Bonus code :

```
import streamlit as st
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from collections import deque
import time
import io

 #Set page config
st.set_page_config(page_title="CPU Scheduling Simulator", layout="wide")

 #Define scheduling algorithms functions first
def run_fcfs(processes):
"""    First Come First Serve (non-preemptive)"""
    processes = sorted(processes, key=lambda x: x['arrival'])
    current_time = 0
    gantt[] =

    for p in processes:
        p['start'] = max(current_time, p['arrival'])
        p['finish'] = p['start'] + p['burst']
        p['waiting'] = p['start'] - p['arrival']
        p['turnaround'] = p['finish'] - p['arrival']
        current_time = p['finish']

        gantt.append((p['pid'], p['start'], p['finish']))

    return processes, gantt

def run_hpf(processes):
"""    Non-Preemptive Highest Priority First"""
    processes = sorted(processes, key=lambda x: (x['priority'], x['arrival']))
```

```
    current_time = 0
    gantt[] =

    while any(p['finish'] == -1 for p in processes):
        ready = [p for p in processes if p['arrival'] <= current_time and p['finish'] == -1]

        if not ready:
            current_time += 1
            continue

#       Select process with highest priority (lowest number)
        selected = min(ready, key=lambda x: x['priority'])
        selected['start'] = current_time
        selected['finish'] = current_time + selected['burst']
        selected['waiting'] = selected['start'] - selected['arrival']
        selected['turnaround'] = selected['finish'] - selected['arrival']

        gantt.append((selected['pid'], selected['start'], selected['finish']))
        current_time = selected['finish']

    return processes, gantt

def run_rr(processes, time_quantum):
"""    Round Robin (preemptive)"""
    processes = sorted(processes, key=lambda x: x['arrival'])
    queue = deque()
    current_time = 0
    completed = 0
    n = len(processes)
    gantt[] =

#    Reset remaining time
    for p in processes:
        p['remaining'] = p['burst']
```

```
        p['start'] = -1
        p['finish'] = -1

    while completed < n:
#       Add arriving processes to queue
        for p in processes:
            if p['arrival'] == current_time:
                queue.append(p)

        if not queue:
            current_time += 1
            continue

        current_process = queue.popleft()

#       Mark start time if this is first run
        if current_process['start'] == -1:
            current_process['start'] = current_time

#       Execute for time quantum or remaining time
        exec_time = min(time_quantum, current_process['remaining'])
        gantt.append((current_process['pid'], current_time, current_time + exec_time))

        current_process['remaining'] -= exec_time
        current_time += exec_time

#       Check if process completed
        if current_process['remaining'] == 0:
            current_process['finish'] = current_time
            current_process['turnaround'] = current_process['finish'] - current_process['arrival']
            current_process['waiting'] = current_process['turnaround'] - current_process['burst']
            completed += 1
        else:
#           Re-add to queue if not finished
```

```
            queue.append(current_process)

    return processes, gantt

def run_srtf(processes):
"""    Shortest Remaining Time First (preemptive)"""
    processes = sorted(processes, key=lambda x: x['arrival'])
    current_time = 0
    completed = 0
    n = len(processes)
    gantt[] =

 #    Reset remaining time
    for p in processes:
        p['remaining'] = p['burst']
        p['start'] = -1
        p['finish'] = -1

    while completed < n:
 #        Find process with shortest remaining time
        ready = [p for p in processes
                if p['arrival'] <= current_time and p['remaining'] > 0[

        if not ready:
            current_time += 1
            continue

        shortest = min(ready, key=lambda x: x['remaining'])

 #        Mark start time if this is first run
        if shortest['start'] == -1:
            shortest['start'] = current_time

 #        Execute for 1 time unit
```

```
        shortest['remaining'] -= 1
        current_time += 1

#        Update Gantt chart
        if gantt and gantt[-1][0] == shortest['pid']:
#            Extend last block
            gantt[-1] = (shortest['pid'], gantt[-1][1], current_time)
        else:
#            New block
            gantt.append((shortest['pid'], current_time - 1, current_time))

#        Check if process completed
        if shortest['remaining'] == 0:
            shortest['finish'] = current_time
            shortest['turnaround'] = shortest['finish'] - shortest['arrival']
            shortest['waiting'] = shortest['turnaround'] - shortest['burst']
            completed += 1

    return processes, gantt

def display_results(results, gantt, algorithm):
#    Convert results to DataFrame
    results_df = pd.DataFrame}])
'        PID': p['pid'],
'        Arrival': p['arrival'],
'        Burst': p['burst'],
'        Priority': p['priority'],
'        Start': p['start'],
'        Finish': p['finish'],
'        Waiting': p['waiting'],
'        Turnaround': p['turnaround']
{    for p in results([

#    Display results in tabs
```

```
    tab1, tab2, tab3, tab4 = st.tabs(["Results Table", "Gantt Chart", "Statistics", "Summary"])

    with tab1:
        st.dataframe(results_df)

    with tab2:
#       Create Gantt chart
        fig, ax = plt.subplots(figsize=(10, 4))

#       Create a mapping from PID to y-axis position
        pids = sorted(list(set([item[0] for item in gantt])))
        pid_to_y = {pid: i for i, pid in enumerate(pids)}

#       Plot each process
        for pid, start, end in gantt:
            ax.broken_barh([(start, end-start)], (pid_to_y[pid]-0.4, 0.8 ,(
                        facecolors=('tab:blue')(

#       Customize the plot
        ax.set_yticks(range(len(pids)))
        ax.set_yticklabels(pids)
        ax.set_xlabel('Time')
        ax.set_title('Gantt Chart')
        ax.grid(True)

#       Ensure x-axis shows integer values
        ax.xaxis.set_major_locator(plt.MaxNLocator(integer=True))

        st.pyplot(fig)

    with tab3:
#       Create statistics charts
        fig, (ax1, ax2) = plt.subplots(2, 1, figsize=(10, 8))
```

```
#       Extract data
      pids = [p['pid'] for p in results]
      waiting_times = [p['waiting'] for p in results]
      turnaround_times = [p['turnaround'] for p in results]

#       Plot waiting times
      bars1 = ax1.bar(pids, waiting_times, color='tab:orange')
      ax1.set_title('Waiting Times')
      ax1.set_ylabel('Time Units')
      ax1.grid(True, axis='y')

#       Plot turnaround times
      bars2 = ax2.bar(pids, turnaround_times, color='tab:green')
      ax2.set_title('Turnaround Times')
      ax2.set_ylabel('Time Units')
      ax2.grid(True, axis='y')

#       Add value labels
      for bars in [bars1, bars2]:
        for bar in bars:
          height = bar.get_height()
          ax = bar.axes
          ax.text(bar.get_x() + bar.get_width()/2., height,
              f'{int(height)},'
              ha='center', va='bottom('

      plt.tight_layout()
      st.pyplot(fig)

    with tab4:
#       Calculate statistics
      avg_wait = sum(p['waiting'] for p in results) / len(results)
      avg_turnaround = sum(p['turnaround'] for p in results) / len(results)
      throughput = len(results) / max(p['finish'] for p in results)
```

