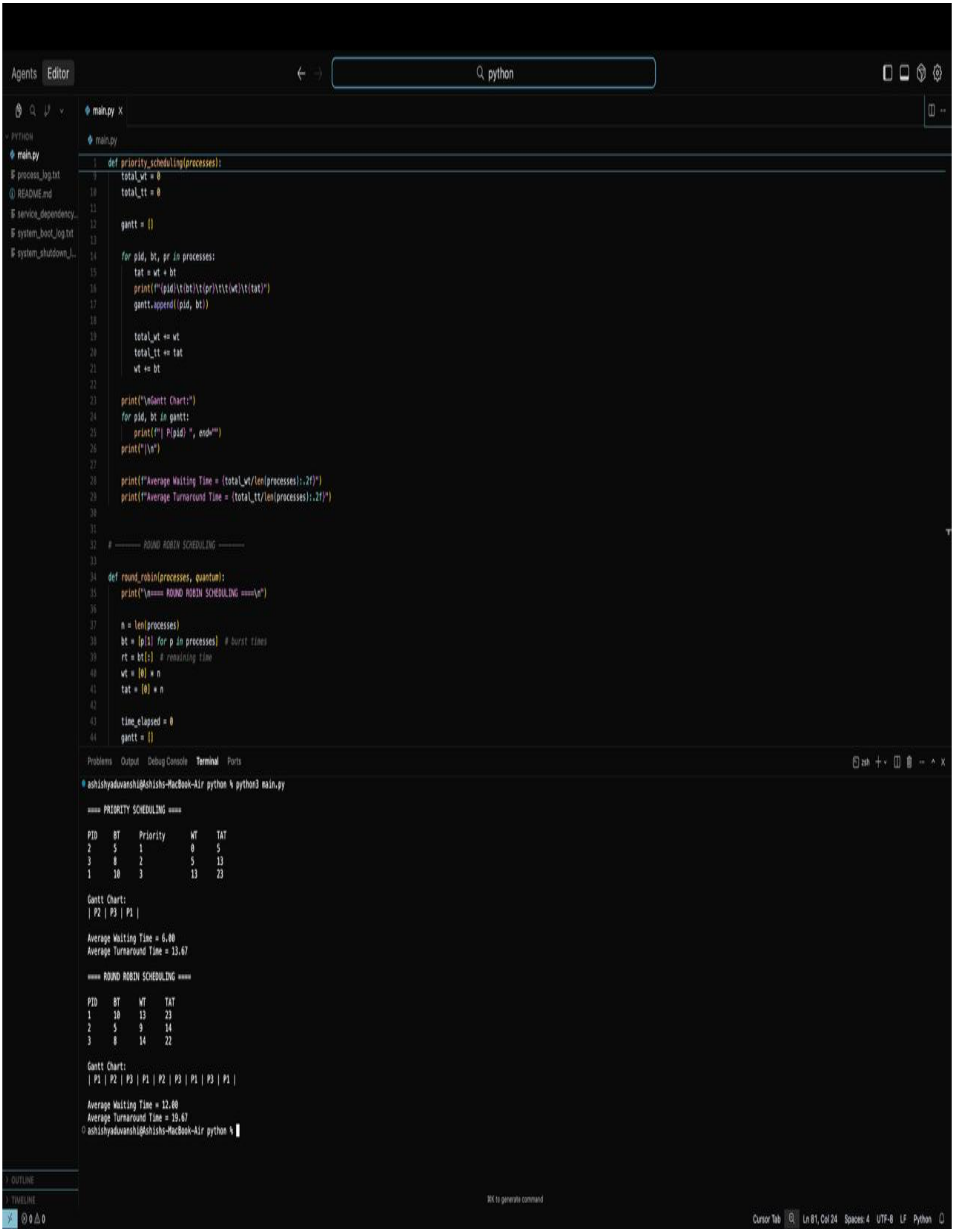
Task 1: CPU Scheduling with Gantt Chart

Write a Python program to simulate Priority and Round Robin scheduling algorithms. Compute average waiting and turnaround times.

```
# Priority Scheduling Simulation
processes = []
n = int(input("Enter number of processes: "))
for i in range(n):
    bt = int(input(f"Enter Burst Time for P{i+1}: "))
    pr = int(input(f"Enter Priority (lower number = higher priority) for P{i+1}: "))
    processes.append((i+1, bt, pr))
processes.sort(key=lambda x: x[2])
wt = 0
total_wt = 0
total_tt = 0
print("\nPriority Scheduling:")
print("PID\tBT\tPriority\tWT\tTAT")
for pid, bt, pr in processes:
    tat = wt + bt
    print(f"{pid}\t{bt}\t{pr}\t\t{wt}\t{tat}")
    total_wt += wt
    total_tt += tat
    wt += bt
print(f"Average Waiting Time: {total_wt / n}")
print(f"Average Turnaround Time: {total_tt / n}")
```



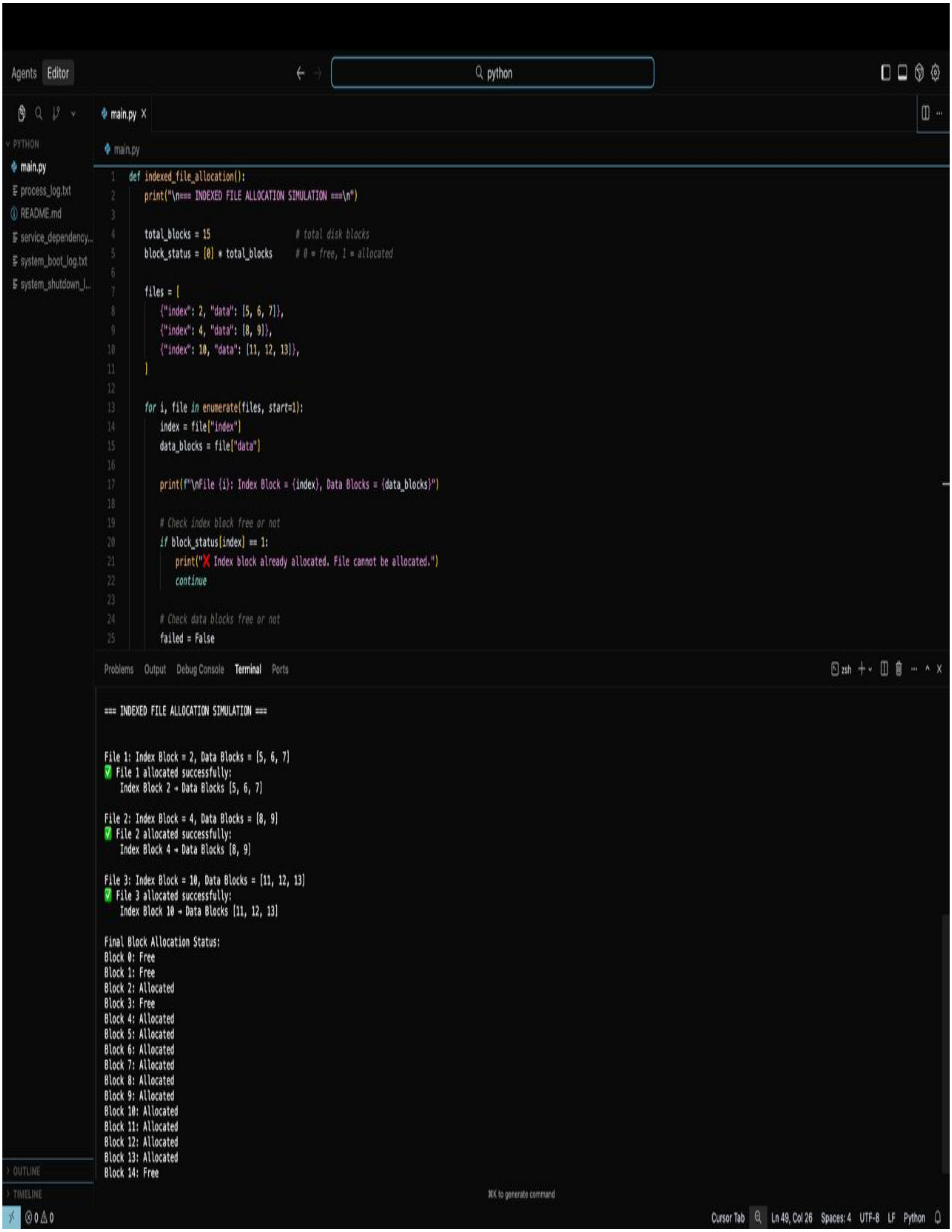
```python
def priority_scheduling(processes):
    total_wt = 0
    total_tt = 0

    gantt = []

    for pid, bt, pr in processes:
        tat = wt + bt
        print(f"{pid}\t{bt}\t{pr}\t\t{wt}\t{tat}")
        gantt.append((pid, bt))

        total_wt += wt
        total_tt += tat
        wt += bt

    print("\nGantt Chart:")
    for pid, bt in gantt:
        print(f"| P{pid} ", end="")
    print("|\n")

    print(f"Average Waiting Time = {total_wt/len(processes):.2f}")
    print(f"Average Turnaround Time = {total_tt/len(processes):.2f}")


# -------- ROUND ROBIN SCHEDULING --------

def round_robin(processes, quantum):
    print("\n==== ROUND ROBIN SCHEDULING ====\n")

    n = len(processes)
    bt = [p[1] for p in processes]  # burst times
    rt = bt[:]  # remaining time
    wt = [0] * n
    tat = [0] * n

    time_elapsed = 0
    gantt = []
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

==== PRIORITY SCHEDULING ====

PID     BT      Priority        WT      TAT
2       5       1               0       5
3       8       2               5       13
1       10      3               13      23

Gantt Chart:
| P2 | P3 | P1 |

Average Waiting Time = 6.00
Average Turnaround Time = 13.67

==== ROUND ROBIN SCHEDULING ====

PID     BT      WT      TAT
1       10      13      23
2       5       9       14
3       8       14      22

Gantt Chart:
| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P3 | P1 |

Average Waiting Time = 12.00
Average Turnaround Time = 19.67
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

Task 2: Sequential File Allocation

Write a Python program to simulate sequential file allocation strategy.

```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks


n = int(input("Enter number of files: "))
for i in range(n):
    start = int(input(f"Enter starting block for file {i+1}: "))
    length = int(input(f"Enter length of file {i+1}: "))
    allocated = True
    for j in range(start, start+length):
        if j >= total_blocks or block_status[j] == 1:
            allocated = False
            break
    if allocated:
        for j in range(start, start+length):
            block_status[j] = 1
        print(f"File {i+1} allocated from block {start} to {start+length-1}")
    else:
        print(f"File {i+1} cannot be allocated.")
```



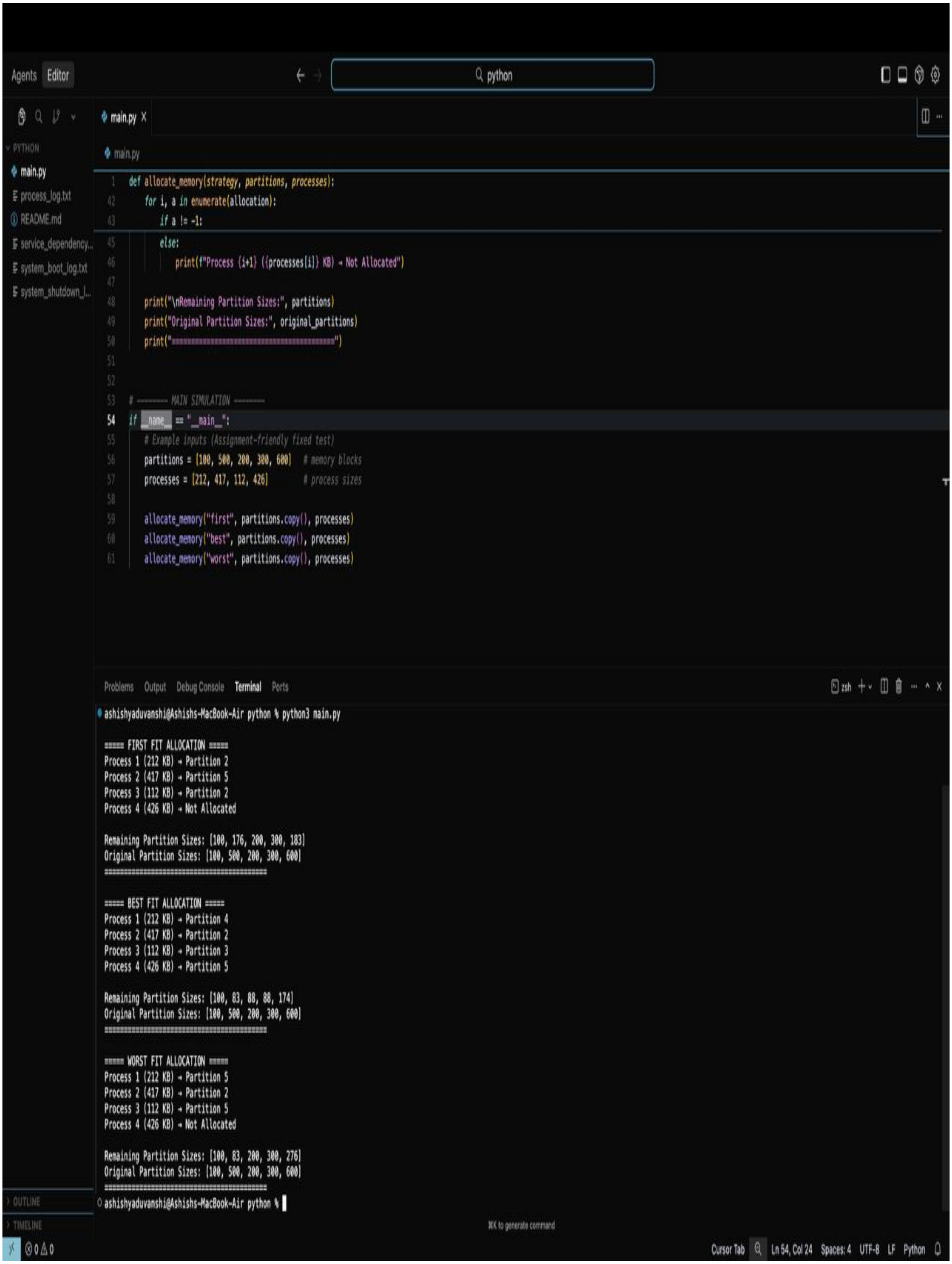
```python
def indexed_file_allocation():
    print("\n=== INDEXED FILE ALLOCATION SIMULATION ===\n")

    total_blocks = 15              # total disk blocks
    block_status = [0] * total_blocks    # 0 = free, 1 = allocated

    files = [
        {"index": 2, "data": [5, 6, 7]},
        {"index": 4, "data": [8, 9]},
        {"index": 10, "data": [11, 12, 13]},
    ]

    for i, file in enumerate(files, start=1):
        index = file["index"]
        data_blocks = file["data"]

        print(f"\nFile {i}: Index Block = {index}, Data Blocks = {data_blocks}")

        # Check index block free or not
        if block_status[index] == 1:
            print("❌ Index block already allocated. File cannot be allocated.")
            continue

        # Check data blocks free or not
        failed = False
```

```
=== INDEXED FILE ALLOCATION SIMULATION ===


File 1: Index Block = 2, Data Blocks = [5, 6, 7]
✅ File 1 allocated successfully:
   Index Block 2 → Data Blocks [5, 6, 7]

