///spos practical 5  
  
/// fcfs code:  
def fcfs\_scheduling():

processes = int(input("Enter the number of processes (up to 6): "))

bt = [0] \* processes # Burst times

wt = [0] \* processes # Waiting times

tt = [0] \* processes # Turnaround times

arrival\_times = list(range(processes)) # Arrival times: 0, 1, 2, ..., processes-1

print("Enter burst times:")

for i in range(processes):

bt[i] = int(input(f"Process {i + 1}: "))

# Calculate exit times and turnaround times

exit\_times = [0] \* processes

for i in range(processes):

if i == 0:

exit\_times[i] = arrival\_times[i] + bt[i] # Exit time for first process

else:

exit\_times[i] = max(arrival\_times[i], exit\_times[i - 1]) + bt[i] # Consider arrival time

tt[i] = exit\_times[i] - arrival\_times[i] # Turnaround time = Exit time - Arrival time

wt[i] = tt[i] - bt[i] # Waiting time = Turnaround time - Burst time

# Calculate total waiting time and total turnaround time

total\_wt = sum(wt)

total\_tt = sum(tt)

# Output the waiting and turnaround times with proper spacing

print("\nProcess\tBurst Time\tArrival Time\tWaiting Time\tTurnaround Time")

for i in range(processes):

print(f"{i + 1:<7}\t{bt[i]:<12}\t{arrival\_times[i]:<14}\t{wt[i]:<14}\t{tt[i]:<15}")

# Output the averages

print("\nAverage waiting time and average turnaround time respectively:")

print(f"Average Waiting Time: {total\_wt / processes:.2f}")

print(f"Average Turnaround Time: {total\_tt / processes:.2f}")

# Call the function to execute

fcfs\_scheduling()

// fcfs output :

Enter the number of processes (up to 6): 5

Enter burst times:

Process 1: 4

Process 2: 3

Process 3: 1

Process 4: 2

Process 5: 5

Process Burst Time Arrival Time Waiting Time Turnaround Time

1 4 0 0 4

2 3 1 3 6

3 1 2 5 6

4 2 3 5 7

5 5 4 6 11

Average waiting time and average turnaround time respectively:

Average Waiting Time: 3.80

Average Turnaround Time: 6.80

=== Code Execution Successful ===

/// priority code :

# Python3 implementation for Priority Scheduling

# with Different Arrival Times, where processes with

# same arrival times are sorted by priority and FCFS is applied

totalprocess = int(input("Enter the number of processes: ")) # Taking total number of processes

proc = []

# Initialize process list with totalprocess empty sublists

for i in range(totalprocess):

l = [0, 0, 0, 0] # Format: [Arrival Time, Burst Time, Priority, Process Number]

proc.append(l)

# Function to calculate Waiting Time using FCFS

def get\_wt\_time(wt):

service = [0] \* totalprocess # Service time array

service[0] = proc[0][0] # First process starts at its arrival time

wt[0] = 0 # First process has no waiting time

for i in range(1, totalprocess):

# Calculate service time as the cumulative burst time

service[i] = service[i - 1] + proc[i - 1][1]

# Calculate waiting time as service time - arrival time

wt[i] = service[i] - proc[i][0]

# Ensure that waiting time is non-negative

if wt[i] < 0:

wt[i] = 0

# Function to calculate Turnaround Time

def get\_tat\_time(tat, wt):

for i in range(totalprocess):

tat[i] = proc[i][1] + wt[i] # Turnaround time = Burst time + Waiting time

# Function to simulate the scheduling process and display Gantt chart

def findgc():

wt = [0] \* totalprocess # Waiting times

tat = [0] \* totalprocess # Turnaround times

wavg = 0 # Total waiting time

tavg = 0 # Total turnaround time

# Calculate waiting time and turnaround time

get\_wt\_time(wt)

get\_tat\_time(tat, wt)

stime = [0] \* totalprocess # Start times

ctime = [0] \* totalprocess # Completion times

stime[0] = proc[0][0] # First process starts at its arrival time

ctime[0] = stime[0] + proc[0][1] # Completion time of first process

# Calculate start time and completion time for each process

for i in range(1, totalprocess):

stime[i] = max(proc[i][0], ctime[i - 1]) # Start time is max of arrival and previous completion

ctime[i] = stime[i] + proc[i][1] # Completion time

# Display the process information and Gantt chart

print("Process\tArrival\tPriority\tBurst\tStart\tComplete\tTurnaround\tWaiting")

for i in range(totalprocess):

wavg += wt[i]

tavg += tat[i]

print(f"P{proc[i][3]}\t{proc[i][0]}\t{proc[i][2]}\t\t{proc[i][1]}\t{stime[i]}\t{ctime[i]}\t\t{tat[i]}\t\t{wt[i]}")

