Name:_____ Section:___ Andrew Id: _____ a

15-112 Fall 2019 Quiz 3a
*** Up to 25 minutes. No calculators, no notes, no books, no computers. * Show your work!**
*** No lists, list indexing, or recursion**

1. **Free Response: getAverage(values)** [35 pts]
   Write the function getAverage(values) that takes a string of comma-separated non-negative integer values, and returns their average as a float (even though the values themselves are integers). Note that some values may not be non-negative integers, and these should be ignored. If there are no integer values, return None (do not crash here).

   For example, getAverage('13,excused,14,absent') ignores the two strings and averages 13 and 14 to return 13.5. Also, getAverage('a,b,c') returns None.

```
def testGetAverage():
    print('Testing getAverage()...', end='')
    assert(getAverage('13,excused,14,absent') == 13.5)
    assert(getAverage('3,excused,4,absent,5') == 4)
    assert(getAverage('3,excused,4,absent,5,-123') == 4)
    assert(getAverage('a,b,0,c') == 0)
    assert(getAverage('a,b,c') == None)
    assert(getAverage('') == None)
    print('Passed!')
```

   Hint: we suggest that you use split() here.

2. **Free Response: encode(s)** [35 pts]:
   Background: a simple way to encode a string s is to add extra letters. Here, we first add an A, then a B, and so on, before each letter in the original string. Thus, if the string is 'cat', our encoded string would be 'AcBaCt'. Do not add anything before non-letters, so we encode 'm333N4' as 'Am333BN4'. Note: we will only add the letters from A to D. If the original string is long enough, then whenever we add D, we wrap around so the next letter we add is A again. Thus, we encode 'qrstuv' as 'AqBrCsDtAuBv'.

   With this in mind, write the function encode(s) that takes a string s and returns the encoded version of that string as just described.

```
def testEncode():
    print('Testing encode()...', end='')
    assert(encode('cat') == 'AcBaCt')
    assert(encode('m3N4') == 'Am3BN4')
    assert(encode('qrstuv') == 'AqBrCsDtAuBv')
    assert(encode('x') == 'Ax')
    assert(encode('') == '')
    print('Passed!')
```

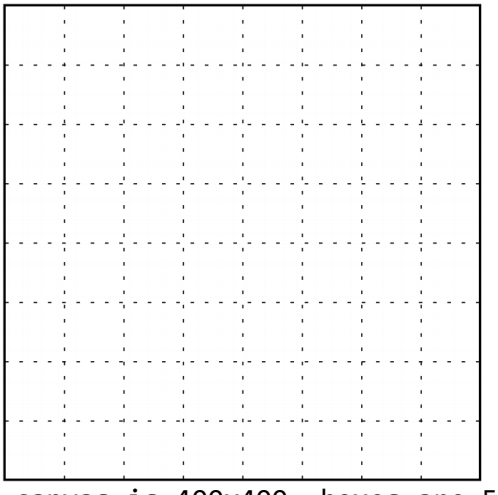
3. **Code Tracing** [10 pts]:
Indicate what this prints. Place your answer (and nothing else) in the box next to the code.

```
def ct1(s, t):
    t0 = t
    for i in range(len(s)):
        if (s[i] == t[-1]):
            print(i)
        else:
            t = s[i].upper() + t
    return t.replace(t0, 'Z')

print(ct1('abc', 'ab'))
```

4. **Graphics Code Tracing** [10 pts]:
Sketch what this will draw when run on a 400x400 canvas:

```
import math
def draw(canvas, width, height):
    canvas.create_line(200, 0, 200, 400)
    canvas.create_line(0, 200, 400, 200)
    x, y = 300, 200
    t = 0
    for q in range(4):
        t += 90
        u, v, = x, y
        x = 200 + 100 * math.cos(math.radians(t))
        y = 200 - 100 * math.sin(math.radians(t))
        canvas.create_line(u, v, x, y)
```



*this canvas is 400x400, boxes are 50x50

5. **Reasoning Over Code** [10 pts]:
Find arguments for the following functions that makes them return True. Place your answers (and nothing else) in the box beside the code:

```
def rc1(s):
    code = ''
    p = -42
    for c in s:
        if (p+1 == ord(c)):
            code += c
        p = ord(c)
    return (code == 'yes')
```

s =

6. **Bonus/Optional: Code Tracing** [2.5 pts]
Indicate what this prints. Very clearly circle your answer (and nothing else):

```
def bonusCt1(r):
    for i in range(128):
        if (chr(i).isalpha()):
            r += ord(chr(i).lower()) - ord('a')
    r = str(r)
    return eval(f'{r[:1]}{r[-1]}/{r[1]}')

print(bonusCt1(4))
```