15-112
Spring 2022 Exam 2
March 29, 2022

Name:

Andrew ID:

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- Show your work on the exam to receive credit.
- You may use the backs of pages as scratch paper. Nothing written on the back of any pages will be graded.
- All code samples run without crashing. Assume any imports are already included as required.
- You may assume that random, math, string, basic_graphics, cmu_112_graphics and copy are imported; do not import any other modules.

Don't write anything in the table below.

Question	Points	Score
1	8	
2	8	
3	10	
4	10	
5	15	
6	20	
7	15	
8	20	
Total:	106	

There are 106 points on this exam, but your final score will be out of 100. (So, 6 points are effectively bonus.) However, the highest score than can be received on this exam is still 100.

1. **Short answers**

Answer each of the following *very briefly*.

(a) (1 point) Write one line of code that would be an MVC violation if it was placed inside `def redrawAll(app, canvas):`

(b) (1 point) Name 2 ways that sets and lists differ. Note: answers that appeal to syntax and method naming will not be accepted.

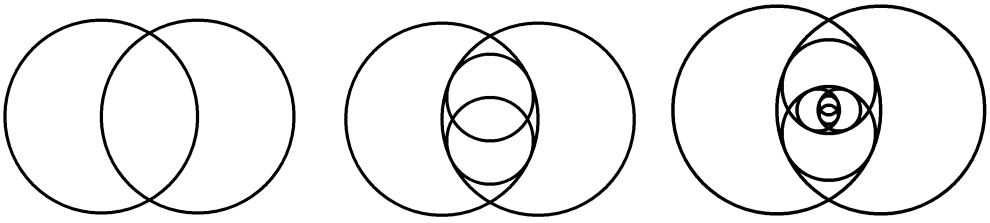
(c) (1 point) In one sentence, explain why this code will crash:

```
s=set()
s.add([1,2])
```

(d) (1 point) The following function is supposed to *reverse a list in a recursive way*. Indicate whether the function is correct or not. If not, explain what's wrong with it.

```
def recursiveReverseList(L):
    if len(L) == 0:
        return []
    return L[::-1]
```

(e) (4 points) Consider the following fractal drawn at levels 0, 1, and 3. Assume that the function `drawGardiFractal` is called to draw the fractal. In the code, how many calls to `drawGardiFractal` and `canvas.create_oval` occur in the base case? What about the recursive case?



Base Case		Recursive Case	
Number of calls to `drawGardiFractal`		Number of calls to `drawGardiFractal`	
Number of calls to `canvas.create_oval`		Number of calls to `canvas.create_oval`	

2. (8 points) **Big-O**

For each function shown below, write next to each line of the function either the Big-O runtime of the line or the number of times the line loops. Then write the total Big-O runtime of the function in terms of N in the box to the right of the code. **All answers must be simplified - do not include lower-order terms!**

```
import string
def bigOh(s):  # s is a string,  N = len(s)
    result = ""                                      _____
    for c in s:                                      _____
        for c in string.ascii_lowercase:             _____
            if s.count(c) == result.count(c):        _____
                result += c                          _____
    return c                                         _____
```

```
def bigOh(L):  # L is a list, N = len(L)
    d = dict()                                       _____
    for i in L:                                      _____
        d[i] = i                                     _____
    return len(d)                                    _____
```

```
def bigOh(L):  # L is a list, N = len(L)
    n = len(L)                                       _____
    for i in range(n**2):                            _____
        L.append(L.count(i))                         _____
```

3. **Code Tracing**

Indicate what each will print. Place your answer (and nothing else) in the box next to or below each block of code.

(a) (5 points) CT1

```
def ct1(L):
    s = set()
    d = dict()
    for i in range(len(L)):
        if (i%2 == 0):
            if (L[i] in d):
                s.add(L[i+1])
            else:
                d[L[i]] = L[i+1]
            print("d:",d)
            print("s:",s)
    return min(d.keys())

print(ct1([42,1,3,5,3,7,42,0]))
```

(b) (5 points) CT2

```
def ct2(n, d):
    print(f"IN n: {n} d: {d}")
    if  (n  <=  2):
        return 2+abs(n)
    else:
        ans = n  +  ct2(n-2, d+1)
        print(f"MID: {ans}")
        ans +=  ct2(n//2, d+1)
        print(f"OUT {ans} d: {d}")
        return ans
print(ct2(5, 0))
```

4. **Reasoning Over Code**

For each function, find values of the parameters so that the following functions will return `True`. Place your answer (and nothing else) in the box below each block of code.

(a) (5 points) ROC1

```
def roc1(d):
    assert(isinstance(d,dict))
    assert(len(d) == 3)
    e = set()
    for k in d:
        e.add(k)
        e.add(d[k])
    return e == set([1,2,3,4])
```

(b) (5 points) ROC2

```
def rocHelper(t,d):
    if len(t) <= 1:
        return False
    if len(t) == 2:
        return d == 0 and t == "ab"
    h = len(t)//2
    if len(t)%2 == 0:
        return rocHelper(t[:h],d-1) and rocHelper(t[h:],d-1)
    else:
        return rocHelper(t[:h],d-1) and rocHelper(t[h+1:],d-1) and int(t[h])==d

def roc2(t):
    return rocHelper(t,2)
```

5. (15 points) **Free Response: Recursive Sum Consecutive Pairs**

   Write the function `recSumConsecutivePairs(L)` that returns a new list with the sums consecutive pairs of elements in `L`, in the corresponding order. If there are no consecutive pairs, it should return an empty list.

