CS521 - Homework 6 - Alexander Bean

1. Matrix Sum

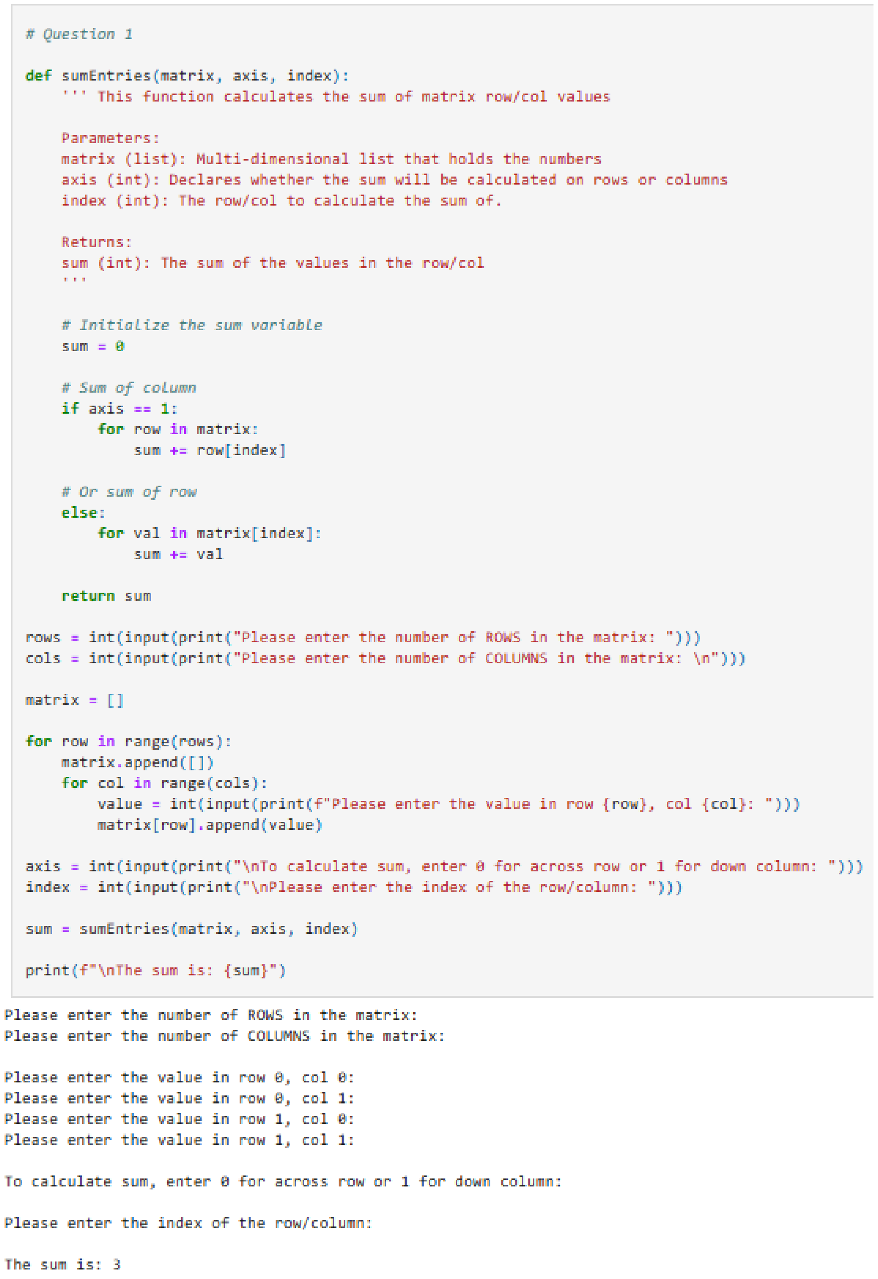
I included a docstring to abide by programming best practices and to explain the function.

In the sumEntries function, I have an if-else statement to determine what calculations to conduct. If it's the sum of a column, I will need to go row by row and add the value in *n* index of each row.

If it's the sum of a row, I need to identify the row in the matrix and add up each value in that row. The sum variable is returned once the sum is calculated and the loops are fully iterated through.

The program begins by taking user inputs for the rows and columns. I then initialize the *matrix* variable with an empty string following the notes.

