**Intelligent CPU Scheduler Simulator**

**A PROJECT REPORT**

***Submitted by***

**ADARSH KEJRIWAL**

**SHOBHEET SHARMA**

***in partial fulfillment for the award of the degree***

***of***

**BACHELOR OF TECHNOLOGY**

**IN**

**COMPUTER SCIENCE AND ENGINEERING**



**Lovely Professional University, Punjab**

**12312119**

**12313187**

**1. Project Overview**

In this undertaking, we've got carried out and analyzed 4 fundamental CPU Scheduling Algorithms normally used in Operating Systems:

* First Come First Serve (FCFS)
* Shortest Job First (SJF) - Non-Preemptive
* Priority Scheduling - Non-Preemptive
* Round Robin (RR)

The objective of this undertaking is to simulate how an Operating System schedules a couple of procedures in a multi-tasking environment. Each algorithm has its unique technique to dealing with CPU time among diverse procedures, and this project highlights the operational common sense at the back of every of them.

Through this venture, we purpose to:

* Understand the operating mechanisms of every scheduling strategy.
* Learn the way to calculate Completion Time (CT), Turnaround Time (TAT), and Waiting Time (WT) for each technique.
* Compare how exclusive algorithms impact procedure execution order and machine overall performance.
* Visualize the execution timeline the usage of Gantt Charts (for relevant algorithms like FCFS and RR).

This venture is implemented using the Python programming language, which permits clear representation of good judgment, clean code structuring, and flexible records control using dictionaries and lists. Ultimately, the task enables bridge the distance among theoretical OS scheduling standards and their realistic implementation, forming a strong basis for further studies in Operating Systems.

**2. Module**

To ensure clarity, the project is divided into the following functional modules, each module with handling a specific schedule:

* Modul 1: FCFS (first to the mill): -

This module simulates the FCFS algorithm where procedures are performed in order after their arrival.

* Main phase:

Sort procedures after arrival\_ time

For each process:

Count Start\_time , Perfection\_time , Touring\_time , Waitt\_Time

Create a Gantt Chart for Visual Representation

* Modul 2: SJF (shortest work first -On-pre -reserved): -

This module handles process execution based on the lowest burst\_time, and selects from available (arrival) procedures.

Main phase:

Choose the process of the smallest burst\_time among them at each time

Calculate planning matrix such as FCFS

* Modul 3: Priority Planning

This module plan processes on the basis of prescribed preferences (low number = high priority).

Main phase:

1. Choose one with the highest priority in the upcoming processes
2. Handle binding of arrival time
3. Calculate and save all time parameters

* Modul 4: Round Robin Scheduling: -

This module uses the RR algorithm using a certain time quantity, so that the execution of the processes.

Main phase:

1. Use a queue to rotate procedures
2. Track the remaining outbreak of each process
3. Handle partial workmanship and stand in line again
4. Product measurements and simulates the process transfer

Each module is written independently, making it easier to troubleshoot, test and increase the code. All modules follow a smooth data structure and output format for a simple comparison of display matrix in algorithms.

**3. Efficiency**

The project provides a clean, educational simulation of the four large CPU planning algorithms. The most important functionality is as follows:

1 User input management: -

Accepts a list of procedures with:

* Process ID (PID)
* Arrival time (arrival\_time)
* Burst\_time
* Priority (only for priority planning)

2. Algorithm simulation: -

Each planning performs:

* Process selection logic: FCFS, SJF, priority or round Robin based on rules
* CPU -Execution simulation: Time spores and administer CPU use
* Gant Chart Representation: (FCF's current) for visual understanding of execution order