File 2: Index Block = 4, Data Blocks = [8, 9]
✅ File 2 allocated successfully:
   Index Block 4 → Data Blocks [8, 9]

File 3: Index Block = 10, Data Blocks = [11, 12, 13]
✅ File 3 allocated successfully:
   Index Block 10 → Data Blocks [11, 12, 13]

Final Block Allocation Status:
Block 0: Free
Block 1: Free
Block 2: Allocated
Block 3: Free
Block 4: Allocated
Block 5: Allocated
Block 6: Allocated
Block 7: Allocated
Block 8: Allocated
Block 9: Allocated
Block 10: Allocated
Block 11: Allocated
Block 12: Allocated
Block 13: Allocated
Block 14: Free
```

Task 3: Indexed File Allocation

Write a Python program to simulate indexed file allocation strategy.

```
total_blocks = int(input("Enter total number of blocks: "))
block_status = [0] * total_blocks
n = int(input("Enter number of files: "))
for i in range(n):
    index = int(input(f"Enter index block for file {i+1}: "))
    if block_status[index] == 1:
        print("Index block already allocated.")
        continue
    count = int(input("Enter number of data blocks: "))
    data_blocks = list(map(int, input("Enter block numbers: ").split()))
    if any(block_status[blk] == 1 for blk in data_blocks) or len(data_blocks) != count:
        print("Block(s) already allocated or invalid input.")
        continue
    block_status[index] = 1
    for blk in data_blocks:
        block_status[blk] = 1
    print(f"File {i+1} allocated with index block {index} -> {data_blocks}")
```



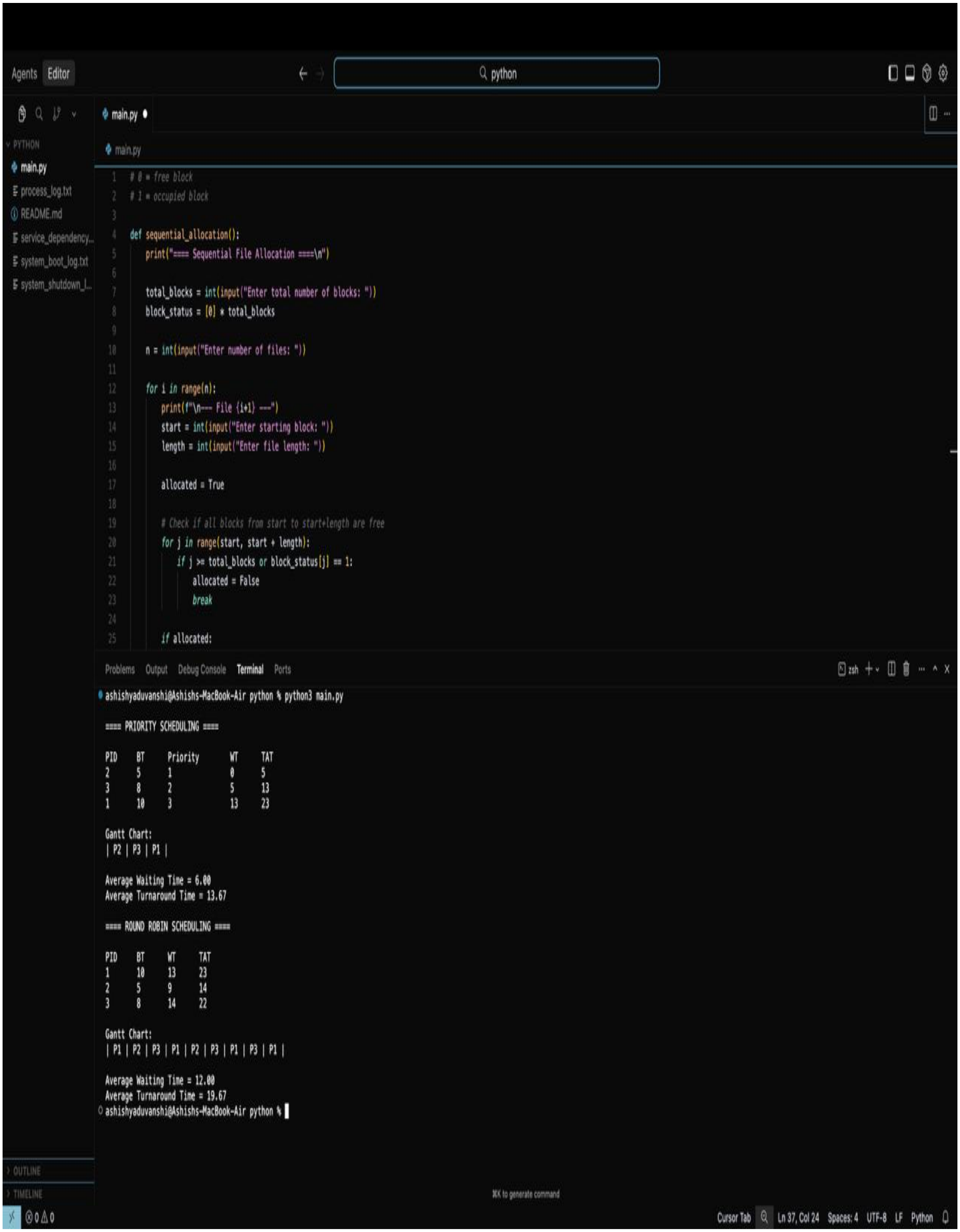
```python
def allocate_memory(strategy, partitions, processes):
    for i, a in enumerate(allocation):
        if a != -1:
        else:
            print(f"Process {i+1} ({processes[i]} KB) → Not Allocated")

    print("\nRemaining Partition Sizes:", partitions)
    print("Original Partition Sizes:", original_partitions)
    print("========================================")


# ------- MAIN SIMULATION -------
if __name__ == "__main__":
    # Example inputs (Assignment-friendly fixed test)
    partitions = [100, 500, 200, 300, 600]   # memory blocks
    processes = [212, 417, 112, 426]         # process sizes

    allocate_memory("first", partitions.copy(), processes)
    allocate_memory("best", partitions.copy(), processes)
    allocate_memory("worst", partitions.copy(), processes)
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

===== FIRST FIT ALLOCATION =====
Process 1 (212 KB) → Partition 2
Process 2 (417 KB) → Partition 5
Process 3 (112 KB) → Partition 2
Process 4 (426 KB) → Not Allocated

Remaining Partition Sizes: [100, 176, 200, 300, 183]
Original Partition Sizes: [100, 500, 200, 300, 600]
========================================

===== BEST FIT ALLOCATION =====
Process 1 (212 KB) → Partition 4
Process 2 (417 KB) → Partition 2
Process 3 (112 KB) → Partition 3
Process 4 (426 KB) → Partition 5

Remaining Partition Sizes: [100, 83, 88, 88, 174]
Original Partition Sizes: [100, 500, 200, 300, 600]
========================================

===== WORST FIT ALLOCATION =====
Process 1 (212 KB) → Partition 5
Process 2 (417 KB) → Partition 2
Process 3 (112 KB) → Partition 5
Process 4 (426 KB) → Not Allocated

Remaining Partition Sizes: [100, 83, 200, 300, 276]
Original Partition Sizes: [100, 500, 200, 300, 600]
========================================
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

