

system SCheduling algorithms

CST1500 Computer Systems Architecture and Operating Systems

TUTOR: VISHAM RAMSURRUN

BSC Computer Science & Systems Engineering



DENZEL GRISON (Student ID: M00973156)

MARK KIPOKOLA (Student ID: M00975543)

NIGEL GAMBIZA (Student ID: M00951628)

Table of Contents

[Introduction 3](#_Toc184963620)

[First Come First Served (FCFS) – Mark Kipokola (M00975543) 4](#_Toc184963621)

[Introduction 4](#_Toc184963622)

[Key Features 5](#_Toc184963623)

[How It Works 5](#_Toc184963624)

[Advantages 7](#_Toc184963625)

[Disadvantages 7](#_Toc184963626)

[Conclusion & Reflection 7](#_Toc184963627)

[Shortest Job First (SJF) Nigel Gambiza M00951628 8](#_Toc184963628)

[Introduction 8](#_Toc184963629)

[Characteristics of SJF Scheduling 10](#_Toc184963630)

[Advantages of SJF 10](#_Toc184963631)

[Disadvantages of SJF 10](#_Toc184963632)

[Conclusion & Reflection 11](#_Toc184963633)

[Round Robin Method Denzel Grison (M00973156) 12](#_Toc184963634)

[Introduction 12](#_Toc184963635)

[Advantages 12](#_Toc184963636)

[Disadvantages 12](#_Toc184963637)

[Code 13](#_Toc184963638)

[Output 14](#_Toc184963639)

[Code explanation: 15](#_Toc184963640)

[Conclusion & Reflection 15](#_Toc184963641)

# Introduction

As mentioned on the Cover Page, this group project was completed by Denzel Grison (Student ID: M00973156), Nigel Gambiza (Student ID: M00951628), and Mark Kipokola (Student ID: M00975543). In this collaborative effort, we chose to address three different system scheduling algorithms, with each member focusing on one. Mark Kipokola worked on the **First Come First Serve scheduling system**, while Nigel Gambiza concentrated on the **Shortest Job First algorithm.** Denzel Grison took on the **Round Robin algorithm system.** The details of how each algorithm was tackled by each team member are provided below.

## First Come First Served (FCFS) – Mark Kipokola (M00975543)

### Introduction

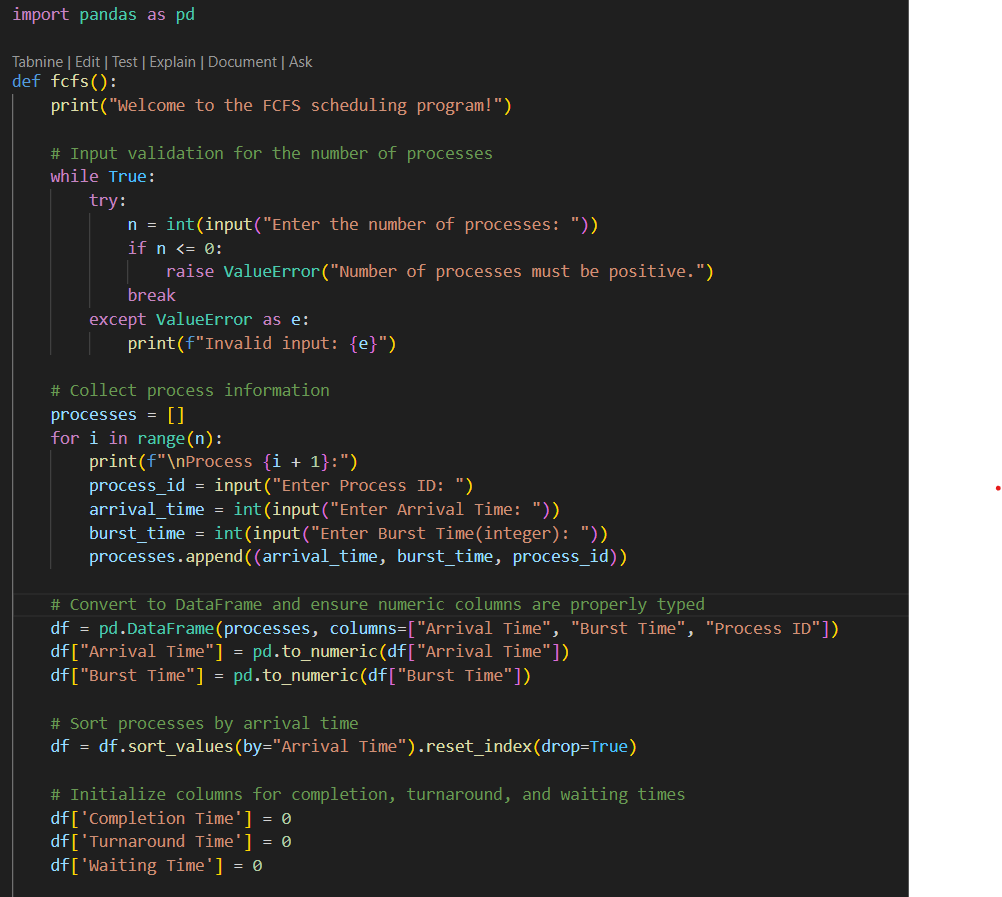
The First Come First Served (FCFS) algorithm is one of the simplest scheduling algorithms used in operating systems and process management. It is a non-pre-emptive scheduling method, meaning that once a process starts execution, it runs until completion without interruption.

