**Kathmandu University**

**Department of Computer Science and Engineering**

**Dhulikhel, Kavre**

****

**A Lab Report #2**

**[ Course title: COMP 307]**

**Submitted by:**

**Bisheshwor Neupane(35)**

**Submitted to:**

**Mr. Dhiraj Shrestha**

**Department of Computer Science and Engineering**

**Submission Date:**

**August 14, 2021**

1. **Consider the Reader writer problem and provide the solutions for Race condition using peterson solutions**

Solution:

Assume that the LOAD and STORE instructions are atomic; i.e. cannot be interrupted.

The two processes share two variables:

int turn;

Boolean flag[2]

The variable turn indicated whose turn it is to enter the critical section.

The flag array is used to indicate if a process is ready to enter the critical section. flag[i] = true implies that process Pi is ready.

Algorithm for Process Pi:

do{

flag[i] = TRUE;

turn = j;

while(flag[j] && turn == j);

critical section

flag[i] = FALSE;

remainder section

}while(TRUE);

Solution to Critical-section problem using locks

do{

aquire lock

critical section

release lock

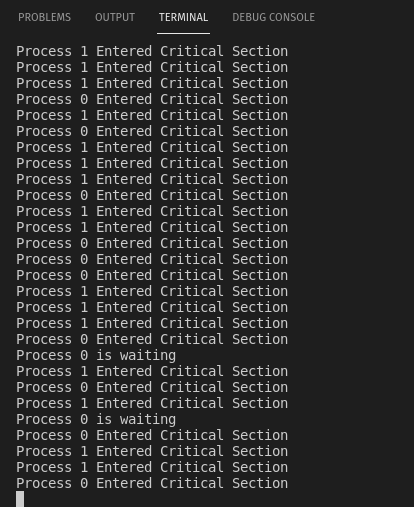
remainder section

}while(TRUE);

**Implementing it in Python:**



Output:



1. **Calculate Waiting time, Turnaround time for each process and also calculate the average Turnaround time and Average Waiting time (Assume 5 different process with different arrival time and burst time):**
   1. **First Come First Serve**
   2. **Round Robin**
   3. **Shortest Job First**
   4. **Priority**

Solution:

1. **First Come First Serve**

**Algorithm:**

1- Input the processes along with their burst time (bt).

2- Find waiting time (wt) for all processes.

3- As first process that comes need not to wait so

waiting time for process 1 will be 0 i.e. wt[0] = 0.

4- Find **waiting time** for all other processes i.e. for

process i ->

wt[i] = bt[i-1] + wt[i-1] .

5- Find **turnaround time** = waiting\_time + burst\_time

for all processes.

