fullName:______________________ andrewID:_______________ section:___

15-112 F25
Midterm2 version 1B
80 Minutes

You **must write your name on this paper and hand this back** in immediately after the assessment. If we do not receive it immediately, you will receive a zero on the assessment. **Do not unstaple any pages**. All pages must be handed in intact.

Do not use your own scrap paper. You should not need it, but if you must absolutely have scrap paper, raise your hand and we will provide some. Write your andrewID clearly on it and hand it in with your exam. We will not grade anything on scrap paper.

You may not ask questions during the exam, except for English-language clarification questions. If you are unsure about a problem, take your best guess.

Before and during the exam, you may not view any other notes, prior work, websites or resources, including any form of AI. You may not use calculators, phones, laptops, or any other devices. You may not communicate with anyone else except for current 112 TAs or faculty during the assessment. All syllabus policies apply.

You may not discuss this test with anyone else, even briefly, in any form, until we have released grades. Failure to abide by these rules may result in an academic integrity violation.

Do not use sets, dictionaries, recursion, or anything we have not yet covered in class or the notes. Do not hardcode your answers. 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.

Do not open this or look inside (even briefly) before the instructors signal for you to begin. When the instructors indicate that time is up, you must *immediately* stop writing, close this document, and hand it in. Do not look at anyone else's exam.

Code Tracing

CT1 [10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def f(x):
    return g(x+1) if x%2 == 0 else g(x)+1

def g(x):
    c = 2
    while x > 0:
        c += 1
        if c == 3:
            continue
        elif c == 6:
            break
        x -= c
    return abs(x)

def ct1(x):
    y = f(x)
    x *= y
    return (y, f(x)%y)

print(ct1(5))
```

CT2 [10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
def ct2(s):
    r = ''
    for t in s.splitlines():
        for u in t.split('.'):
            if u.isdigit():
                r = u + r
            else:
                v = [c if c.isalpha() else '' for c in u.upper()]
                for i in range(len(u)):
                    if v[i] == u[i]:
                        r += v[i]
    return r

s = '''\
5.6
9.2 Plus 4.7
Yes!'''

print(ct2(s))
```

CT3 [10 pts]:

Indicate what the following code prints. Place your answer (and nothing else) in the box below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

```
import copy
def ct3(L):
    A = copy.deepcopy(L)
    B = copy.copy(L)
    C = L
    B[0].append(A[1].pop())
    B.extend(C[0])
    C = C + B[1:3]
    print(f'A:{A}')
    print(f'B:{B}')
    print(f'C:{C}')
    return L.sort()

D = [[3], [4]]
ct3(D)
print(f'D:{D}')
```

Fill in the Blank (FitB): encode [10 pts]:

You are given the function encode(s) that solves the following problem, only with some parts removed. You need to fill in the blanks with the missing code.

Note: when you fill in blanks, you must not add any newlines and you must not use any semicolons.

With that, here is the writeup for encode(s):

The function encode(s) takes a string s that only contains uppercase letters, and returns an encoding of s where each character in s is replaced with a two-digit integer which is the difference from that character and 'A' (so 'A' is 00, 'B' is 01, and 'Z' is 25). Then, these two-digit values are all placed in a single string separated by dashes. Thus:

```
    assert(encode('ABYZ') == '00-01-24-25')
```

And here is the FitB solution (fill in the missing blanks)

```
def encode(s):

    L = [ ]

    for c in s:

        d = _________________________________________________________ # BLANK #1

        s = str(d)

        if d < 10:

            s = _____________________________________________________ # BLANK #2

        L.append(s)

    return '-'.___________________________________________________(L) # BLANK #3
```

Free Response

Your functions should work generally for the kinds of inputs specified in the problem statement, and we may test your code using additional test cases. We will manually grade free responses for partial credit if you do not pass all the test cases.

FR1 [20 pts]: nthPickety(n)

Note: You may not use strings or lists for this problem.

First, we will say that a positive integer n with an even number of digits can be "picketed" into two integers x and y formed by the alternating digits of n.

For example, if n = 123456, then x = 135 and y = 246.

