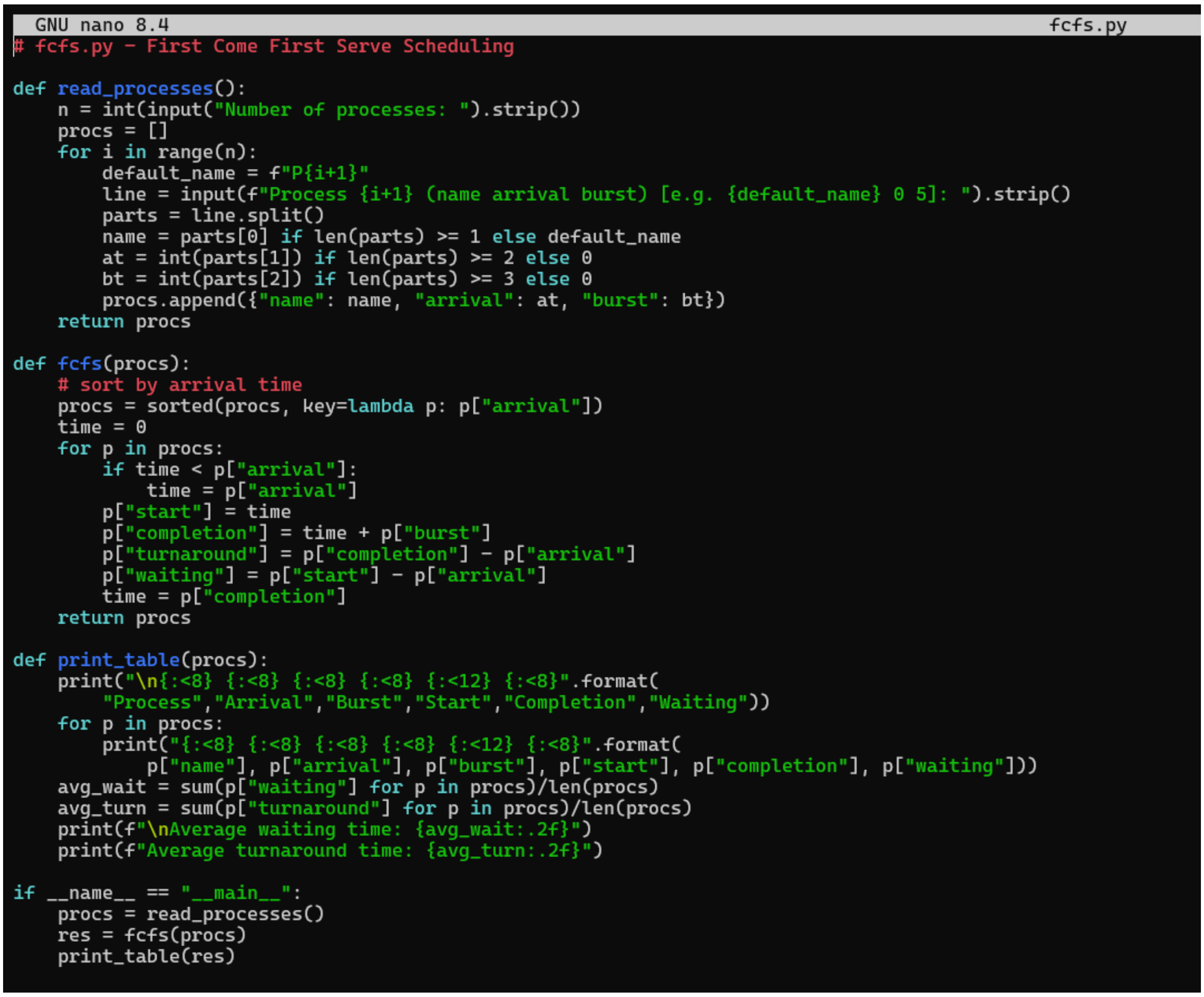
ASSIGNMENT-3

NAME-ANJALI
ROLL NO.-2301420021
COURSE-BTech CSE(DS)

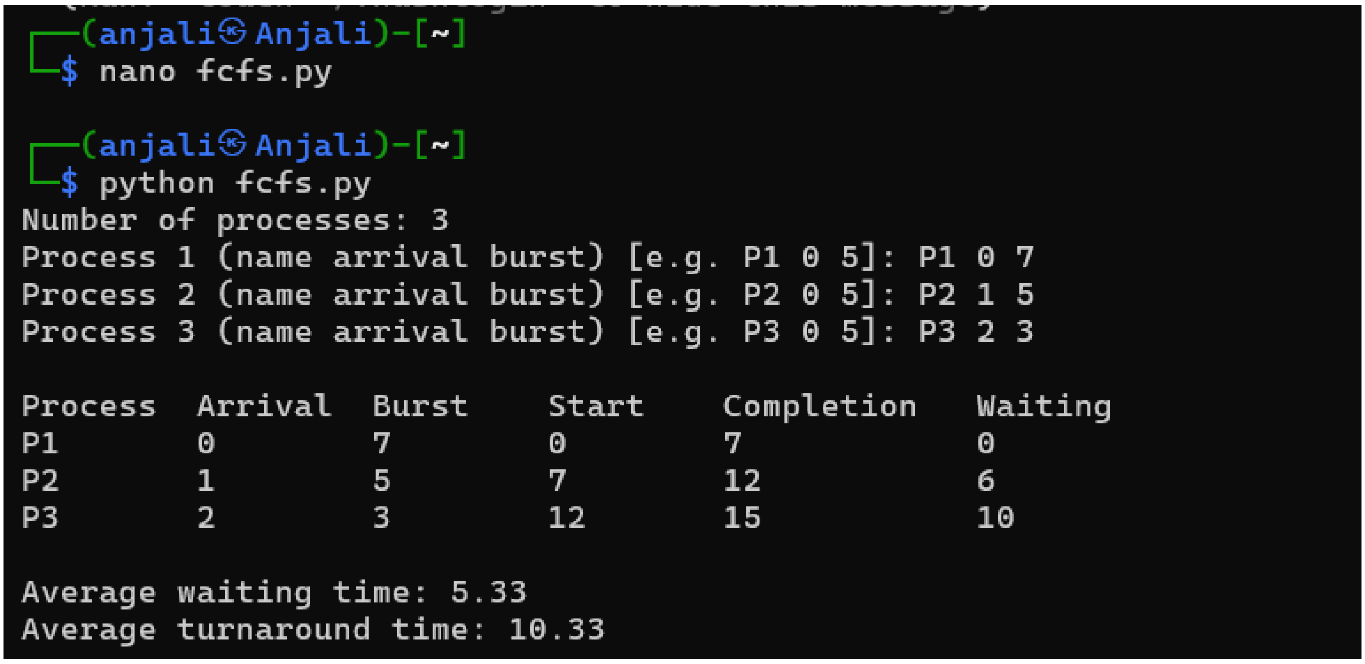
1. First Come First Serve Scheduling (FCFS):

Code:



```
  GNU nano 8.4                                                                 fcfs.py
# fcfs.py - First Come First Serve Scheduling

def read_processes():
    n = int(input("Number of processes: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt})
    return procs

def fcfs(procs):
    # sort by arrival time
    procs = sorted(procs, key=lambda p: p["arrival"])
    time = 0
    for p in procs:
        if time < p["arrival"]:
            time = p["arrival"]
        p["start"] = time
        p["completion"] = time + p["burst"]
        p["turnaround"] = p["completion"] - p["arrival"]
        p["waiting"] = p["start"] - p["arrival"]
        time = p["completion"]
    return procs

def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Start","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["start"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs = read_processes()
