fullName:_____andrewID:_____
recitationLetter:_____

15-112 N22

Quiz3 version B [35 min.]

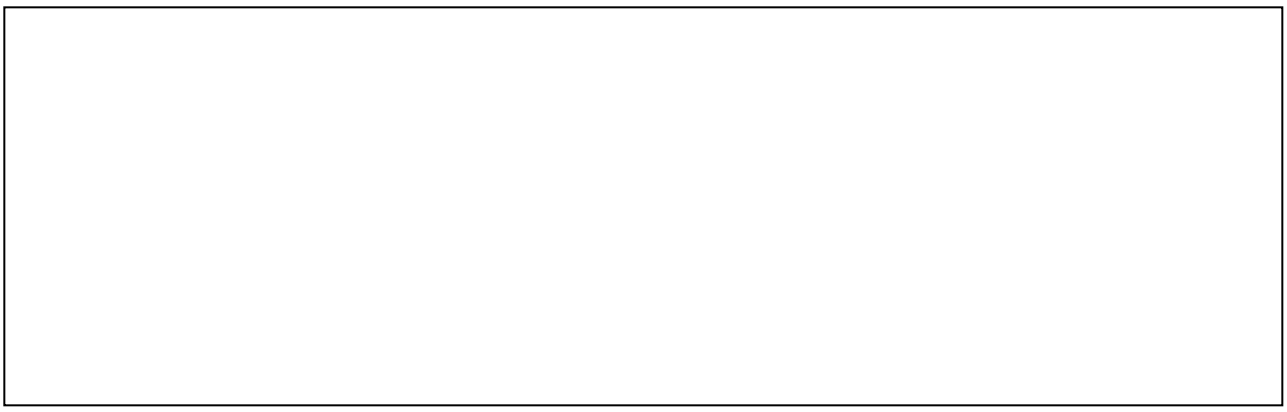
You **MUST** stop writing and hand in this **entire** quiz when instructed in lecture.

- You may not unstaple any pages.
- You may not use your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand this in with your paper quiz, and we will not grade it.
- Failure to hand in an intact quiz will be considered cheating. Discussing the quiz with anyone in any way, even briefly, is cheating. (You may discuss it only once the quiz has been posted to the course website.)
- You may not use any concepts (including builtin functions) we have not covered in the notes this semester.
- You may not use dictionaries, sets, or recursion.
- We may test your code using additional test cases. Do not hardcode.
- We do not deduct points for bad style on quizzes
- Assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you. You must write all other helper functions you wish to use.

CT1: Code Tracing [10pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
# Hint: Draw a box and arrow diagram!
# prints 5 lines total
def ct1(a):
    (b, c) = (a, a[:2])
    b[0] += 3
    a += [3]
    a = a + [9]
    print(c + [b[0]])
    print(c.append(b[1]))
    print(a)
    print(b)
    return c
print(ct1([1, 6]))
```



CT2: Code Tracing [12pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
# Hint: Draw a box and arrow diagram!
# prints 4 lines total
import copy
def ct2(a):
    b = a
    c = copy.copy(a)
    b[0] = 42
    a.append("owo")
    c[3] = 2 * c[3]
    b = b[:2] + ["hi"] + b[2:]
    c.remove(1)
    a[-2] = 112
    print(a)
    print(b)
    print(c)
z = ["jun", "28", 1, 100]
ct2(z)
print(z)
```

Free Response 1: destructive shortenLongRuns(L, k) [32 points]

Write the destructive function shortenLongRuns(L, k) that takes a non-empty list of integers L and a positive integer k and destructively modifies L by removing any values in L that would otherwise produce a run of k consecutive equal values in L.

```
def testShortenLongRuns():
    print("Testing destructiveShortenLongRuns...", end="")
    L = [2, 3, 5, 5, 5, 3]
    shortenLongRuns(L, 2)
    assert(L == [2, 3, 5, 3])

    L = [2, 3, 5, 5, 5, 3]
    shortenLongRuns(L, 3)
    assert(L == [2, 3, 5, 5, 3])

    L = [-1, -1, -1, 3, 5, 5, 5, 5]
    shortenLongRuns(L, 3)
    assert(L == [-1, -1, 3, 5, 5])

    L = [2, 3, 5, 5, 5, 3]
    shortenLongRuns(L, 1)
    assert(L == [])

    print("Passed!")
```

You may continue your FR1 answer here, if you wish

Free Response 2: Vehicle and Car Classes [32pts]

Write the classes Vehicle and Car so that they pass the following test cases. You may not hardcode any test cases. For full credit, Car must be a subclass of Vehicle, and it should inherit as many methods and attributes from the Vehicle class as possible.

```
def testVehicleAndCarClasses():
    # A Vehicle has one property: whether or not it is currently moving.
    v1 = Vehicle(False)
    assert(v1.isMoving == False)
    # A vehicle can move and brake
    v1.move()
    assert(v1.isMoving == True)
    assert(str(v1) == "Vehicle(Moving)")
    v1.brake()
    assert(v1.isMoving == False)
    assert(str(v1) == "Vehicle(Stopped)")

    # A Car is a vehicle with an engine. The engine must be on for the car to move.
    # Note that the first param is related to moving; the second checks the engine.
    c1 = Car(False, False)
    assert(c1.isMoving == False and c1.engineOn == False)
    assert(str(c1) == "Car(Stopped, Engine Off)")

    c1.move()
    # c1.move() should not change c1.isMoving if the engine is off
    assert(str(c1) == "Car(Stopped, Engine Off)")
    c1.startEngine()
    assert(str(c1) == "Car(Stopped, Engine On)")
    c1.move()
    assert(str(c1) == "Car(Moving, Engine On)")
    c1.brake()
    assert(str(c1) == "Car(Stopped, Engine On)")
    # Nothing stops us from making cars with weird start states
    assert(str(Car(True, False)) == "Car(Moving, Engine Off)")
    # Check for inheritance
    assert(isinstance(c1, Vehicle) == True)
    assert(isinstance(v1, Car) == False)

    # Finally, calling v1.startEngine() should crash. Only Cars have engines!
testVehicleAndCarClasses()
```

You may continue your FR2 answer here, if you wish

You may continue your FR2 answer here, if you wish

True or False [2pts ea]

Write only the whole word "True" or "False" (and not just T or F).

_____ **TF1:** Objects and lists can have aliases.

_____ **TF2:** A class is an instance of an object.

_____ **TF3:** .sort() and .reverse() are destructive, while sorted(L) and reversed(L)are nondestructive.

_____ **TF4:** In the Snake case study, we represented the snake as a list of (row, col) tuples.

_____ **TF5:** In the Snake case study, doStep(app) calls timerFired(app).

_____ **TF6:** The .__init__() method is also known as a "constructor."

_____ **TF7:** In 112, OOP usually stands for "Out Of Parentheses."

bonusCT: Code Tracing [2pts]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def bonusCt(L, M):
    while L != M:
        L,M = M,L
        M.append(L.pop(0))
        M.pop(0)
        L[-1] = sum(M)
    return L
print(bonusCt([1,2,3],[4,5,6]))
```