Task 4: Contiguous Memory Allocation

Simulate Worst-fit, Best-fit, and First-fit memory allocation strategies.

```
def allocate_memory(strategy):
    partitions = list(map(int, input("Enter partition sizes: ").split()))
    processes = list(map(int, input("Enter process sizes: ").split()))
    allocation = [-1] * len(processes)

    for i, psize in enumerate(processes):
        idx = -1
        if strategy == "first":
            for j, part in enumerate(partitions):
                if part >= psize:
                    idx = j
                    break
        elif strategy == "best":
            best_fit = float("inf")
            for j, part in enumerate(partitions):
                if part >= psize and part < best_fit:
                    best_fit = part
                    idx = j
        elif strategy == "worst":
            worst_fit = -1
            for j, part in enumerate(partitions):
                if part >= psize and part > worst_fit:
                    worst_fit = part
                    idx = j
        if idx != -1:
            allocation[i] = idx
            partitions[idx] -= psize

    for i, a in enumerate(allocation):
        if a != -1:
            print(f"Process {i+1} allocated in Partition {a+1}")
        else:
            print(f"Process {i+1} cannot be allocated")

allocate_memory("first")
allocate_memory("best")
allocate_memory("worst")
```



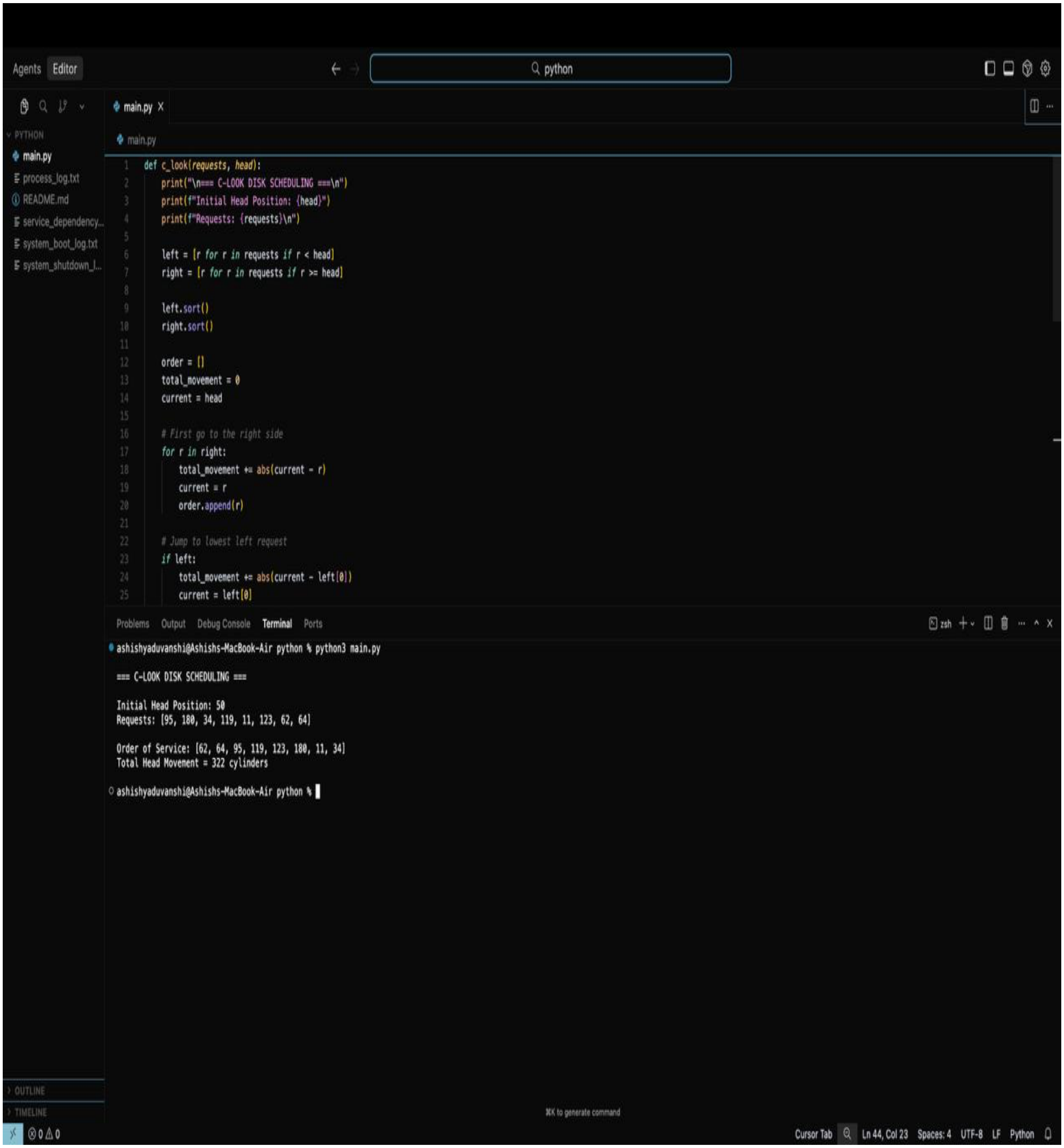
```python
# 0 = free block
# 1 = occupied block

def sequential_allocation():
    print("==== Sequential File Allocation ====\n")

    total_blocks = int(input("Enter total number of blocks: "))
    block_status = [0] * total_blocks

    n = int(input("Enter number of files: "))

    for i in range(n):
        print(f"\n--- File {i+1} ---")
        start = int(input("Enter starting block: "))
        length = int(input("Enter file length: "))

        allocated = True

        # Check if all blocks from start to start+length are free
        for j in range(start, start + length):
            if j >= total_blocks or block_status[j] == 1:
                allocated = False
                break

        if allocated:
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

==== PRIORITY SCHEDULING ====

PID     BT      Priority        WT      TAT
2       5       1               0       5
3       8       2               5       13
1       10      3               13      23

Gantt Chart:
| P2 | P3 | P1 |

Average Waiting Time = 6.00
Average Turnaround Time = 13.67

==== ROUND ROBIN SCHEDULING ====

PID     BT      WT      TAT
1       10      13      23
2       5       9       14
3       8       14      22

Gantt Chart:
| P1 | P2 | P3 | P1 | P2 | P3 | P1 | P3 | P1 |

Average Waiting Time = 12.00
Average Turnaround Time = 19.67
ashishyaduvanshi@Ashishs-MacBook-Air python % 
```