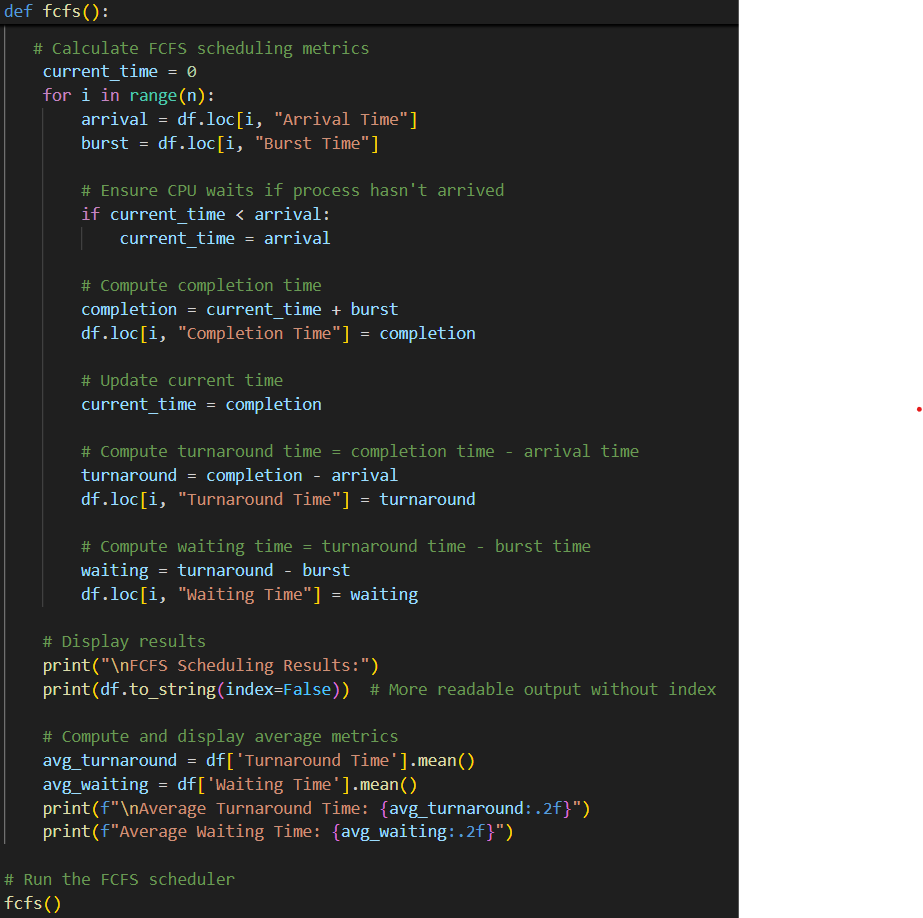
### Key Features

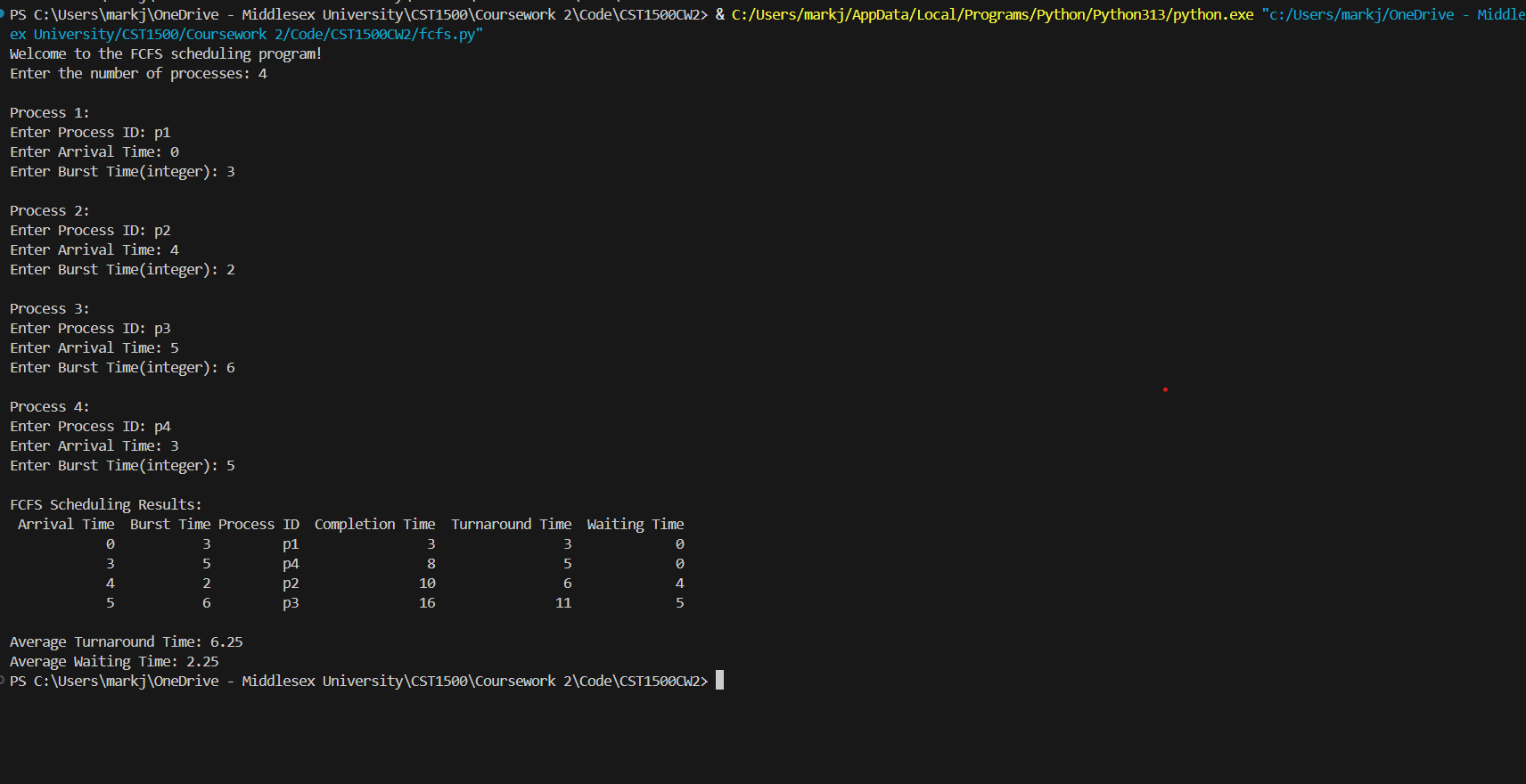
FCFS is queue-based, meaning processes are scheduled in the order of arrival in the ready queue. The process that arrives first gets executed first. Once a process starts, it runs to completion without interruption making it non-pre-emptive. Each process is treated equally based on its arrival time, but it can lead to inefficiencies like long wait times for short tasks if a long process is scheduled first. This kind of scheduling is easy to implement using a queue structure.