3. Metric calculation: -

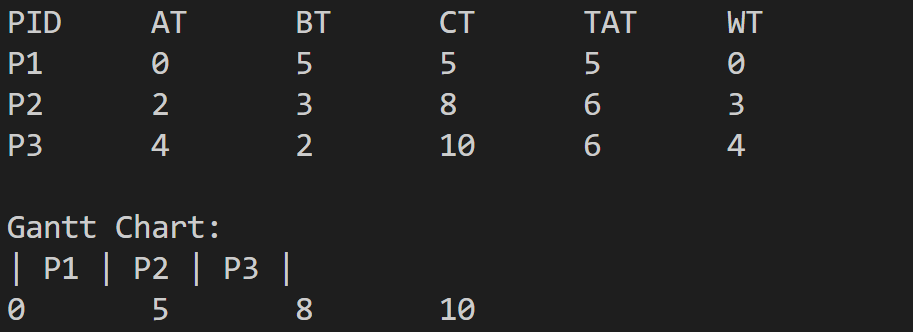
Automatically calculates for each process:

* Completion -time (CT)
* Turnaround Time (TAT) = CT - Arrival time
* Waiting time (WT) = TAT - BURST TIME
* Start time (for internal use)

These matrices help to compare the efficiency and justice of different algorithms.

4. Console -based output: -

Each algorithm shows a clean table:



The GANT diagram (FCFS) also prints for better visual learning.

Modular and expandable design: -

Easily add new algorithms such as preemptive SJF or multi -level queue.

Independent modules can be integrated into the GUI or online simulator in the future.

**4. Used technology**

The project was developed using light, available technologies to ensure simple understanding for students and teachers. The distribution of all equipment and technologies used below is:

1. Programming language: -

* Python

Logic -based planning of early friendly and powerful language is ideal for simulating algorithms.

1. libraries and equipment: -

No external libraries require

The entire project is written with pure pythons, which are easily performed in any basic python environment

1. GitHub:-

used for:

* Version control - Tracking all the changes in the code base.
* Collaboration - if the team's contribution is extended to include.
* Showing - Public portfolio performance for educational or location purposes.

1. Vs code

**6. Revision Tracking on GitHub: -**

* **Repository Name: OS-Process-Scheduling-Algorithms**
* **GitHub Link:** [**https://github.com/username/OS-Process-Scheduling-Algorithms**](https://github.com/username/OS-Process-Scheduling-Algorithms)

**7. Conclusion and future scope**

Conclusion

The process planning algorithm simulator was successfully developed to demonstrate the internal feature of the popular CPU planning strategies: Including:

* First come and first serve (FCFS)
* The shortest job first (SJF)
* Priority Scheduling
* Round Robin (RR)

The project provided an understanding of how the operating systems handle many processes, plan them effectively and manage the use of CPU. Code accurately required process matrix as:

* Completion Time (CT)
* Turnaround Time (TAT)
* Waiting time (WT)

In addition, the inclusion of the Gantt diagram in the imagination of the process's performance order to a large extent improved.

**Future scope**

While this simulator is designed for educational and performance purposes, many campaigns can be used to make it more advanced and interactive:

* Graphic user interface (GUI):

Creating a web or stationary user interface can make the interaction more comfortable and attractive to users.

* Preemptive version:

At the moment, algorithms such as SJF and priority are used in non-PRE-PRE form. Adding the preemptive variants will improve the simulation of the real world.

* Live Simulation Visualization:

A time -based animation or bar simulation can be added to show how the CPU performs procedures over time.

* Performance Comparison Chart:

Performing the comparative chart will help to analyze the efficiency between the algorithms that add to the calculations (average WT, TAT, CPU passive time).7. Conclusion and future scope

