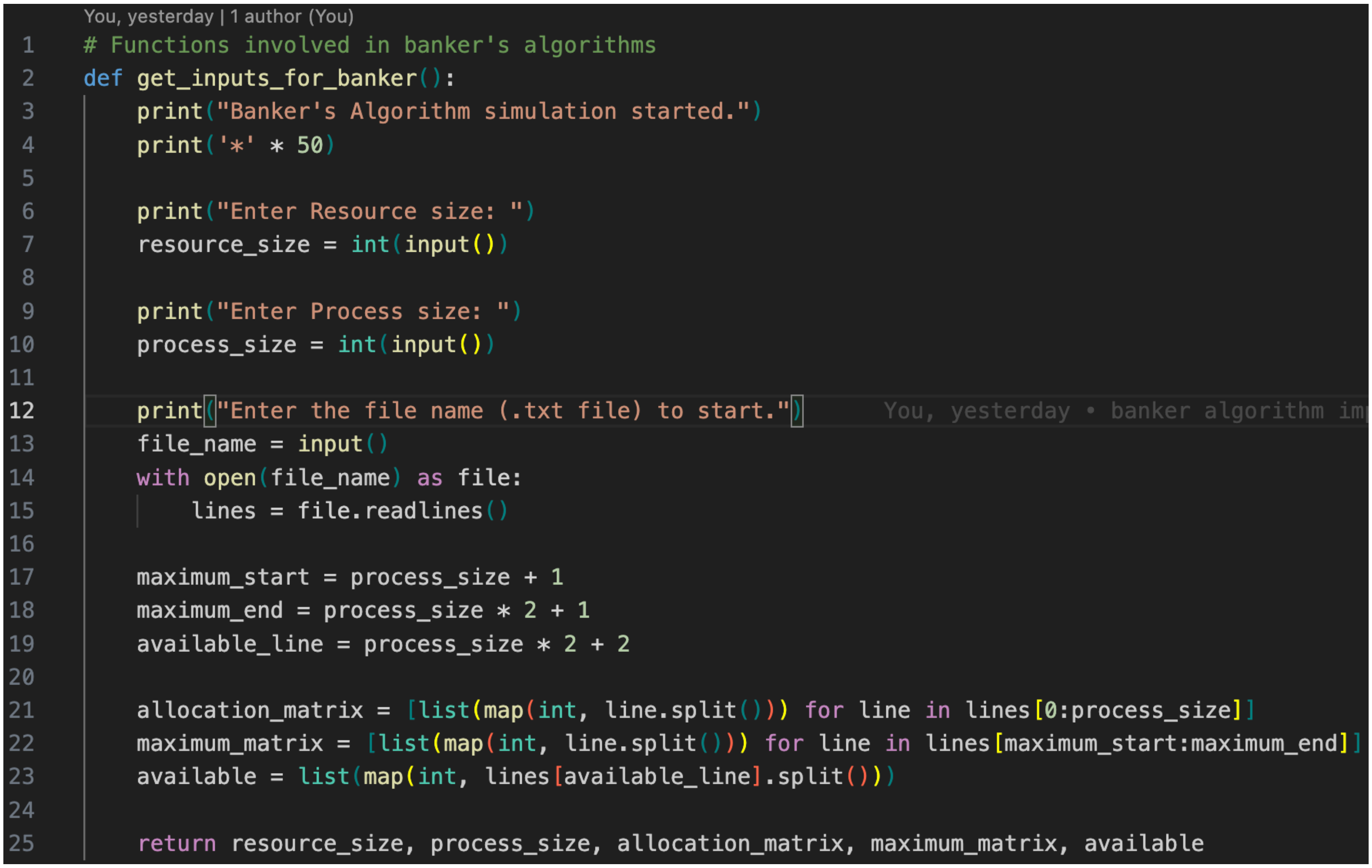
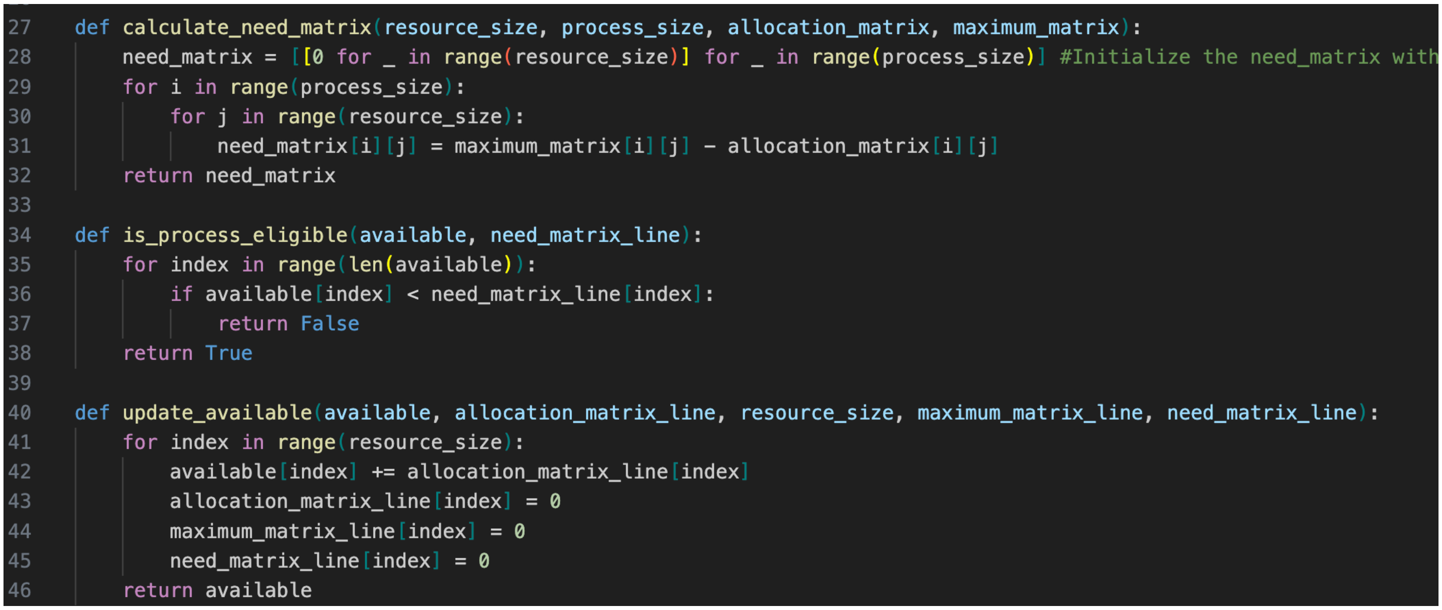
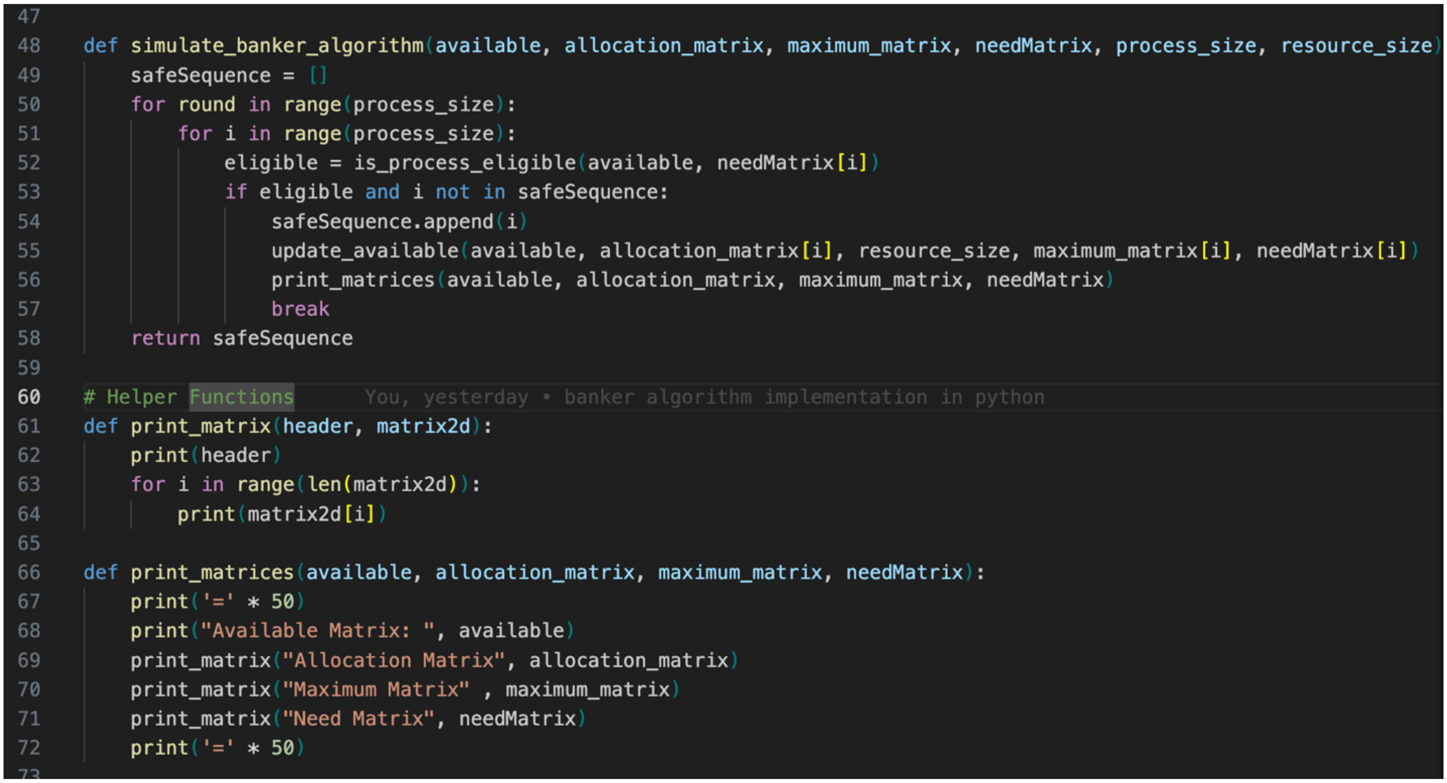

You, yesterday | 1 author (You)
You, yesterday • banker algorithm im

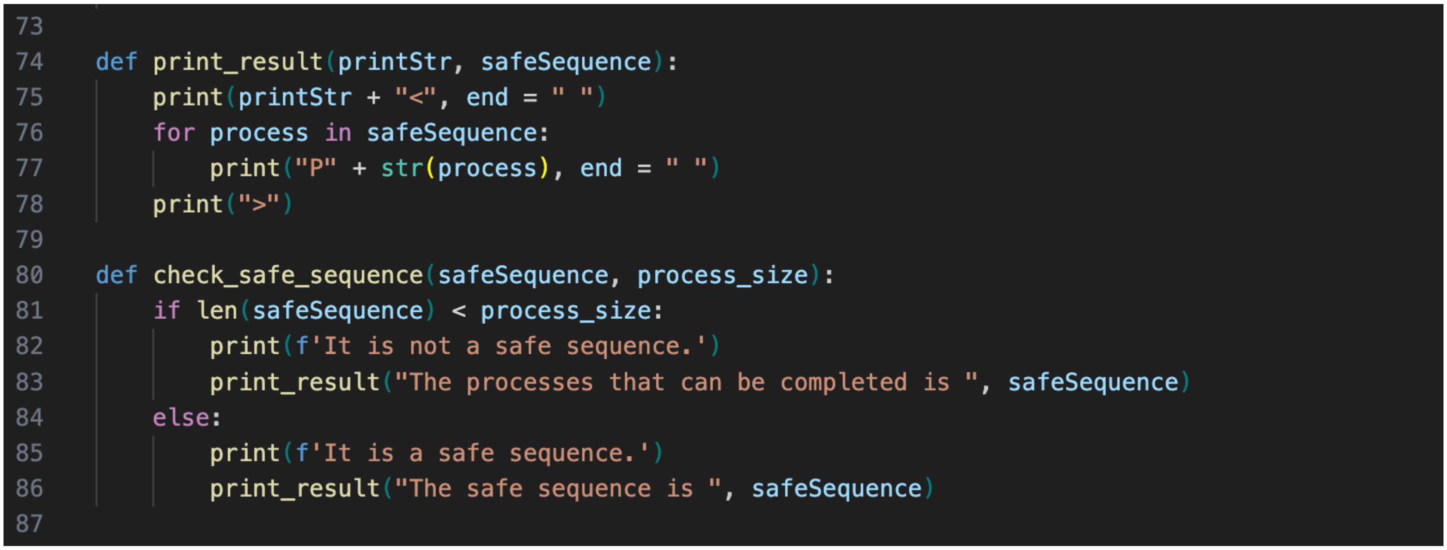
```python
# Functions involved in banker's algorithms
def get_inputs_for_banker():
    print("Banker's Algorithm simulation started.")
    print('*' * 50)

    print("Enter Resource size: ")
    resource_size = int(input())

    print("Enter Process size: ")
    process_size = int(input())

    print("Enter the file name (.txt file) to start.")
    file_name = input()
    with open(file_name) as file:
        lines = file.readlines()

    maximum_start = process_size + 1
    maximum_end = process_size * 2 + 1
    available_line = process_size * 2 + 2

    allocation_matrix = [list(map(int, line.split())) for line in lines[0:process_size]]