### How It Works

When processes arrive in the ready queue, they are lined up based on their arrival times.







The process at the front of the queue is executed first. The next process begins execution only after the current process is completed. At completion, the process’s turnaround time, waiting time, and completion time are calculated.

### Advantages

It is simple and easy to implement and understand. The order of the execution is straightforward making it predictable.

### Disadvantages

Possibility of a convoy effect meaning long processes can delay the execution of shorter ones, leading to high waiting times for short jobs. It is not optimal as it does not minimize average waiting time or turnaround time compared to other algorithms like Shortest Job First (SJF) or Round Robin (RR).

### Conclusion & Reflection

FCFS is best suited for batch systems where simplicity is more important than efficiency. However, for modern multitasking operating systems, its inefficiencies often make it less desirable. Researching and creating this algorithm has taught me the importance of a computer system being able to efficiently manage tasks to reduce the possibility of a system crash. There are multiple ways to manage tasks and the first come first served algorithm is simply one of them. I see it as a straightforward algorithm to implement, however, because of its simplicity, I also see it would be difficult to use it in more complex systems which require efficiency. The FCFS algorithm does not discriminate and instead treats every task fairly. It does not choose the task with the shortest operation time; instead, it chooses one that comes first. Therefore, if the first task to come is a task with a long operation time, the operation after it with a shorter operation time would instead have an unnecessarily long waiting time. Me and my teammates used Github to update each on our codes’ progress by allowing each other to see the code and to crosscheck it themselves to see if everything was working properly. This ensured that we were kept up to date with not only our own codes but each other’s as well. Definitely recommend using Github when in a group. Working on this algorithm has also taught me the importance of the tasks arrival time and burst time. Both of these are important factors to consider when choosing which task, the CPU needs to do first to ensure efficiency.

## Shortest Job First (SJF) Nigel Gambiza M00951628

### Introduction

Shortest job first is a CPU scheduling algorithm that allocates the CPU to the process with the shortest execution time. It’s a non-preemptive algorithm which means that once a process starts running it cannot be stopped until it is finished or blocked.