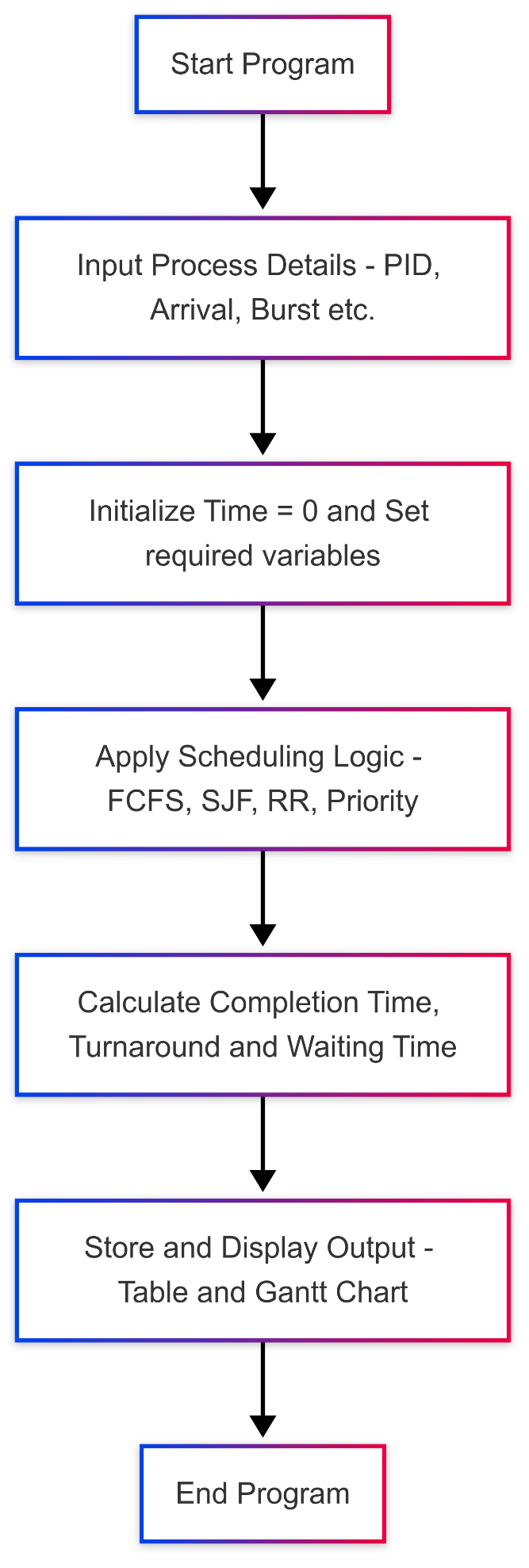
**7. Conclusion:**

The project aims to simulate and implement the most important CPU process planning algorithms that make back legs for modern operating systems. By creating a modular code for FCF, SJF, Priority Scheduling and Round Robin, we could find out the argument behind an OS plan and processes.

* Through this simulator we gained valuable insight:
* The importance of planning decisions in improving system efficiency.
* How each algorithm performs under different arrival and time conditions.
* By working on this project, we not only strengthen our ideological understanding of the planning of the OS level, but also improved our coding, problem solving and troubleshooting skills.

This simulator acts as an excellent basis to pursue, and it is a strong representation of how core theoretical concepts can be brought into life through practical programming.

**8. Flow Diagram:**

****

**9. References:**

* Silberschatz, Galvin, and Gagne – *Operating System Concepts*, 9th Edition
* **Python Documentation** – *Built-in Functions and Data Structures*
* **GitHub** – *Version Control & Collaborative Development*

**Appendix:**

**A. AI-Generated Project Elaboration/Breakdown Report**

This project simulates and evaluates different CPU Scheduling Algorithms—a core aspect of Operating Systems—by implementing the following major types:

**1. First Come First Serve (FCFS)**

* **Concept**: Non-preemptive algorithm where processes are scheduled in the order they arrive.
* **Key Logic**:
  + Sort processes by arrival time.
  + Start each process when the CPU becomes free.
  + Calculate Completion Time (CT), Turnaround Time (TAT), and Waiting Time (WT).

**2. Shortest Job First (SJF)**

* **Concept**: Non-preemptive algorithm that selects the process with the shortest burst time among the arrived processes.
* **Key Logic**:
  + At each moment, pick the job with the minimum burst time among the available ones.
  + Manage CPU idle time if no process has arrived.

**3. Priority Scheduling**

* **Concept**: Non-preemptive priority-based scheduling where lower numerical value indicates higher priority.
* **Key Logic**:
  + Among available processes at any given time, pick the one with the highest priority.
  + Handle tie-breakers using arrival time.

