Rohit Kumar

2301010365

LAB ASSIGNMENT-03

Experiment Title: Simulation of File Allocation, Memory Management, and Scheduling in Python

Task 1: CPU Scheduling with Gantt Chart

Write a Python program to simulate Priority and Round Robin scheduling

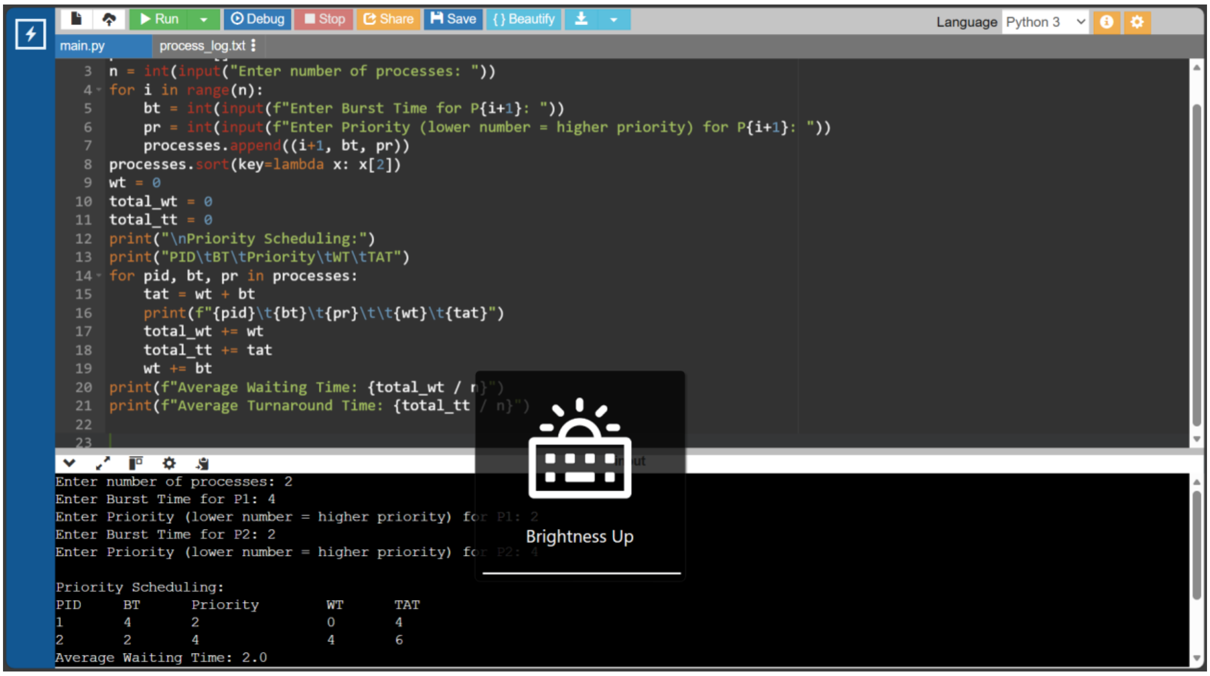
algorithms. Compute average waiting and turnaround times.

Implementation:

```
# 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}")
```

Output:



Task 2: Sequential File Allocation

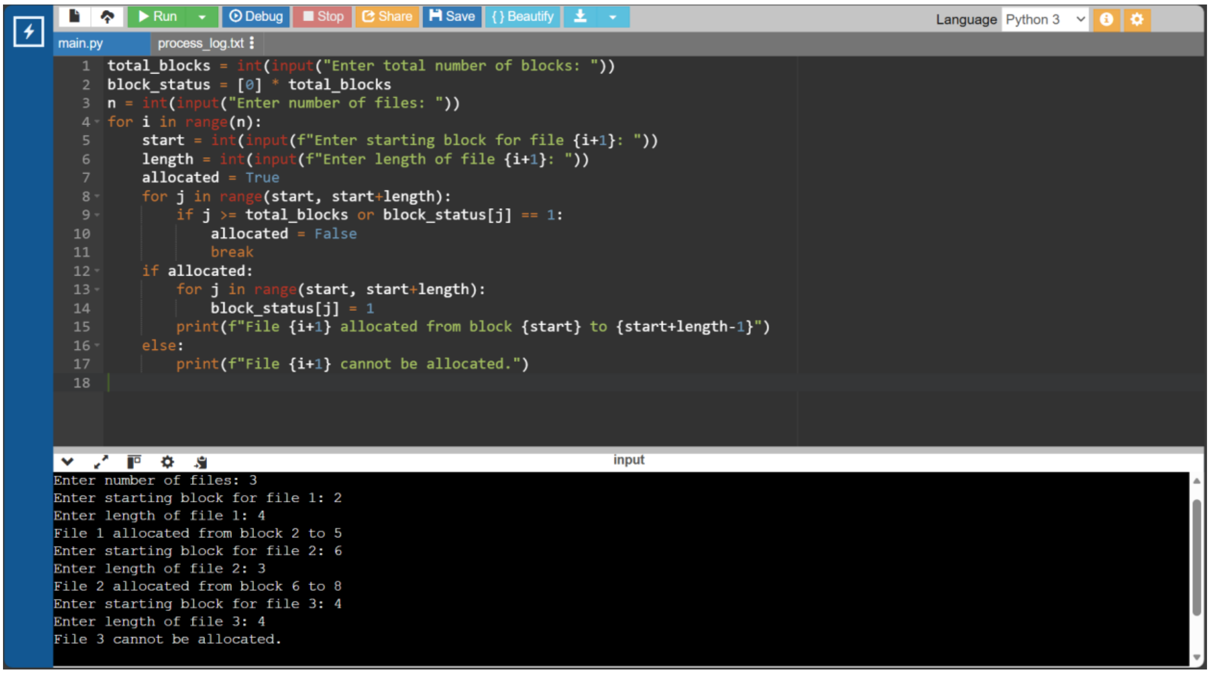
Write a Python program to simulate sequential file allocation strategy.

Implementation:

```
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.")
```

Output:



Task 3: Indexed File Allocation

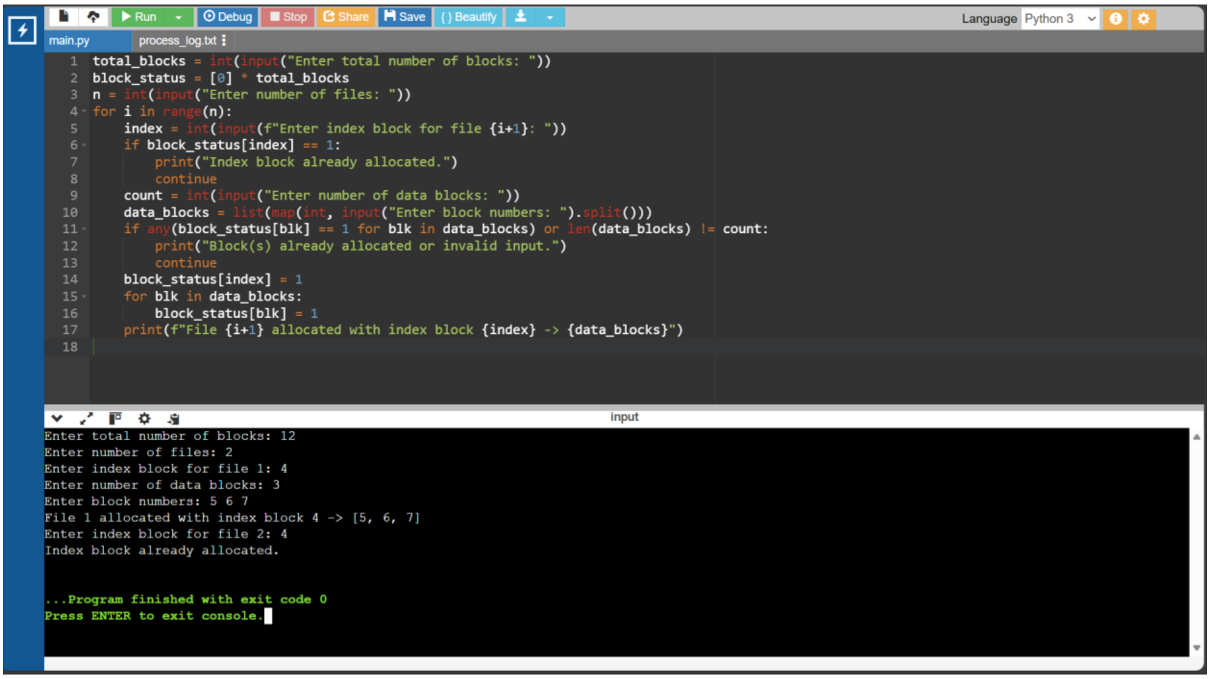
Write a Python program to simulate indexed file allocation strategy.

Implementation:

```
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}")
```

Output:



Task 4: Contiguous Memory Allocation

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

Implementation:

```
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")
```

Output:

Task 5: MFT & MVT Memory Management

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

Implementation:

```
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()

Output: