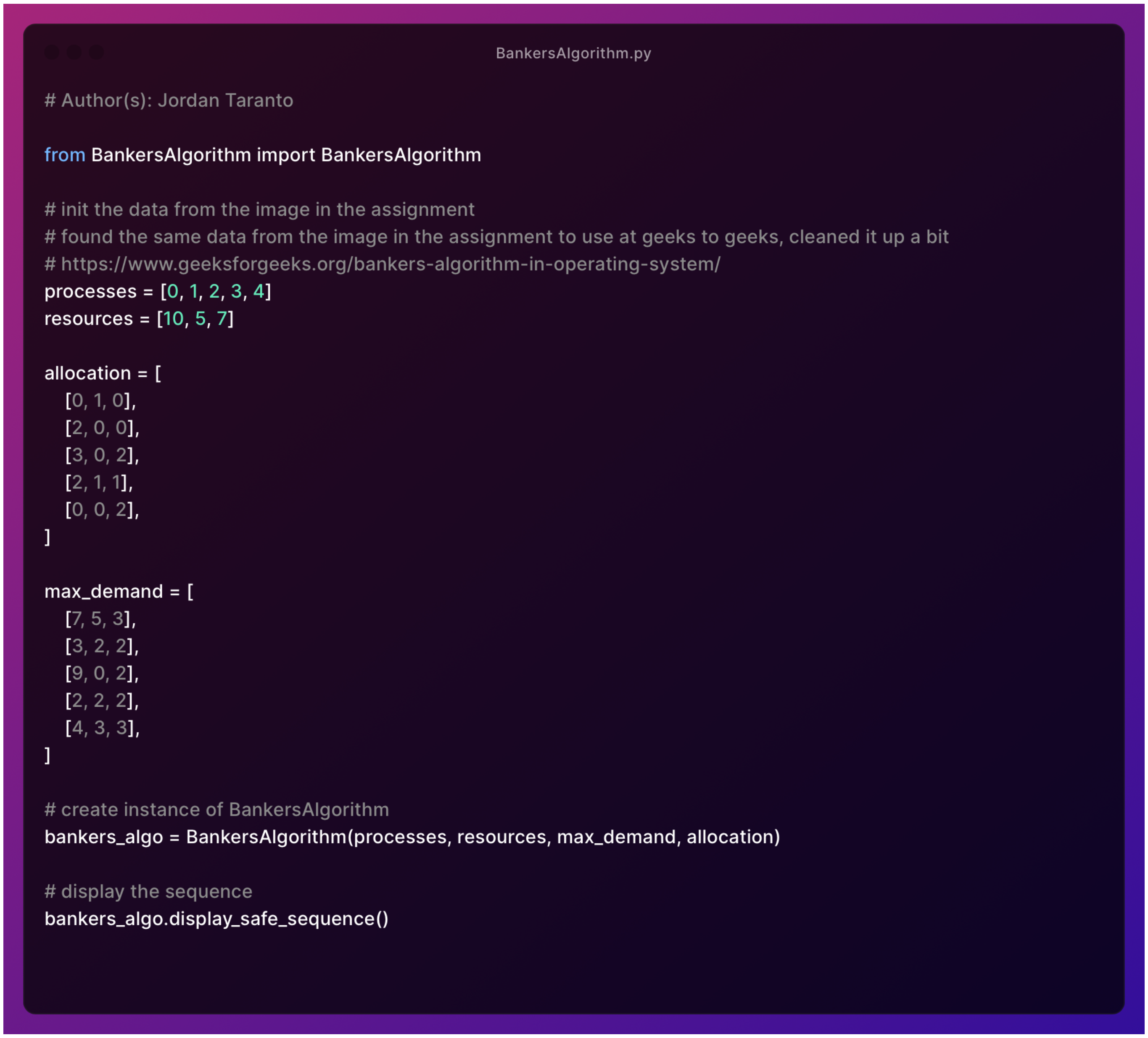
Author: Jordan Taranto
Assignment 2
CS431