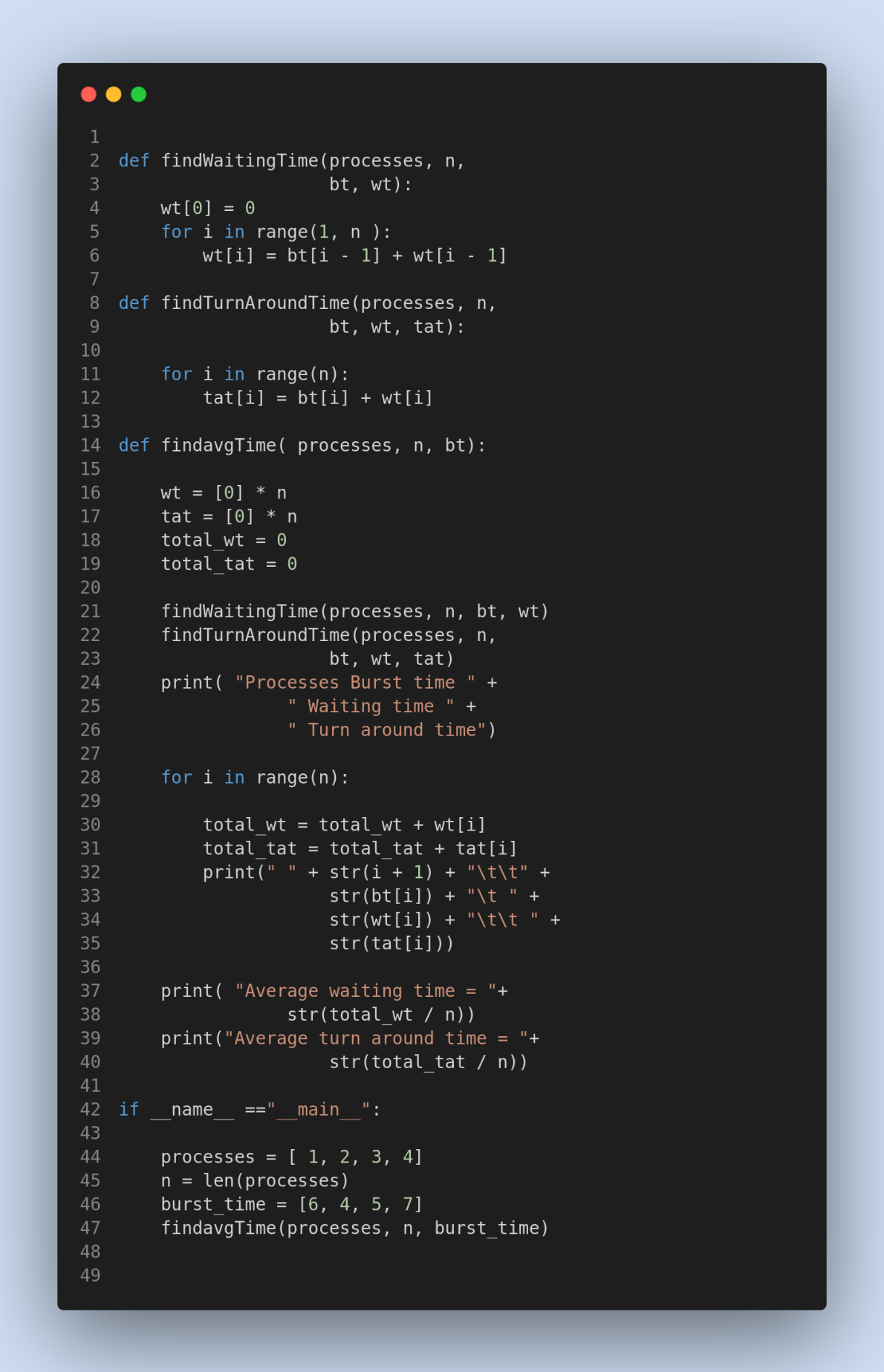
6- Find **average waiting time** =

total\_waiting\_time / no\_of\_processes.

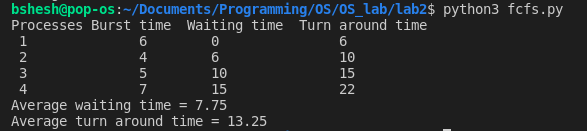
7- Similarly, find **average turnaround time** =

total\_turn\_around\_time / no\_of\_processes.

**Implementation in Python:**



**Output:**



1. **Round Robin:**

**Algorithm:**

1- Create an array **rem\_bt[]** to keep track of remaining

burst time of processes. This array is initially a

copy of bt[] (burst times array)

2- Create another array **wt[]** to store waiting times

of processes. Initialize this array as 0.

3- Initialize time : t = 0

4- Keep traversing the all processes while all processes

are not done. Do following for i'th process if it is

not done yet.

a- If rem\_bt[i] > quantum

(i) t = t + quantum

(ii) bt\_rem[i] -= quantum;