    res = fcfs(procs)
    print_table(res)
```

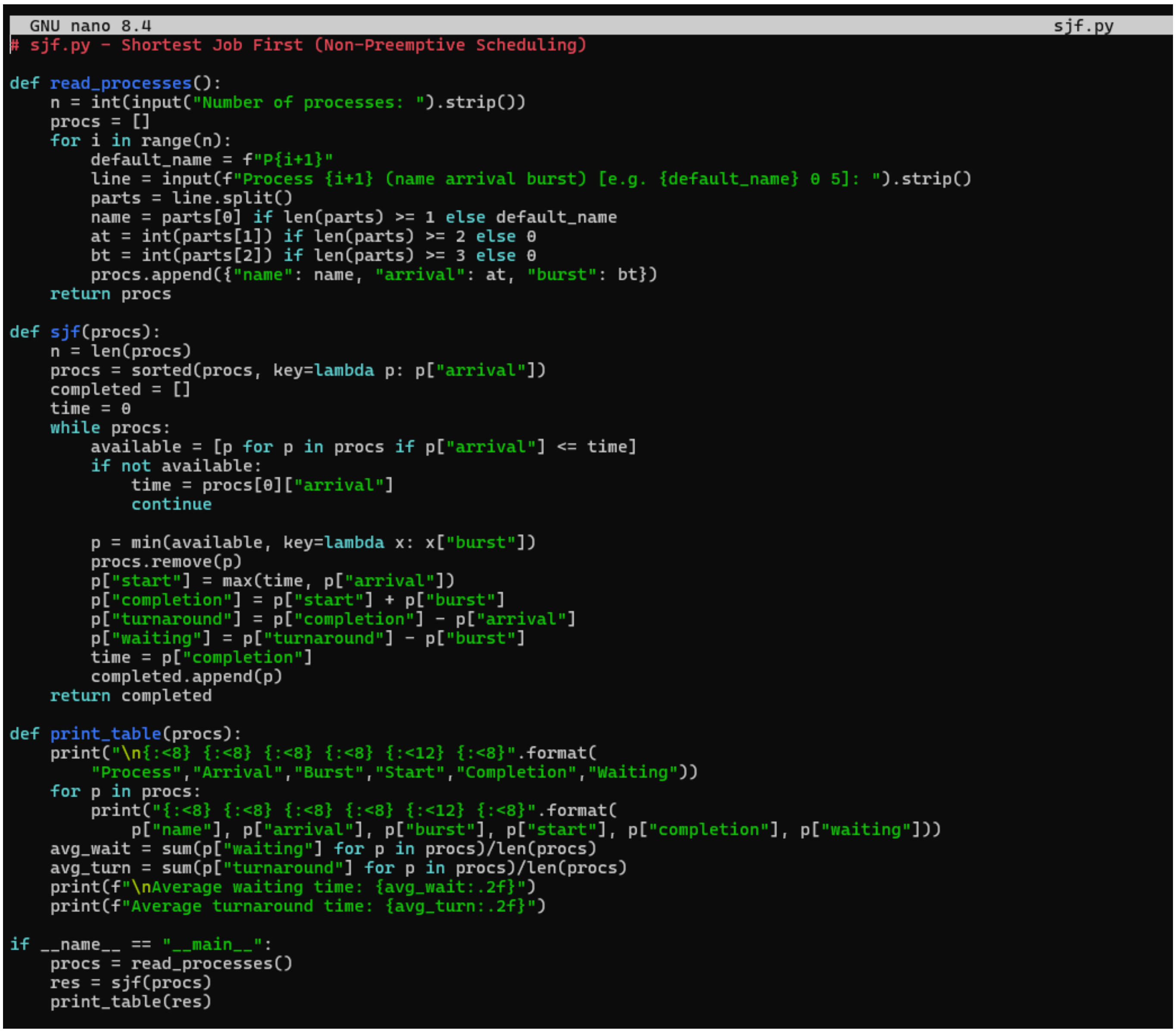
Output:



```
┌──(anjali㉿Anjali)-[~]
└─$ nano fcfs.py

┌──(anjali㉿Anjali)-[~]
└─$ python fcfs.py
Number of processes: 3
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 7
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 5
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 3

Process  Arrival  Burst    Start    Completion   Waiting
P1       0        7        0        7            0
P2       1        5        7        12           6
P3       2        3        12       15           10

Average waiting time: 5.33
Average turnaround time: 10.33
```

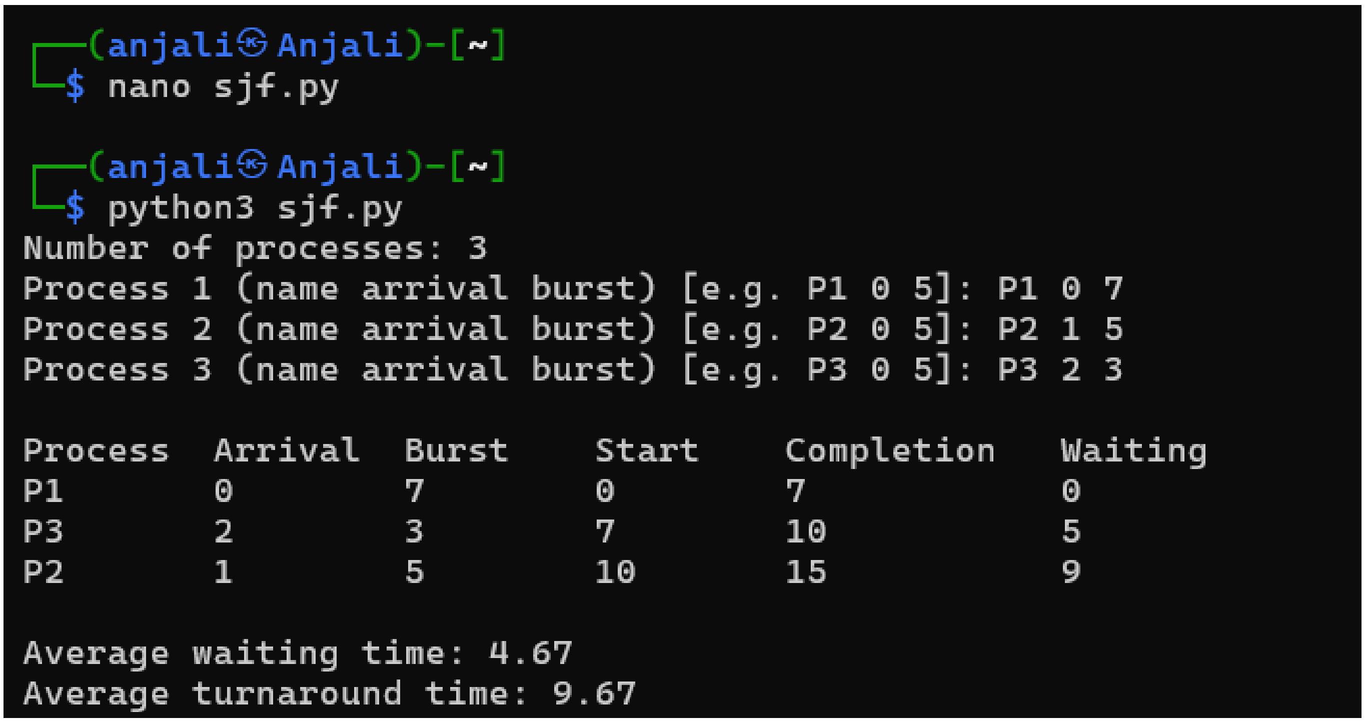
2. Shortest Job First (SJF) : Non-preemptive

Code:



```
  GNU nano 8.4                                                                                        sjf.py
# sjf.py - Shortest Job First (Non-Preemptive Scheduling)

def read_processes():
    n = int(input("Number of processes: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt})
    return procs

def sjf(procs):
    n = len(procs)