**4. Round Robin (RR)**

* **Concept**: Preemptive scheduling using time quantum, ensuring fair sharing of CPU.
* **Key Logic**:
  + Each process is given a fixed time slice in a cyclic order.
  + If a process doesn’t finish in one quantum, it’s pushed back to the queue.

**Learning Objectives & Takeaways:**

* Understand how scheduling affects CPU utilization, turnaround, and waiting times.
* Apply theoretical concepts practically using Python.
* Visualize process execution through Gantt charts (for FCFS).
* Analyze performance metrics for different scenarios**.**

**B. Problem Statement**

Design and implement a CPU Scheduler Simulator that executes and compares four major scheduling algorithms:  
FCFS, SJF (Non-Preemptive), Priority (Non-Preemptive), and Round Robin (Preemptive).

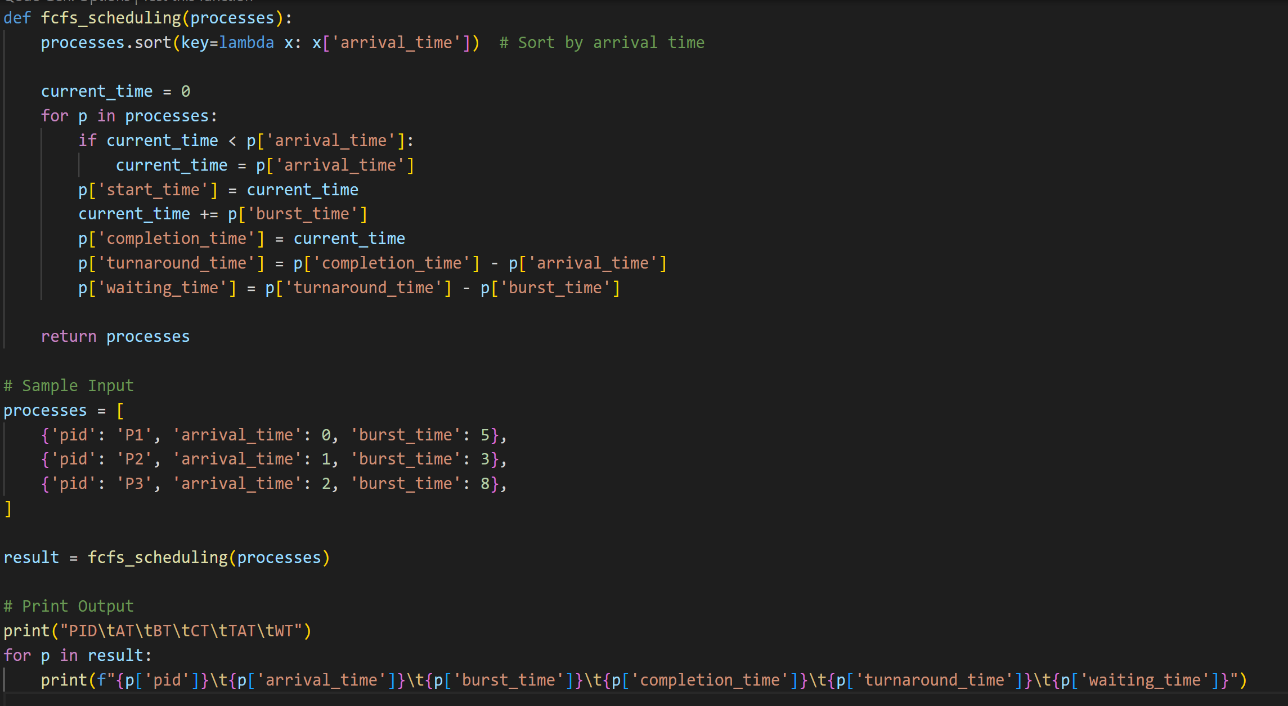
The simulator should take process details (PID, Arrival Time, Burst Time, [Priority, if needed]) and display:

* Completion Time
* Turnaround Time
* Waiting Time

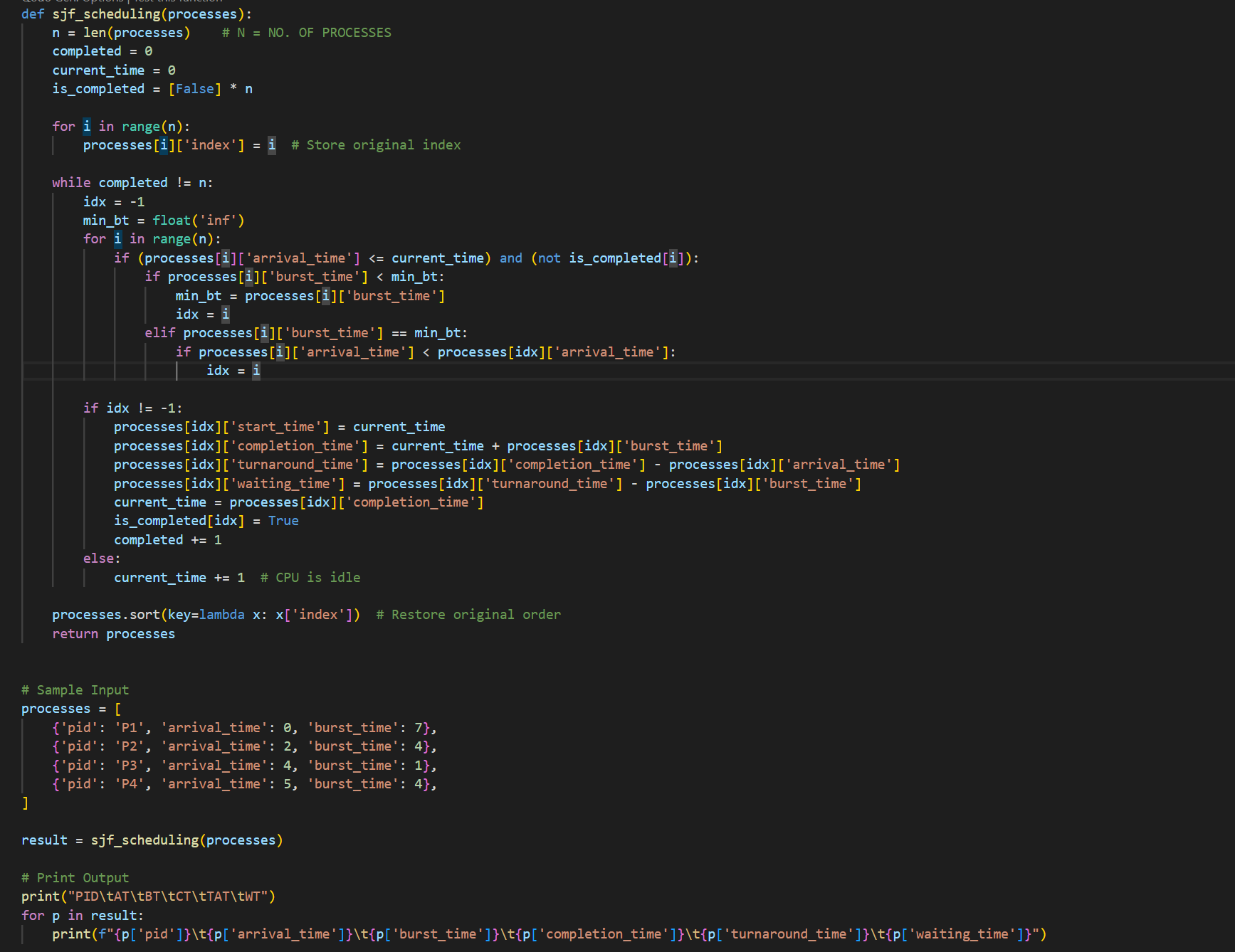
In addition, it should optionally generate a Gantt Chart (for FCFS) and demonstrate output readability through structured tabular representation.