Also, we will say that a non-negative integer n is "pickety" (a coined term) if it does not contain any zeros, it is positive, it has an even number of digits, and if n is picketed into x and y, then y is exactly 5 times x.

For example, say n = 1875. This is picketed into x=17 and y=85, and 17*5 equals 85. So 1875 is pickety.

As another example, say n = 151635. This is picketed into x=113 and y=565, and 113*5 equals 565. So 151635 is pickety.

With that in mind, write the function nthPickety(n) that takes a non-negative integer n and returns the nth pickety number.

Here are some test cases for you:

```
    assert(nthPickety(0) == 15)   # 1 * 5 == 5
    assert(nthPickety(1) == 1515) # 11 * 5 == 55
    assert(nthPickety(2) == 1635) # 13 * 5 == 65
    assert(nthPickety(3) == 1755) # 15 * 5 == 75
    assert(nthPickety(4) == 1875) # 17 * 5 == 85
    assert(nthPickety(5) == 1995) # 19 * 5 == 95
    assert(nthPickety(6) == 151515) # 111 * 5 == 555
```

Write your answer on the following page

FR2 [20 pts]: removeSmallestCols(L)

Write the function removeSmallestCols(L) that takes a rectangular 2d list L of integers, where L has at least 2 rows and at least 2 cols, and mutatingly changes L by removing the column in L that has the smallest sum. If more than one column has the smallest sum, remove all such columns. Your function should return None.

For example:

```
    # Here, the middle col sums to 3, which is smallest:
    L = [[5, 1, 3],
         [4, 2, 7]]
    M = [[5, 3],
         [4, 7]]
    assert(removeSmallestCols(L) == None)
    assert(L == M)

    # Here, all but the middle col sum to 2 which is smallest
    L = [[1, 2, 3, 3, 0],
         [1, 0, 7, 0, 1],
         [0, 0, 0,-1, 1]]
    M = [[3],
         [7],
         [0]]
    assert(removeSmallestCols(L) == None)
    assert(L == M)
```

`Write your answer on the following pages`

FR3 [20 pts]: Animation

Notes:

- You may not assume the canvas is 400x400, but you may assume it is never smaller than that.
- Assume app.stepsPerSecond is 25.
- You may not use align= for this problem.

Write an animation such that:

- At first, a 100x100 black square is drawn centered in the canvas.
- When 'g' is pressed, the square turns green and starts growing (with its center unchanged) until it is 200x200 at which point it stops growing and becomes black again.
- When 's' is pressed, the square turns silver and starts shrinking (with its center unchanged) until it is 100x100 at which point it stops shrinking and becomes black again.
- The rate of growth should be such that it takes 2 seconds to grow from 100x100 to 200x200.
- When the mouse is pressed outside of the square, the square becomes centered on that mouse press, drawn in black, and returns to 100x100 without growing (it will grow on the next 'g' press).
    - Note that this means the square may be partly off the canvas. You can ignore this.
- Mouse presses inside the square are ignored.

`Write your answer on the following pages`

BonusCT

These CTs are optional, and intended to be very challenging. They are worth very few points.
Indicate what the following code prints. Place your answers (and nothing else) in the boxes below. If a line of code crashes, just print "crash" (without quotes) and stop the CT at that point.

bonusCT1[1pt]:

```
def bonusCt1(n):
    def f(n): return [[n]*n for _ in range(n)]
    def g(L): return sum(L) if type(L[0]) == int else [g(M) for M in L]
    M = [f(v) for v in list(range(1, 6)) if v != v**2 - 12]
    while True:
        try: M = g(M)
        except: return M - n
print(bonusCt1(6))
```

bonusCT2[1pt]:

```
def bonusCt2(s):
    L = ['m', 'n']
    M = [ lambda x, y: x < y,
          lambda x, y: x > y ]
    for c in s:
        if c.isalpha():
            for i in range(2):
                f = M[i]
                if f(c, L[i]):
                    L[i] = c
    return s.split()[2].replace(s[-2],s[1]).join(L)
print(bonusCt2('Is this so cool'))
```