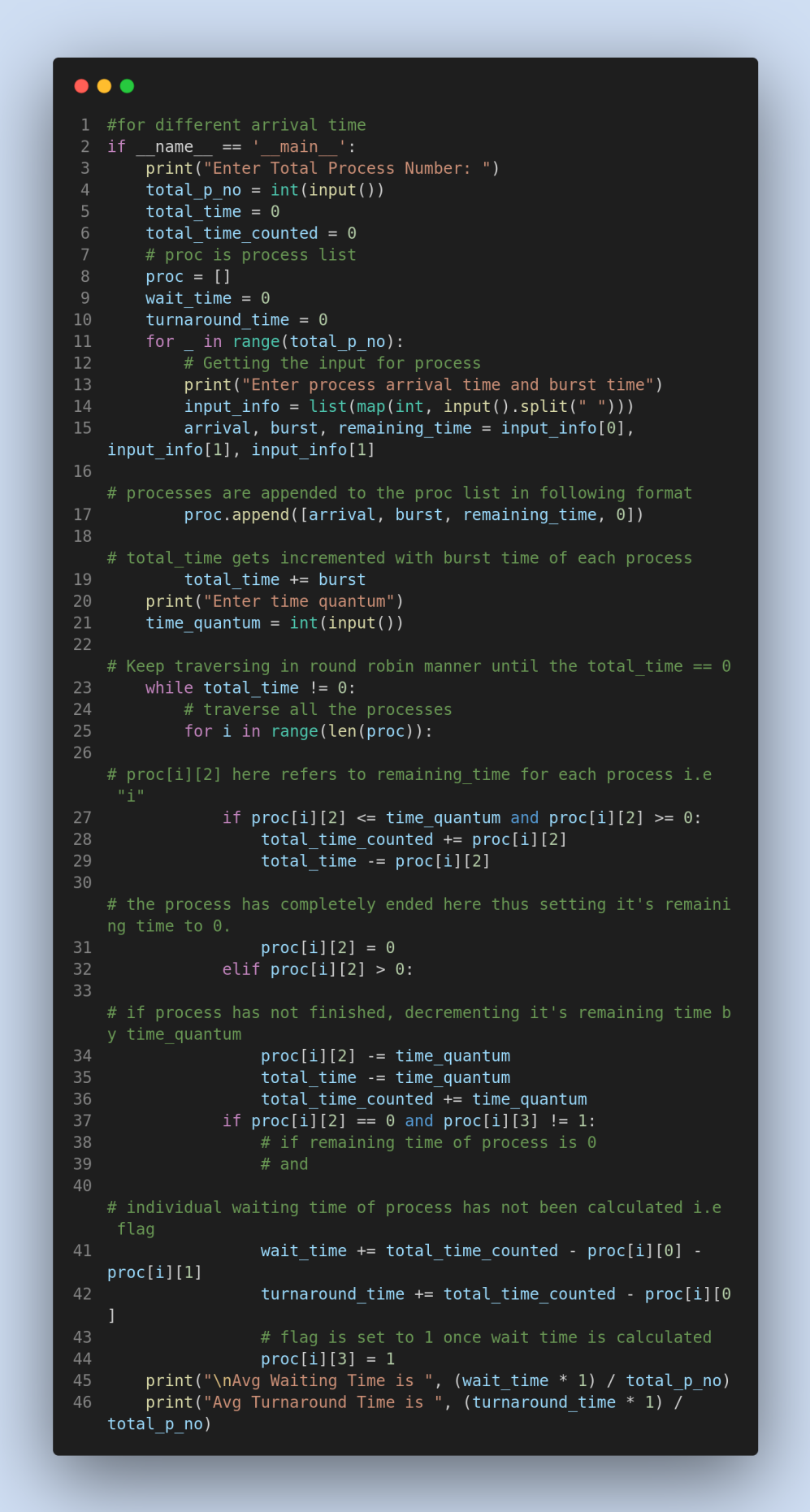
c- Else // Last cycle for this process

(i) t = t + bt\_rem[i];

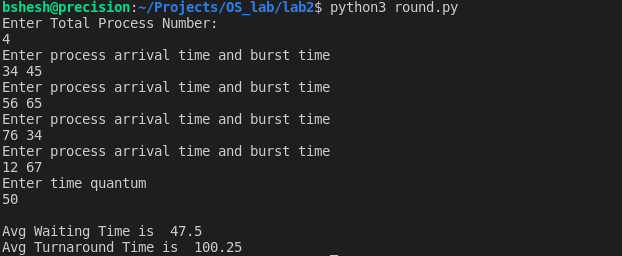
(ii) wt[i] = t - bt[i]