   For instance,

```
recSumConsecutivePairs([3,2,5,1]) == [5,7,6]          #  3+2, 2+5,  5+1
recSumConsecutivePairs([-1,4,10,2,0]) == [3,14,12,2]  # -1+4, 4+10, 10+2, 2+0
recSumConsecutivePairs([1]) == []  # no consecutive pairs
recSumConsecutivePairs([]) == []   # no consecutive pairs
```

   Your solution must use recursion. If you use any loops, comprehensions, or iterative functions, you will receive no points on this problem.

6. (20 points) **Free Response: OOP**

Consider the following code designed to test two different classes: Cart and WishList:

```
# A (Shopping) Cart takes a string (not a list) of comma-separated items
# A cart stores items, and items are case insensitive
m1 = Cart('milk,eggs,MILK')
assert(m1.itemCount() == 2) # milk and eggs
# The .getItems method should return a set of unique items present in the cart
assert(m1.getItems() == {'milk', 'eggs'})
# Note that these items are lowercase. Ignore case and duplicates
m2 = Cart('Tomatoes,ONION,water,chips,CHIPS')
assert(m2.getItems() == {'tomatoes','onion','water','chips'})
assert(m2.itemCount() == 4) # Still just four items
assert(str(m2) == "Shopping Cart: 4 item(s)")
m3 = Cart('MILK')
assert(str(m3) == "Shopping Cart: 1 item(s)")
# You can add one new item at a time:
assert(m3.addItem('milk') == False) # Return False if the item is already present
assert(m3.addItem('eggs') == True) # Return True if we added a new item
assert(m3.itemCount() == 2)
assert(m3.getItems() == {'milk', 'eggs'})
assert((m3.getItems() == m1.getItems()) and (m3.getItems != m2.getItems()))
# Once more, but with a new item:
assert(m3.addItem('water') == True) # True means the item was added!
# and so these all change:
assert(m3.itemCount() == 3)
assert(m3.getItems() == {'milk', 'eggs','water'})
# Lastly, all carts start out open
assert(m1.status == 'open')
# ...unless you pay them
m1.checkout()
assert(m1.status == 'closed')
m1.checkout()
assert(m1.status == 'closed') # still closed
# and you cannot add any more items
assert(m1.addItem('tomatoes') == False) # Return False if the cart has been checked out
assert(m1.status == 'closed') # still closed

# A Wish List is a shopping cart that it always remains open
w1 = WishList("LEMON")
assert(str(w1) == "Wish List: 1 item(s)")
assert(w1.itemCount() == 1)
assert(w1.getItems() == {'lemon'})
assert(w1.addItem("Salt") == True)
assert(w1.getItems() == {'lemon', 'salt'})
assert(w1.status == 'open')
w1.checkout()   # does nothing, the wish list stays open
assert(w1.status == 'open')
```

Write the classes Cart and WishList so that the test code runs as specified. Do not hardcode against the values used in the testcases, though you can assume the testcases cover the needed functionality. For full credit, you must use proper object-oriented design, including good inheritance and avoiding unnecessary code duplication.

Additional Answer Space for Question 6

7. (15 points) **Free Response: Fluke Numbers** A *fluke number* (coined term) is an integer that has a frequency in the list equal to its value

   Write the function `findFlukeNumbers(L))` that is given a list `L` of objects (not necessarily integers). The function should return a set containing all the fluke numbers in the list. Your solution should run in $O(N)$ time.

   For example,

```
assert(findFlukeNumbers([1,'a','a',[4], 3, False, 3, 3]) == {1, 3})
assert(findFlukeNumbers([1, 2, 2, 3, 3, 3, 4]) == {1, 2, 3})
assert(findFlukeNumbers([0, False, 'hello']) == set())
```

8. (20 points) **Free Response: Animation**

   Write `appStarted`, `keyPressed`, `mousePressed`, `redrawAll`, and `timerFired` that implement the game described below.

   1. The game begins with a red square (width of 30 pixels) placed in a random location on the screen and 20 blue dots (radius of 10 pixels) at random locations.
   2. In the upper left hand corner of the screen is a timer. It counts down from 10 seconds to 0 seconds, only displaying whole seconds.
   3. When the user clicks on the screen, the red square teleports to the location of the click. Any blue dots that are covered by the squared are permanently removed from the screen.
   4. If all blue dots have been removed, the user wins. The game stops and displays "You Win".
   5. After the 10 second timer has counted down to 0, the game stops and displays "Game Over".
   6. When the game is stopped, the time left stays at 0 and the user cannot get additional points.
   7. At any time, the user can press "r" to reset the game and play again.

Answer space for Question 8

Answer space for Question 8