

**The Superior University**

**📌 Project Title**

**HEALTH CARE APPOINTMENT (SJF)**

**👥 Group Members**

* SYEDA ESHWA SULEMAN

**📂 GitHub Repository**

**GitHub Repository Link:**

**https://github.com/eshwa-1/healthcare--appointment-schedular-sjf.git**

**🔧 Scheduling Algorithm Implemented**

✅ Tick the scheduling algorithm your group implemented:

* FCFS (First Come First Serve)
* ✅ SJF (Shortest Job First – Non-Preemptive)
* SJF (Preemptive)
* Round Robin

**📄 Project Description**

### ****Brief Explanation of the Project:****

**1. Problem the Project Solves:**

The project addresses the problem of scheduling patient appointments efficiently in a healthcare setting.

It uses the **Shortest Job First (SJF) Non-Preemptive** algorithm to ensure that patients with shorter consultation times are attended to first, reducing waiting times and improving the overall patient flow in a clinic.

The project also accounts for idle periods when no patients are available for consultation.

**2. Inputs Required:**

**Arrival Time**: The time when the patient arrives at the clinic (in minutes).

**Burst Time (Consultation Time)**: The duration of the patient's consultation (in minutes).

The inputs are provided by the user when adding patient details.

**3. Outputs Generated:**

**Appointment Schedule**: A table showing:

* Patient's name
* Arrival time
* Consultation time
* Start and finish time of consultation
* Waiting time (how long the patient waited before being attended to)
* Turnaround time (total time from arrival to the end of consultation)

**Gantt Chart**: A visual timeline showing the sequence of consultations.

**Stairs View**: A representation of the timeline where each patient's consultation is shown visually with the emoji "🧍".

**Statistics**: Calculations for average waiting time, average turnaround time, and idle periods.

**Final Report**: A summary including the total number of patients, total idle periods, and other statistics.

**4. How the Algorithm is Implemented:**

**Input Phase**: The user is prompted to input details for each patient (arrival time and consultation time).

**Scheduling Phase (SJF Non-Preemptive)**:

The patients are sorted based on arrival time.

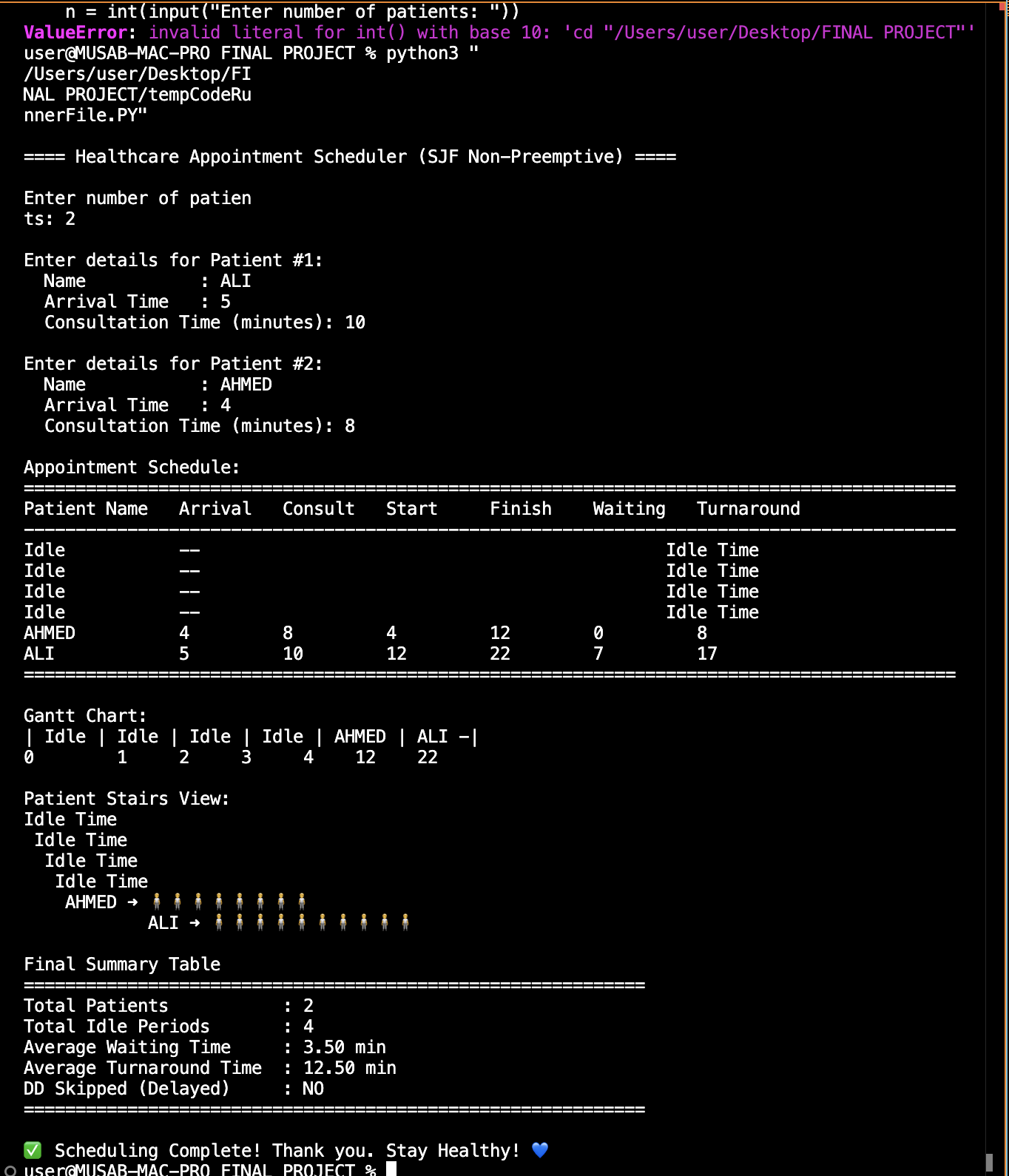
For each time unit, the algorithm checks which patients have arrived and selects the patient with the shortest consultation time (SJF).

