15-112
Fall 2026 Exam 2
October 20, 2026

Name:

Andrew ID:

- You may not use any books, notes, or electronic devices during this exam.
- You may not ask questions about the exam except for language clarifications.
- **Do not tear any pages out of the exam**. Doing so may invalidate your exam.
- **Write legibly**; you may not receive credit if the instructor cannot read your scribbles.
- Show your work on the exam to receive credit.
- If you write an answer on the back of any pages, make sure to make a note of this in the expected answer location.
- All code samples run without crashing. Assume any imports are already included as required.
- Do not use these post-exam 2 topics/constructs: sets, maps/dictionaries, recursion, or classes/OOP.

Do not write anything in the table below.

Question	Points	Score
1	30	
2	15	
3	25	
4	30	
Total:	100	

There are 100 points on this exam, lasting 75 minutes.

1. **Code Tracing**

   (a) (10 points) Indicate what the following program prints. Place your answers (and nothing else) in the box below the code. (Hint: Drawing a picture as you work will help.)

```
def ct1(A, B):
    C = A
    A = A + B[:1]
    B.append(C.pop())
    C.insert(0, B[-1] * 2)
    B = B[1:]
    B += [len(A)]
    A[0] = len(C)
    print(A)
    print(B)
    return C

L = [4, 7, 1]
M = [2, 5]
N = ct1(L, M)
N.append(0)
print(L)
print(M)
print(N)
```

(b) (10 points) Indicate what the following program prints. Place your answers (and nothing else) in the box below the code. (Hint: Drawing a picture as you work will help.)

```
def ct2(M):
    A, B, C = M, copy.copy(M), copy.deepcopy(M)
    A[0].append(B[1][1])
    B[1] = A[0]
    B[1][0] = 9
    C[0] = M[1]
    M[1] += [7]
    A = [M[1], M[1]]
    A[0][0] = 'x'
    print(A)
    print(B)
    print(C)

L = [[1, 2], [3, 4]]
print(ct2(L))
print(L)
```

(c) (10 points) Indicate what the following program prints. Place your answers (and nothing else) in the box below the code.

```
def ct3(L):
    M = []
    for i in range(len(L)):
        if L[i] % 3 == 0:
            M.insert(0, L[i])
        elif i % 2 == 1:
            M.append(L[i] * 2)
    print(M)
    L.extend(M[-2:])
    L.remove(max(M))
    print(L)
    while L[0] < L[-1]:
        L.append(L.pop(0))
    print(L)
    return L.index(6) + L.count(6)

print(ct3([4, 9, 2, 7, 6, 5, 3]))
```

2. (15 points) **Free Response: Smaller Before**

Write the **destructive** function `smallerBefore(L)` that takes a list of integers `L` and modifies it so that the value at index `i` becomes the number of elements that come *before* index `i` in the original `L` and are strictly smaller than the original `L[i]`. The function returns `None`.

Restrictions:

- You may not make a copy of `L` (for example, with slicing, `copy.copy`, or a loop that copies its values into another list).
- Your approach must be destructive in nature. You may not, for example, build a new list and then copy it into `L` at the end.

Hint: Think carefully about the order in which you want to check (and modify) the values in `L`.

Examples:

```
L = [5, 2, 6, 1]
assert smallerBefore(L) == None
assert L == [0, 0, 2, 0]
# 5 has nothing before it; 2 has 5 (not smaller);
# 6 has 5 and 2; 1 has nothing smaller before it

L = [4, 9, 4, 1, 9]
smallerBefore(L)
assert L == [0, 1, 0, 0, 3]

L = [1, 2, 3]
smallerBefore(L)
assert L == [0, 1, 2]

L = [3, 3, 3]
smallerBefore(L)
assert L == [0, 0, 0]          # strictly smaller

L = [-1, -5, 0, -2]
smallerBefore(L)
assert L == [0, 0, 2, 1]

L = []
smallerBefore(L)
assert L == []
```

Additional Space for Answer to Question 2

3. (25 points) **Free Response: Word Path**

Write the function `wordPath(L, word)` that takes a non-empty, non-ragged 2D list `L` of single letters, **where every letter is unique**, and a non-empty string `word`. It returns the path that spells out `word` in the grid, as a list of `(row, col)` tuples, one per letter of the word, in order.

- Consecutive letters in the word must be in adjacent cells, either vertically or horizontally. Diagonal moves are not allowed.
- A cell may be visited more than once, but two consecutive letters can't come from the same cell.
- If the word can't be spelled this way, because a letter isn't in the grid or two consecutive letters aren't adjacent, the function returns an empty list.
- The grid does not have to be square.
- Your function must be non-destructive.

For example, consider this grid (row and column indices shown in gray):

	0	1	2
0	c	a	t
1	o	r	s
2	d	e	n

The word `"code"` goes down the left column and then right to `e`, so its path is `[(0, 0), (1, 0), (2, 0), (2, 1)]`. The word `"art"` cannot be spelled, because `r` at `(1, 1)` and `t` at `(0, 2)` are only diagonally adjacent.

Examples:

```
L = [['c', 'a', 't'],
     ['o', 'r', 's'],
     ['d', 'e', 'n']]
assert wordPath(L, 'cat') == [(0, 0), (0, 1), (0, 2)]
assert wordPath(L, 'code') == [(0, 0), (1, 0), (2, 0), (2, 1)]
assert wordPath(L, 'rat') == [(1, 1), (0, 1), (0, 2)]
assert wordPath(L, 'tat') == [(0, 2), (0, 1), (0, 2)] # revisit
assert wordPath(L, 'a') == [(0, 1)]
assert wordPath(L, 'art') == []     # r -> t is diagonal
assert wordPath(L, 'ten') == []     # t -> e not adjacent
assert wordPath(L, 'dog') == []     # no 'g' in the grid
assert wordPath(L, 'error') == []   # 'rr' reuses one cell
assert wordPath(L, 'Cat') == []     # 'C' is not 'c'

L2 = [['h', 'i'],
      ['o', 'p'],
      ['u', 's']]
assert wordPath(L2, 'hip') == [(0, 0), (0, 1), (1, 1)]
assert wordPath(L2, 'hop') == [(0, 0), (1, 0), (1, 1)]
assert wordPath(L2, 'sup') == []    # u -> p is diagonal
```

Additional Space for Answer to Question 3

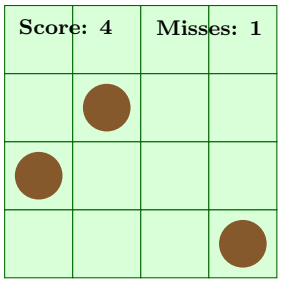
Additional Space for Answer to Question 3

4. (30 points) **Free Response: Whack-a-Mole**

Write an animation app with the following functionality:

1. The board is a 4×4 grid of light green cells with dark green borders that fills the entire canvas.
2. The game starts with no moles, a score of 0, and 0 misses. The score is shown in bold at the top-left of the canvas as `Score: X`, and the misses are shown in bold at the top-right as `Misses: Y`.
3. Once every second, a mole appears in a randomly chosen cell that does not already have a mole. A mole is drawn as a brown circle centered in its cell.
4. Each mole stays on the board for exactly 2 seconds. If it is not whacked in that time, it disappears and the number of misses goes up by 1.
5. Clicking a cell that has a mole *whacks* it: the mole disappears immediately and the score goes up by 1. Clicking a cell with no mole does nothing.
6. **Game Over:** When the number of misses reaches 3, the game ends, and the only thing displayed is the centered message `Game over! Score: X`, where X is the final score.
7. Pressing `r` at any time restarts the game.

Here is an example of the game in progress:



Notes and hints:

- You may assume `runApp(width=400, height=400)` is called.
- To solve this problem, you need to write `onAppStart`, `onKeyPress`, `onMousePress`, `onStep`, and `redrawAll`.
- You must leave `app.stepsPerSecond` at its default value of 30, and manage time by counting steps.
- One way to store the moles is a 4×4 2D list where each cell holds how many steps that mole has left (with 0 meaning no mole).

Additional Space for Answer to Question 4

Additional Space for Answer to Question 4