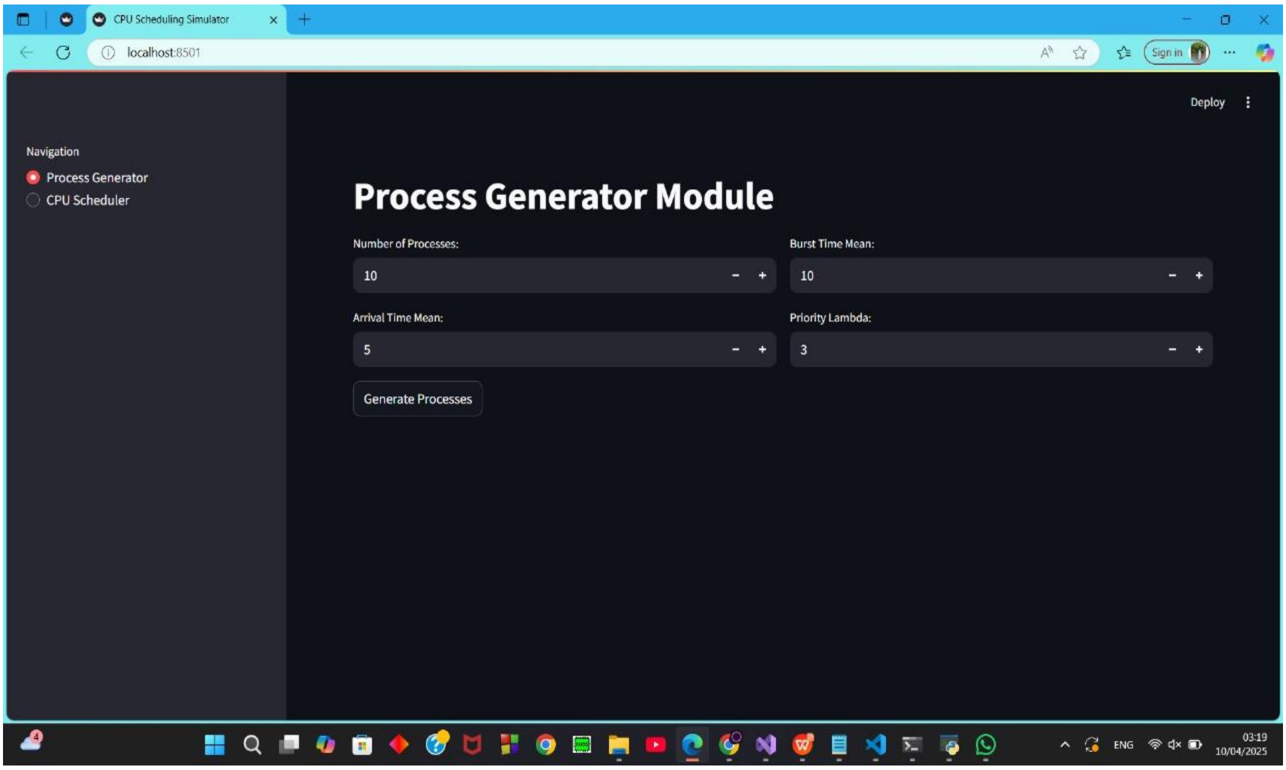
```
#       Create summary
        st.subheader("Performance Summary")
        st.write(f"**Algorithm:** {algorithm}")
        st.write(f"**Average Waiting Time:** {avg_wait:.2f} time units")
        st.write(f"**Average Turnaround Time:** {avg_turnaround:.2f} time units")
        st.write(f"**Throughput:** {throughput:.4f} processes per time unit")

        st.subheader("Process Execution Details")
        for p in results:
            st.write(f"""
**          {p['pid']}**: Arrived at {p['arrival']}, Burst {p['burst']}, Priority {p['priority']}
            Started at {p['start']}, Finished at {p['finish']}
            Waiting Time: {p['waiting']}, Turnaround Time: {p['turnaround']}
("""

#Main application code
def main:()
#   Sidebar for navigation
    page = st.sidebar.radio("Navigation", ["Process Generator", "CPU Scheduler"])

    if page == "Process Generator:"
        st.title("Process Generator Module")

#       Parameters
        col1, col2 = st.columns(2)
        with col1:
            num_processes = st.number_input("Number of Processes:", min_value=1, max_value=100,
value=10)
            arrival_mean = st.number_input("Arrival Time Mean:", min_value=0, max_value=100, value=5)
        with col2:
            burst_mean = st.number_input("Burst Time Mean:", min_value=1, max_value=100, value=10)
            priority_lambda = st.number_input("Priority Lambda:", min_value=1, max_value=10, value=3)

#       Generate button
```

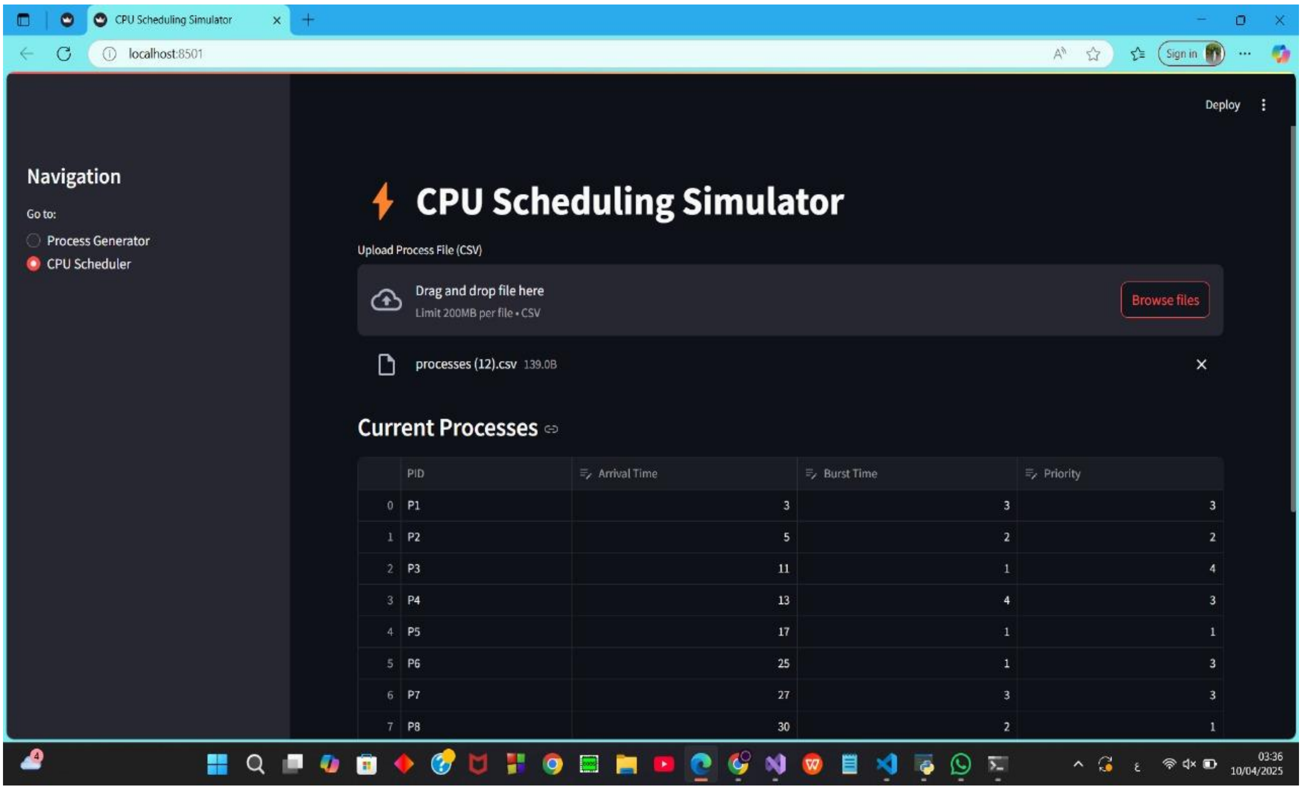
```
    if st.button("Generate Processes"):
#        Generate arrival times (normal distribution)
        arrival_times = np.abs(np.random.normal(arrival_mean, arrival_mean/2,
num_processes)).astype(int)
        arrival_times = np.cumsum(arrival_times)

#        Generate burst times (normal distribution)
        burst_times = np.abs(np.random.normal(burst_mean, burst_mean/2, num_processes)).astype(int)
        burst_times = np.where(burst_times < 1, 1, burst_times)

#        Generate priorities (Poisson distribution)
        priorities = np.random.poisson(priority_lambda, num_processes)
        priorities = np.where(priorities < 1, 1, priorities)

#        Create DataFrame
        processes = pd.DataFrame})
'            PID': [f"P{i+1}" for i in range(num_processes)],
'            Arrival': arrival_times,
'            Burst': burst_times,
'            Priority': priorities
({

        st.session_state.processes = processes

#    Display and save processes
    if 'processes' in st.session_state:
        st.subheader("Generated Processes")
        st.dataframe(st.session_state.processes)

#        Download button
        csv = st.session_state.processes.to_csv(index=False).encode('utf-8')
        st.download_button)
            label="Download as CSV,"
            data=csv,
            file_name='processes.csv,'
```