I then created a nested for loop based on the range of the value entered for rows and columns. For each row, an empty list is appended to the matrix variable and initializes the row for the next loop values to be inserted into.



```python
# Question 1

def sumEntries(matrix, axis, index):
    ''' This function calculates the sum of matrix row/col values

    Parameters:
    matrix (list): Multi-dimensional list that holds the numbers
    axis (int): Declares whether the sum will be calculated on rows or columns
    index (int): The row/col to calculate the sum of.

    Returns:
    sum (int): The sum of the values in the row/col
    '''

    # Initialize the sum variable
    sum = 0

    # Sum of column
    if axis == 1:
        for row in matrix:
            sum += row[index]

    # Or sum of row
    else:
        for val in matrix[index]:
            sum += val

    return sum

rows = int(input(print("Please enter the number of ROWS in the matrix: ")))
cols = int(input(print("Please enter the number of COLUMNS in the matrix: \n")))

matrix = []

for row in range(rows):
    matrix.append([])
    for col in range(cols):
        value = int(input(print(f"Please enter the value in row {row}, col {col}: ")))
        matrix[row].append(value)

axis = int(input(print("\nTo calculate sum, enter 0 for across row or 1 for down column: ")))
index = int(input(print("\nPlease enter the index of the row/column: ")))

sum = sumEntries(matrix, axis, index)

print(f"\nThe sum is: {sum}")
```

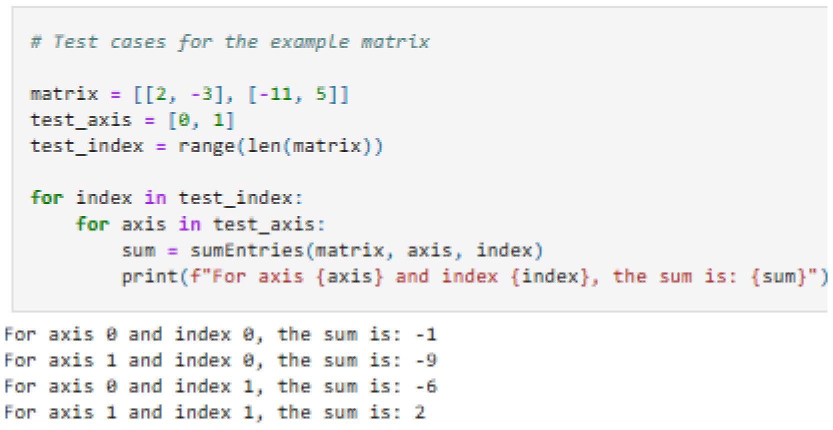
```
Please enter the number of ROWS in the matrix:
Please enter the number of COLUMNS in the matrix:

Please enter the value in row 0, col 0:
Please enter the value in row 0, col 1:
Please enter the value in row 1, col 0:
Please enter the value in row 1, col 1:

To calculate sum, enter 0 for across row or 1 for down column:

Please enter the index of the row/column:

The sum is: 3
```

The nested loop then requests the value in this row and column from the user and appends it to the corresponding row in the matrix. The loop will iterate through the entire row before moving on to the next, appending a new empty list, and asking for the next row's values.

By structuring the matrix insertion like this, the program can dynamically create a matrix with any number of rows or columns. There is no need to manually create the lists and sublists of the matrix like in previous assignments. This program can do that just based on what the user enters.

Once all the row and column loops have been iterated through, the user is prompted to enter either a 0 or 1. A 0 states to calculate across a row, and a 1 states to calculate down a column. The final user input is to enter the index number of the row/column to calculate on. I call the sumEntries function and enter the matrix, axis, and index values as arguments. I assign the returned value to the *sum* variable and print it to the user.

In a separate block, I hard-coded the example matrix, axis list, and index list to validate the correctness of my calculations. The examples provided are -1 for across a row and -9 down a column, so I included them in this test to validate.



```python
# Test cases for the example matrix

matrix = [[2, -3], [-11, 5]]
test_axis = [0, 1]
test_index = range(len(matrix))

for index in test_index:
    for axis in test_axis:
        sum = sumEntries(matrix, axis, index)
        print(f"For axis {axis} and index {index}, the sum is: {sum}")
```

```
For axis 0 and index 0, the sum is: -1
For axis 1 and index 0, the sum is: -9
For axis 0 and index 1, the sum is: -6
For axis 1 and index 1, the sum is: 2
```