    procs = sorted(procs, key=lambda p: p["arrival"])
    completed = []
    time = 0
    while procs:
        available = [p for p in procs if p["arrival"] <= time]
        if not available:
            time = procs[0]["arrival"]
            continue

        p = min(available, key=lambda x: x["burst"])
        procs.remove(p)
        p["start"] = max(time, p["arrival"])
        p["completion"] = p["start"] + p["burst"]
        p["turnaround"] = p["completion"] - p["arrival"]
        p["waiting"] = p["turnaround"] - p["burst"]
        time = p["completion"]
        completed.append(p)
    return completed

def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Start","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["start"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs = read_processes()
    res = sjf(procs)
    print_table(res)
```

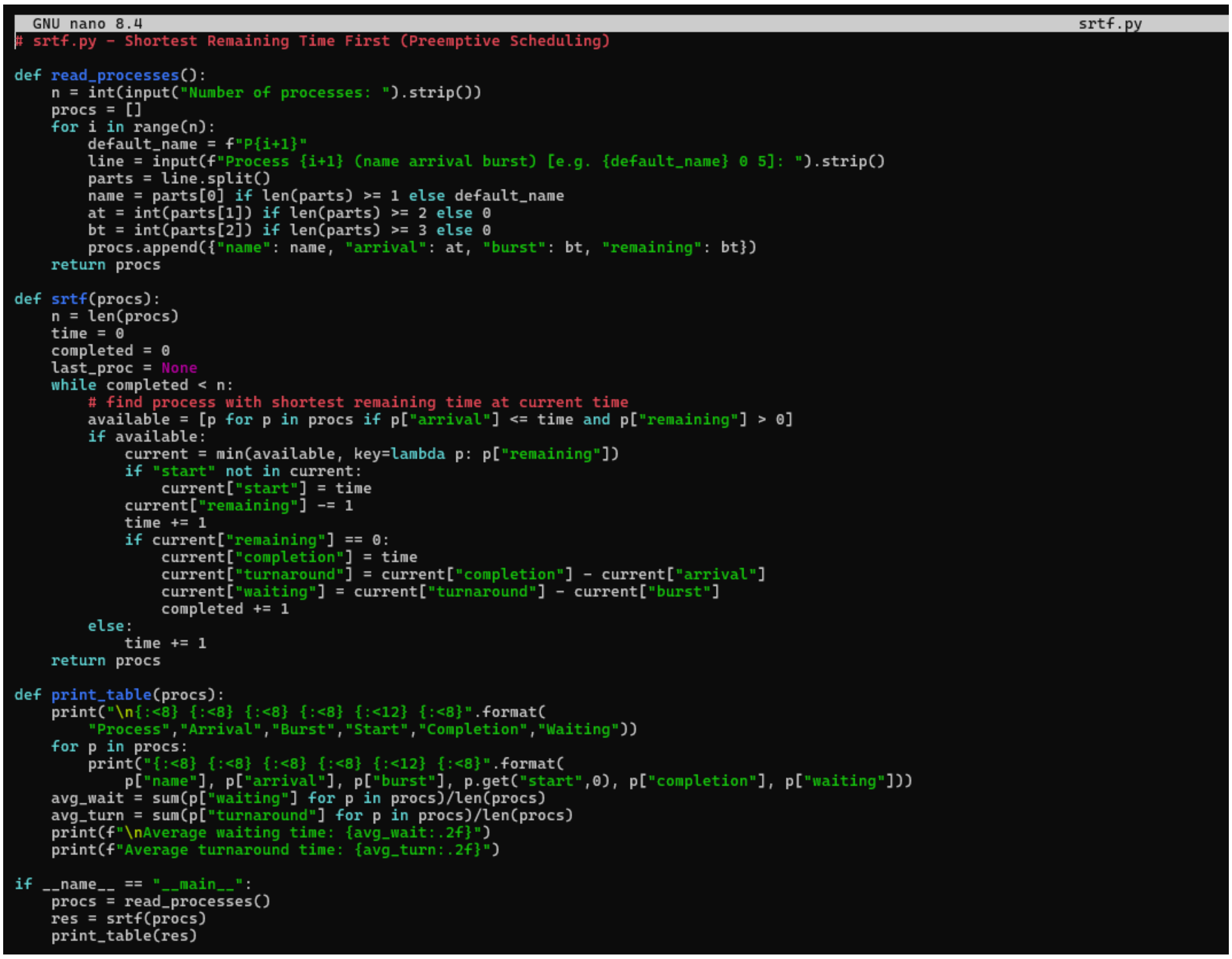
Output:



```
┌──(anjali㉿Anjali)-[~]
└─$ nano sjf.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 sjf.py
Number of processes: 3
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 7