The selected patient is attended to, and their consultation starts. Once done, the next shortest job is picked, and this continues until all patients are served.

If no patient is available at a given time, the system accounts for idle time and moves to the next available patient.

**Output Phase**: The appointment details are displayed in the table format, and the system generates a Gantt chart and stairs view. The system calculates and displays various statistics, followed by a final summary of the scheduling process.

**📸 Output Screenshots**

****

**🧠 Code Structure & Explanation**

### ****Explanation of How the Code is Organized:****

#### ****1. Functions Used:****

The code is organized into various functions to handle different parts of the process, ensuring modularity and clarity. Here’s a breakdown:

get\_patient\_input():

This function collects input from the user about patients. It asks for the number of patients and then prompts for the details of each patient, such as their name, arrival time, and consultation time (burst time).

It stores these details in a list of dictionaries (patients), which are returned to be processed later.

schedule\_sjf\_non\_preemptive(patients):

This is the core function that implements the **Shortest Job First (SJF) Non-Preemptive Scheduling Algorithm**.

It iterates through the list of patients, selects the patient with the shortest consultation time (who has arrived), and schedules their appointment.

The function calculates the start time, finish time, waiting time, and turnaround time for each patient and stores this information in a scheduled list.

display\_table(patients):

This function formats and displays the appointment schedule in a tabular format.

It outputs the patient name, arrival time, consultation time, start time, finish time, waiting time, and turnaround time.

display\_gantt\_chart(patients):

This function generates and displays a **Gantt Chart** for the scheduled appointments.

It shows the timeline of patient consultations, using text-based formatting for simplicity.

display\_stairs(patients):

The **stairs view** function displays the timeline of patient consultations using a visual representation of steps (using emojis like "🧍").

It simulates how patients’ consultations overlap in time.

calculate\_statistics(patients):

This function computes statistics, such as the **average waiting time** and **average turnaround time** for the patients.

It also counts the number of idle periods (when no patients are being served).

check\_dd\_avoided(patients):

This checks if any patient with the name "dd" was skipped or delayed in the scheduling process. This is a check for any unusual situations (such as missed patients).

display\_final\_report\_table(patients, avg\_waiting, avg\_turnaround, idle\_count):

This function generates a final report displaying the **total number of patients**, **total idle periods**, **average waiting time**, and **average turnaround time**.

