Name: _____ Andrew Id: _____

**15-112 Fall 2021 Quiz 8**

Up to 25 minutes. No calculators, no notes, no books, no computers. Show your work!
Do not use try/except on this quiz.

1. (5 points) **Code Tracing**: Indicate what the following program prints. Place your answer (and nothing else) in the box next to the code.

```
def ct(a, b, d=0):
    print(f"a:{a} b:{b} d:{d}")
    n = b - a
    if n <= 1:
        return
    ct(a, a+n//2, d+1)
    ct(a+n//2, b, d+1)

ct(0,3)
```

2. (5 points) **Reasoning Over Code**: Find an argument, `L`, for the function `roc` to cause it to return `True`. Place your answer (and nothing else) in the box below the code.

```
def roc(L):
    return mystery(L, 2, 0)

def mystery(L, v, d):
    assert (v == L[0])
    if len(L) == 1:
        return d == 4
    return mystery(L[1:], v+L[0], d+1)
```

3. (10 points) **Free Response**

Consider the following code designed to test two different classes: Vehicle and Motorcycle:

```
tester():
    # You can create a vehicle with a name and capacity
    v = Vehicle("Row Boat", 2)
    assert(str(v) == "Row Boat contains 0:")

    # You can add passengers up to the capacity. The following two lines each
    # add a passenger.
    assert(v.addPassenger("Ken") == True)
    assert(v.addPassenger("Barbie") == True)
    assert(str(v) == "Row Boat contains 2: Ken Barbie")

    # Once the vehicle is full, you can't add any more passengers.
    assert(v.addPassenger("Ted") == False)
    assert(str(v) == "Row Boat contains 2: Ken Barbie")

    # Remove the most recently added passenger.
    assert(v.removePassenger() == True)
    assert(str(v) == "Row Boat contains 1: Ken")

    # If a passenger is removed, then space is freed up to add another one.
    assert(v.addPassenger("Ted") == True)
    assert(str(v) == "Row Boat contains 2: Ken Ted")

    # You can't remove a passenger from an empty vehicle
    assert(v.removePassenger() == True)
    assert(v.removePassenger() == True)
    assert(v.removePassenger() == False)
    assert(str(v) == "Row Boat contains 0:")

    # A Motorcycle is a vehicle with an assumed name of Motorcycle and
    # a capacity of 1
    m = Motorcycle()
    assert(str(m) == "Motorcycle contains 0:")
    assert(m.addPassenger("Ken") == True)
    assert(m.addPassenger("Ted") == False)
    assert(str(m) == "Motorcycle contains 1: Ken")

    # A motorcycle is both a Motorcycle and a Vehicle
    assert(isinstance(m, Motorcycle) == True)
    assert(isinstance(m, Vehicle) == True)

    # While a motorcycle is doing a wheelie, you can't remove a passenger
    m.popWheelie()
    assert(m.removePassenger() == False)
    assert(str(m) == "Motorcycle contains 1: Ken")

    # Once the wheelie stops, you can remove as usual
    m.stopWheelie()
    assert(m.removePassenger() == True)
    assert(str(m) == "Motorcycle contains 0:")
```

Write the classes Vehicle and Motorcycle 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.

Answer space for Question 3