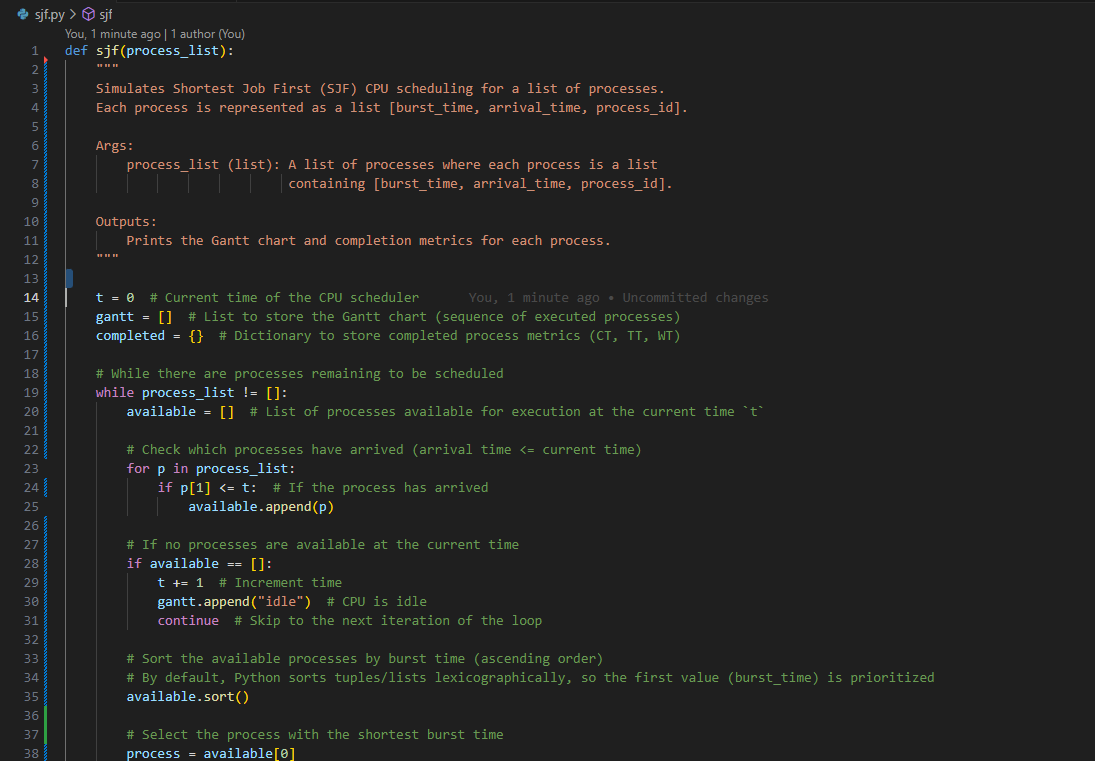


Fig.1

Above is the first part of the code in python with comments and an explanation what the program is supposed to do.

Fig.2 More of the code with comments.