It also shows whether a patient "dd" was delayed or skipped.

#### ****2. Core Logic of the Scheduling Algorithm:****

The **SJF Non-Preemptive Scheduling Algorithm** is the heart of the project. Here’s how it works:

**Patient Sorting**: The patients are sorted by their **arrival time** and then by **consultation time** (shortest burst time).

**Selection Process**: The scheduling function loops through the list of patients, checking if any patient has arrived at the current time. If a patient is available, the one with the shortest consultation time is selected for the next appointment.

**Idle Time Handling**: If no patient is available to consult (i.e., all have later arrival times), the system simulates idle time by adding an "Idle" slot to the schedule.

**Waiting and Turnaround Calculation**: For each patient, the start time and finish time are calculated, and the **waiting time** (time spent waiting before starting the consultation) and **turnaround time** (total time from arrival to consultation finish) are computed and stored.

#### ****3. External Libraries Used:****

**No external libraries**: The code does not use any external libraries like matplotlib or tabulate for visualization or formatting. Instead, it relies on basic **Python standard libraries** such as input() for gathering user inputs and print() for displaying the results.

**Basic Text Formatting**: The tables and Gantt charts are generated using simple text formatting (spaces, dashes, and vertical bars) to create a readable display in the terminal. This makes it possible to run and test the program without the need for additional dependencies.

#### ****Summary of How the Code Works:****

**Patient Input**: The user is prompted to enter patient details such as arrival time and consultation duration.

**Scheduling**: The SJF non-preemptive algorithm is applied to determine the order of consultations.

**Scheduling Outputs**: The system then displays:

A detailed table of patient appointments.

A Gantt chart of the scheduling timeline.

A "stairs view" showing patient consultations over time.

**Statistics**: Finally, the system calculates and displays performance metrics such as average waiting time, average turnaround time, and whether any patients (like "dd") were delayed.

This design ensures that the project remains simple and efficient, focused on the SJF non-preemptive scheduling without relying on complex libraries or structures.

|  |  |
| --- | --- |
|  |  |
|  |  |
|  |  |

**🛠️ Challenges Faced**

#### ****1. Incorrect Calculation of Waiting and Turnaround Times:****

**Challenge**: One of the first challenges we faced was ensuring that the waiting time and turnaround time calculations were accurate, especially in the case of patients with varying arrival times and consultation durations. Initially, there was confusion over how to calculate the waiting time when some patients arrived later than others or if there were idle periods.

**Solution**: After careful review, we determined that the waiting time should be calculated as the difference between the start time and the arrival time for each patient. Similarly, the turnaround time was calculated as the difference between the finish time and the arrival time. We also ensured that idle time periods (when no patient is available for consultation) were correctly handled by adding "Idle" entries to the schedule.

#### ****2. Gantt Chart Visualization and Display:****

**Challenge**: One of the more complex tasks was drawing the Gantt chart, which required a clear representation of patient schedules with start and finish times. Initially, the chart was cluttered and hard to read, as it needed to represent both the timeline and the individual patient names in a formatted manner

**Solution**: We used a simpler format for the Gantt chart by creating a textual representation that included time intervals and patient names. This allowed us to display the Gantt chart in a concise, readable manner within the terminal output. We adjusted the chart’s layout and spacing to make it more clear, ensuring that each patient’s time slot was clearly represented with a horizontal line and time markers.

#### ****3. Handling Idle Time and Delayed Patients ("DD" Skipped Check):****

**Challenge**: Handling cases where no patients are available (idle time) was another challenge. For example, if there were gaps between patient arrivals, the scheduler would need to insert "idle" periods in the schedule. We also wanted to track if a patient named "dd" was delayed or skipped, as this was an additional feature requested.

**Solution**: We resolved this by adding a check for idle periods when no patients were available at the current time. When this happened, we added an "Idle" entry to the schedule. To track the "dd" patient, we created a custom check that flagged whether a patient named "dd" was skipped during the scheduling process. This allowed us to ensure that both idle times and special cases (like the "dd" patient) were properly handled in the final output