2. Which Points are Nearest?

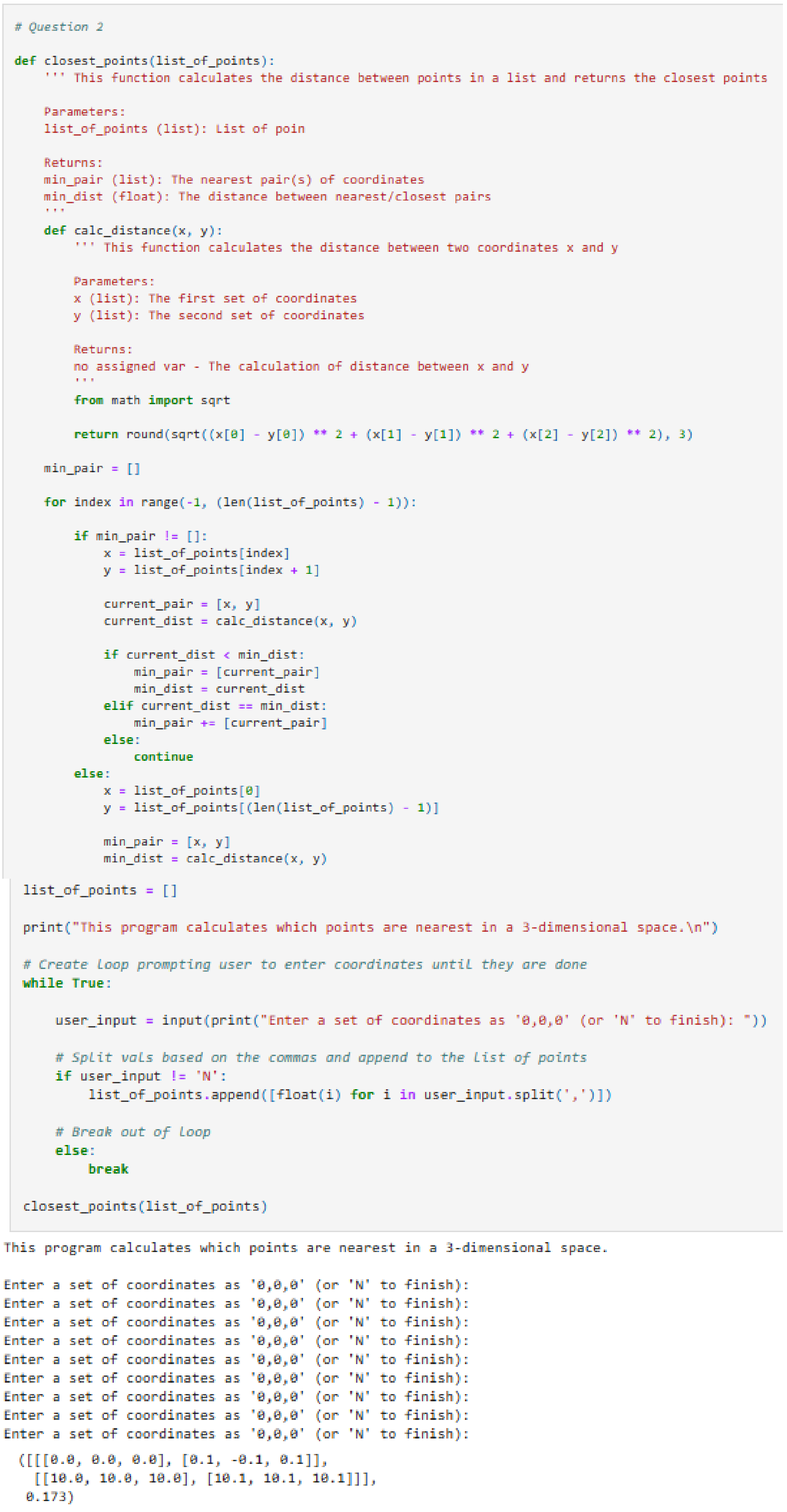
I included a docstring to abide by programming best practices and to explain the function.

I started with importing the sqrt function from the math module, since I need to use this to do the calculation. I initialized a *min_pair* variable with an empty list as a way to track the nearest pairs.

The function runs a for loop with a few nested if-else statements. From the argument entered, the for loop iterates through every set of coordinates in the list. I assign x and y values for the function to compare each neighbor.

If the current pair distance is less than the minimum distance, the minimum pair is assigned the current pair value and the minimum distance is assigned the current distance value. In this statement, I identify the scenario where there is more than 1 nearest pair.

