1. FCFS

```
def fcfs_scheduling(processes):
    # Sort processes by arrival time
    processes.sort(key=lambda x: x['arrival_time'])
    current_time = 0
    for process in processes:
        if current_time < process['arrival_time']:
            current_time = process['arrival_time']
        process['start_time'] = current_time
        process['completion_time'] = current_time + process['burst_time']
        process['turnaround_time'] = process['completion_time'] - process['arrival_time']
        process['waiting_time'] = process['turnaround_time'] - process['burst_time']
        current_time += process['burst_time']
    return processes


# Example usage
if __name__ == "__main__":
    processes = [
        {'id': 'P1', 'arrival_time': 0, 'burst_time': 5},
        {'id': 'P2', 'arrival_time': 2, 'burst_time': 3},
        {'id': 'P3', 'arrival_time': 4, 'burst_time': 1},
        {'id': 'P4', 'arrival_time': 6, 'burst_time': 2}
    ]


    scheduled = fcfs_scheduling(processes)


    print("FCFS Scheduling:")
    print("PID\tArrival\tBurst\tStart\tCompletion\tTurnaround\tWaiting")
    for p in scheduled:

print(f"{p['id']}\t{p['arrival_time']}\t{p['burst_time']}\t{p['start_time']}\t{p['completion_time']}\t\t{p['
turnaround_time']}\t\t{p['waiting_time']}")
```

OUTPUT:

FCFS Scheduling:

PID	Arrival	Burst	Start	Completion	Turnaround	Waiting
P1	0	5	0	5	5	0
P2	2	3	5	8	6	3
P3	4	1	8	9	5	4
P4	6	2	9	11	5	3

2. SJF Scheduling

```
def sjf_preemptive_scheduling(processes):
    n = len(processes)
    completed = 0
    current_time = 0
    min_burst = float('inf')
    short = None
    check = False

    # Initialize all fields
    for process in processes:
        process['remaining_time'] = process['burst_time']
        process['completion_time'] = 0
        process['turnaround_time'] = 0
        process['waiting_time'] = 0

    while completed != n:
        for process in processes:
            if (process['arrival_time'] <= current_time and
                process['remaining_time'] < min_burst and
                process['remaining_time'] > 0):
                min_burst = process['remaining_time']
```

```
            short = process
            check = True

        if not check:
            current_time +=1
            continue

        # Execute the process
        short['remaining_time'] -=1
        min_burst = short['remaining_time']
        if min_burst == 0:
            min_burst = float('inf')

        if short['remaining_time'] == 0:
            completed +=1
            check = False
            short['completion_time'] = current_time +1
            short['turnaround_time'] = short['completion_time'] - short['arrival_time']
            short['waiting_time'] = short['turnaround_time'] - short['burst_time']

        current_time +=1

    return processes

# Example usage
if __name__ == "__main__":
    processes = [
        {'id': 'P1', 'arrival_time': 0, 'burst_time': 8},
        {'id': 'P2', 'arrival_time': 1, 'burst_time': 4},
        {'id': 'P3', 'arrival_time': 2, 'burst_time': 9},
        {'id': 'P4', 'arrival_time': 3, 'burst_time': 5}
```

```
    ]

    scheduled = sjf_preemptive_scheduling(processes)

    print("SJF Preemptive Scheduling:")
    print("PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting")
    for p in scheduled:

print(f"{p['id']}\t{p['arrival_time']}\t{p['burst_time']}\t{p['completion_time']}\t\t{p['turnaround_time
']}\t\t{p['waiting_time']}")
```

OUTPUT:

SJF Preemptive Scheduling:

PID	Arrival	Burst	Completion	Turnaround	Waiting
P1	0	8	17	17	9
P2	1	4	5	4	0
P3	2	9	26	24	15
P4	3	5	12	9	4

3. Priority

```
def priority_scheduling(processes):
    n = len(processes)
    completed = 0
    current_time = 0
    prev = -1

    # Initialize all fields
    for process in processes:
        process['completion_time'] = 0
        process['turnaround_time'] = 0
        process['waiting_time'] = 0
        process['is_completed'] = False
```

```
    while completed != n:
        # Find process with highest priority which has arrived and not completed
        idx = -1
        highest_priority = float('inf')
        for i in range(n):
            if (processes[i]['arrival_time'] <= current_time and
                not processes[i]['is_completed'] and
                processes[i]['priority'] < highest_priority):
                highest_priority = processes[i]['priority']
                idx = i

        if idx != -1:
            process = processes[idx]
            process['start_time'] = current_time
            process['completion_time'] = current_time + process['burst_time']
            process['turnaround_time'] = process['completion_time'] - process['arrival_time']
            process['waiting_time'] = process['turnaround_time'] - process['burst_time']
            current_time += process['burst_time']
            process['is_completed'] = True
            completed +=1
        else:
            current_time +=1

    return processes

# Example usage
if __name__ == "__main__":
    processes = [
        {'id': 'P1', 'arrival_time': 2, 'burst_time': 3, 'priority': 2},
        {'id': 'P2', 'arrival_time': 0, 'burst_time': 4, 'priority': 1},
```

```
    {'id': 'P3', 'arrival_time': 4, 'burst_time': 1, 'priority': 3},
    {'id': 'P4', 'arrival_time': 5, 'burst_time': 2, 'priority': 2}
]

scheduled = priority_scheduling(processes)

print("Priority Scheduling (Non-Preemptive):")
print("PID\tArrival\tBurst\tPriority\tCompletion\tTurnaround\tWaiting")
for p in scheduled:

print(f"{p['id']}\t{p['arrival_time']}\t{p['burst_time']}\t{p['priority']}\t\t{p['completion_time']}\t\t{p['turnaround_time']}\t\t{p['waiting_time']}")
```