Task 5: MFT & MVT Memory Management

Implement MFT (fixed partitions) and MVT (variable partitions) strategies in Python.

```
def MFT():
    mem_size = int(input("Enter total memory size: "))
    part_size = int(input("Enter partition size: "))
    n = int(input("Enter number of processes: "))
    partitions = mem_size // part_size
    print(f"Memory divided into {partitions} partitions")
    for i in range(n):
        psize = int(input(f"Enter size of Process {i+1}: "))
        if psize <= part_size:
            print(f"Process {i+1} allocated.")
        else:
            print(f"Process {i+1} too large for fixed partition.")


def MVT():
    mem_size = int(input("Enter total memory size: "))
    n = int(input("Enter number of processes: "))
    for i in range(n):
        psize = int(input(f"Enter size of Process {i+1}: "))
        if psize <= mem_size:
            print(f"Process {i+1} allocated.")
            mem_size -= psize
        else:
            print(f"Process {i+1} cannot be allocated. Not enough memory.")
print("MFT Simulation:")

MFT()

print("\nMVT Simulation:")

MVT()
```



```python
def c_look(requests, head):
    print("\n=== C-LOOK DISK SCHEDULING ===\n")
    print(f"Initial Head Position: {head}")
    print(f"Requests: {requests}\n")

    left = [r for r in requests if r < head]
    right = [r for r in requests if r >= head]

    left.sort()
    right.sort()

    order = []
    total_movement = 0
    current = head

    # First go to the right side
    for r in right:
        total_movement += abs(current - r)
        current = r
        order.append(r)

    # Jump to lowest left request
    if left:
        total_movement += abs(current - left[0])
        current = left[0]
```

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py

=== C-LOOK DISK SCHEDULING ===

Initial Head Position: 50
Requests: [95, 180, 34, 119, 11, 123, 62, 64]

Order of Service: [62, 64, 95, 119, 123, 180, 11, 34]
Total Head Movement = 322 cylinders

ashishyaduvanshi@Ashishs-MacBook-Air python %
```