andrewID:_____ recitationLetter:_____

Quiz 4 version A (25min)

Part 1: Code Tracing

You may not run any code in Part 1. Once you move to Part 2, you may not return to Part 1.

Note that there are **three** FRs this time (but they are not very big). Still, save time for them.

CT1: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not run this code.

```
#prints 5 lines total
def ct1():
    a = list(range(2,10,6))
    (b, c) =  (a, a[:2])
    b[0]  +=  3
    a +=  [3]
    a = a + [7]
    print(c + [b[0]])
    print(c.append(b[1]))
    print(a)
    print(b)
    return c
print(ct1())
```

CT2: Code Tracing [15pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not run this code.

```
#Prints 2 lines total
import  copy
def f(a,  b):
    a = copy.copy(a)
    a[0]  = b[1]
    b[0]  = a[1]
    return  a + b

def ct2():
    a1 = list(range(3,5))
    b1 = list(range(2))
    a2, b2 = a1, b1
    a1  = f(a1, b1)
    print(a1 + ['x'] + a2)
    print(b1 + ['x'] + b2)
ct2()
```

Part 2

Note: Part 3 is a (very challenging) bonus code tracing problem worth 2 points. Once you move on, you may not return to Part 1 or Part 2.

Free Response 1: median(a) [20pts]

Write the non-destructive function median(a) that takes a list of floats and returns the median value, which is the value of the middle element, or the average of the two middle elements. If the list is empty, return None.

You may not import or use any module other than copy. You may not use any method, function, or concept that we have not covered this semester. We may use additional test cases not shown here. Do not hardcode.

```
# Note: almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
def almostEqual(d1, d2, epsilon=10**-7): #helper-fn
    return (abs(d2 - d1) < epsilon)

import decimal
def roundHalfUp(d): #helper-fn
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))

import copy
#---------------

#Write your function here:
def median(a):
    return 42

def testMedian():
    print('Testing median()...', end='')
    a = [0.5, 3.2, 5.6, 7.3, 10.4]
    b = copy.copy(a)
    assert(almostEqual(median(a), 5.6))
    assert(a == b)                              #Test for non-destructiveness
    assert(almostEqual(median([0.5, 3.2, 5.0, 6.0, 7.3, 10.4]), 5.5))
    assert(almostEqual(median([7.3, 0.5, 6.0, 10.4, 3.2, 5.0]), 5.5))
    assert(almostEqual(median([-7.3, -6.0, 10.4, -3.2, 0.5]), -3.2))
    assert(almostEqual(median([-3.0, 3.0]), 0.0))
    assert(almostEqual(median([11.2]), 11.2))
```

```
    assert(median([]) == None)                #Return None for empty lists
    print('Passed!')

testMedian()
```

Free Response 2: prepend(a,b) [25pts]

Write the function prepend(a, b) which destructively modifies a (without modifying b) by adding all the values of b onto the front of a, and returns the usual value returned by destructive functions. So this code works:

```
a = [1, 2]
b = [3, 4]
prepend(a, b)
assert((a == [3,4,1,2]) and (b == [3,4]))
```

You may not import or use any module other than copy. You may not use any method, function, or concept that we have not covered this semester. We may use additional test cases not shown here. Do not hardcode.

```
# Note: almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
def almostEqual(d1, d2, epsilon=10**-7): #helper-fn
    return (abs(d2 - d1) < epsilon)

import decimal
def roundHalfUp(d): #helper-fn
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))

import copy
#----------------

#Write your function here:
def prepend(a, b):
    return 42

def testPrepend():
    print('Testing prepend()...', end='')
    a = [1, 2]
    b = [3, 4]
    prepend(a, b)
    assert((a == [3,4,1,2]) and (b == [3,4]))
    prepend(b, a)
    assert((a == [3,4,1,2]) and (b == [3,4,1,2,3,4]))
    c = [5]
    prepend(a, c)
    assert((a == [5, 3, 4, 1, 2]) and (c == [5]))
```

```
    print('Passed!')

testPrepend()
```

Free Response 3: noDuplicateTypes(a) [25pts]

Write the function noDuplicateTypes(a) that takes a list of values and returns True if no two values in the list are the same Python type, and False otherwise.

You may not import or use any module other than copy. You may not use any method, function, or concept that we have not covered this semester. We may use additional test cases not shown here. Do not hardcode.

```
# Note: almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
def almostEqual(d1, d2, epsilon=10**-7): #helper-fn
    return (abs(d2 - d1) < epsilon)

import decimal
def roundHalfUp(d): #helper-fn
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))

import copy
#---------------

#Write your function here:
def noDuplicateTypes(a):
    return 42

def testNoDuplicateTypes():
    print('Testing noDuplicateTypes()...', end='')
    assert(noDuplicateTypes([42]) == True)
    assert(noDuplicateTypes([42, True]) == True)
    assert(noDuplicateTypes([42, True, 24]) == False)
    assert(noDuplicateTypes([42, True, 24.0, 'False']) == True)
    assert(noDuplicateTypes([True, 57, False]) == False)
    assert(noDuplicateTypes(['One string and one tuple', (1, 2, 3)]) == Tr
    assert(noDuplicateTypes([]) == True)
    print('Passed!')

testNoDuplicateTypes()
```

Bonus Part 3

This part is optional. You may not return to Parts 1 or 2.

bonusCT1: Code Tracing [2pts]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not run this code.

```
def bonusCt(a,  n):
    try:  a.append(2*n  + a[0]);  return  n + f(a[3:],  n+1)
    except: return  42
a = list(range(3));
print(bonusCt(a,  3))
print(a)
```