EECS 1015 Lab 11

Goal

- Be able to write a script that solves problems using nested collections

Tasks

- Learn to break down problems into smaller pieces with nested collections
- Learn to debug a script with nested collections
- Learn to write a relatively complicated script with nested collections
- Learn to write a relatively complicated script with nested collections

Total Credit: 100 pts

It is an academic offense to copy code from other students, provide code to other students, let other people (including the teaching staff) debug your code, or debug other students' code.

We will check code similarities at the end of the semester.

Task 1: Follow the Steps (30 pts)

For this task, you will create a function named separate_numbers() that takes a nested list as input. The function's objective is to separate odd and even numbers within each inner list and organize the results in a dictionary-style format. The output should be a dictionary where each key represents a list from the input, and the corresponding value is a nested dictionary containing the original list, odd numbers, and even numbers.

You are highly recommended to attempt the questions yourself before you look at the solution as a way to learn.

Q1.1 Based on the description above, declare the function and write the function recipe including test cases. You can fill the rest of the body of the function with a pass statement for now.

Note: You may need to include appropriate import statements yourself.

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
```

Q1.2 Planning out each step you need to take to achieve the function goal.

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    #Step 4: Create empty dictionary for results
    #Step 5: Iterate through the nested list
        #Step 6: Create empty lists for odd and even numbers
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.3 Check pre-conditions, i.e., make sure the user passed in positive integers.

Solution:

```
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    #Step 5: Iterate through the nested list
        #Step 6: Create empty lists for odd and even numbers
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.4 Create empty dictionary for results

Solution:

```
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
        #Step 6: Create empty lists for odd and even numbers
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.5 Iterate throughout the nested list

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
    for i in range(len(nested_list)):
        inner_list = nested_list[i]
        #Step 6: Create empty lists for odd and even numbers
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.6 Create empty lists for odd and even numbers

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
    for i in range(len(nested_list)):
        inner_list = nested_list[i]
        #Step 6: Create empty lists for odd and even numbers
        odds_list = []
        even_list = []
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.7: Create a Function for Checking Odd Numbers

Solution:

```python
#Step 7: Create a Function for Checking Odd Numbers
def is_odd(num: int) -> bool:
    """
    Check if a number is odd.

    Parameters:
    - num (int): The number to be checked.

    Returns:
    - bool: True if the number is odd, False otherwise.
    """
    return num % 2 != 0
```

Think about why do we need this function?

Q1.8: Separate Odd and Even Numbers

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
    for i in range(len(nested_list)):
        inner_list = nested_list[i]
        #Step 6: Create empty lists for odd and even numbers
        odds_list = []
        even_list = []
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        for num in inner_list:
            if is_odd(num):
                odds_list.append(num)
            else:
                even_list.append(num)
        #Step 9: Organize results in a dictionary for each inner list
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.9: Organize Results in a Dictionary

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
    for i in range(len(nested_list)):
        inner_list = nested_list[i]
        #Step 6: Create empty lists for odd and even numbers
        odds_list = []
        even_list = []
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        for num in inner_list:
            if is_odd(num):
                odds_list.append(num)
            else:
                even_list.append(num)
        #Step 9: Organize results in a dictionary for each inner list
        list_dict = {
            "list": inner_list,
            "odd_numbers": odds_list,
            "even_numbers": even_list
        }
        #Step 10: Add the dictionary to the main result dictionary
        Pass
```

Q1.10: Add Dictionary to Main Result Dictionary

Solution:

```python
def separate_numbers(nested_list: List[List[int]]) -> Dict[str, Dict[str, List[int]]]:
    """
    Takes a nested list, separates odd and even numbers within each inner list,
    and organizes the results in a dictionary-style format.

    Parameters:
    - nested_list (List[List[int]]): A nested list containing inner lists of numerical values.

    Test Cases:
    >>> separate_numbers([[1, 2, 3, 4, 5], [1, 2, 34, 5]])
    {'list_number_1': {'list': [1, 2, 3, 4, 5], 'odd_numbers': [1, 3, 5], 'even_numbers': [2, 4]}, 'list_number_2': {'list': [1, 2, 34, 5], 'odd_numbers': [1,
        5], 'even_numbers': [2, 34]}}
    """
    #Step 3: Check pre-conditions
    for inner_list in nested_list:
        for num in inner_list:
            assert isinstance(num, int) and num >= 0, "All elements must be non-negative integers."
    #Step 4: Create empty dictionary for results
    result_dict = {}
    #Step 5: Iterate through the nested list
    for i in range(len(nested_list)):
        inner_list = nested_list[i]
        #Step 6: Create empty lists for odd and even numbers
        odds_list = []
        even_list = []
        #Step 8: Iterate throughout the inner lists and append to the appropriate list:
        for num in inner_list:
            if is_odd(num):
                odds_list.append(num)
            else:
                even_list.append(num)
        #Step 9: Organize results in a dictionary for each inner list
        list_dict = {
            "list": inner_list,
            "odd_numbers": odds_list,
            "even_numbers": even_list
        }
        #Step 10: Add the dictionary to the main result dictionary
        result_dict[f"list_number_{i + 1}"] = list_dict
    return result_dict
```