**C. Solution/Code:**

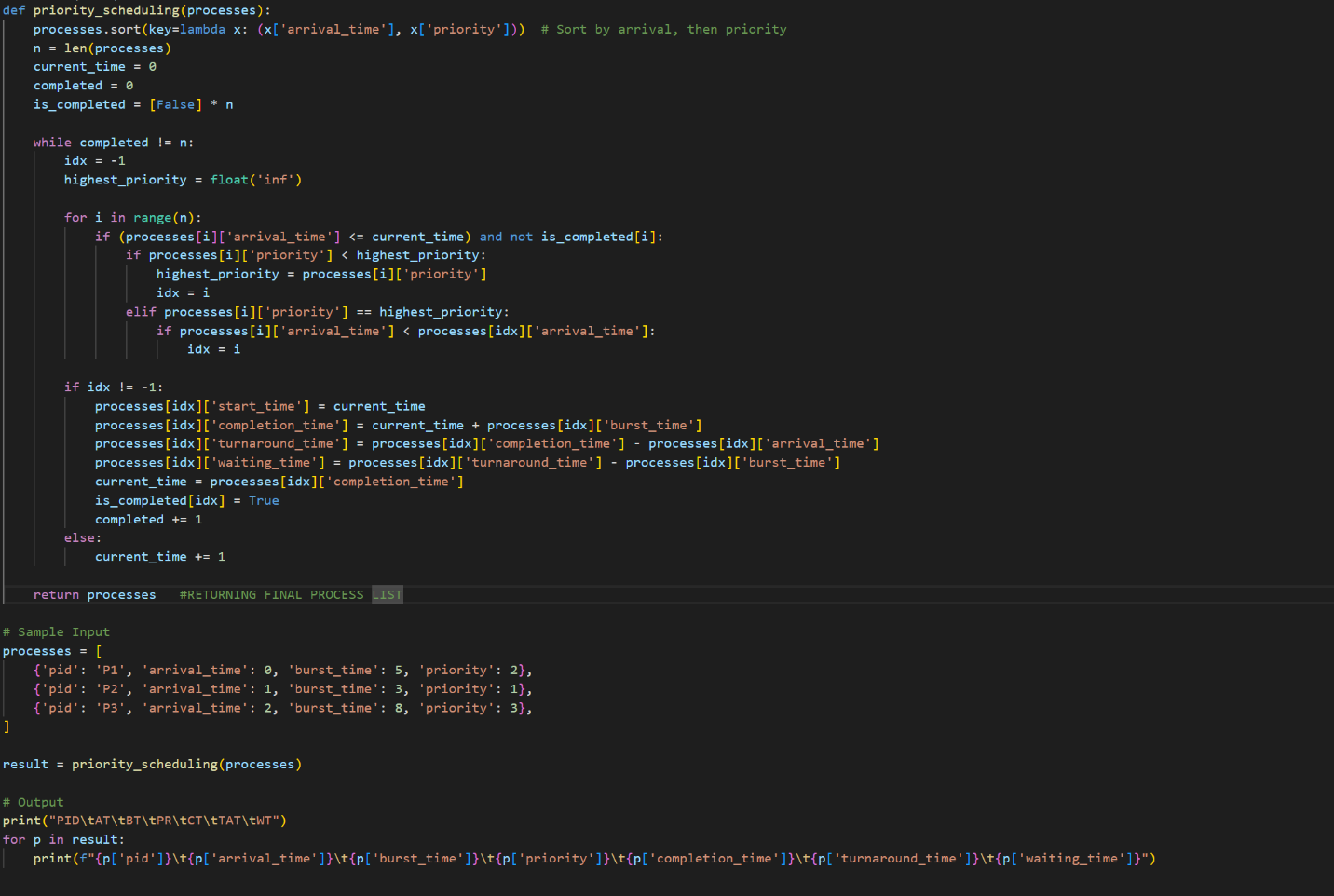
* FCFS with Gantt Chart: -

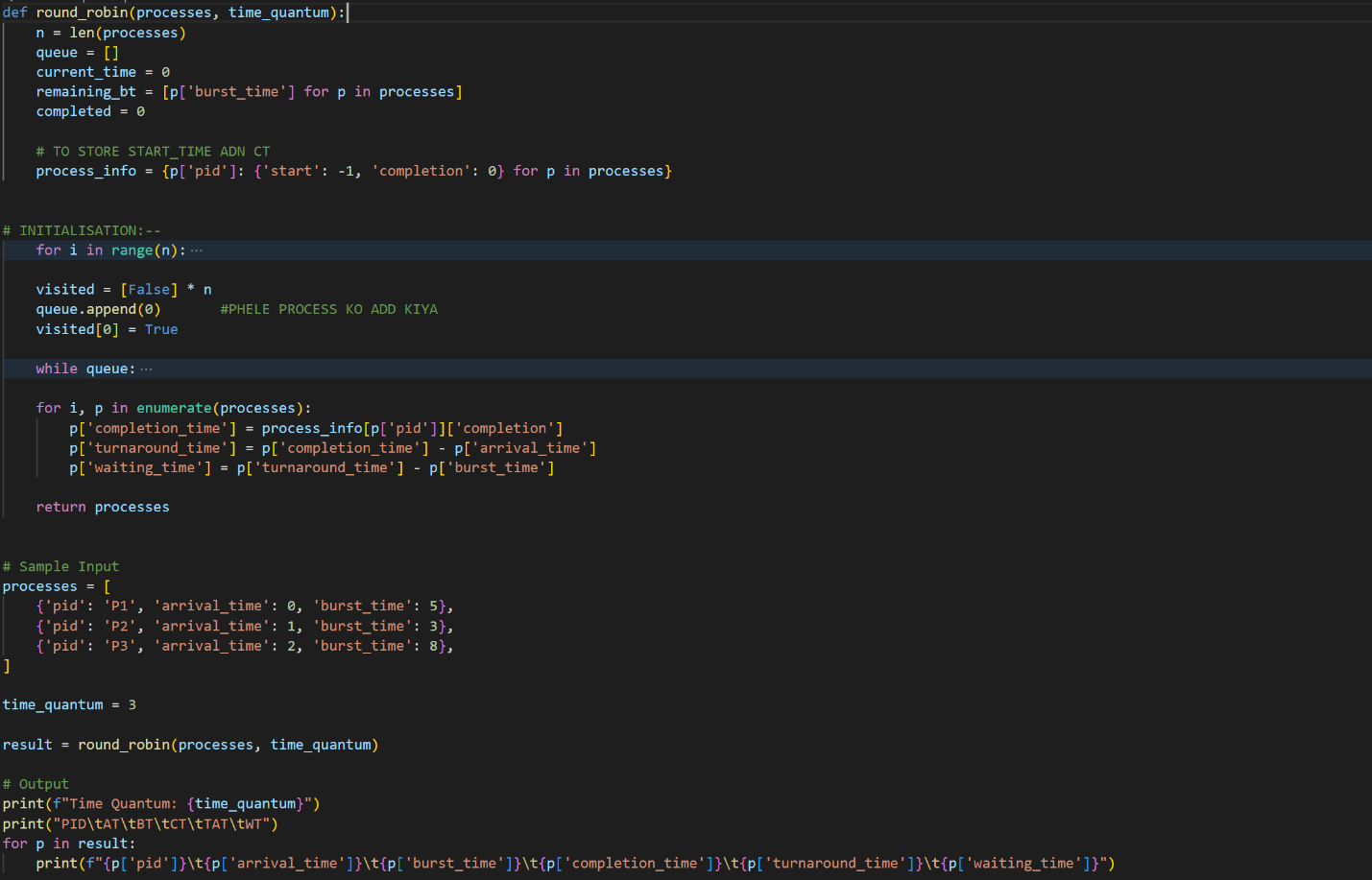


* SJF (non-preemptive)



* Priority Scheduling



* Round Robin Scheduling  
  
  + Fcfs with gantt chart:

