Name:_____ andrewID:_____ Recitation:___

15-112 F24
Quiz3 version B (25 min)

Read these instructions carefully before starting:

1. Quiz versions are color-coded. You must have a different version (color) of this quiz than the students sitting to your left and right.
2. Stop writing and submit the entire quiz when instructed by the proctor.
    - Do not unstaple any pages.
    - You must submit the entire quiz with all pages intact.
3. Do not discuss the quiz with anyone else until after 4pm.
    - This applies to everyone, including students in either lecture.
4. Do not use your own scrap paper.
    - You should not need scrap paper, there is plenty of room for you on the quiz.
    - However, if you absolutely must use scrap paper, raise your hand and we will provide some. Then, you must write your andrew id clearly on the scrap paper, and hand in the scrap paper with your paper quiz. We will not grade anything on your scrap paper.
5. You may not ask questions during the quiz.
    - The one exception is for English-language clarifications.
    - If you are unsure how to interpret a problem, just take your best guess.
6. Do not use any concepts (including built-in functions) not covered in the notes through week 3 / unit 3.
    - Do not use lists, tuples, dictionaries, sets, or recursion.
7. Do not hardcode your solutions.
    - We may test your code using additional test cases.
    - Hardcoding will receive zero points.
8. 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.
9. Good luck!

Code Tracing (CT)) [15 pts, 5 pts each]
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

CT1:

```
def ct1(s):
    t = ''
    while s != '':
        t += s[-1]
        s = s[1:-1]
        s = s[::-1]
    return t

print(ct1('CDEFG'))
```

CT2:

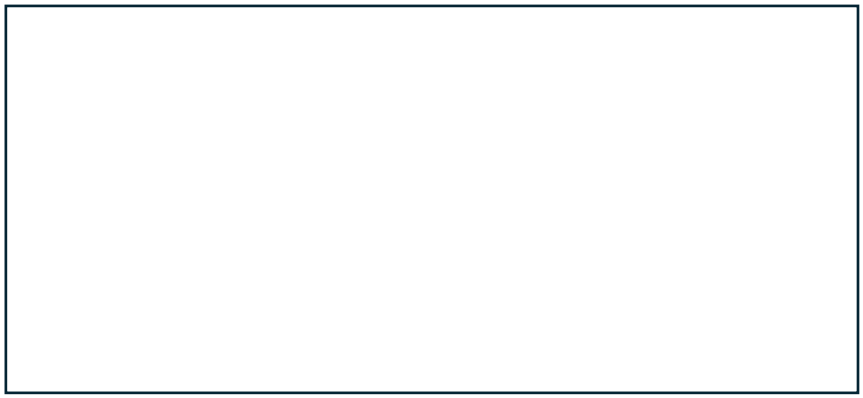
```
def ct2(s):
    t = s
    s = s.lower()
    s = s.replace('c', f'e,{t},')
    for t in s.split(','):
        print(t)

ct2('BCE')
```

CT3:

```
def ct3(s):
    t = ''
    d = 1
    for i in range(len(s)):
        t += chr(ord(s[i]) + d)
        d = -d
    for c in s:
        t = f'{c}{t.find(c)}.' + t
    return t

print(ct3('ceb'))
```



Free Response / FR: duplicateThreeLetterSubstrings [85 pts]

For this problem, assume that all strings contain only letters or spaces.

Given a string s, we will say a substring u starting at index i of s is a "duplicate three-letter substring" if:

- u is of length 3
- u contains no spaces

AND there is another substring v starting at index j of s where:

- u == v (but this test is case-insensitive, so lowercase matches uppercase).

With this in mind, write the function duplicateThreeLetterSubstrings(s), which you can abbreviate as DTLS(s), that takes a string s containing only letters (upper or lower case) or spaces, and returns a single string with the comma-separated duplicate three-letter substrings in s, in uppercase, listed in the order they first appear in s.

For example, in the string s = 'abc def def abc', we see the first 'abc' before the first 'def', so 'ABC' comes before 'DEF' in the result.

Also:

- Each DTLS should be included only once in the result, no matter how many times it occurs in s.
- If there are no DTLS in s, return None (not the empty string).

Here are some test cases for you:

```
assert(DTLS('aaa') == None)
assert(DTLS('aaaa') == 'AAA')
assert(DTLS('aaaaaaaaaa') == 'AAA')
assert(DTLS('abc def def abc') == 'ABC,DEF')
assert(DTLS('abcdc cdc abc') == 'ABC,CDC')
assert(DTLS('cdc abcdc abc') == 'CDC,ABC')
assert(DTLS('CDc aBCDc ABc') == 'CDC,ABC')
assert(DTLS('cdc abcd abc abcdcba') == 'CDC,ABC,BCD')
assert(DTLS('abc bcd') == None)
assert(DTLS('') == None)
```

Begin your FR answer on the next page.

Begin your answer to the FR here:

Bonus Code Tracing (BonusCT)) [Optional, 4 pts, 2 pts each]
Bonus problems are not required.
For each CT, indicate what the code prints
Place your answer (and nothing else) in the box below the code.

BonusCT1:

```
import string
def bonusCt1(s):
    i, t = 0, string.ascii_uppercase
    while len(s) < 25:
        i, s = i+1, s.replace(t[i], t[i:i+3])
    return len(s)
print(bonusCt1('ABD'))
```

BonusCT2:

```
# hint: chr(66) == 'B'
def bonusCt2(t):
    s = (string.ascii_uppercase + string.digits +
         string.ascii_lowercase)
    for c in s: t += ord(c) % 2
    s = ''
    for c in string.ascii_uppercase:
        s = s + string.ascii_uppercase[(ord(c) - t)%26]
        t -= 1
        if len(s) == 4: break
    return s
print(bonusCt2(2))
```