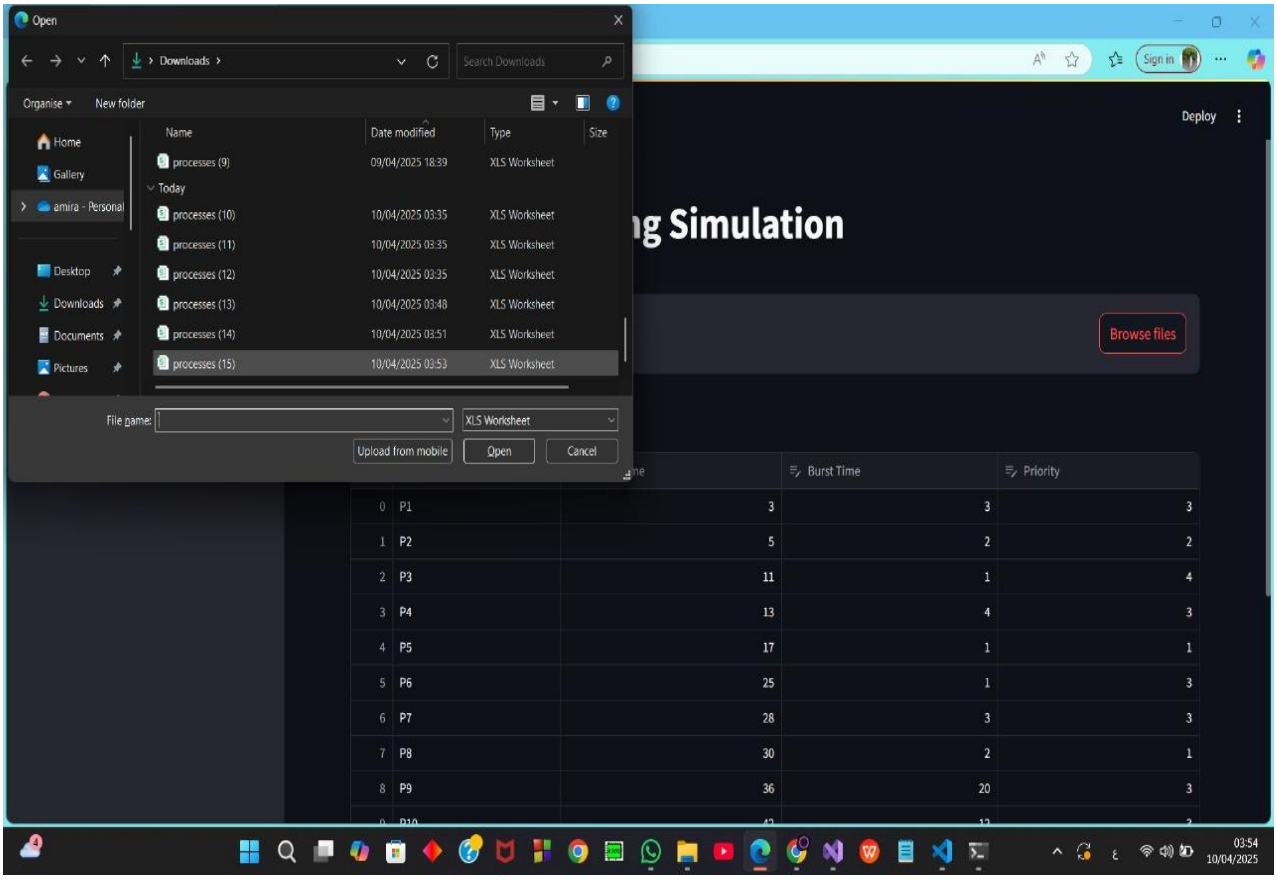
```
            mime='text/csv'
(

    elif page == "CPU Scheduler:"
        st.title("CPU Scheduling Simulator")

 #        File upload
        uploaded_file = st.file_uploader("Upload Process File (CSV)", type=['csv'])

        if uploaded_file is not None:
            processes = pd.read_csv(uploaded_file)
            st.session_state.processes = processes.to_dict('records')

 #            Convert to list of dictionaries with additional fields
            processes_list[] =
            for i, row in processes.iterrows:()
                processes_list.append})
'                pid': row['PID'],
'                arrival': row['Arrival'],
'                burst': row['Burst'],
'                priority': row.get('Priority', 1),
'                remaining': row['Burst'],
'                start': -1,
'                finish': -1,
'                waiting': 0,
'                turnaround': 0
({

            st.session_state.processes_list = processes_list

 #        Algorithm selection
        algorithm = st.selectbox)
"            Scheduling Algorithm,":
]
```

