15-112 Spring 2023 Lecture 3+4
Quiz 7 Handout

- **Use this handout to complete quiz7.**
- **Submit this handout, but do not write on it.**

```
def testLetterCounterClass():
    print('Testing LetterCounter class...', end='')
    # A "LetterCounter" instance is constructed with some target letters ('AB' in
    # the first example below).  It is then given text (with the addText method),
    # and it counts the number of times any of those target letters appear
    # in the text (case-insensitively).
    lc1 = LetterCounter('AB')
    assert(lc1.getCount() == 0)
    assert(str(lc1) == "LetterCounter(letters='AB', count=0)")
    assert(str([lc1]) == "[LetterCounter(letters='AB', count=0)]")

    lc1.addText('A big cat!') # adding 2 As and 1 B
    assert(lc1.getCount() == 3)
    assert(str(lc1) == "LetterCounter(letters='AB', count=3)")

    lc1.addText('Another big one!') # adding 1 more A and 1 more B
    assert(lc1.getCount() == 5)
    assert(str(lc1) == "LetterCounter(letters='AB', count=5)")

    # Note that the letters are always displayed in alphabetical order,
    # and without duplicates, and always in uppercase, so:

    lc2 = LetterCounter('cbc') # counts the number of Bs or Cs in text
                               # this is exactly the same as LetterCounter('BC')
    assert(lc2.getCount() == 0)
    assert(str(lc2) == "LetterCounter(letters='BC', count=0)")

    lc2.addText('A big cat!') # adding 1 B and 1 C
    assert(lc2.getCount() == 2)
    assert(str(lc2) == "LetterCounter(letters='BC', count=2)")

    # (continued on next page)
    # To add two LetterCounters, combine the letters they are counting (without
    # creating duplicates), and add their counts, so:
    lc3 = lc1 + lc2
    assert(lc3.getCount() == 7)
    assert(str(lc3) == "LetterCounter(letters='ABC', count=7)")
    print('Passed!')

# End of testLetterCounterClass()
```

```
def testLineAndVerticalLineClasses():
    print('Testing Line and VerticalLine classes...', end='')
    line1 = Line((4, 5), (7, 1))
    assert(str(line1) == 'Line from (4, 5) to (7, 1)')
    assert(str([line1]) == '[Line from (4, 5) to (7, 1)]')

    # getLength returns the length between the endpoints of the line:
    assert(almostEqual(line1.getLength(), 5.0))

    # two lines are equal if their endpoints match in either order:
    assert(line1 == Line((4, 5), (7, 1)))
    assert(line1 == Line((7, 1), (4, 5)))

    # and they are not equal if either of their endpoints do not match:
    assert(line1 != Line((4, 5), (7, 0)))

    # and do not crash here:
    assert(line1 != 'yikes!')

    # The class VerticalLine is a subclass of Line.
    # A VerticalLine only takes one point (x, y) and makes a vertical line
    # from that point to the x axis:
    line2 = VerticalLine((3, 7))
    assert(str(line2) == 'VerticalLine from (3, 7) to (3, 0)')
    assert(almostEqual(line2.getLength(), 7))

    # Verify that it is a subclass
    assert(type(line2) == VerticalLine)
    assert(isinstance(line2, Line))

    # A VerticalLine can equal a Line if their endpoints match (in either order):
    line3 = Line((3, 0), (3, 7))
    assert(line2 == line3)

    # (continued on next page)
    # Lines can be added to sets:
    s = set()
    line4 = Line((4, 0), (4, 3))
    s.add(line4)
    assert(line4 in s)

    # Note that a line with the points in the other order still matches.
    # Be sure that hash(line4) == hash(line5).  To do this, we suggest you
    # sort the endpoints before hashing them, though other approaches will
    # also work:
    line5 = Line((4, 3), (4, 0))
    assert(hash(line4) == hash(line5))
    assert(line5 in s)

    # This hash also has to match a VerticalLine that is equal to the Line in s:
    line6 = VerticalLine((4, 3))
    assert(hash(line4) == hash(line6))
    assert(line6 in s)

    print('Passed!')
# End of testLineAndVerticalLineClasses()
```

15-112 Spring 2023 Lecture 3+4
Quiz 7
32 minutes

Name: ____________________________________

Andrew ID: ____________________@andrew.cmu.edu

Section: _______

- **You may not use any books, notes, or electronic devices during this quiz.**
- **You may not ask questions about the quiz except for language clarifications.**
- **Show your work on the quiz (not scratch paper) to receive credit.**
- **If you use scratch paper, you must submit it with your andrew id on it, and we will ignore it.**
- **All code samples run without crashing unless we state otherwise. Assume any imports are already included as required.**
- **Do not use these topics: recursion.**
- **You may use almostEqual() and rounded() without writing them. You must write everything else.**

Do not write below here

Question	**Points**	**Score**
1. FR: LetterCounter	50	
2. FR: Line and VerticalLine	50	
3. Bonus	5 (bonus)	
TOTAL	100	

1. FR: LetterCounter Class [50 pts]

Write the LetterCounter class so that the test function (on a separate handout) works properly. To receive full credit, you must not hardcode methods, and you must use OOP properly.

This page is for your LetterCounter class.

2. FR: Line and VerticalLine Classes [50 pts]

Write the Line and VerticalLine classes so that the test function (on a separate handout) works properly. To receive full credit, you must not hardcode methods, and you must use OOP properly (for example, you must call the super's __init__ method appropriately if you override its __init__ method).

This page is for your Line and VerticalLine classes.

3. Bonus [5 pts, 2.5 pts each]
Indicate what these print. Place your answers (and nothing else) in the box next to each block of code.

```
def bonusCt1(x, y):
    def inc(L):
        for i in range(y-1, -1, -1):
            L[i] += 1
            if L[i] < x:
                break
            else:
                L[i] = 0
    L = [0] * y
    L[-1] = 1
    z = 0
    while L != [0] * y:
        inc(L)
        z += 1
    return z
print(bonusCt1(3, 4))
```

```
def bonusCt2(j):
    code = '''
class A:
    def __init__(self, x): self.x = x
    def f(self): return self.x*10'''
    for i in range(j):
        A = chr(ord('A') + i)
        B = chr(ord('B') + i)
        code += f'''
class {B}({A}):
    def __init__(self, x): super().__init__({i+2}*x)
z = {B}(2).f()'''
    exec(code, globals()) # hint: this evaluates the given code
    return z
print(bonusCt2(3))
```