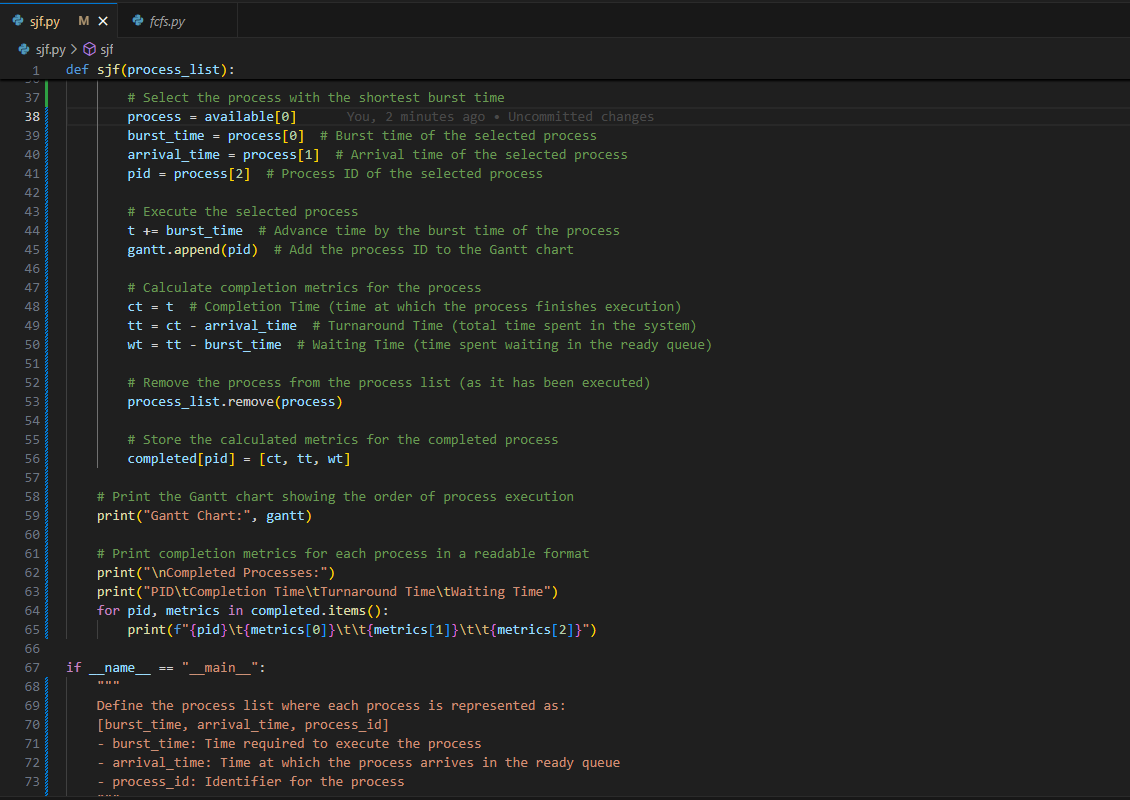


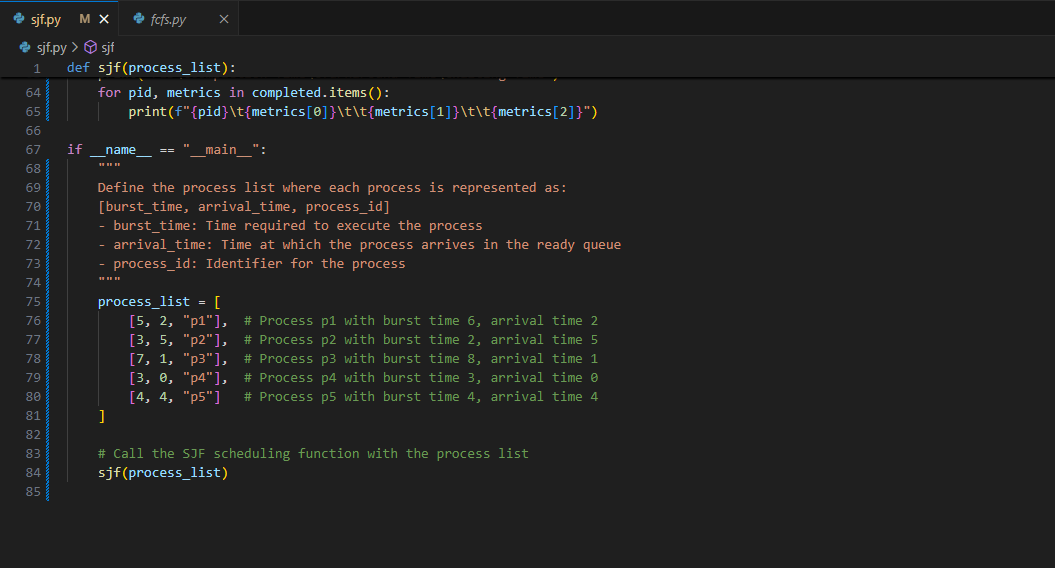
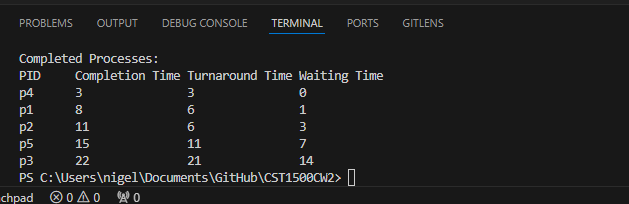
Fig.3 the process list this has the minimum burst times from the question in process 1,2 and 3

Fig.4 output



### Characteristics of SJF Scheduling

* Associated with every job as it requires a unit of time for a job to complete.
* Helpful for batch-type processing.
* Improves process throughput as shorter jobs execute first, further reducing the turnaround time.
* Offers shorter jobs that improve job output.

### Advantages of SJF

* Used for long-term scheduling.
* Reduces average waiting time.
* Helpful for batch-type processing where runtimes are known in advance.

### Disadvantages of SJF

* It is necessary to know the job completion time beforehand as it is hard to predict.
* Used for long-term scheduling in a batch system.
* Can’t implement this algorithm for CPU scheduling for the short term as we can’t predict the length of the upcoming CPU burst.

### Conclusion & Reflection

Writing this project was interesting for me because it taught me the importance of understanding process scheduling in operating systems and how decisions like this impact CPU efficiency. When I was figuring out how to write this code I overcame a few challenges, the biggest one handling idle CPU time. Incorporating the handle cases where no processes were available for execution, with more research I came across the “Idle” state, which I included in my code. I also learned how sorting helps prioritize processes dynamically and how to calculate important metrics like Completion time (CT), Turnaround time (TT), and Waiting time (WT) for each process. Overall, I enjoyed doing this algorithm it strengthened my knowledge in algorithms and definitely in writing code in python.

## Round Robin Method Denzel Grison (M00973156)

### Introduction

The Round Robin method is a system scheduling algorithm designed to enable programs to share system resources effectively by using time slices allocated to each program. Once a program utilizes its designated time slice, it is interrupted and returned to the ready queue, where it waits for its next turn to execute. This highlights the pre-emptive nature of the Round Robin method. While Round Robin scheduling offers several advantages, it also has disadvantages that can impact its effectiveness in various scenarios.