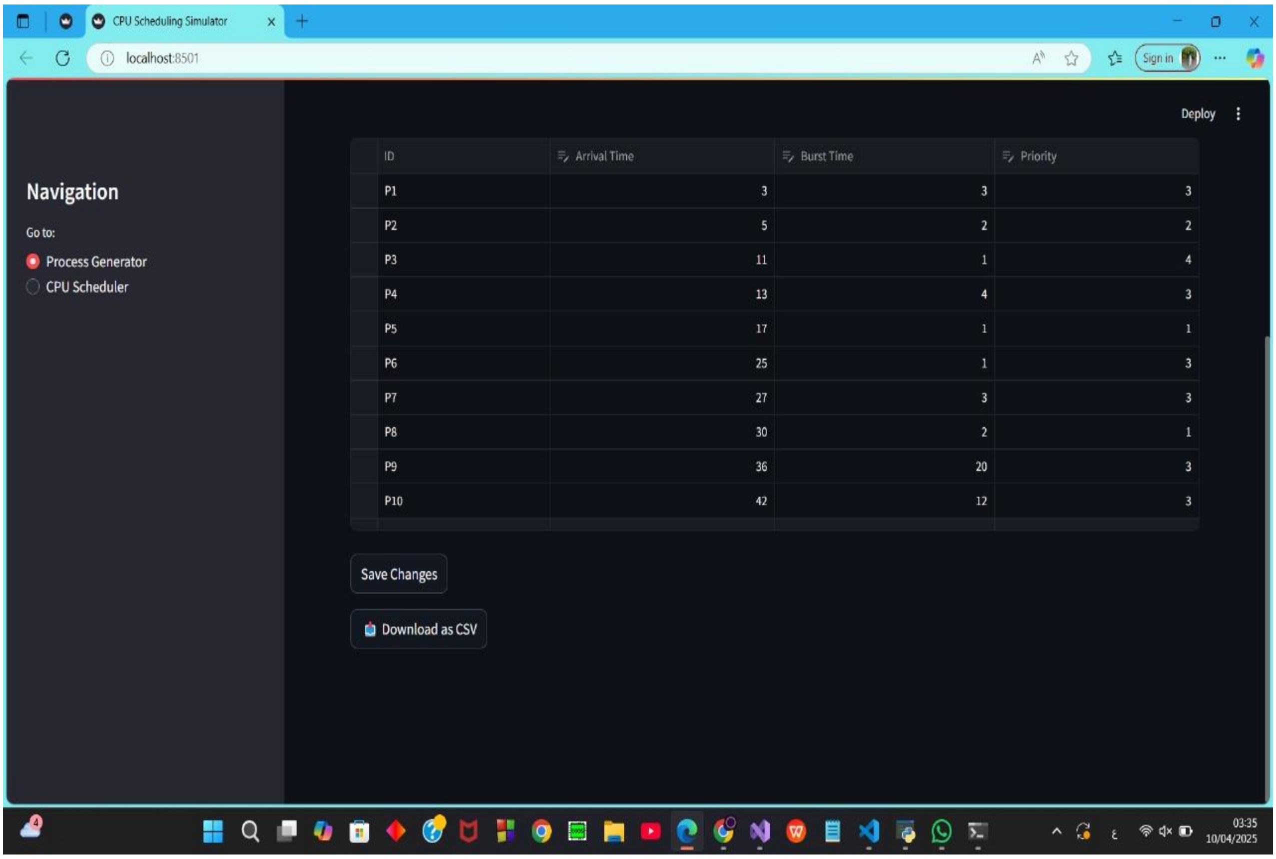
```
"            First Come First Serve (FCFS),"
"            Non-Preemptive Highest Priority First (HPF),"
"            Round Robin (RR),"
"            Preemptive Shortest Remaining Time First (SRTF)"
[
(

 #      Time quantum for RR
    time_quantum = 4
    if "Round Robin" in algorithm:
      time_quantum = st.number_input("Time Quantum:", min_value=1, max_value=100, value=4)

 #      Run scheduling
    if st.button("Run Scheduling") and 'processes_list' in st.session_state:
      with st.spinner('Running simulation...'):
        if "FCFS" in algorithm:
          results, gantt = run_fcfs(st.session_state.processes_list)
        elif "HPF" in algorithm:
          results, gantt = run_hpf(st.session_state.processes_list)
        elif "RR" in algorithm:
          results, gantt = run_rr(st.session_state.processes_list, time_quantum)
        elif "SRTF" in algorithm:
          results, gantt = run_srtf(st.session_state.processes_list)

        display_results(results, gantt, algorithm)

if __name__ == "__main:"__
  main()
```

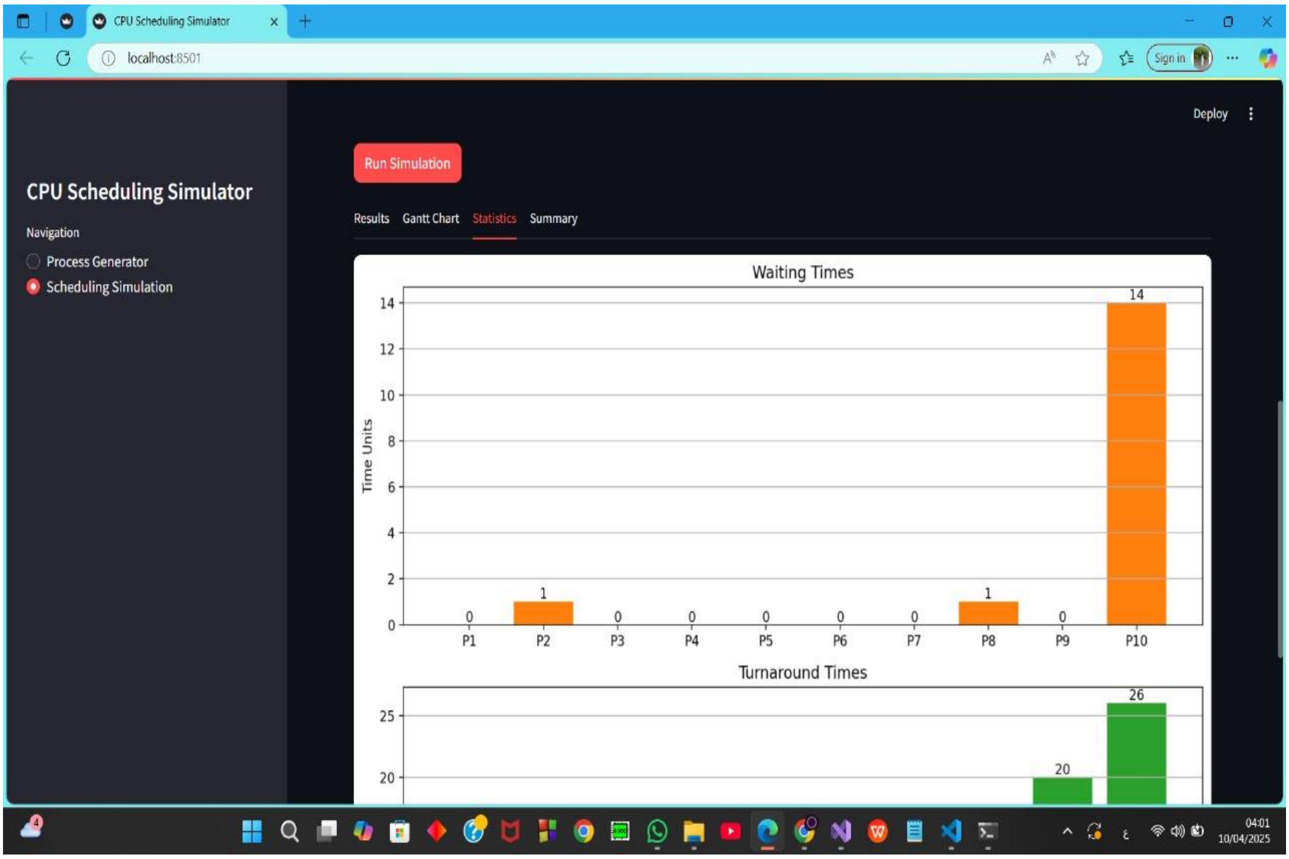
output :

