fullName:____________________________andrewID:____________________________
recitationLetter:_____ Lecture (1, 2, or 3)_____

submissionCode:________________

Midterm1 version B

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

- You may not unstaple any pages.
- 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 this in with your paper exam, and we will not grade it.
- Failure to hand in an intact exam will be considered cheating.
- Discussing the exam with anyone in any way, even briefly, is cheating.

Exam Outline:

- Part 1 contains 3 CTs and 3 MCs, and must be solved on paper. You will transcribe your CT answers in coconut during Part 3.
- Part 2 contains FR1 and FR2, which must be solved on paper only. You will not transcribe these in coconut.
- Part 3 contains FR3 and FR4, which must be solved in coconut only (though you may use exam pages for scrap work, which we will not grade).
- Part 4 contains two optional bonusCTs at the end. For credit, your answers must be in coconut.

You may not open your computer until instructed to begin the coconut portion of the exam.

Do not open this exam until instructed. Doing so will be considered cheating.

Part 1: Code Tracing and Multiple Choice

You may not run any code in Part 1.

CT1: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not ever run this code. You must copy your final answers into coconut.

```
#prints 5 lines
def ct1(x, m):
    while x >= 10:
        x, n = foo(x)
        print(x, n)
        m += n
    return m

def foo(x):
    n = x // 10
    x = x % 100
    return n, x

print(ct1(12515, 2))
```

CT2: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not ever run this code. You must copy your final answers into coconut.

```python
# Prints 1 line
def ct2(s):
    z = ""
    i = 0
    while (len(z) < 5) and (i < len(s)):
        c = s[i]
        if c.isdigit():
            i += int(c)
        elif c.isupper():
            i //= 2
        elif c.isalpha() == False:
            i *= -1
        else :
            i -= 1
        z += c
    return z
print(ct2("23aB?1"))
```

CT3: Code Tracing [6pts]

Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not ever run this code. You must copy your final answers into coconut.

```
# Prints 4 lines
import copy
def ct3(a):
    b = copy.copy(a)
    c = copy.deepcopy(a)
    a[1] = "hack"
    b[0][1] = 20
    c[1][0] = a[1][1]
    c.append(b[0].pop(0))
    a.extend([a[0]])
    a[0][0]+=1
    print(f"a:{a}")
    print(f"b:{b}")
    print(f"c:{c}")
z = [ [ "cat", 4 ], [ "dog", 12 ] ]
ct3(z)
print(f"z:{z}")
```

Multiple Choice [2pts ea, 6pts total]

Clearly place an X in the blank next to the letter of each correct answer for the following 3 questions. You must answer these questions on paper, and not in coconut.

MC1: Recall the wordSearch case study, which takes `board` (a 2d list) and `word` (a string). Which of the following options **best** describes which function uses the variable `word`?
____ A) In wordSearch, when we make sure that `word` fits in our board.
____ B) In wordSearchFromCell, when we make sure that `board[startRow][startCol]` equals `word[0]`.
____ C) In wordSearchFromCell, when we make sure that the result of wordSearchFromCellInDirection equals `word`.
____ D) In wordSearchFromCellInDirection, when we access each character of `word`.

MC2: Which of the following options **best** describes how we generate a new piece of food in the snake case study?
____ A) Pick a random row and column. If it is occupied by the snake already, pick a new row and column. Repeat as many times as needed to find an empty cell.
____ B) Create a list of tuples representing all unoccupied cells and pick a random tuple from the list to put the food in.
____ C) Place the food in the cell that the snake most recently left behind.
____ D) Pick a random row and column. If the food is placed on the snake, the snake just immediately eats it, grows longer, and we place another piece.

MC3: Which of the following statements is NOT true?
____ A) Slicing a list *always* produces a shallow copy of the sliced elements.
____ B) *Every* destructive/mutating list method returns \`None\`.
____ C) When rotating a piece in Tetris, we blindly rotate the piece, then check to see if its position is legal, and undo the rotation if not.
____ D) Integers, strings, tuples, and booleans are all immutable.