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 5
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 3

Process   Arrival  Burst    Start    Completion   Waiting
P1        0        7        0        7            0
P3        2        3        7        10           5
P2        1        5        10       15           9

Average waiting time: 4.67
Average turnaround time: 9.67
```

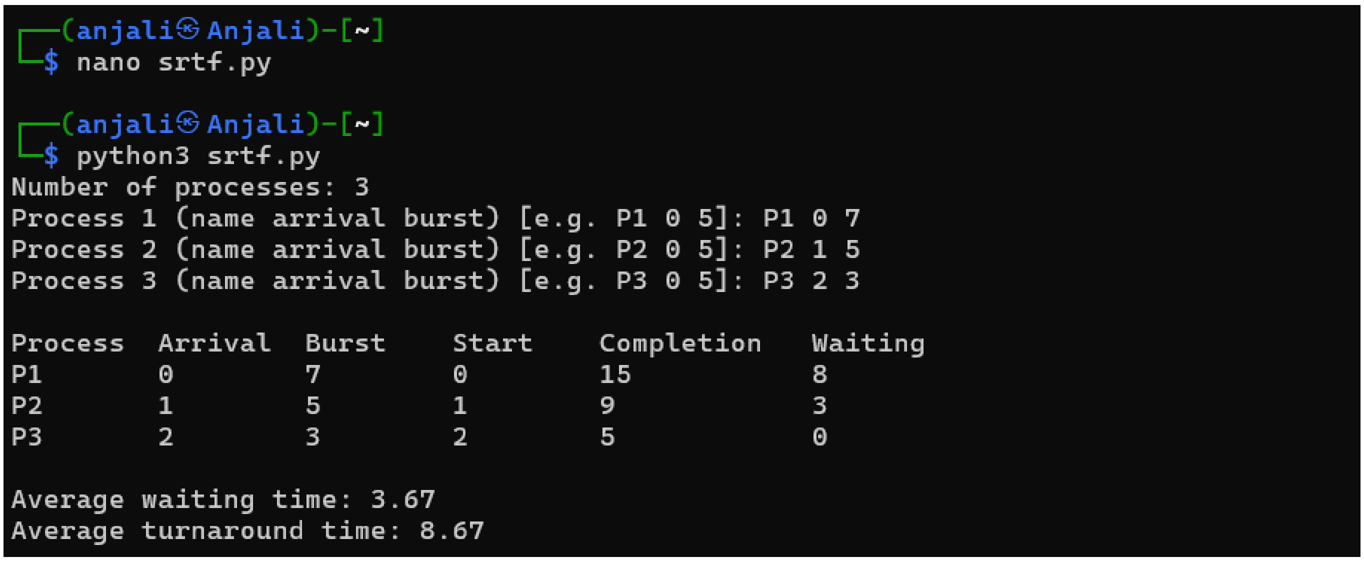
3. Shortest Remaining Time First (SRTF) : Preemptive SJF

Code:



```
  GNU nano 8.4                                                                                          srtf.py
# srtf.py - Shortest Remaining Time First (Preemptive Scheduling)

def read_processes():
    n = int(input("Number of processes: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt, "remaining": bt})
    return procs

def srtf(procs):
    n = len(procs)
    time = 0
    completed = 0
    last_proc = None
    while completed < n:
        # find process with shortest remaining time at current time
        available = [p for p in procs if p["arrival"] <= time and p["remaining"] > 0]