# Display the average waiting time and turnaround time

print(f"\nAverage Waiting Time: {wavg / totalprocess:.2f}")

print(f"Average Turnaround Time: {tavg / totalprocess:.2f}")

# Driver code

if \_\_name\_\_ == "\_\_main\_\_":

# Input data from user

for i in range(totalprocess):

arrivaltime = int(input(f"Enter arrival time for process {i + 1}: "))

bursttime = int(input(f"Enter burst time for process {i + 1}: "))

priority = int(input(f"Enter priority for process {i + 1} (lower number means higher priority): "))

# Initialize process information: [Arrival Time, Burst Time, Priority, Process Number]

proc[i][0] = arrivaltime # Arrival Time

proc[i][1] = bursttime # Burst Time

proc[i][2] = priority # Priority

proc[i][3] = i + 1 # Process Number

# First sort by arrival time, then by priority

proc.sort(key=lambda x: (x[0], x[2]))

# Calling function to generate Gantt chart and find scheduling details

findgc()

/// code output:

Enter the number of processes: 3

Enter arrival time for process 1: 0

Enter burst time for process 1: 5

Enter priority for process 1 (lower number means higher priority): 2

Enter arrival time for process 2: 1

Enter burst time for process 2: 3

Enter priority for process 2 (lower number means higher priority): 1

Enter arrival time for process 3: 2

Enter burst time for process 3: 8

Enter priority for process 3 (lower number means higher priority): 3  
Process Arrival Priority Burst Start Complete Turnaround Waiting

P1 0 2 5 0 5 5 0

P2 1 1 3 5 8 7 4

P3 2 3 8 8 16 14 6

Average Waiting Time: 3.33

Average Turnaround Time: 8.67

=== Code Execution Successful ===  
  
// round robin code :

def round\_robin\_scheduling():

# Input: number of processes and time quantum

processes = int(input("Enter the number of processes: "))

quantum = int(input("Enter the quantum time: "))

# Input: arrival times and burst times for each process

at = [0] \* processes # arrival times

bt = [0] \* processes # burst times

rem\_bt = [0] \* processes # remaining burst times

print("Enter the arrival times:")

for i in range(processes):

at[i] = int(input(f"Process {i + 1} arrival time: "))

print("Enter the burst times:")

for i in range(processes):

bt[i] = int(input(f"Process {i + 1} burst time: "))

rem\_bt[i] = bt[i] # Initialize remaining burst time

# Initialize variables

wt = [0] \* processes # waiting times

ct = [0] \* processes # completion times

tt = [0] \* processes # turnaround times

t = 0 # current time

complete = 0 # number of completed processes

ready\_queue = [] # queue to hold processes ready to be executed

arrived = [False] \* processes # to track if a process has arrived

# Start Round Robin scheduling

while complete < processes:

# Check for newly arrived processes at time 't'

for i in range(processes):

if at[i] <= t and not arrived[i]:

ready\_queue.append(i)

arrived[i] = True

if ready\_queue:

# Get the first process in the ready queue (FIFO order)

i = ready\_queue.pop(0)

if rem\_bt[i] > quantum:

t += quantum # Increment time by quantum

rem\_bt[i] -= quantum # Reduce remaining burst time

# Check for any new arrivals during this quantum slice

for j in range(processes):

if at[j] <= t and not arrived[j]:

ready\_queue.append(j)

arrived[j] = True

# Re-add the current process to the ready queue if not finished

ready\_queue.append(i)

else:

t += rem\_bt[i] # Increment time by remaining burst time

rem\_bt[i] = 0 # Process completed

ct[i] = t # Set completion time

tt[i] = ct[i] - at[i] # Turnaround time

wt[i] = tt[i] - bt[i] # Waiting time

complete += 1 # Increment completed process count

else:

t += 1 # If no process is ready, increment time to next available process

# Output the results in a table format

print("\nProcess\tArrival\tBurst\tCompletion\tWaiting\tTurnaround")

for i in range(processes):

print(f"P{i + 1}\t\t{at[i]}\t\t{bt[i]}\t\t{ct[i]}\t\t\t{wt[i]}\t\t{tt[i]}")

