fullName:_____andrewID:_____ recitationLetter:_____

15-112 S24

Midterm1 version B

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

- You may not unstaple any pages.
- Failure to hand in an intact exam will be considered cheating. Discussing the exam with anyone in any way, even briefly, is cheating. (You may discuss it only once the exam 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 exam, and we will not grade it.
- You may not ask questions during the exam, 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 1-5 / units 1-4.
- You may not use dictionaries, sets, or recursion.
- We may test your code using additional test cases.
- 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.

CT1: Code Tracing [6pts]

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

```
def ct1(x, y):
    x, y = y, x
    print(x, y)
    print(int(bool(x-2)), int(bool(y-2)))
    print(float(x)*x, str(x)*x)
    n = 5**4**3
    print(n == (5**4)**3, n == 5**(4**3))
print(ct1(3, 2))
```

CT2: Code Tracing [6pts]

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

```
import math

def g(n):
    return n - math.ceil(n/10)

def f(n):
    n += g(n) if n%2 == 0 else g(2*n)
    return n + g(n)

def ct2(n):
    print(f(n), f(n+1))

ct2(4)
```

CT3: Code Tracing [7pts]

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():
        if t.isalpha():
            u += t.lower()
            continue
        elif t.isdigit():
            t = chr(ord('C') + int(t)%10)
        elif t[0] == t[-1]:
            break
        u += f't{t}'
    return u

print(ct3('AB 4C 23 A4A FF'))
```

CT4: Code Tracing [7pts]

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

```
def ct4(L):
    M = L
    N = [v//10 for v in L]
    print(N)
    M.append(N.pop(0))
    L = sorted(L)
    print(L)
    print(M)
    print(N)

L = [2, 12]
ct4(L)
print(L) # do not miss this!
```

Free Response 1: isEightish(n) and nthEightish(n) [18pts]

Note: you may not use strings or lists here.

Background: we will say that a number is eightish (a coined term) if it is at least 100 and if the sum of its first and last digit is 8, and the sum of all the digits not including its first and last digit is also 8.

For example: 12067 is eightish because 1+7==8 and 2+0+6==8.

The first 10 eightish numbers are: 187, 286, 385, 484, 583, 682, 781, 880, 1087, 1177

With this in mind, without using strings or lists, **write the function isEightish(n)** which takes an integer n and returns True if it is eightish and False otherwise. **Also write the function nthEightish(n)** that takes a non-negative integer n and returns the nth eightish number where nthEightish(0) returns 187.

```
def testIsEightish():
    assert(isEightish(12067) == True)
    assert(isEightish(12064) == False) # 1+4 != 8
    assert(isEightish(12047) == False) # 2+0+4 != 8
    assert(isEightish(3200000065) == True)
    assert(isEightish(8) == False)  # too small
    assert(isEightish(71) == False) # too small
    assert(isEightish(-12067) == False) # too small

def testNthEightish():
    assert(nthEightish(0) == 187)
    assert(nthEightish(1) == 286)
    assert(nthEightish(2) == 385)
    assert(nthEightish(9) == 1177)
```

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: decode123(t) [18pts]

Background: recall that in a Caesar Cipher, each letter in a string s is shifted by some amount d, preserving case (so uppercase letters become uppercase letters, and lowercase letters become lowercase letters), and wrapping around from 'Z' back to 'A'.
So a Caesar Cipher with d==2 shifts 'A' to 'C', shifts 'd' to 'f', and shifts 'Z' to 'B'. Thus, it shifts 'AdZ' to 'CfB'. Also, non-letters are included as-is, so it shifts 'Ad Z!' to 'Cf B!'.

We just invented the 123 Cipher, which works like a Caesar Cipher but the first letter is shifted by 1, the second by 2, the third by 3, and then this repeats, so the fourth letter is shifted by 1, the fifth letter by 2, the sixth letter by 3, and so on. Non-letters stay the same.

For example, if s == 'Abc 12 xyZ!':

- 'A' is shifted by 1 to 'B'
- 'b' is shifted by 2 to 'd'
- 'c' is shifted by 3 to 'f'
- 'x' is shifted by 1 to 'y'
- 'y' is shifted by 2 to 'a'
- 'Z' is shifted by 3 to 'C'
- the non-letters stay the same and do not affect the shift

Thus, encode123('Abc 12 xyZ!') returns 'Bdf 12 yaC!'.

For this problem, you will write the DECODER for this encoder. That is, write the function decode123(t) that takes t, which is the result of encode123(s), and return the original string s.

Thus, decode123('Bdf 12 yaC!') returns 'Abc 12 xyZ!'.

In general, for any string s, decode123(encode123(s)) == s.

Hint: While it is not required, we recommend you write the helper function shift(c, delta), which takes a string c of length 1 and an integer delta. If c is a letter, it returns the letter properly shifted by delta. Otherwise, it just returns the unmodified character.