(ii) bt\_rem[i] = 0; // This process is over

**Implementation in Python:**

****

**Output:**

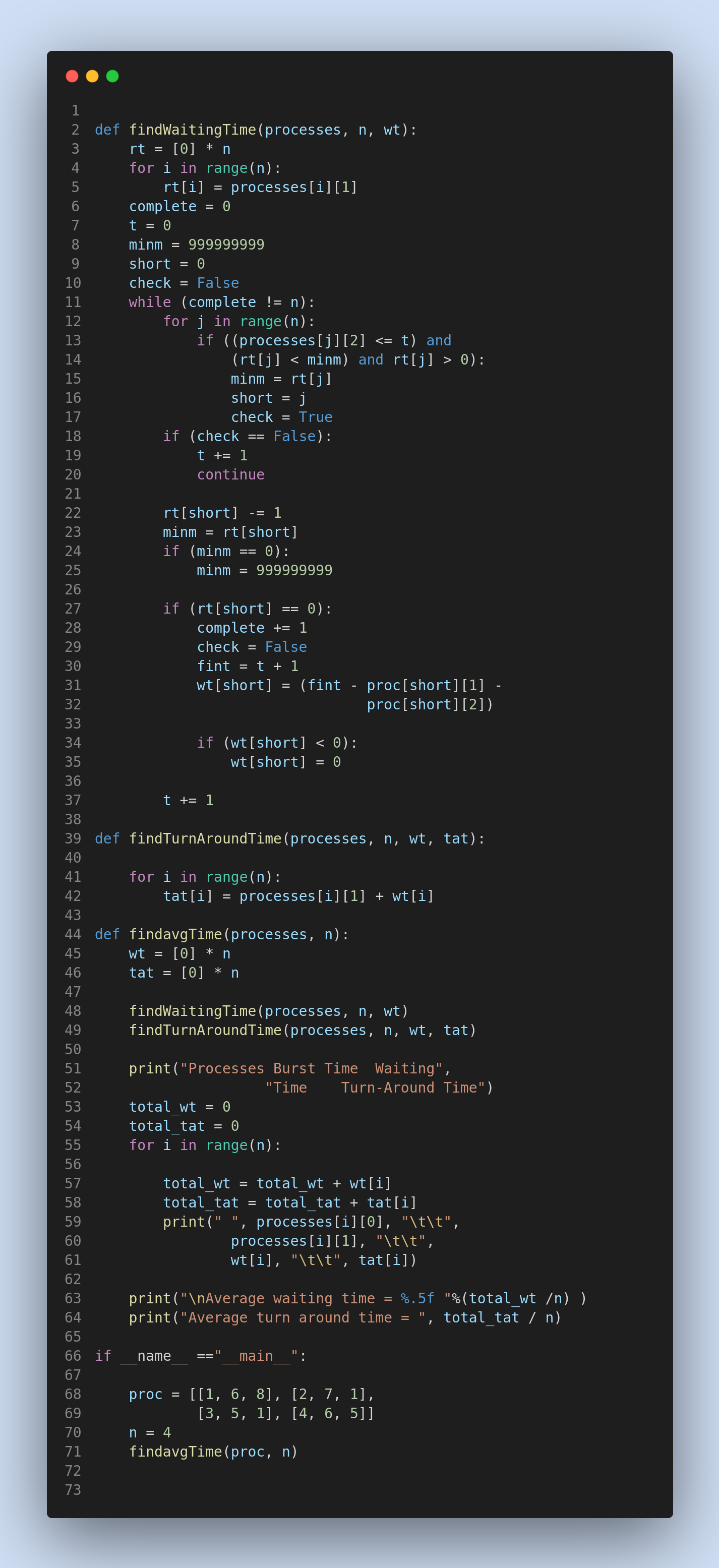


1. **Shortest Job First:**

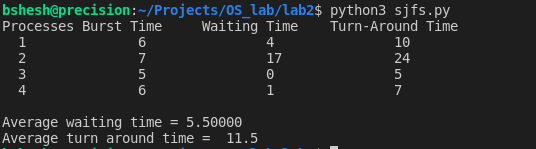
**Algorithm**:

1. Sort all the process according to the arrival time.
2. Then select that process which has minimum arrival time and minimum Burst time.
3. After completion of process make a pool of process which after till the completion of previous process and select that process among the pool which is having minimum Burst time.

**Implementation in Python:**



**Output:**



1. **Priority Scheduling:**

Algorithm:

1. First input the processes with their arrival time, burst time and priority.
2. First process will schedule, which have the lowest arrival time, if two or more processes will have lowest arrival time, then whoever has higher priority will schedule first.
3. Now further processes will be schedule according to the arrival time and priority of the process. (Here we are assuming that lower the priority number having higher priority). If two process priority are same then sort according to process number.  
   **Note:** In the question, They will clearly mention, which number will have higher priority and which number will have lower priority.
4. Once all the processes have been arrived, we can schedule them based on their priority.

**Implementation in Python:**



**Output:**