Part 2: Paper-Only FRs

You must answer the Part 2 problems on paper only. They will not appear in coconut, and you will not need to transcribe your answers. Note that you will not be able to run any code for these Part 2 problems.

Free Response 1: Dot animation [20pts]

Write an animation such that:

1. A red dot of radius 10 appears centered in the canvas (the dimensions of which may vary).
2. The red dot radius grows by 5 every 50ms, until some part of the red dot would extend beyond the canvas, at which time its radius becomes 10 and starts growing again.
3. Wherever the user presses the mouse, a blue dot of radius 20 appears, centered on that point. If the user presses several times, there will be several blue dots.
4. When the red dot intersects any blue dot, the blue dot disappears.
5. If the user ever presses any key, the red dot permanently disappears (and so it will no longer intersect any blue dots), but the user can continue to make more blue dots.

Hint: Two circles intersect if the distance between their centers is less than the sum of their radii.

You may not use any concepts we have not covered in the notes this semester. Assume the graphics framework has already been imported for you.

Free Response 2: containsAliases(L) [12pts]

Write the function containsAliases(L) that takes a 2d list L (that you can assume is rectangular) and returns True if any two rows in L are aliases of each other, and False otherwise. For example:

```
M = [1, 2]
L = [ [3, 4], [5, 6], M, [7, 8], M ] # contains aliases!
print(containsAliases(L)) # True
```

And:

```
M = [1, 2]
L = [ [3, 4], [5, 6], M, [7, 8], [1, 2] ] # contains no aliases!
print(containsAliases(L)) # False
```

Hint: the 'is' operator may be very useful here. Also, our solution is less than 10 lines long, and could be shorter still.
You may not use any concepts we have not covered in the notes this semester. We may test your code using additional test cases. Do not hardcode.

```
# Note: assume almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
#---------------
def containsAliases(L):
```

```
def testContainsAliases():
    print('Testing containsAliases(L)...', end='')
    M = [1, 2]
    L = [ [3, 4], [5, 6], M, [7, 8], M ] # contains aliases!
    assert(containsAliases(L) == True)
    L = [ [3, 4], [5, 6], M, [7, 8], [1, 2] ] # no aliases!
    assert(containsAliases(L) == False)
    L = [[0]*5] * 6   # contains aliases!
    assert(containsAliases(L) == True)
    L = [[0]*5 for i in range(6)] # no aliases!
    assert(containsAliases(L) == False)
    print('Passed!')

testContainsAliases()
```

Part 3: Coconut-Only FRs

You must answer the Part 3 problems in coconut. We will not grade any work for the Part 3 problems on paper. Note that you may run your code for these problems (and these problems only).

Free Response 3: nearestTastyNumber(n) [20pts]

We will say that a positive integer n is tasty (a coined term) if it contains no digits smaller than 5 (so, no 4's, 3's, 2's, 1's, or 0's) and the sum of its even digits equals the sum of its odd digits.

The smallest tasty number is 5667 (note that 5+7 == 6+6). Thus, nearestTastyNumber(0) == 5667.

With this in mind, first write the helper function isTastyNumber(n) which takes a possibly-negative integer n and returns True if it is a tasty number and False otherwise.

Then write the function nearestTastyNumber(n) that takes a possibly-negative integer n and returns the nearest tasty number to n. If there is a tie, return the larger tasty number.

For full credit, nearestTastyNumber(n) must check significantly fewer than n numbers to see if it is tasty (unless n itself is very small). If your submission naively checks all or most numbers between 0 and n, you will receive a -10 deduction.

In order to receive any credit, you may not use strings anywhere in this problem. You may not use any concepts we have not covered in the notes this semester. We may test your code using additional test cases. Do not hardcode.

```
# Note: almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
def almostEqual(d1, d2, epsilon=10**-7): #helper-fn
    return (abs(d2 - d1) < epsilon)

import decimal
def roundHalfUp(d): #helper-fn
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))
#---------------
#Write your function in coconut
#(We will not grade your paper work)
def isTasty(n):
    return 42

#Write your function in coconut
#(We will not grade your paper work)
def nearestTastyNumber(n):
    return 42

def testNearestTastyNumber():
    print('Testing nearestTastyNumber(n)...', end='')
    assert(nearestTastyNumber(0) == 5667)
    assert(nearestTastyNumber(6000) == 5986)
    assert(nearestTastyNumber(-100) == 5667)
                        #Negatives should still work
    assert(nearestTastyNumber(8668) == 8677)
                        #Halfway between 8659 and 8677
    assert(nearestTastyNumber(66666) == 66699)
                        #Some solutions may be too slow!
    print('Passed!')

testNearestTastyNumber()
```

Free Response 4: getRecord(team, scores) [24pts]

Background: in a certain sport, the scores of games are given like so:

```
scores = '''\
Chi 2 - Pit 1
Chi 2 - Pit 11
Mia 13 - Pit 0
Pit 4 - Mia 4
Chi 2 - Mia 3'''
```

The scores are a multiline string with no blank lines. Teams are always represented as three-letter strings. Each line contains these values, in order:

1. The home team's name
2. The home team's score
3. A dash
4. The visiting team's name
5. The visiting team's score

From this, we can compute the record of a team, which is a tuple of 3 values -- that team's number of wins, losses, and ties, in that order.

For example, for Pit, in the example above, they lost the first game to Chi, then won the next game against Chi, then lost to Mia, then tied Mia. So overall, Pit's record is (1, 2, 1) -- 1 win, 2 losses, 1 tie.

With this in mind, write the function getRecord(team, scores) that takes a team name and a list of scores as described above, and returns a tuple of 3 values, (wins, losses, ties), for that team using those scores.

You may not use dictionaries to solve this problem, and you may not use any concepts we have not covered in the notes this semester. We may test your code using additional test cases. Do not hardcode.

```
# Note: almostEqual(x, y) and roundHalfUp(n) are both supplied for you.
# You must write all other helper functions you wish to use.
def almostEqual(d1, d2, epsilon=10**-7): #helper-fn
    return (abs(d2 - d1) < epsilon)

import decimal
def roundHalfUp(d): #helper-fn
    # Round to nearest with ties going away from zero.
    rounding = decimal.ROUND_HALF_UP
    return int(decimal.Decimal(d).to_integral_value(rounding=rounding))
#---------------

#Write your function in coconut
#(We will not grade your paper work)
def getRecord(team, scores):
    return 42

def testGetRecord():
    print('Testing getRecord()...', end='')
    scores = '''\
    Chi 2 - Pit 1
    Chi 2 - Pit 11
    Mia 13 - Pit 0
    Pit 4 - Mia 4
    Chi 2 - Mia 3'''
    assert(getRecord('Pit', scores) == (1, 2, 1))
    assert(getRecord('Mia', scores) == (2, 0, 1))
    assert(getRecord('Chi', scores) == (1, 2, 0))
    assert(getRecord('Det', scores) == (0, 0, 0))
    print('Passed')

testGetRecord()
```

Bonus Part 4

This part is optional. We will only grade your answers in coconut.

bonusCT1: Code Tracing [2pts]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not run this code.

```
def bonusCT1(L): #(2pts)
    def f(L): return L[:]*len(L[:]) + [val*len(L) for val in L]
    def g(L, n): return len(L) if (len(L) >= n) else g(f(L), n)
    return g(L, 42)
print(bonusCT1(list(range(5))))
```

bonusCT2: Code Tracing [2pts]

This question is optional. Indicate what the following code prints. Place your answers (and nothing else) in the box below. Note that you may not run this code.

```
def bonusCT2(): #(2pts)
    import math
    a = [int(math.log(x,3)) for x in range(1,3)]
    for k in range(1234): a = [a[0] or  k,  a[1]  and k]
    return int("".join([str(x)*x**y for x in a for y in a]))
print(bonusCT2())
```