Section 1: Code Implementation

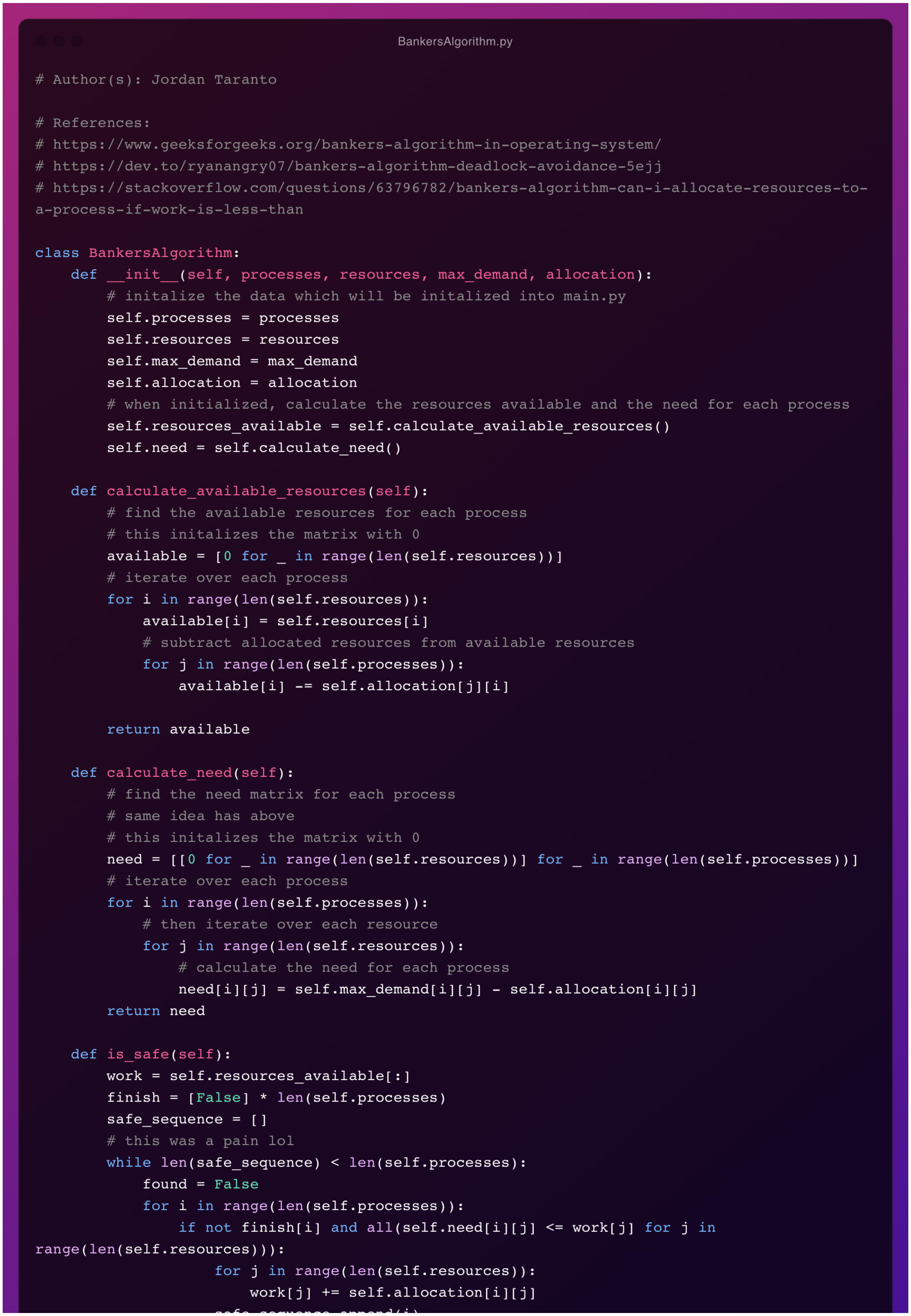
main.py



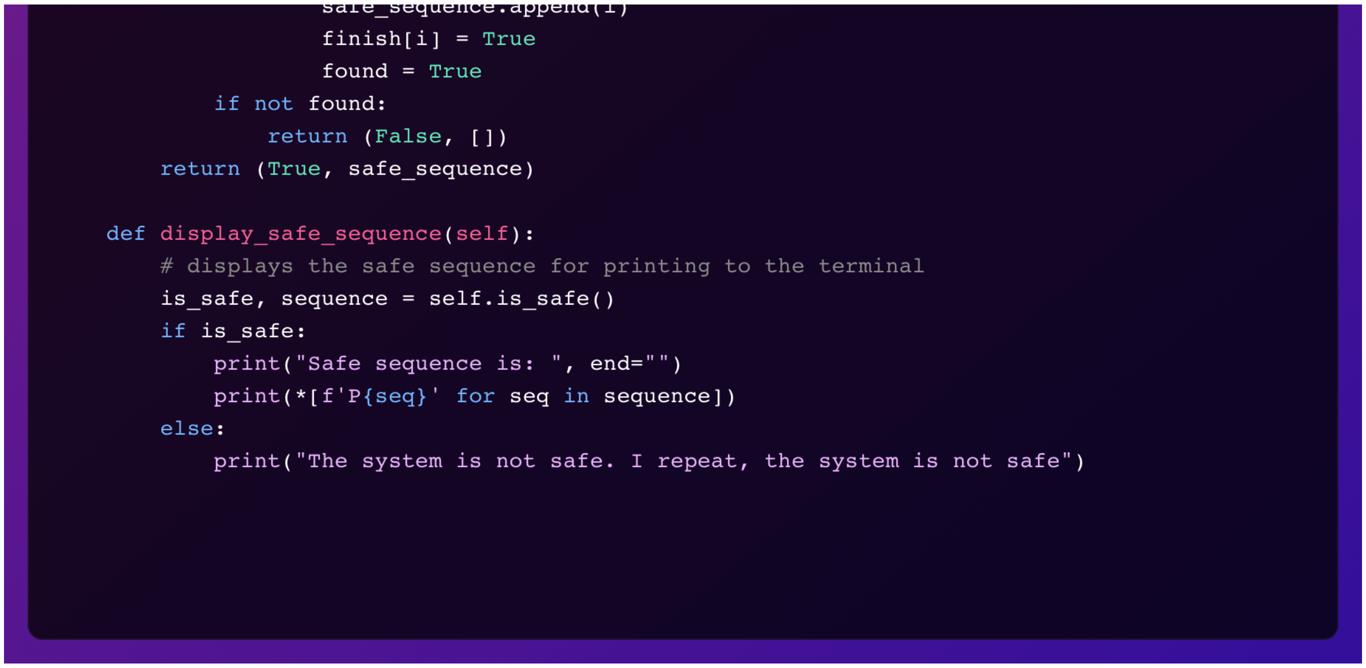
BankersAlgorithm.py

```python
# Author(s): Jordan Taranto

from BankersAlgorithm import BankersAlgorithm

# init the data from the image in the assignment
# found the same data from the image in the assignment to use at geeks to geeks, cleaned it up a bit
# https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system/
processes = [0, 1, 2, 3, 4]
resources = [10, 5, 7]

allocation = [
    [0, 1, 0],
    [2, 0, 0],
    [3, 0, 2],
    [2, 1, 1],
    [0, 0, 2],
]

max_demand = [
    [7, 5, 3],
    [3, 2, 2],
    [9, 0, 2],
    [2, 2, 2],
    [4, 3, 3],
]

# create instance of BankersAlgorithm
bankers_algo = BankersAlgorithm(processes, resources, max_demand, allocation)

# display the sequence
bankers_algo.display_safe_sequence()
```