        if available:
            current = min(available, key=lambda p: p["remaining"])
            if "start" not in current:
                current["start"] = time
            current["remaining"] -= 1
            time += 1
            if current["remaining"] == 0:
                current["completion"] = time
                current["turnaround"] = current["completion"] - current["arrival"]
                current["waiting"] = current["turnaround"] - current["burst"]
                completed += 1
        else:
            time += 1
    return procs

def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Start","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p.get("start",0), p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs = read_processes()
    res = srtf(procs)
    print_table(res)
```

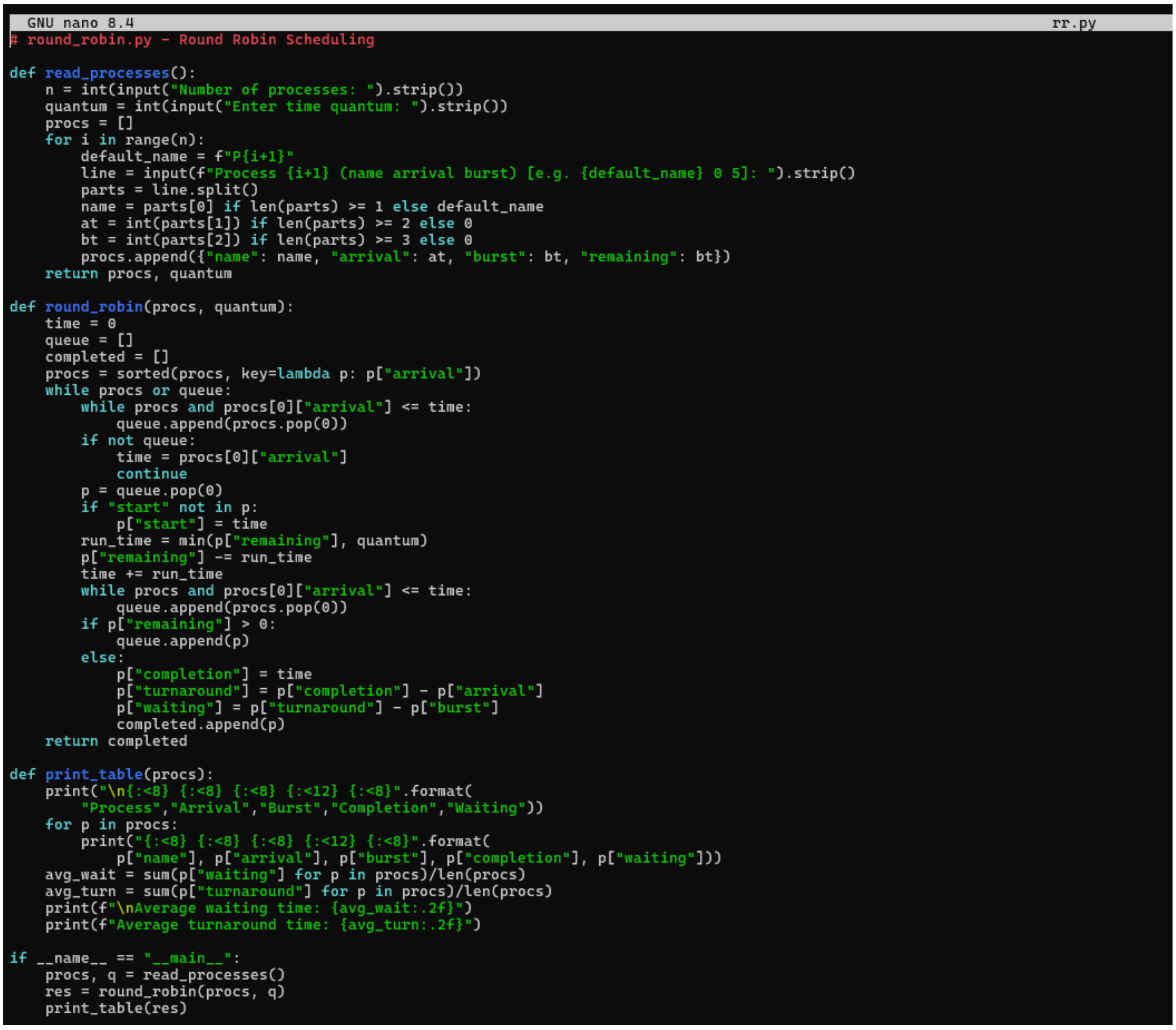
Output:



```
┌──(anjali㉿Anjali)-[~]
└─$ nano srtf.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 srtf.py
Number of processes: 3
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 7
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 5
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 3

Process  Arrival  Burst    Start    Completion   Waiting
P1       0        7        0        15           8
P2       1        5        1        9            3
P3       2        3        2        5            0

Average waiting time: 3.67
Average turnaround time: 8.67
```

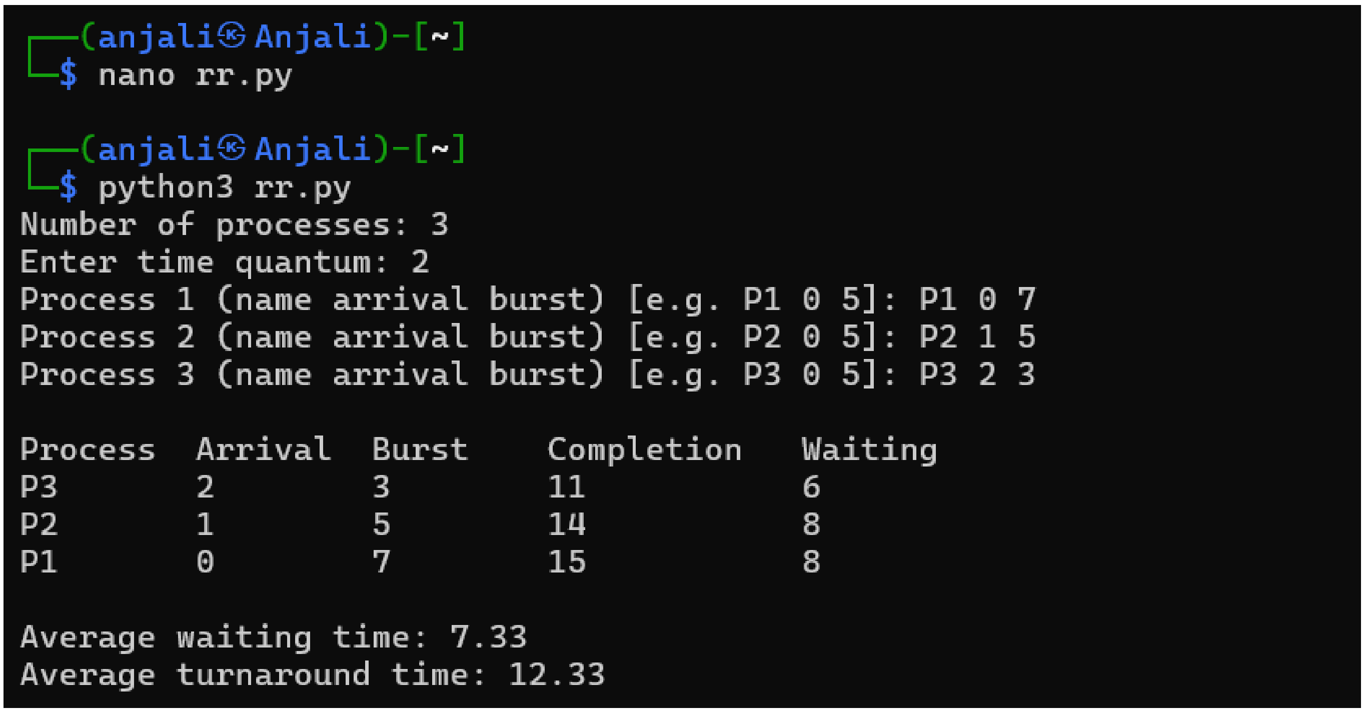
4. Round Robin (RR) :

Code:



```
 GNU nano 8.4                                                                                          rr.py
# round_robin.py - Round Robin Scheduling

def read_processes():
    n = int(input("Number of processes: ").strip())
    quantum = int(input("Enter time quantum: ").strip())
    procs = []
    for i in range(n):
        default_name = f"P{i+1}"
        line = input(f"Process {i+1} (name arrival burst) [e.g. {default_name} 0 5]: ").strip()
        parts = line.split()
        name = parts[0] if len(parts) >= 1 else default_name
        at = int(parts[1]) if len(parts) >= 2 else 0
        bt = int(parts[2]) if len(parts) >= 3 else 0