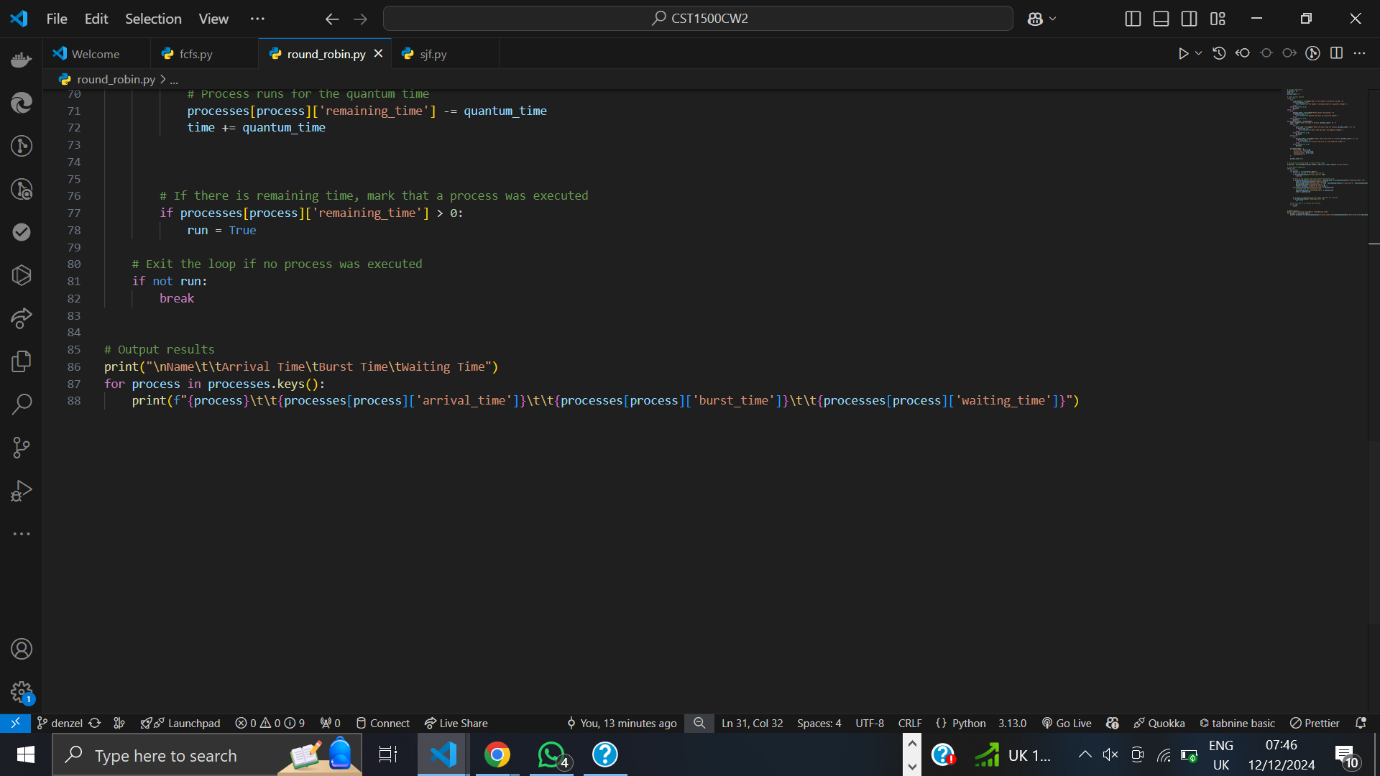
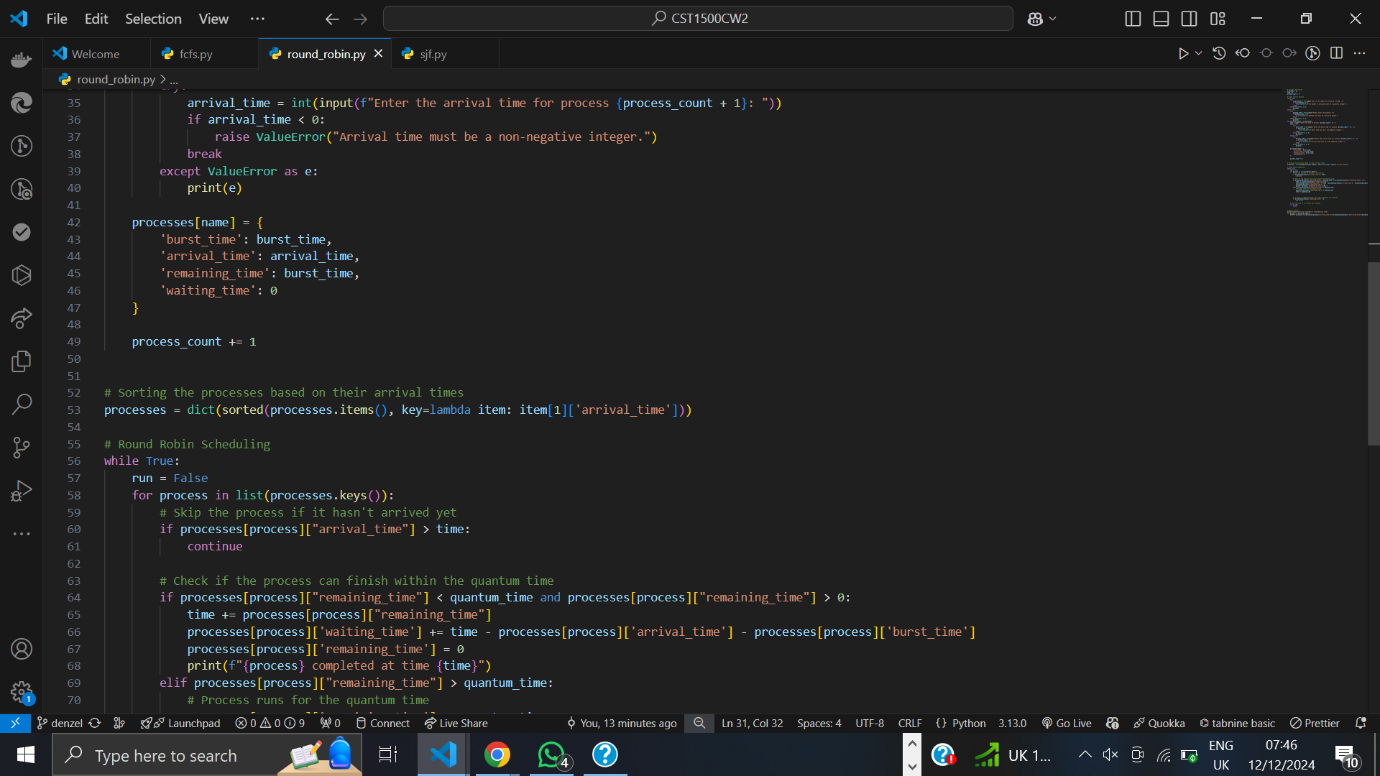