```
def testDecode123():
    assert(decode123('Bdf 12 yaC!') == 'Abc 12 xyZ!')
    s = 'Cwqev fbmht cuf lxtv eji gppxuu!'
    assert(decode123(s) == 'Bundt cakes are just big donuts!')
    assert(decode123('') == '')
```

Begin your FR2 answer on the following page

Begin your FR2 answer here:

You may continue your FR2 answer here, if needed:

Free Response 3: mutateIntoColumn(L, M) [18pts]

Background: for this problem, you will take a 1d list L and a rectangular 2d list M. Your goal is to mutate L by inserting exactly one value somewhere into L so that L then equals at least one of the columns in M.

For example, say:

```
M = [ [ 1, 4, 7 ],
      [ 2, 5, 8 ],
      [ 3, 6, 9 ] ]
```

and:

```
L = [1, 2]
```

Here, we can insert the value 3 at index 2 in L, so L becomes [1, 2, 3], which equals column 0 of M.

As another example, if M is the same as above, but

```
L = [4, 6]
```

Here, we can insert the value 5 at index 1 of L, so L becomes [4, 5, 6], which equals column 1 of M.

To continue, if:

```
L = [8, 9]
```

Here we can insert the value 7 at index 0 of L, so L becomes [7, 8, 9], which equals column 2 of M.

As a counterexample, if:

```
L = [1, 2, 3]
```

L is too big to turn into a column of M with one insertion, so this is impossible.

Also, if:

```
L = [1]
```

L is too small to turn into a column of M with just one insertion.

With this in mind, **write the function mutateIntoColumn(L, M)** that takes a possibly-empty 1d list L and a non-empty rectangular 2d list M. If there is a way to turn L into a column of M with a single insertion, your function mutates L with that one insertion and returns True. Otherwise, if it is impossible, your function leaves L unmutated and returns False.
If L can be mutated into more than one column in M, you should use the leftmost column that is possible.

```
def testMutateIntoColumn():
    M = [ [ 1, 4, 7 ],
          [ 2, 5, 8 ],
          [ 3, 6, 9 ] ]
    L = [1, 2]
    assert(mutateIntoColumn(L, M) == True)
    assert(L == [1, 2, 3])

    L = [4, 6]
    assert(mutateIntoColumn(L, M) == True)
    assert(L == [4, 5, 6])

    L = [8, 9]
    assert(mutateIntoColumn(L, M) == True)
    assert(L == [7, 8, 9])

    L = [1, 2, 3]
    assert(mutateIntoColumn(L, M) == False)
    assert(L == [1, 2, 3])

    L = [1]
    assert(mutateIntoColumn(L, M) == False)
    assert(L == [1])

    M = [ [1, 2] ]
    L = []
    assert(mutateIntoColumn(L, M) == True)
    assert(L == [1])

    M = [ [ 1, 3, 3 ],
          [ 2, 1, 2 ],
          [ 3, 2, 2 ],
          [ 1, 1, 1 ] ]
    L = [3, 2, 1]
    assert(mutateIntoColumn(L, M) == True)
    assert(L == [3, 1, 2, 1])
```

Begin your FR3 answer on the following page

Begin your FR3 answer here:

You may continue your FR3 answer here, if needed:

Free Response 4: Move Dot to Left or Right Edge [20pts]

Write an app that:

- Starts with a 400x400 canvas with a pink dot of radius 20 centered in the canvas.
- Sets app.stepsPerSecond = 10 and keeps it at that value.
- If the user presses the mouse outside the pink dot, then the pink dot moves so it is centered on that mouse press.
- If the user presses the mouse inside the pink dot, then:
    - The dot turns blue.
    - The blue dot moves horizontally either towards the left edge or the right edge of the canvas, whichever is closer to the blue dot's center. (Either direction is fine if it is perfectly centered.) The blue dot stays at the same position vertically on the canvas.
    - The blue dot moves at a constant speed so that exactly 1 second after it turns blue its center reaches the edge. At that time, the blue dot is removed from the canvas and a new pink dot of radius 20 appears centered in the canvas.
    - While the blue dot is moving, all mouse presses are ignored.

You must adhere to MVC rules. Code that violates MVC will be ignored, and will result in very large deductions.

Begin your FR4 answer here or on the following page:

Begin or continue your FR4 answer here:

Continue your FR4 answer here:

Continue your FR4 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]

```
def bonusCt1(L):
    def f(L):
        # Hint: you need to see the *pattern* here and not
        # just code trace
        M = [ ]
        for i in range(1, len(L), 2):
            x, y, z = L[i], L[i-1], 1
            while z < x: z *= y
            M.append(z-x)
        return M
    while L[1:]: L = f(L)
    return L
print(bonusCt1([4,2,5,20,2,5,4,11]))
```

bonusCt2 [optional, 1pt]

```
def bonusCt2(z):
    c = [0]*10
    for n in range(10**z):
        while n: c[n%10] += 1; n //= 10
    return 1+max(c)-min(c)
print(bonusCt2(2))
```