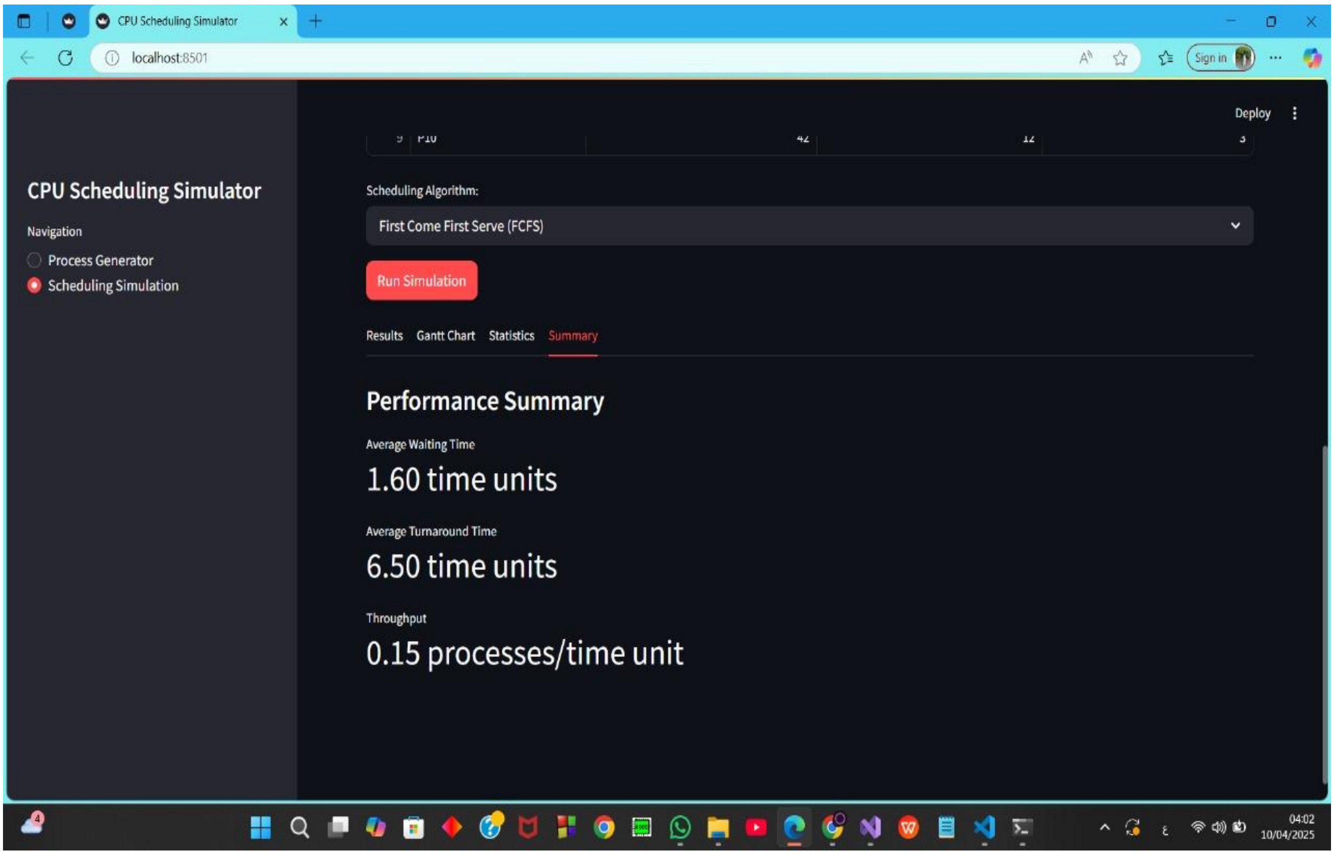

CPU Scheduling Simulator
localhost:8501
Deploy
Navigation
Process Generator
CPU Scheduler
Process Generator Module
Number of Processes:
10
Burst Time Mean:
10
Arrival Time Mean:
5
Priority Lambda:
3
Generate Processes
ENG
03:19
10/04/2025