In the elif block, I identify the scenario where there is more than 1 nearest pair. The elif condition is true if the current pair's distance is equal to the minimum distance, and the pair is added to the *min_pair* list if true. There is no need to reassign the *min_dist* value because the distance is the same.



```
# Question 2

def closest_points(list_of_points):
    ''' This function calculates the distance between points in a list and returns the closest points

    Parameters:
    list_of_points (list): List of poin

    Returns:
    min_pair (list): The nearest pair(s) of coordinates
    min_dist (float): The distance between nearest/closest pairs
    '''
    def calc_distance(x, y):
        ''' This function calculates the distance between two coordinates x and y

        Parameters:
        x (list): The first set of coordinates
        y (list): The second set of coordinates

        Returns:
        no assigned var - The calculation of distance between x and y
        '''
        from math import sqrt

        return round(sqrt((x[0] - y[0]) ** 2 + (x[1] - y[1]) ** 2 + (x[2] - y[2]) ** 2), 3)

    min_pair = []

    for index in range(-1, (len(list_of_points) - 1)):

        if min_pair != []:
            x = list_of_points[index]
            y = list_of_points[index + 1]

            current_pair = [x, y]
            current_dist = calc_distance(x, y)

            if current_dist < min_dist:
                min_pair = [current_pair]
                min_dist = current_dist
            elif current_dist == min_dist:
                min_pair += [current_pair]
            else:
                continue
        else:
            x = list_of_points[0]
            y = list_of_points[(len(list_of_points) - 1)]

            min_pair = [x, y]
            min_dist = calc_distance(x, y)
```

```
list_of_points = []

print("This program calculates which points are nearest in a 3-dimensional space.\n")

# Create loop prompting user to enter coordinates until they are done
while True:

    user_input = input(print("Enter a set of coordinates as '0,0,0' (or 'N' to finish): "))

    # Split vals based on the commas and append to the list of points
    if user_input != 'N':
        list_of_points.append([float(i) for i in user_input.split(',')])

    # Break out of loop
    else:
        break

closest_points(list_of_points)
```

```
This program calculates which points are nearest in a 3-dimensional space.

Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):
Enter a set of coordinates as '0,0,0' (or 'N' to finish):

  ([[[0.0, 0.0, 0.0], [0.1, -0.1, 0.1]],
    [[10.0, 10.0, 10.0], [10.1, 10.1, 10.1]]],
   0.173)
```

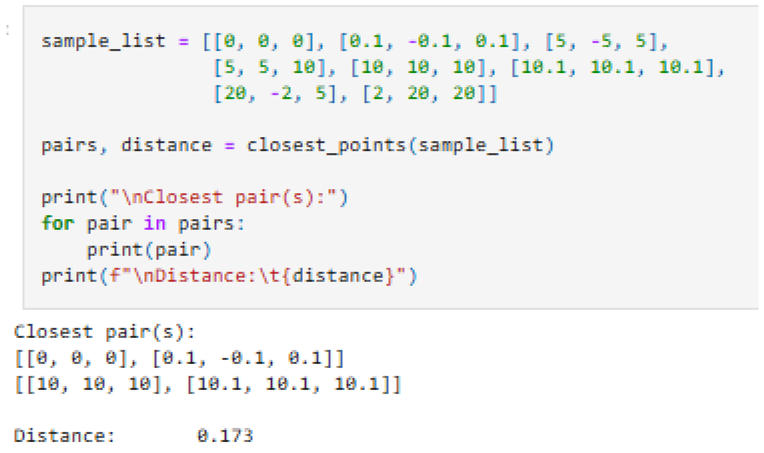
Originally, I had the else statement to set *min_pair* to the current pair and *min_dist* to the current distance if it was the first iteration. However, I realized that there was one distance not calculated by my program: the last and first coordinates. So, I changed the else statement to instead calculate the distance between the first and last sets of coordinates as the first iteration of the loop. This way, the program can correctly be initialized and the one missing distance can be calculated.

In addition to this, I created a sub-function in the closest_points() function that houses the sqrt() function calculation. Since I used it more than once in the program, it was a good practice to create a sub-function like we have discussed in class.

Now that the functions have been defined, the next step is the actual program. I create an empty list as instructed in the description, and this list will store all the sets of coordinates. The program runs a while loop to take in as many sets of coordinates as the user would like.

I used list comprehension to achieve the results of converting the values from strings to floats and grouping them as a list. The list comprehension takes the string input and splits each item on the comma. Now, we have 3 distinct strings that can be converted into float values for the calculations. Then, once converted, they are grouped as a list and appended to the *list_of_points* list.