    maximum_matrix = [list(map(int, line.split())) for line in lines[maximum_start:maximum_end]]
    available = list(map(int, lines[available_line].split()))

    return resource_size, process_size, allocation_matrix, maximum_matrix, available
```



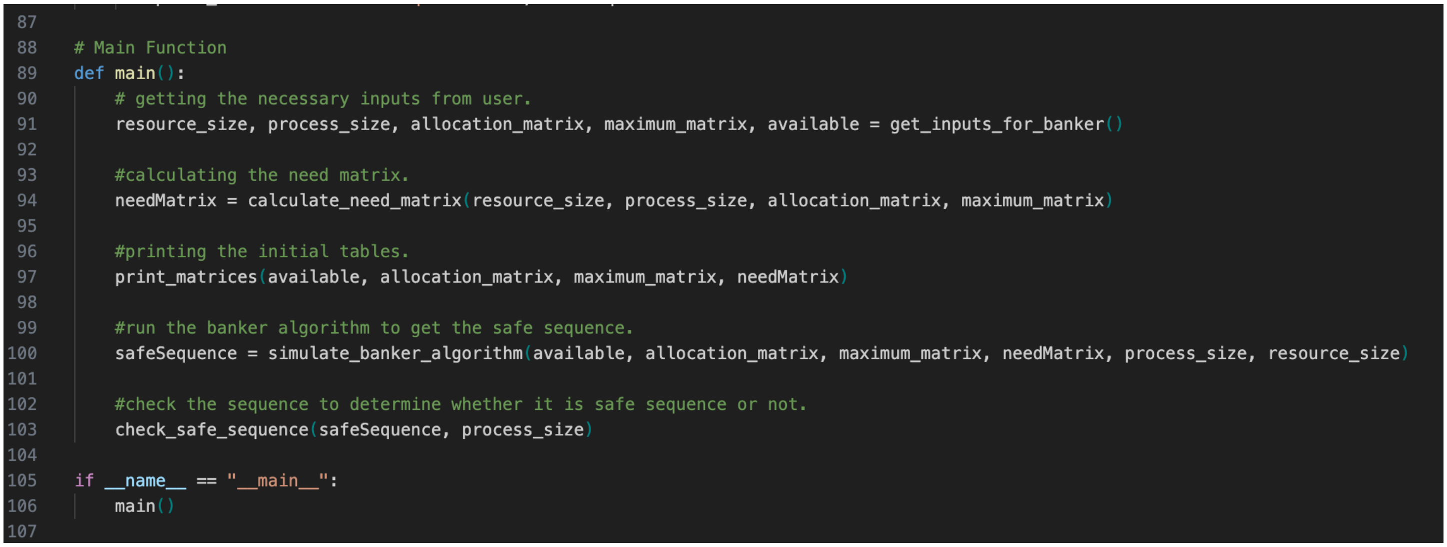
```python
def calculate_need_matrix(resource_size, process_size, allocation_matrix, maximum_matrix):
    need_matrix = [[0 for _ in range(resource_size)] for _ in range(process_size)] #Initialize the need_matrix with
    for i in range(process_size):
        for j in range(resource_size):
            need_matrix[i][j] = maximum_matrix[i][j] - allocation_matrix[i][j]
    return need_matrix

def is_process_eligible(available, need_matrix_line):
    for index in range(len(available)):
        if available[index] < need_matrix_line[index]:
            return False
    return True

def update_available(available, allocation_matrix_line, resource_size, maximum_matrix_line, need_matrix_line):