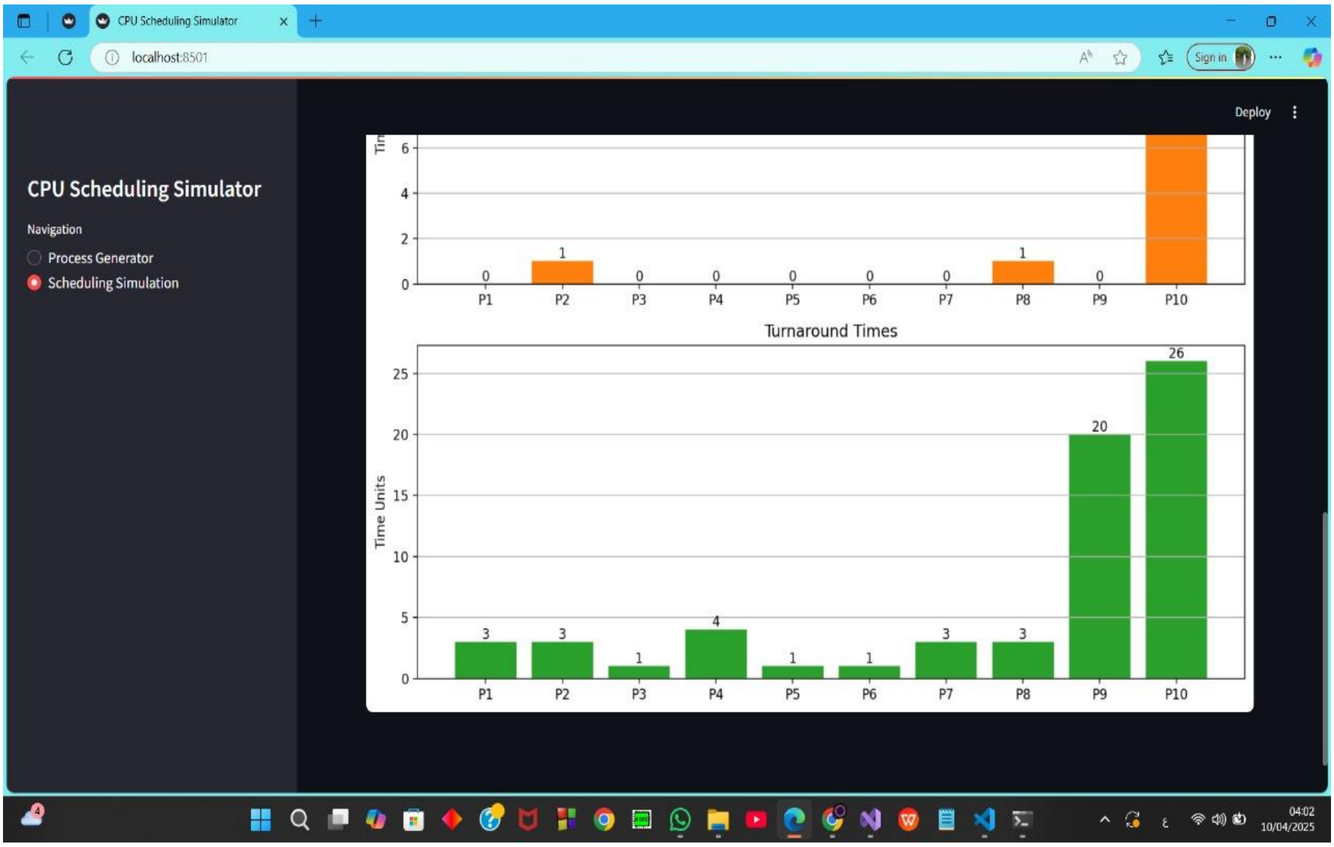

CPU Scheduling Simulator
localhost:8501
Deploy
Navigation
Go to:
Process Generator
CPU Scheduler
CPU Scheduling Simulator
Upload Process File (CSV)
Drag and drop file here
Limit 200MB per file • CSV
Browse files
processes (12).csv 139.0B
Current Processes
PID Arrival Time Burst Time Priority
0 P1 3 3 3
1 P2 5 2 2
2 P3 11 1 4
3 P4 13 4 3
4 P5 17 1 1
5 P6 25 1 3
6 P7 27 3 3
7 P8 30 2 1
03:36
10/04/2025

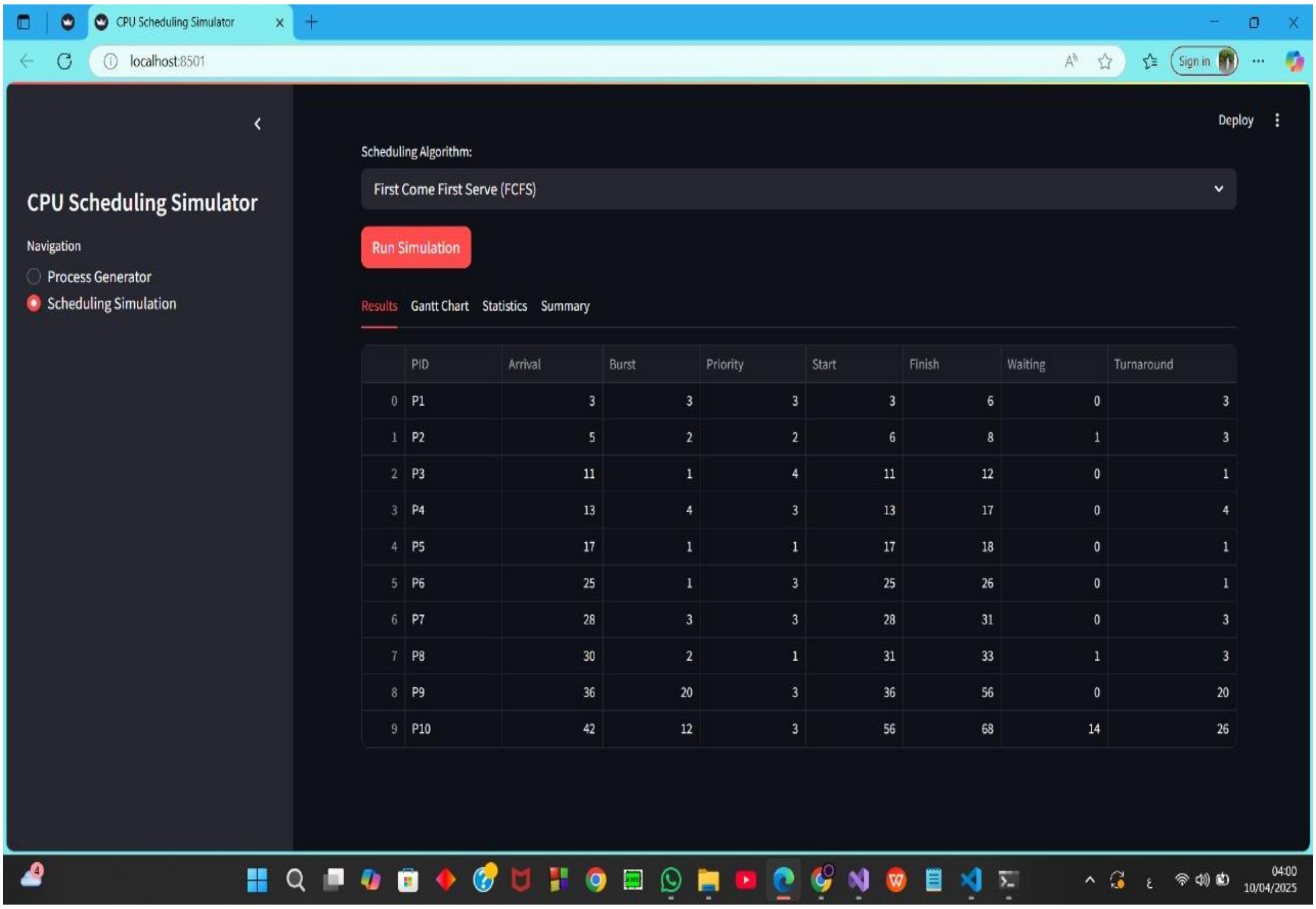

Open
Downloads
Search Downloads
Organise
New folder
Name
Date modified
Type
Size
processes (9) 09/04/2025 18:39 XLS Worksheet
Today
processes (10) 10/04/2025 03:35 XLS Worksheet
processes (11) 10/04/2025 03:35 XLS Worksheet
processes (12) 10/04/2025 03:35 XLS Worksheet
processes (13) 10/04/2025 03:48 XLS Worksheet
processes (14) 10/04/2025 03:51 XLS Worksheet
processes (15) 10/04/2025 03:53 XLS Worksheet
Home
Gallery
amira - Personal
Desktop
Downloads
Documents
Pictures
File name:
XLS Worksheet
Upload from mobile
Open
Cancel
Deploy
ng Simulation
Browse files
Burst Time
Priority
0 P1 3 3 3
1 P2 5 2 2
2 P3 11 1 4
3 P4 13 4 3
4 P5 17 1 1
5 P6 25 1 3
6 P7 28 3 3
7 P8 30 2 1
8 P9 36 20 3
03:54
10/04/2025


CPU Scheduling Simulator
localhost:8501
Deploy
Navigation
Go to:
Process Generator
CPU Scheduler
ID
Arrival Time
Burst Time
Priority
P1 3 3 3
P2 5 2 2
P3 11 1 4
P4 13 4 3
P5 17 1 1
P6 25 1 3
P7 27 3 3
P8 30 2 1
P9 36 20 3
P10 42 12 3
Save Changes
Download as CSV
03:35
10/04/2025

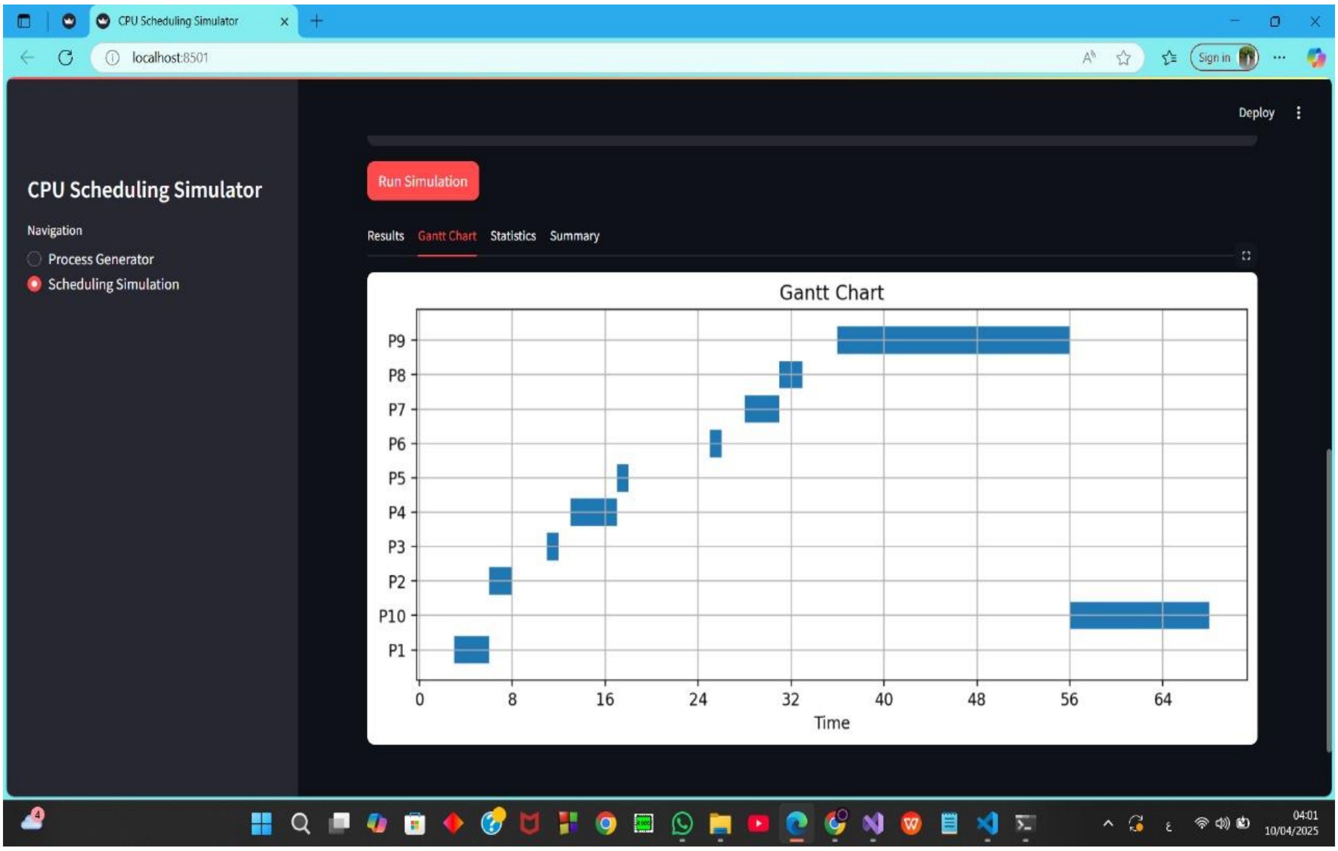

CPU Scheduling Simulator
localhost:8501
Deploy
CPU Scheduling Simulator
Navigation
Process Generator
Scheduling Simulation
Run Simulation
Results
Gantt Chart
Statistics
Summary
Waiting Times
Time Units
P1
P2
P3
P4
P5
P6
P7
P8
P9
P10
Turnaround Times