# Calculate total waiting and turnaround times

total\_wt = sum(wt)

total\_tt = sum(tt)

print(f"\nAverage Waiting Time: {total\_wt / processes:.2f}")

print(f"Average Turnaround Time: {total\_tt / processes:.2f}")

# Run the scheduling function

round\_robin\_scheduling()  
  
// output :

Enter the number of processes: 4

Enter the quantum time: 2

Enter the arrival times:

Process 1 arrival time: 0

Process 2 arrival time: 1

Process 3 arrival time: 2

Process 4 arrival time: 4

Enter the burst times:

Process 1 burst time: 5

Process 2 burst time: 4

Process 3 burst time: 2

Process 4 burst time: 1

Process Arrival Burst Completion Waiting Turnaround

P1 0 5 12 7 12

P2 1 4 11 6 10

P3 2 2 6 2 4

P4 4 1 9 4 5

Average Waiting Time: 4.75

Average Turnaround Time: 7.75

=== Code Execution Successful ===  
  
def sjf\_preemptive\_scheduling():

processes = int(input("Enter the number of processes: "))

arrival\_times = [0] \* processes

burst\_times = [0] \* processes

print("Enter the arrival times:")

for i in range(processes):

arrival\_times[i] = int(input(f"Process {i + 1}: "))

print("Enter the burst times:")

for i in range(processes):

burst\_times[i] = int(input(f"Process {i + 1}: "))

remaining\_times = burst\_times[:]

completion\_times = [0] \* processes

waiting\_times = [0] \* processes

turnaround\_times = [0] \* processes

time = 0

completed = 0

while completed < processes:

min\_remaining\_time = float('inf')

shortest\_process = None

# Find the process with the shortest remaining time

for i in range(processes):

if arrival\_times[i] <= time and remaining\_times[i] > 0:

if remaining\_times[i] < min\_remaining\_time:

min\_remaining\_time = remaining\_times[i]

shortest\_process = i

# If no process is ready, move time forward

if shortest\_process is None:

time += 1

continue

# Execute the selected process for 1 unit of time

remaining\_times[shortest\_process] -= 1

# If the process finishes, calculate times

if remaining\_times[shortest\_process] == 0:

completed += 1

completion\_times[shortest\_process] = time + 1

waiting\_times[shortest\_process] = (completion\_times[shortest\_process] -

arrival\_times[shortest\_process] -

burst\_times[shortest\_process])

turnaround\_times[shortest\_process] = (waiting\_times[shortest\_process] +

burst\_times[shortest\_process])

time += 1

# Calculate total waiting and turnaround times

total\_wt = sum(waiting\_times)

total\_tt = sum(turnaround\_times)

# Print the results in table format

print("\nProcess\t\tArrival\t\tBurst\t\tCompletion\t\tWaiting\t\tTurnaround")

for i in range(processes):

print(f"P{i + 1}\t\t{arrival\_times[i]}\t\t{burst\_times[i]}\t\t{completion\_times[i]}\t\t\t{waiting\_times[i]}\t\t\t{turnaround\_times[i]}")

print(f"\nAverage Waiting Time: {total\_wt / processes:.2f}")

print(f"Average Turnaround Time: {total\_tt / processes:.2f}")

sjf\_preemptive\_scheduling()

\\OUTPUT :

Enter the number of processes: 4

Enter the arrival times:

Process 1: 1

Process 2: 2

Process 3: 1

Process 4: 4

Enter the burst times:

Process 1: 3

Process 2: 4

Process 3: 2

Process 4: 4

Process Arrival Burst Completion Waiting Turnaround

P1 1 3 6 2 5

P2 2 4 10 4 8

P3 1 2 3 0 2

P4 4 4 14 6 10

Average Waiting Time: 3.00

Average Turnaround Time: 6.25

=== Code Execution Successful ===