
```

Paste your entire class, including the definition, in the box below.

Do not leave any debugging print statements.

Problem 7

You are given the following two classes.

Do not change the Location or Campus classes.

Location class is the same as in lecture.

```
class Location(object):
  def __init__(self, x, y):
    self.x = x
    self.y = y
  def move(self, deltaX, deltaY):
    return Location(self.x + deltaX, self.y + deltaY)
  def getX(self):
    return self.x
  def getY(self):
    return self.y
  def dist_from(self, other):
    xDist = self.x - other.x
    yDist = self.y - other.y
    return (xDist**2 + yDist**2)**0.5
  def __eq__(self, other):
    return (self.x == other.x and self.y == other.y)
  def __str__(self):
    return '<' + str(self.x) + ',' + str(self.y) + '>'
class Campus(object):
  def __init__(self, center_loc):
    self.center_loc = center_loc
  def __str__(self):
    return str(self.center_loc)
```

Implement a class that meets the specifications below.

```
class MITCampus(Campus):
   """ A MITCampus is a Campus that contains tents """
   def __init__(self, center_loc, tent_loc = Location(0,0)):
      """ Assumes center_loc and tent_loc are Location objects
      Initializes a new Campus centered at location center_loc
      with a tent at location tent_loc """
      # Your code here
   def add_tent(self, new_tent_loc):
      """ Assumes new_tent_loc is a Location
      Adds new_tent_loc to the campus only if the tent is at least 0.5 distance
      away from all other tents already there. Campus is unchanged otherwise.
      Returns True if it could add the tent, False otherwise. """
      # Your code here


   def remove_tent(self, tent_loc):
      """ Assumes tent_loc is a Location
      Removes tent_loc from the campus.
      Raises a ValueError if there is not a tent at tent_loc.
      Does not return anything """
     # Your code here
   def get_tents(self):
      """ Returns a list of all tents on the campus. The list should contain
      the string representation of the Location of a tent. The list should
      be sorted by the x coordinate of the location. """
      # Your code here
```

For example, if c = MITCampus(Location(1,2)) then executing the following sequence of commands:

c.add_tent(Location(2,3)) should return True

c.add_tent(Location(1,2)) should return True

c.add_tent(Location(0,0)) should return False c.add_tent(Location(2,3)) should return False

c.get_tents() should return ['<0,0>', '<1,2>', '<2,3>']

Paste your entire class MITCampus in the box below. Do not leave any debugging print statements.