fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

WS2+Quiz2 version B (30 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.

Writing Session 2 (10% of HW2)

WS1. Here is an alphabetized list of some exercises that were assigned in hw2, and a couple of others that were not:

```
A) findZeroWithBisection
B) longestIncreasingRun
C) mostFrequentDigit
D) nthEinNumber
E) nthEmirpPrime
F) stepsToReach495
G) sumOfEvenDigits
```

Each of the following quotes is an excerpt from a writeup of an exercise that was assigned in hw2. For each, give the letter of the exercise in the list above which includes the excerpt.

1.1) "Construct two new numbers by arranging the digits of n in ascending and descending order."

1.2) "Find the midpoint between x0 and x1, which we will call xMid."

1.3) "The odd digits in the number can be in any order."

Quiz2

CT1: Code Tracing [15pts]

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

```
def ct1(n):
    for x in range(n//2, n):
        for y in range(n, x, -2):
            print(f'{x},{y+1}')
ct1(5)
```

CT2: Code Tracing [15pts]

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

```
def ct2(n):
    x = 0
    for v in range(n, 3*n, 3):
        while v > 0:
            x = 10*x + (v%10)
            v //= 10
    return x
print(ct2(5))
```

CT3: Code Tracing [15pts]

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

```
def ct3(n):
    z = 3*n
    for y in range(-n, n):
        if abs(y) % 3 == 0:
            continue
        elif y == n//2:
            break
        else:
            z += abs(y)
    return z
print(ct3(5))
```

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

Background: we will say that an integer n is threeish (a coined term) if:

- `n >= 100`
- and the difference between each consecutive digit in n is 3

Consider the number `n = 1474`:

- `1474 >= 100`
- and the difference between each consecutive digit is:

```
|1 - 4| == 3
|4 - 7| == 3
|7 - 4| == 3
```

So we see that 1474 is threeish.

Here are the first 10 threeish numbers: 141, 147, 252, 258, 303, 363, 369, 414, 474, 525

With that, write the function nthThreeish(n) that takes a non-negative integer n, and returns the nth threeish number, so nthThreeish(0) returns 141.

```
def testNthThreeish():
    assert(nthThreeish(0) == 141)
    assert(nthThreeish(1) == 147)
    assert(nthThreeish(2) == 252)
    assert(nthThreeish(3) == 258)
    assert(nthThreeish(9) == 525)
```

Begin your FR1 answer on the following page

Begin your FR1 answer here:

You may continue your FR1 answer here, if needed:

Free Response 2: nearestZany(n) [30pts]

Background: we will say that a number is 'zany' (a coined term) if it is a positive integer where the sum of the even digits equals the sum of the odd digits.

For example, 112 is zany because the sum of the even digits is 2, and the sum of the odd digits is `1+1 == 2`.

The first several zany numbers are: 112, 121, 134, 143, 156, 165, 178, 187, 211, ...

With this in mind, write the function `nearestZany(n)` that takes a possibly-negative integer n and returns the nearest zany number to n. If there is a tie, return the smaller one.

For example:

- nearestZany(0) returns 112
- nearestZany(116) returns 112
- nearestZany(117) returns 121

To be clear, 199 is 12 away from two zany numbers: 187 and 211. Here, we return the smaller, so `nearestZany(199)` returns 187.

Hint: if you cannot solve `nearestZany(n)`, at least try to write the helper function `isZany(n)` for partial credit.

```
def testNearestZany():
    assert(nearestZany(0) == 112)
    assert(nearestZany(116) == 112)
    assert(nearestZany(117) == 121)
    assert(nearestZany(198) == 187)
    assert(nearestZany(199) == 187) # 12 away from 187 and 211
    assert(nearestZany(200) == 211)
    assert(nearestZany(-1000) == 112) # Handle negative numbers
```

Begin your FR2 answer on the following page

Begin your FR2 answer here:

You may continue your FR2 answer here, if needed:

The problems below are not required. Indicate what the following code prints. Place your answers (and nothing else) in the boxes below.

bonusCt1 [optional, 1pt]

```
def bonusCt1(n):
    def z(n):
        m = 0
        while n > 0: m,n = m+n%2, n//2
        return m == 1
    def f(n, d):
        m = n
        while not z(n): n += d
        return abs(m - n)
    return f'{f(n,+1)}.{f(n,-1)}'
print(bonusCt1(99))
```

bonusCt2 [optional, 1pt]

```
def bonusCt2(n):
    t = 1
    for d in range(10):
        for k in range(n):
            t += (str(d) in str(k))
    return t
print(bonusCt2(100))
```