OUTPUT:

Priority Scheduling (Non-Preemptive):

PID	Arrival	Burst	Priority	Completion	Turnaround	Waiting
P1	2	3	2	8	6	3
P2	0	4	1	4	4	0
P3	4	1	3	9	5	4
P4	5	2	2	11	6	4

4. Round Robin

```
def round_robin_scheduling(processes, quantum):
    from collections import deque

    n = len(processes)
    queue = deque()
    time = 0
    completed = 0

    # Initialize remaining burst time for each process
```

```
for process in processes:
    process['remaining_time'] = process['burst_time']
    process['completion_time'] = 0
    process['turnaround_time'] = 0
    process['waiting_time'] = 0

# Sort processes by arrival time
processes.sort(key=lambda x: x['arrival_time'])

# Add all processes that have arrived at time 0 to the queue
for process in processes:
    if process['arrival_time'] <= time and process['remaining_time'] > 0:
        queue.append(process)

# If no process has arrived at time 0, increment time
if not queue:
    time = processes[0]['arrival_time']
    queue.append(processes[0])

while completed != n:
    if queue:
        current_process = queue.popleft()

        # If the process has not arrived yet, skip it
        if current_process['arrival_time'] > time:
            time = current_process['arrival_time']

        # Calculate execution time
        exec_time = min(quantum, current_process['remaining_time'])
        time += exec_time
        current_process['remaining_time'] -= exec_time
```

```
            # Check for other processes that have arrived during execution
            for process in processes:
                if (process['arrival_time'] > current_process['arrival_time'] and
                    process['arrival_time'] <= time and
                    process not in queue and
                    process['remaining_time'] > 0):
                    queue.append(process)

            if current_process['remaining_time'] == 0:
                completed += 1
                current_process['completion_time'] = time
                current_process['turnaround_time'] = current_process['completion_time'] -
current_process['arrival_time']
                current_process['waiting_time'] = current_process['turnaround_time'] -
current_process['burst_time']
            else:
                queue.append(current_process)
        else:
            # If queue is empty, jump to the next process arrival time
            next_arrival = min([p['arrival_time'] for p in processes if p['remaining_time'] > 0])
            time = next_arrival
            for process in processes:
                if process['arrival_time'] == next_arrival and process['remaining_time'] > 0:
                    queue.append(process)

    return processes

# Example usage
if __name__ == "__main__":
    # Define processes with their arrival and burst times
    processes = [
```

```
        {'id': 'P1', 'arrival_time': 0, 'burst_time': 5},
        {'id': 'P2', 'arrival_time': 1, 'burst_time': 3},
        {'id': 'P3', 'arrival_time': 2, 'burst_time': 8},
        {'id': 'P4', 'arrival_time': 3, 'burst_time': 6}
    ]

    time_quantum = 2

    scheduled = round_robin_scheduling(processes, time_quantum)

    print("Round Robin Scheduling:")
    print("PID\tArrival\tBurst\tCompletion\tTurnaround\tWaiting")
    for p in scheduled:

print(f"{p['id']}\t{p['arrival_time']}\t{p['burst_time']}\t{p['completion_time']}\t\t{p['turnaround_time
']}\t\t{p['waiting_time']}")

    # Calculate average turnaround time and waiting time
    total_turnaround = sum(p['turnaround_time'] for p in scheduled)
    total_waiting = sum(p['waiting_time'] for p in scheduled)
    avg_turnaround = total_turnaround / len(scheduled)
    avg_waiting = total_waiting / len(scheduled)

    print(f"\nAverage Turnaround Time: {avg_turnaround:.2f}")
    print(f"Average Waiting Time: {avg_waiting:.2f}")
```

OUTPUT:

Round Robin Scheduling:

PID	Arrival	Burst	Completion	Turnaround	Waiting
P1	0	5	12	12	7
P2	1	3	7	6	3
P3	2	8	17	15	7

P4	3	6	16	13	7

Average Turnaround Time: 11.50

Average Waiting Time: 6.00