fullName:_____andrewID:_____ recitationLetter:_____

15-112 F23

Quiz2 version A (25 min)

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

- You may not unstaple any pages.
- 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 your own scrap paper. If you must use additional scrap paper, raise your hand and we will provide some. You must hand any scrap paper in with your paper quiz, and we will not grade it.
- You may not ask questions during the quiz, except for English-language clarifications. If you are unsure how to interpret a problem, take your best guess.
- You may not use any concepts (including builtin functions) which we have not covered in the notes in weeks 1-2 / units 1-2.
- You may not use strings, lists, indexing, tuples, dictionaries, sets, or recursion.
- We may test your code using additional test cases. Do not hardcode.
- Assume almostEqual(x, y) and rounded(n) are both supplied for you. You must write all other helper functions you wish to use, unless we specify otherwise.

True or False [3pts ea]

Mark each of the following statements as True or False.

1. ◯ True ◯ False

When writing the function nthPrime (or nthEinNumber or another similar nth-pattern problem) we should usually use at least one while loop.

2. ◯ True ◯ False

The following code will eventually print 'done'

```
x = 1
while x != 20:
    x *= 2
print('done')
```

3. ◯ True ◯ False

The following code will print 'beep' 30 times:

```
for i in range(6):
    for j in range(6, 13, 2):
        print('beep')
```

4. ◯ True ◯ False

The following code will eventually print 'done':

```
x = 1
while True:
    x += 1
    if x == 5:
        break
print('done')
```

5. ◯ True ◯ False

The following code will eventually print 'done':

```
x = 1
while True:
    if x > 5:
        continue
    x += 1
print('done')
```

CT1: Code Tracing [14pts]

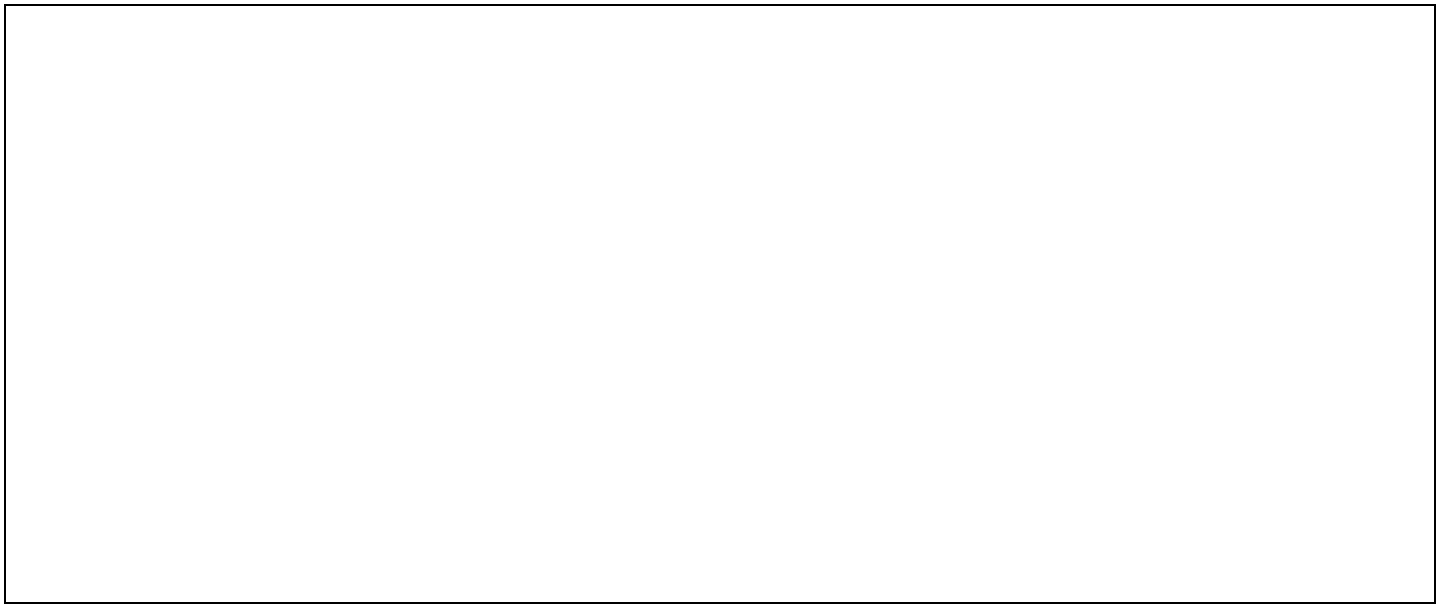
Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct1(m, n):
    x = 1
    while x < 5:
        x += 2
        print('x =', x)
        for y in range(m, n, x):
            print('y =', y)
    return x
print(ct1(1, 7))
```

CT2: Code Tracing [14pts]

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

```
def ct2(a, b):
    for x in range(a):
        y = x
        if x % 2 == 1:
            print(f'x = {x}')
            continue
        while y < b:
            y += 2
            print(f'y = {y}')
            if y == 2 * x:
                print('break')
                break
        if x == y:
            print('x == y')
ct2(4, 5)
```



Free Response 1: iterationsUntilOne(n) [25pts]

The Collatz algorithm takes an integer n and repeats the following steps until n becomes 1:

- If n is is even, divide it by two and assign the new value to n
- Otherwise, if n is odd, n becomes `3*n + 1`

Write the function `iterationsUntilOne(n)` that takes a positive integer n and returns the number of steps it takes the algorithm to reach 1. Note, we will only test your function on integers which eventually reach 1.

Here are some examples:
iterationsUntilOne(1) returns 0 because we begin at 1, and no steps are needed to reach 1

iterationsUntilOne(2) returns 1, because 2 is even, and we perform 1 step to divide it by two and reach 1 (so its sequence is 2, then 1)

iterationsUntilOne(7) returns 16. The sequence of n is as follows: 7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

Remember, do not use strings, lists, or any material not covered in the notes or in lecture for weeks 1 and 2.

```
assert(iterationsUntilOne(1) == 0)
assert(iterationsUntilOne(2) == 1)
assert(iterationsUntilOne(3) == 7)
assert(iterationsUntilOne(4) == 2)
assert(iterationsUntilOne(5) == 5)
assert(iterationsUntilOne(7) == 16)
assert(iterationsUntilOne(112) == 20)
```

Begin your FR1 answer on the following page

Begin your FR1 answer here

Free Response 2: biggestTriple(n) [32pts]

Write the function `biggestTriple(n)` that takes an integer n (which may be negative) and returns the largest three-digit number formed by consecutive (i.e. neighboring) digits in n. The digits of the result should be next to each other and in the same order as they appear in n. For example: biggestTriple(12341234) should return 412.

The function should return None if no three-digit numbers can be formed. For example: biggestTriple(42) should return None.

If n is negative, the negative sign should be ignored, so: biggestTriple(-623412) should return 623.

Remember, do not use strings, lists, or any material not covered in the notes or in lecture for weeks 1 and 2.

```
assert(biggestTriple(12341234) == 412)
assert(biggestTriple(249027) == 902)
assert(biggestTriple(3414089) == 414)
assert(biggestTriple(100) == 100)
assert(biggestTriple(42) == None)      #No three-digit number can be formed
assert(biggestTriple(-623412) == 623)  #Ignore the negative sign
assert(biggestTriple(-3414089) == 414)
```

Begin your FR2 answer here or on the following page

You may begin or continue your FR2 answer here, if you wish