    for index in range(resource_size):
        available[index] += allocation_matrix_line[index]
        allocation_matrix_line[index] = 0
        maximum_matrix_line[index] = 0
        need_matrix_line[index] = 0
    return available
```



```python

def simulate_banker_algorithm(available, allocation_matrix, maximum_matrix, needMatrix, process_size, resource_size)
    safeSequence = []
    for round in range(process_size):
        for i in range(process_size):
            eligible = is_process_eligible(available, needMatrix[i])
            if eligible and i not in safeSequence:
                safeSequence.append(i)
                update_available(available, allocation_matrix[i], resource_size, maximum_matrix[i], needMatrix[i])
                print_matrices(available, allocation_matrix, maximum_matrix, needMatrix)
                break
    return safeSequence

# Helper Functions       You, yesterday • banker algorithm implementation in python
def print_matrix(header, matrix2d):
    print(header)
    for i in range(len(matrix2d)):
        print(matrix2d[i])

def print_matrices(available, allocation_matrix, maximum_matrix, needMatrix):
    print('=' * 50)
    print("Available Matrix: ", available)
    print_matrix("Allocation Matrix", allocation_matrix)
    print_matrix("Maximum Matrix" , maximum_matrix)
    print_matrix("Need Matrix", needMatrix)
    print('=' * 50)
```



```python

def print_result(printStr, safeSequence):
    print(printStr + "<", end = " ")
    for process in safeSequence:
        print("P" + str(process), end = " ")
    print(">")

def check_safe_sequence(safeSequence, process_size):
    if len(safeSequence) < process_size:
        print(f'It is not a safe sequence.')
        print_result("The processes that can be completed is ", safeSequence)
    else:
        print(f'It is a safe sequence.')
        print_result("The safe sequence is ", safeSequence)

```



```python

# Main Function
def main():
    # getting the necessary inputs from user.
    resource_size, process_size, allocation_matrix, maximum_matrix, available = get_inputs_for_banker()

    #calculating the need matrix.
    needMatrix = calculate_need_matrix(resource_size, process_size, allocation_matrix, maximum_matrix)

    #printing the initial tables.
    print_matrices(available, allocation_matrix, maximum_matrix, needMatrix)

    #run the banker algorithm to get the safe sequence.
    safeSequence = simulate_banker_algorithm(available, allocation_matrix, maximum_matrix, needMatrix, process_size, resource_size)

    #check the sequence to determine whether it is safe sequence or not.
    check_safe_sequence(safeSequence, process_size)

if __name__ == "__main__":
    main()

```