        procs.append({"name": name, "arrival": at, "burst": bt, "remaining": bt})
    return procs, quantum

def round_robin(procs, quantum):
    time = 0
    queue = []
    completed = []
    procs = sorted(procs, key=lambda p: p["arrival"])
    while procs or queue:
        while procs and procs[0]["arrival"] <= time:
            queue.append(procs.pop(0))
        if not queue:
            time = procs[0]["arrival"]
            continue
        p = queue.pop(0)
        if "start" not in p:
            p["start"] = time
        run_time = min(p["remaining"], quantum)
        p["remaining"] -= run_time
        time += run_time
        while procs and procs[0]["arrival"] <= time:
            queue.append(procs.pop(0))
        if p["remaining"] > 0:
            queue.append(p)
        else:
            p["completion"] = time
            p["turnaround"] = p["completion"] - p["arrival"]
            p["waiting"] = p["turnaround"] - p["burst"]
            completed.append(p)
    return completed

def print_table(procs):
    print("\n{:<8} {:<8} {:<8} {:<12} {:<8}".format(
        "Process","Arrival","Burst","Completion","Waiting"))
    for p in procs:
        print("{:<8} {:<8} {:<8} {:<12} {:<8}".format(
            p["name"], p["arrival"], p["burst"], p["completion"], p["waiting"]))
    avg_wait = sum(p["waiting"] for p in procs)/len(procs)
    avg_turn = sum(p["turnaround"] for p in procs)/len(procs)
    print(f"\nAverage waiting time: {avg_wait:.2f}")
    print(f"Average turnaround time: {avg_turn:.2f}")

if __name__ == "__main__":
    procs, q = read_processes()
    res = round_robin(procs, q)
    print_table(res)
```

Output:



```
┌──(anjali㉿Anjali)-[~]
└─$ nano rr.py

┌──(anjali㉿Anjali)-[~]
└─$ python3 rr.py
Number of processes: 3
Enter time quantum: 2
Process 1 (name arrival burst) [e.g. P1 0 5]: P1 0 7
Process 2 (name arrival burst) [e.g. P2 0 5]: P2 1 5
Process 3 (name arrival burst) [e.g. P3 0 5]: P3 2 3

Process  Arrival  Burst    Completion   Waiting
P3       2        3        11           6
P2       1        5        14           8
P1       0        7        15           8

Average waiting time: 7.33
Average turnaround time: 12.33
```