Any inputs are appended to the list_of_points variable as a float. When the user is done entering coordinates, they will enter “N” and the loop will break. Then, the list of entered coordinates will be used as the input into the closest_points() function



```python
sample_list = [[0, 0, 0], [0.1, -0.1, 0.1], [5, -5, 5],
               [5, 5, 10], [10, 10, 10], [10.1, 10.1, 10.1],
               [20, -2, 5], [2, 20, 20]]

pairs, distance = closest_points(sample_list)

print("\nClosest pair(s):")
for pair in pairs:
    print(pair)
print(f"\nDistance:\t{distance}")
```

```
Closest pair(s):
[[0, 0, 0], [0.1, -0.1, 0.1]]
[[10, 10, 10], [10.1, 10.1, 10.1]]

Distance:       0.173
```

3. Bank Loans

```python
# Question 3

def unsafeBanks(bank_list, safe_banks):
    ''' This function takes a total list of banks and the current list of safe banks
        and identifies, of that list, what banks are unsafe

    Parameters:
    bank_list (list): The total list of banks
    safe_banks (list): The list of currently safe banks

    Returns:
    unsafe (str): The IDs of banks that are unsafe
    '''

    limit = banks[0][1]
    risks_found = 0

    for bankID, bank in enumerate(banks[1:]):
        assets = bank[0]

        for loan in bank[2:]:   # Add up all assets
            assets += loan[1]

        if assets < limit and bankID in safe_banks: # banks that are below limit and not marked as "unsafe" yet
            safe_banks.remove(bankID)               # Removes from safe banks list
            unsafe = str(bankID)

            # Goes through all banks and sets any loans with bankID to value of '0'
            for i, subbank in enumerate(banks[1:]):
                for j, subloan in enumerate(subbank[2:]):
                    if subloan[0] == bankID:
                        banks[i + 1][j + 2] = (bankID, 0)
                        risks_found += 1

    # Base case
    if risks_found == 0:
        return ""

    # Recursive case to check list of banks again for risk
    else:
        unsafe += " " + str(unsafeBanks(banks, safe_banks))
        return unsafe
```

I included a docstring to abide by programming best practices and to explain the function.

The function has two arguments, *bank_list* and *safe_banks*. The first variable is the original list of banks, and the second is the list of banks currently deemed safe. I added the second variable for future use in recursion. There were multiple variations I attempted without the *safe_banks* variable but this was the only solution that worked successfully.

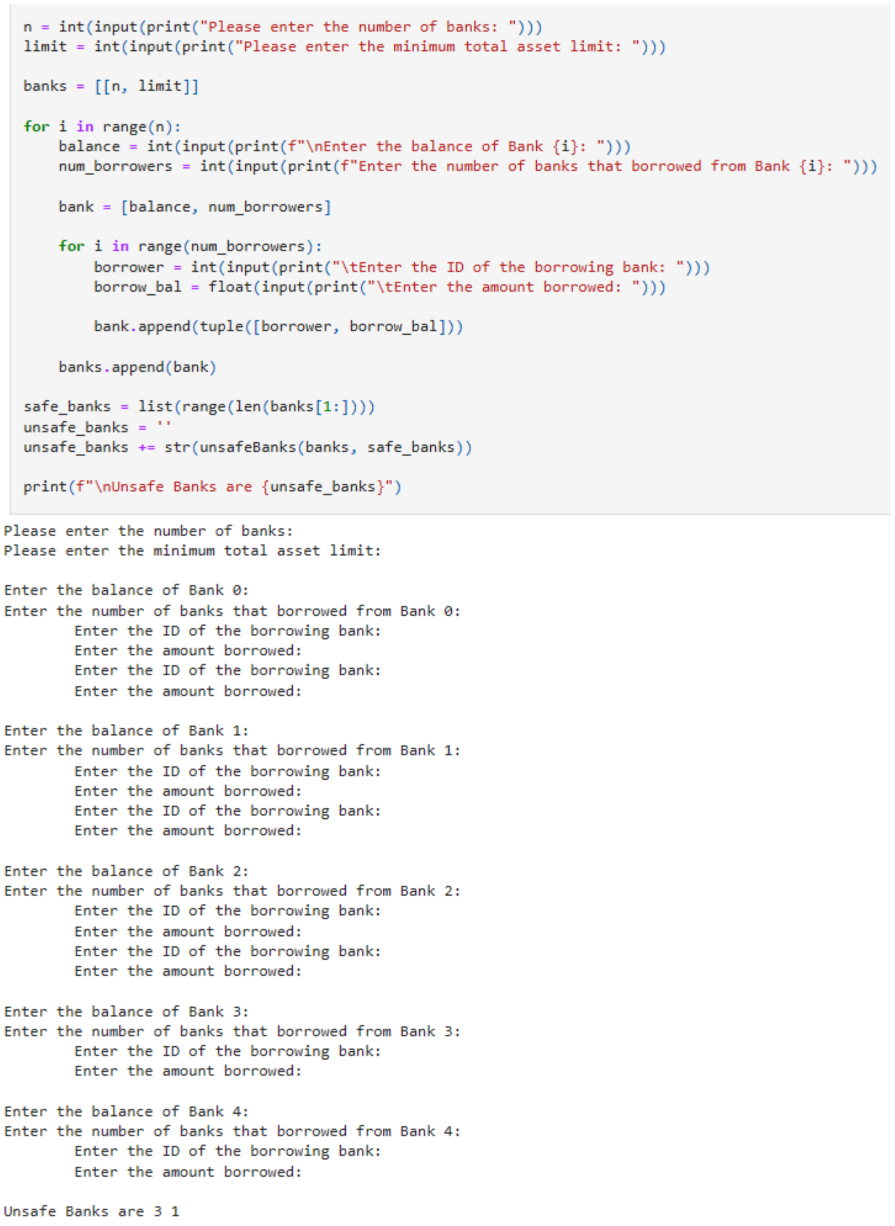
The function iterates through each bank in the list of banks, starting at the second index since the first sublist contains the number of banks and asset limit value. I used enumerate to identify the bank ID (index) of each bank to be used in the following loops.

