fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

WS3+Quiz3 version A (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 1-3 / units 1-2.
- You may not use lists, list 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 3 (10% of HW3)

WS1. What does this print:

```
print(decodeCaesarCipher('Bdc', 2))
```

WS2. What does this print:

```
print(interleave('abc', 'de'))
```

WS3. In solvesCryptarithm, the solution to the puzzle 'NUMBER + NUMBER = PUZZLE' is 'UMNZP-BLER'. What is the numeric value assigned to the letter 'Z' in this solution?

WS4. Write the few lines of arithmetic (just arithmetic, no code) required to show that 2212 is a happy number.
(Reminder: this problem somehow involves sums of squares of digits.)

Quiz3

CT1: Code Tracing [15pts]

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

```
def ct1(s):
    t = s
    c = 'd'
    while len(s) < 5:
        s += c
        c = chr(ord(c) + 2)
    return f'{c}-{s}-{t}'
print(ct1('ab'))
```

CT2: Code Tracing [15pts]

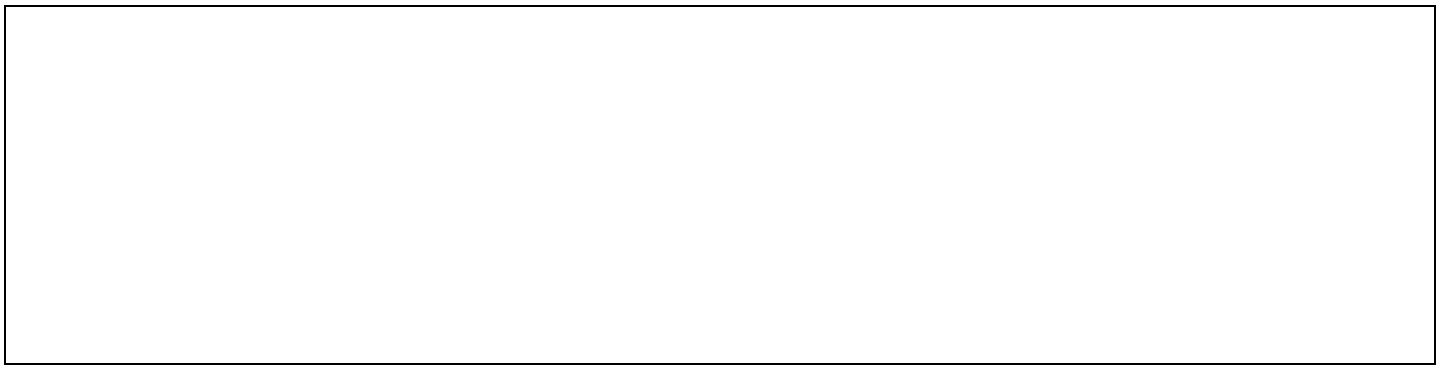
Indicate what the following code prints. Place your answers (and nothing else) in the box below.

```
def ct2(s):
    t = str(s.find('b'))
    while s[0] in s[1:]:
        t += s[0].upper()
        s = s[2:len(s)-1]
    return s.replace('b', t)
print(ct2('aBbDabcdE'))
```

CT3: Code Tracing [15pts]

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

```
def ct3(s):
    u = ''
    for t in s.split('-'):
        u += t + '\n'
    for t in s.split():
        u += t + '\n'
    for v in u.splitlines():
        if v.isdigit():
            print(f'{v*2}{int(v)*2}')
        else:
            print(v)
ct3('1-2 3')
```



Free Response 1: areRotatedStrings(s, t) [25pts]

Write the function `areRotatedStrings(s, t)` that takes two values, s and t, that can be of any type and returns True if they are both strings and s is a rotation of t, and False otherwise.
Do not crash if s or t are not strings.

Note that rotations preserve order, so the only rotations of 'abc' are 'bca' and 'cab'. Also, a string is not a rotation of itself.

Hence:

```
assert(areRotatedStrings('abc', 'bca') == True)
assert(areRotatedStrings('abc', 'cab') == True)
assert(areRotatedStrings('abc', 'bac') == False) # not a rotation!
assert(areRotatedStrings('abc', 'abc') == False) # same string!

assert(areRotatedStrings('abc def', 'bc defa') == True)
assert(areRotatedStrings('abc def', ' defabc') == True)
assert(areRotatedStrings('abc def', 'efabc d') == True)
assert(areRotatedStrings('abc def', 'efabc ') == False)
assert(areRotatedStrings('', '') == False)
assert(areRotatedStrings(123, '231') == False)
assert(areRotatedStrings('123', 231) == False)
assert(areRotatedStrings(True, 3.14) == False)
```

Begin your FR1 answer here or on the following page:

You may begin or continue your FR1 answer here:

Free Response 2: getShortestPath(path) [30pts]

Background: we will consider a "path" to be a possibly-empty string containing any combination of the letters U, D, L, R, which stand for the directions Up, Down, Left, and Right.

You can imagine that we start at the origin (0,0), and take one step in the given direction for each character in the path.

For example, the path 'UUR' goes up to (0, 1), up to (0, 2), and right to (1, 2). Note that the path 'UURRL' also winds up at (1, 2), but it takes some extra steps to do so.

With that in mind, write the function `getShortestPath(path)` that takes a path as just described, and returns the shortest path that ends at the same point. Since there will be more than one such shortest paths, you should return the first one alphabetically.

For example: for the path 'UURRL' from above, which ends at (1,2) there are 3 shortest paths that end at (1, 2): 'UUR', 'URU', and 'RUU'. Of these, 'RUU' is the first alphabetically. Hence:
`assert(getShortestPath('UURRL') == 'RUU')`

Remember, you may not use lists.

```
assert(getShortestPath('UURRL') == 'RUU')
assert(getShortestPath('UUDR') == 'RU')
assert(getShortestPath('DLUDUDUDLRLRLRD') == 'DDL')
```

Begin your FR2 answer here or on the following page:

You may begin or continue your FR2 answer here:

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]

```
import string

def bonusCt1(s):
    def f(s, t): return ('' if s[0] in t else s[0]) + f(s[1:], t) if s else s
    return int(f(string.digits[:0:-1], s)) - 13

print(bonusCt1('The CMU SCS phone number is 412-268-7884!  No kidding!'))
```

bonusCt2 [optional, 1pt]

```
def bonusCt2(s):
    def f(s):
        t = 0
        for v in s.split(): t += int(v)
        return t
    while f(s) < 250: s *= 2
    return f(s)
print(bonusCt2('1 23 4'))
```