andrewID:_____ recitationLetter:_____

Quiz 2 version A (25min)

Part 1: CTs and Fill-In-The-Blank

You may not run any code in Part 1. Once you move to Part 2, you may not return to Part 1. **We recommend leaving most of your time for Part 2**

CT1: Code Tracing [20pts]

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 foo(x, d):
    return d + (x//3 if (x < 10) else x//2)

def ct1(x):
    counter = 0
    target  = 6  #does not start at 0
    while (target < 8):
        if (foo(counter, x) == target):
            print(f'c:{counter}, t:{target}')
            target += 1
        counter += 1
    return  f'c:{counter}, t:{target}'  #careful!
print(ct1(0))  #prints 3 lines
```

CT2: Code Tracing [20pts]

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 ct2(x, y):
    for z in range(x, y):
        if (z < y/2):
            if (z%4 == 0): print(f'b:{z}')
            elif (z%2 == 0): print(f'e:{z}')
        elif (x + y + z == 25):
            print(f'c:{z}')
    for z in range(y, x):
        print(f'a:{z}')
    w = 0
    for z in range(x, 10*x, x):
        if (z < 5*x): continue
        elif (z >= 7*x): return f'd:{z}+{w}'
        w += z
    return True

print(ct2(4,  14)) #prints 4 lines
```

Part 2: Free Response

Free Response: nthOnePrime(n) [60pts]

Write the function nthOnePrime(n) which takes a non-negative integer n and returns the nth "onePrime" (a coined term), which is a prime number that begins with 1 or ends with 1, but **NOT** both. So 41 is a onePrime, as is 193, but 11 is not, and 191 is not. See the test cases for more examples.

Hint: You may want to write one or two well-chosen helper functions. To receive any credit, you must not use strings, lists, sets, dictionaries, or other concepts we have not covered in the notes this semester. Also, we may test your code against 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))
#---------------

#Write your function here:
def nthOnePrime(n):
    return 42



# We have provided these test cases:
def testNthOnePrime():
    print('Testing nthOnePrime()...', end='')
    assert(nthOnePrime(0) == 13)
    assert(nthOnePrime(1) == 17)
    assert(nthOnePrime(2) == 19)
    assert(nthOnePrime(3) == 31)
    assert(nthOnePrime(7) == 103)
    assert(nthOnePrime(23) == 211)
    assert(nthOnePrime(155) == 2011)
```

```
    print('Passed!')
testNthOnePrime()
```

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(x, y, z=32):
    return (x+y) if (z>32) else bonusCT(y, x+y, 2*(x+y))
print(bonusCT(5, 11))
```