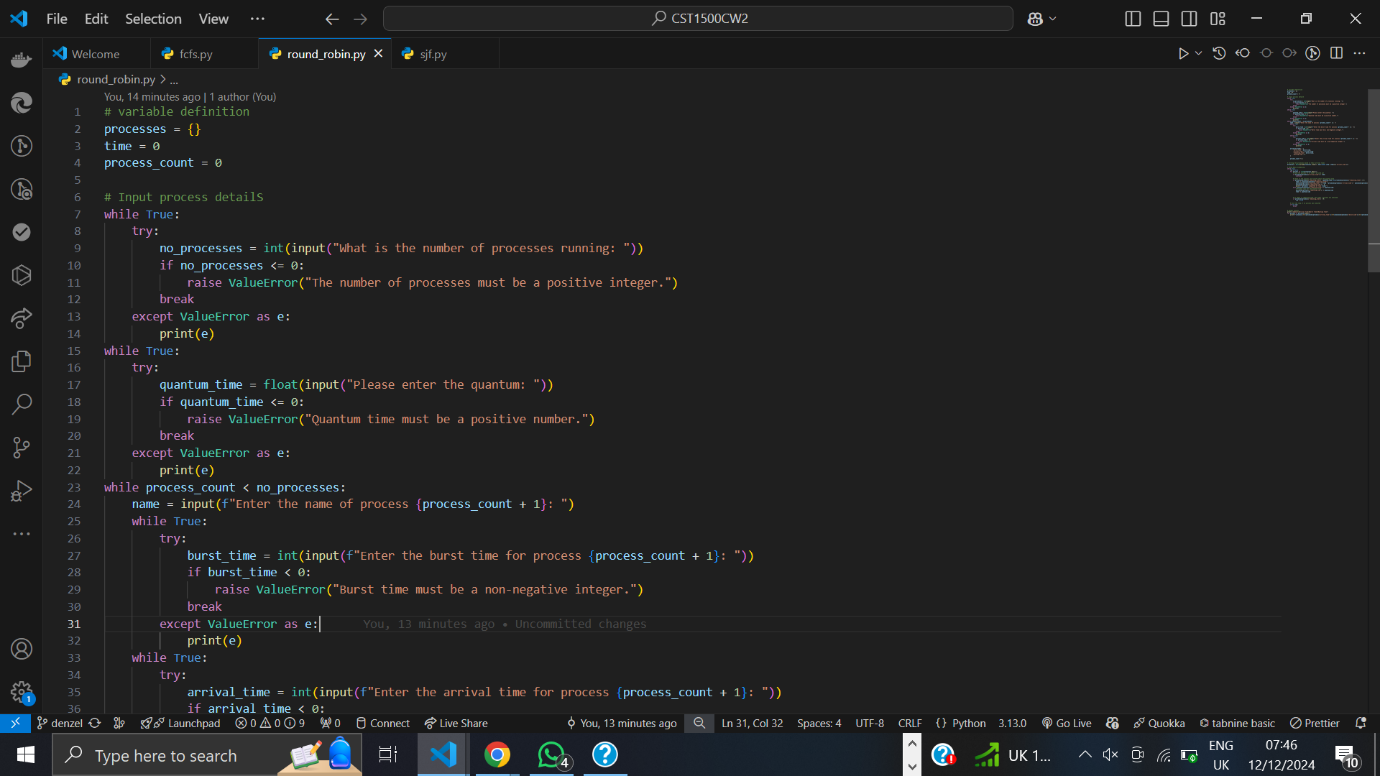
### Advantages

One of the key benefits of Round Robin scheduling is its fairness, as it ensures that each process gets an equal share of CPU time, preventing any single process from taking over resources. The algorithm is straightforward, making it easy to understand and implement, thanks to its fixed time quantum that simplifies how processes are managed. Moreover, it supports pre-emption, meaning that if a process exceeds its time quantum, the CPU can be allocated to another process. This feature is particularly useful in time-sharing systems. Round Robin also offers reasonable response times for shorter processes and reduces the risk of starvation by guaranteeing that all processes eventually receive CPU time.

### Disadvantages

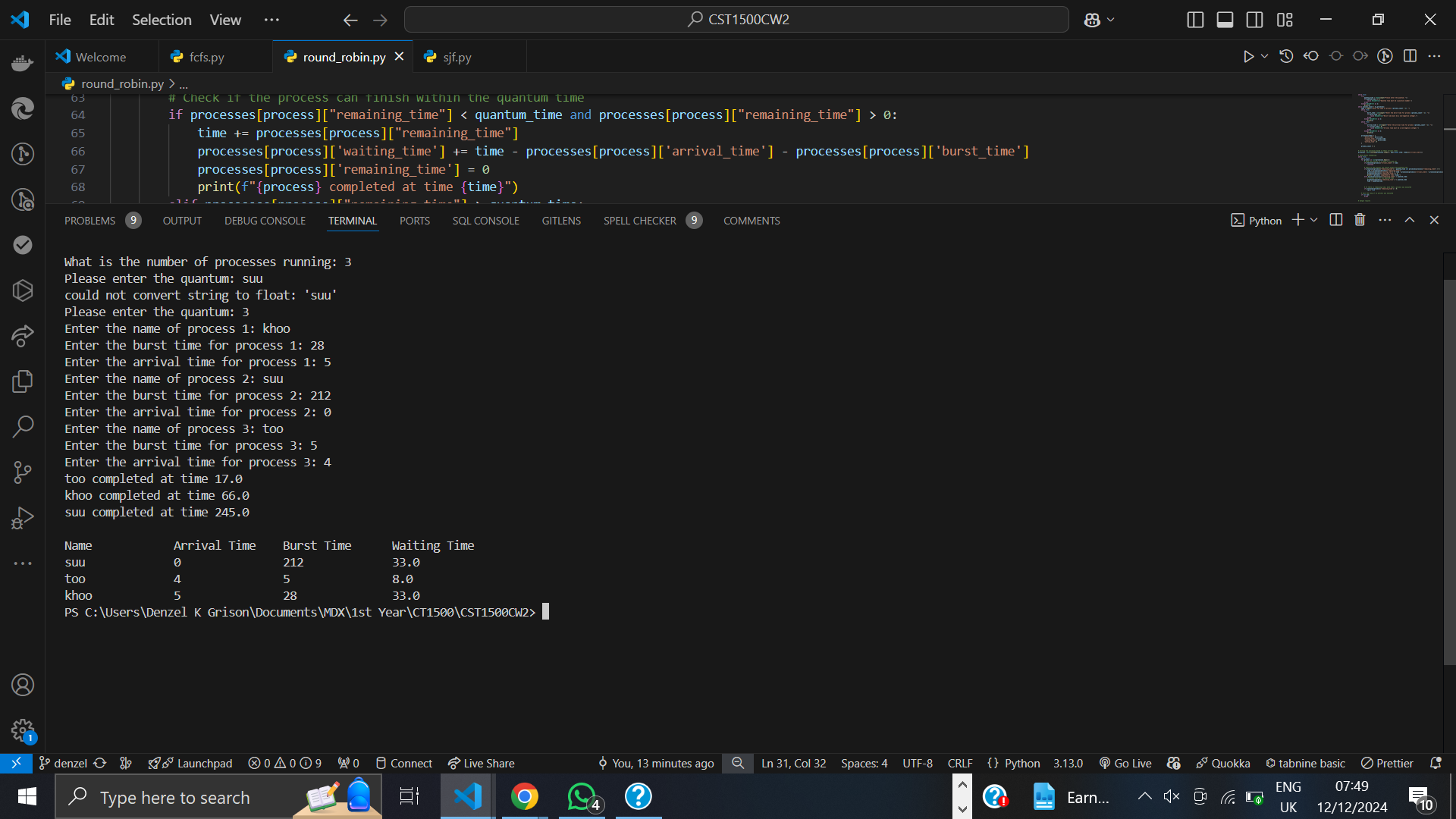
However, Round Robin scheduling does have some significant drawbacks. One major issue is that frequent context switching can lead to increased overhead, which may negatively impact overall system performance, especially when using a small-time quantum. Longer processes might face delays in completion due to the time-slicing nature of the algorithm, resulting in inefficiencies. The choice of time quantum is critical; if it’s too short, it can lead to excessive context switching, while a longer quantum might increase wait times for shorter processes. Additionally, Round Robin is not particularly well-suited for CPU-bound processes, as they may struggle to complete their tasks efficiently within the limited time slices. In summary, while Round Robin scheduling is effective in many scenarios, especially in time-sharing systems, it requires careful consideration of the types of processes and the selection of an appropriate time quantum to optimize performance.

### Code

Below the code for the Round Robin Method will be provided followed by the output:

### Output

Below is the output produced from the code above:



### Code explanation:

The code above accepts user input and prompts the user to enter the number of processes to be expected before asking the user to enter each process’s details (burst time, arrival time, name of process). The program then sorts the processes, which are stored in a dictionary, in ascending order according the arrival time. Thus, a program can also be allocated a time slice even when processing has already started. The code also includes error handling for user input, this ensures that only accurate information is obtained from the user.

### Conclusion & Reflection

This coursework enhanced my understanding of system scheduling by providing an environment for experimenting with various scheduling systems, enabling thorough testing. We utilized resources like GitHub, which helped us maintain a clear record of our progress. This platform not only improved our collaboration but also facilitated debugging as a team. GitHub served as an excellent backup solution for our code, particularly in situations where physical storage devices may fail, making it a valuable recommendation for colleagues. Working in a group fostered better teamwork and communication. Additionally, observing how others wrote their code was beneficial, promoting concise and clear coding practices. The experience allowed us to identify each other's strengths and weaknesses, aiding in mutual development in those areas. For future projects, using a cloud-based tool like Microsoft Word could be advantageous, as it supports simultaneous collaboration among multiple users on the same document. Overall, the group project was successful, with everyone actively participating in the report writing, coding, and video presentations.