Note: The code provided above works functional, but has a problem with the coding style. Can you identify it and fix it? (Note: this is optional)

Submission

- Go to eClass -> our course -> Week 12 -> Practice -> Lab -> Lab 11 Submission
- Copy your code to Task 1
- You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission.

Rubrics:

- You will get full marks for this question if you submit the solution (with no typo) to the required location on Prairielearn before the deadline.

Task 2: Debugging (30 pts)

In this task, Sam implemented a function called find_highest_correlation in lab11_task2.py that takes two arguments:

1. reference_list: A list of numerical values representing a reference sequence.
2. nested_list: A nested list where each sublist contains numerical values.

The goal is to find the sublist in the nested_list that has the highest Pearson correlation with the reference_list.

Sam implemented a separate function called pearson_correlation that calculates the Pearson correlation coefficient between two lists.

Pearson Correlation Coefficient: The Pearson correlation coefficient, often denoted as r, measures the linear relationship between two sets of data. For two lists, X and Y, with n data points, the formula for Pearson correlation is:

$$r = \frac{\sum (X_i - \bar{X})(Y_i - \bar{Y})}{\sqrt{\sum (X_i - \bar{X})^2 \sum (Y_i - \bar{Y})^2}}$$

Where:

- X_i and Y_i are individual data points in lists X and Y.
- $\overline{X}$ and $\overline{Y}$ are the means of lists X and Y, respectively.

Let's illustrate this with an example:

Suppose we have two lists:

X = [2, 4, 6]
Y = [1, 3 ,5]

- Calculate the means :
  - $\overline{X} = \frac{2+4+6}{3} = 4$
  - $\overline{Y} = \frac{1+3+5}{3} = 3$
- Calculate the numerator:

$$\sum_{i=1}^{3} (X_i - \bar{X})(Y_i - \bar{Y})$$

$$= (X_1 - \bar{X})(Y_1 - \bar{Y}) + (X_2 - \bar{X})(Y_2 - \bar{Y}) + (X_3 - \bar{X})(Y_3 - \bar{Y})$$

$$= (2 - 4) \times (1 - 3) + (4 - 4) \times (3 - 3) + (6 - 4) \times (5 - 3) = 8$$

- Calculate the denominators:

$$\sqrt{\sum_{i=1}^{3}(X_i-\bar{X})^2 \sum_{j=1}^{3}(Y_i-\bar{Y})^2}$$

$$=\sqrt{[(X_1-\bar{X})^2+(X_2-\bar{X})^2+(X_3-\bar{X})^2]\times[(Y_1-\bar{Y})^2+(Y_2-\bar{Y})^2+(Y_3-\bar{Y})^2]}$$

$$=\sqrt{[(2-4)^2+(4-4)^2+(6-4)^2]\times[(1-3)^2+(3-3)^2+(5-3)^2]}$$

$$=\sqrt{64}=8$$

- Calculate the Pearson correlation co-efficient
    - $\frac{8}{8}=1.0$

Requirements:

- Debug the code so that it provides the correct result
- Note: To ensure the autograder functions properly,
    - **Do not change the name of the functions.**
    - **Do not change the order of the arguments.**
- Optional:

Submission:

- Go to eClass -> our course -> Week 11 -> Practice -> Lab -> Lab 11 Submission
- Copy your code to Task 2
- You can resubmit your code as many times as you need, but you need to wait for 5 minutes before submission.

Rubric:

- You will not receive any marks if you fail to submit before the deadline.
- You will not receive any marks if your code violates the above requirements.
- Otherwise,
    - You will receive 5 points if your script correctly provides a solution for the general case, e.g., find_highest_correlation([1, 2, 3], [[4, 5, 6], [1, 2, 3], [7, 8, 9]]), ans: ([4, 5, 6], 1.0)
    - You will receive 5 points if your script correctly provides a solution for condition – list include negative numbers, e.g., find_highest_correlation([1, 3, 6, 7, 8], [[2, 6, 10, 6, 1], [-3, 7, 10, 12, 23], [7, 3, 1, -3, -6]]), ans: ([-3, 7, 10, 12, 23], 0.9248)

- You will receive 5 points if your script correctly provides a solution for contract violation – sublist has different length, e.g., find_highest_correlation([1, 2, 3], [[4, 5], [1, 2, 3], [7, 8, 9]]), ans: AssertionError
- You will receive 5 points if your script correctly provides a solution for corner case – division by zero, e.g., find_highest_correlation([1, 2, 3], [[4, 5, 6], [4, 4, 4], [7, 8, 9]]), ans: AssertionError
- You will receive 5 points for each hidden case (2 hidden cases in total)
    - We will not check for preconditions in these two hidden cases

Task 3: Implementation (20 pts)

In the previous question, we restrict the board to be 3 x 3, and accept a board that is n x n where n can be any integers larger than 2. In the previous task, you may hard code some functions and the game will still work. However, you need to make your code work for a more general case.

Here are the functions in the starter code in lab11_task4.py:

1. **Game Initialization:**
   - Initialize an n x n game board as a nested list, where each cell is initially filled with a space (" ").
   - def initialize_board(n) -> List[List[str]]:
2. **Printing the Board (Provided):**
   - A function called print_board to display the current state of the Tic-Tac-Toe game board. Print the board with row and column indices.
3. **Dropping Pieces:**
   - Implement a function called drop_piece that allows a player to drop their piece ('X' or 'O') into a specified cell on the board. Ensure that a player cannot overwrite a cell that is already occupied.
   - def drop_piece(board: List[List[str]], row: int, col: int, player: str) -> bool:
4. **Checking for a Winner:**
   - Implement a function called is_winner to check if a player has won the game. Check for winning conditions in rows, columns, and diagonals. To win a game, the player should occupy an entire row, or an entire column, or the diagonal from top right to bottom left, or the diagonal from top left to bottom right.
   - def is_winner(board: List[List[str]], player: str) -> bool:
5. **Checking for a Tie:**
   - Implement a function called is_board_full to check if the game board is full, indicating a tie.
   - def is_board_full(board: List[List[str]]) -> bool:
6. **Playing the Game (Provided):**
   - A function named play_tic_tac_toe that orchestrates the game. Allow two players to take turns entering their moves (row and column indices). Continuously print the board after each move. This is the function you call to play the game
   - def play_tic_tac_toe() -> None:

Requirements:

- Use clear and concise variable names.
- Ensure that your functions handle user input appropriately and provide feedback on the game's progress.
- Please note that there are fewer starter code then the previous one. Try to come up with test cases yourself so that you can debug each function.

Submission:

- Copy your code to the designated location on Prairielearn.
- You can resubmit your code as needed, but there's a 5-minute wait before each submission.
- Note: To ensure the autograder functions properly,
    - **Do not change the name of the functions.**
    - **Do not change the order of the arguments.**

Rubric:

- You will not receive any marks if you fail to submit before the deadline.
- You will not receive any marks if your code violates the above requirements.
- Function description, signature and test cases are provided as a guide for you to implement the function (It is also an exercise of how to read documentation!). So we will not test for those.
- Otherwise,
    1. You will receive 3 pts for correctly implement the initialize_board function to initialize a board of any size
    2. You will receive 2 pts for correctly implement the drop_piece function given a random board where the spot to place already occupied by the opponent
    3. You will receive 2 pts for correctly implement the drop_piece function given a random board where the spot to place already occupied by oneself
    4. You will receive 2 pts for correctly implement the drop_piece function given a random board where the spot to place that is not occupied
    5. You will receive 1 pts for correctly implement the is_winner function given a random board with a column occupied by the player
    6. You will receive 1 pts for correctly implement the is_winner function given a random board with a column occupied by the opponent
    7. You will receive 1 pts for correctly implement the is_winner function given a random board with a row occupied by the player
    8. You will receive 1 pts for correctly implement the is_winner function given a random board with a row occupied by the opponent
    9. You will receive 1 pts for correctly implement the is_winner function given a random board with a diagonal from top right to bottom left occupied by the player
    10. You will receive 1 pts for correctly implement the is_winner function given a random board with a diagonal from top left to bottom right occupied by the player
    11. You will receive 2.5 pts for correctly implement the is_board_full function given a random board that is not full
    12. You will receive 2.5 pts for correctly implement the is_board_full function given a random board that is full