BankersAlgorithm.py


BankersAlgorithm.py

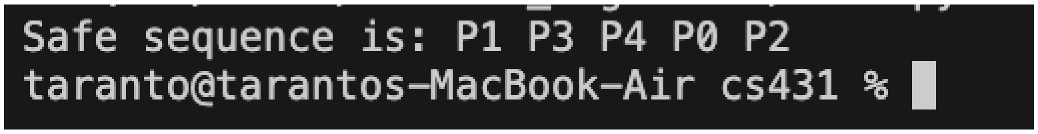
```python
# Author(s): Jordan Taranto

# References:
# https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system/
# https://dev.to/ryanangry07/bankers-algorithm-deadlock-avoidance-5ejj
# https://stackoverflow.com/questions/63796782/bankers-algorithm-can-i-allocate-resources-to-
a-process-if-work-is-less-than

class BankersAlgorithm:
    def __init__(self, processes, resources, max_demand, allocation):
        # initalize the data which will be initalized into main.py
        self.processes = processes
        self.resources = resources
        self.max_demand = max_demand
        self.allocation = allocation
        # when initialized, calculate the resources available and the need for each process
        self.resources_available = self.calculate_available_resources()
        self.need = self.calculate_need()

    def calculate_available_resources(self):
        # find the available resources for each process
        # this initalizes the matrix with 0
        available = [0 for _ in range(len(self.resources))]
        # iterate over each process
        for i in range(len(self.resources)):
            available[i] = self.resources[i]
            # subtract allocated resources from available resources
            for j in range(len(self.processes)):
                available[i] -= self.allocation[j][i]

        return available

    def calculate_need(self):
        # find the need matrix for each process
        # same idea has above
        # this initalizes the matrix with 0
        need = [[0 for _ in range(len(self.resources))] for _ in range(len(self.processes))]
        # iterate over each process
        for i in range(len(self.processes)):
            # then iterate over each resource
            for j in range(len(self.resources)):
                # calculate the need for each process
                need[i][j] = self.max_demand[i][j] - self.allocation[i][j]
        return need

    def is_safe(self):
        work = self.resources_available[:]
        finish = [False] * len(self.processes)
        safe_sequence = []
        # this was a pain lol
        while len(safe_sequence) < len(self.processes):
            found = False
            for i in range(len(self.processes)):
                if not finish[i] and all(self.need[i][j] <= work[j] for j in
range(len(self.resources))):
                    for j in range(len(self.resources)):
                        work[j] += self.allocation[i][j]
```



```
                    safe_sequence.append(i)
                    finish[i] = True
                    found = True
            if not found:
                return (False, [])
        return (True, safe_sequence)

    def display_safe_sequence(self):
        # displays the safe sequence for printing to the terminal
        is_safe, sequence = self.is_safe()
        if is_safe:
            print("Safe sequence is: ", end="")
            print(*[f'P{seq}' for seq in sequence])
        else:
            print("The system is not safe. I repeat, the system is not safe")
```

Section 2: Outputs

Output of sequence



```
Safe sequence is: P1 P3 P4 P0 P2
taranto@tarantos-MacBook-Air cs431 %
```

Section 3: Implementation Details

Thought process:

The first part was converting the data we will be using into the necessary matrixes. Fortunately I found a GeeksForGeeks post with the same data so I copied that over and cleaned it up.

Challenges:

This was much more straightforward then the scheduling algorithm. The first assignment helped get me in the mindset for how to conceptualize operating systems and writing code moving forward. So didn't really have many challenges.

Insights:

Provided a good foundation for avoiding deadlocks. I noticed that it doesn't prioritize processes. So moving forward I think that would be very beneficial, based off of metrics like minimum waiting time, FIFO, etc. just like what we talked about in class with the restaurant example.

References:

- https://ray.so/#code=&title=&padding=16&theme=breeze
- https://www.geeksforgeeks.org/bankers-algorithm-in-operating-system/
- https://dev.to/ryanangry07/bankers-algorithm-deadlock-avoidance-5ejj
- https://stackoverflow.com/questions/63796782/bankers-algorithm-can-i-allocate-resources-to-a-process-if-work-is-less-than