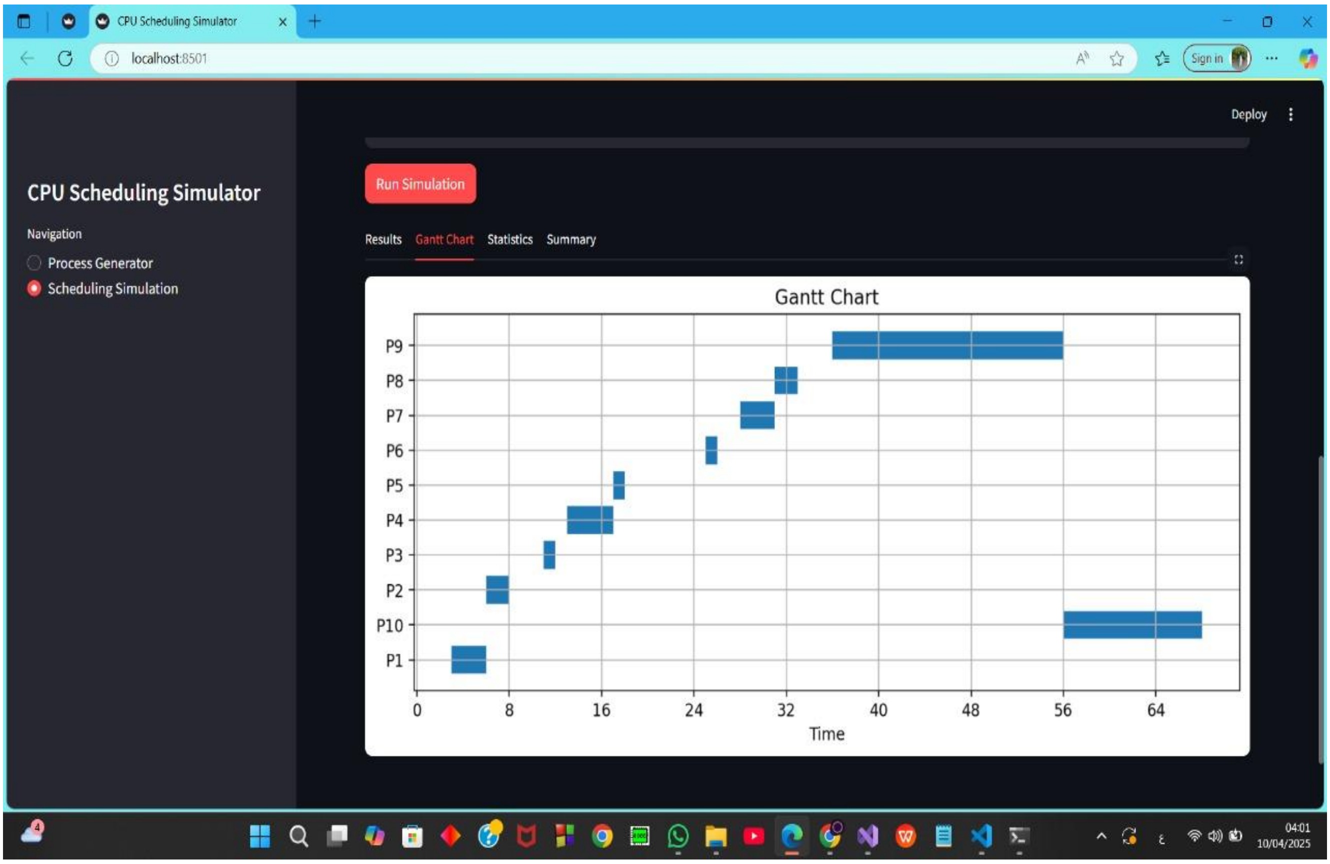

CPU Scheduling Simulator
localhost:8501
Deploy
CPU Scheduling Simulator
Navigation
Process Generator
Scheduling Simulation
Scheduling Algorithm:
First Come First Serve (FCFS)
Run Simulation
Results
Gantt Chart
Statistics
Summary
Performance Summary
Average Waiting Time
1.60 time units
Average Turnaround Time
6.50 time units
Throughput
0.15 processes/time unit

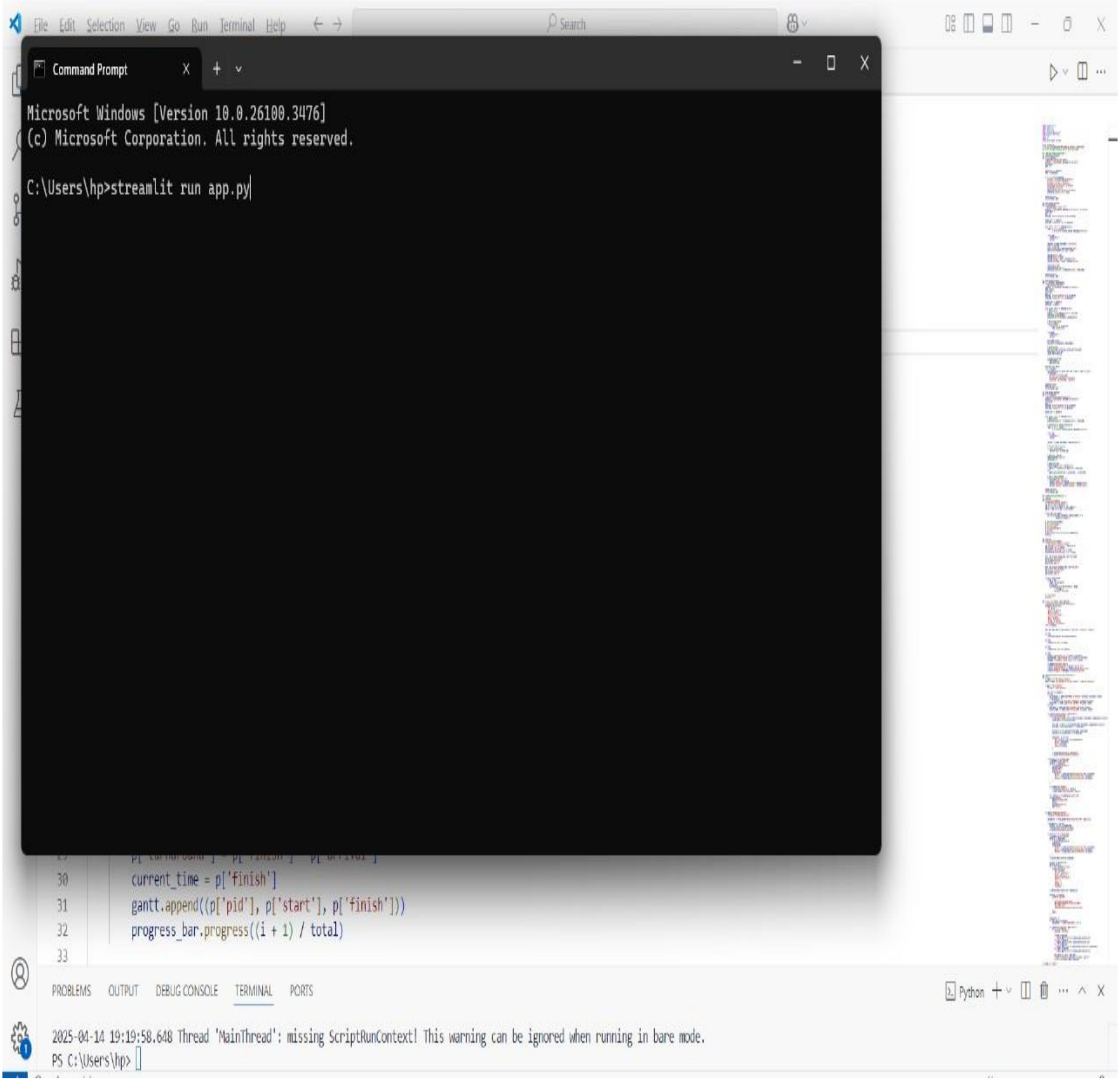




	PID	Arrival	Burst	Priority	Start	Finish	Waiting	Turnaround
0	P1	3	3	3	3	6	0	3
1	P2	5	2	2	6	8	1	3
2	P3	11	1	4	11	12	0	1
3	P4	13	4	3	13	17	0	4
4	P5	17	1	1	17	18	0	1
5	P6	25	1	3	25	26	0	1
6	P7	28	3	3	28	31	0	3
7	P8	30	2	1	31	33	1	3
8	P9	36	20	3	36	56	0	20
9	P10	42	12	3	56	68	14	26


CPU Scheduling Simulator
localhost:8501
Deploy
CPU Scheduling Simulator
Navigation
Process Generator
Scheduling Simulation
Run Simulation
Results
Gantt Chart
Statistics
Summary
Gantt Chart
P9
P8
P7
P6
P5
P4
P3
P2
P10
P1
0
8
16
24
32
40
48
56
64
Time
04:01
10/04/2025


CPU Scheduling Simulator
localhost:8501
Deploy
CPU Scheduling Simulator
Navigation
Process Generator
Scheduling Simulation
Run Simulation
Results
Gantt Chart
Statistics
Summary
Gantt Chart
P9
P8
P7
P6
P5
P4
P3
P2
P10
P1
0
8
16
24
32
40
48
56
64
Time
04:01
10/04/2025


File Edit Selection View Go Run Terminal Help
Search
Command Prompt
Microsoft Windows [Version 10.0.26100.3476]
(c) Microsoft Corporation. All rights reserved.
C:\Users\hp>streamlit run app.py
30 current_time = p['finish']
31 gantt.append((p['pid'], p['start'], p['finish']))
32 progress_bar.progress((i + 1) / total)
33
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS
Python
2025-04-14 19:19:58.648 Thread 'MainThread': missing ScriptRunContext! This warning can be ignored when running in bare mode.
PS C:\Users\hp>