If a risk is identified, the bank is removed from the safe bank list and added to the *unsafe* variable. Then, a complex for loop iterates through every bank in the main list to find any occurrences of the bank ID in their borrowers. If found, the borrowed balance is reduced to 0.

Since I created this function with recursion in mind, I needed to add a base and recursive case. I used the *risks_found* variable for recursion control. This variable is set to 0 at the start of the function, and is counted up if any risks are identified If there were no risks found, then the recursive loop ends. If 1 or more risks were identified, the function is called again with the bank list and an updated list of safe banks (defined in the function's first if statement).

In the nested if statement, I was encountering a lot of issues with the tuple not being replaced. I realized that my indices were not correct when replacing the value. Since I was starting the list indexes at the 1st and 2nd indices respectively, the enumerate count was 1 and 2 off from the expected total. So, to remediate this, I added +1 and +2 to the indices when reassigning the value and was successful.

In the actual program, I had nested for loops to go through however many banks were entered by the user and, for each bank, however many borrowers were identified. The first two indices of the bank were set to the *balance* and *num_borrwers* variables. Each borrower value was appended as a tuple to the specific bank sublist, and this produced the expected result when the test values were entered.



```python
n = int(input(print("Please enter the number of banks: ")))
limit = int(input(print("Please enter the minimum total asset limit: ")))

banks = [[n, limit]]

for i in range(n):
    balance = int(input(print(f"\nEnter the balance of Bank {i}: ")))
    num_borrowers = int(input(print(f"Enter the number of banks that borrowed from Bank {i}: ")))

    bank = [balance, num_borrowers]

    for i in range(num_borrowers):
        borrower = int(input(print("\tEnter the ID of the borrowing bank: ")))
        borrow_bal = float(input(print("\tEnter the amount borrowed: ")))

        bank.append(tuple([borrower, borrow_bal]))

    banks.append(bank)

safe_banks = list(range(len(banks[1:])))
unsafe_banks = ''
unsafe_banks += str(unsafeBanks(banks, safe_banks))

print(f"\nUnsafe Banks are {unsafe_banks}")
```

```
Please enter the number of banks:
Please enter the minimum total asset limit:

Enter the balance of Bank 0:
Enter the number of banks that borrowed from Bank 0:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:

Enter the balance of Bank 1:
Enter the number of banks that borrowed from Bank 1:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:

Enter the balance of Bank 2:
Enter the number of banks that borrowed from Bank 2:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:

Enter the balance of Bank 3:
Enter the number of banks that borrowed from Bank 3:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:

Enter the balance of Bank 4:
Enter the number of banks that borrowed from Bank 4:
        Enter the ID of the borrowing bank:
        Enter the amount borrowed:

Unsafe Banks are 3 1
```

At the bottom, I initialized the safe_banks and unsafe_banks variables to track what banks for the function to sort through and save as "unsafe".