

Task 1: Batch Processing Simulation (Python)

Write a Python script to execute multiple .py files sequentially, mimicking batch processing.

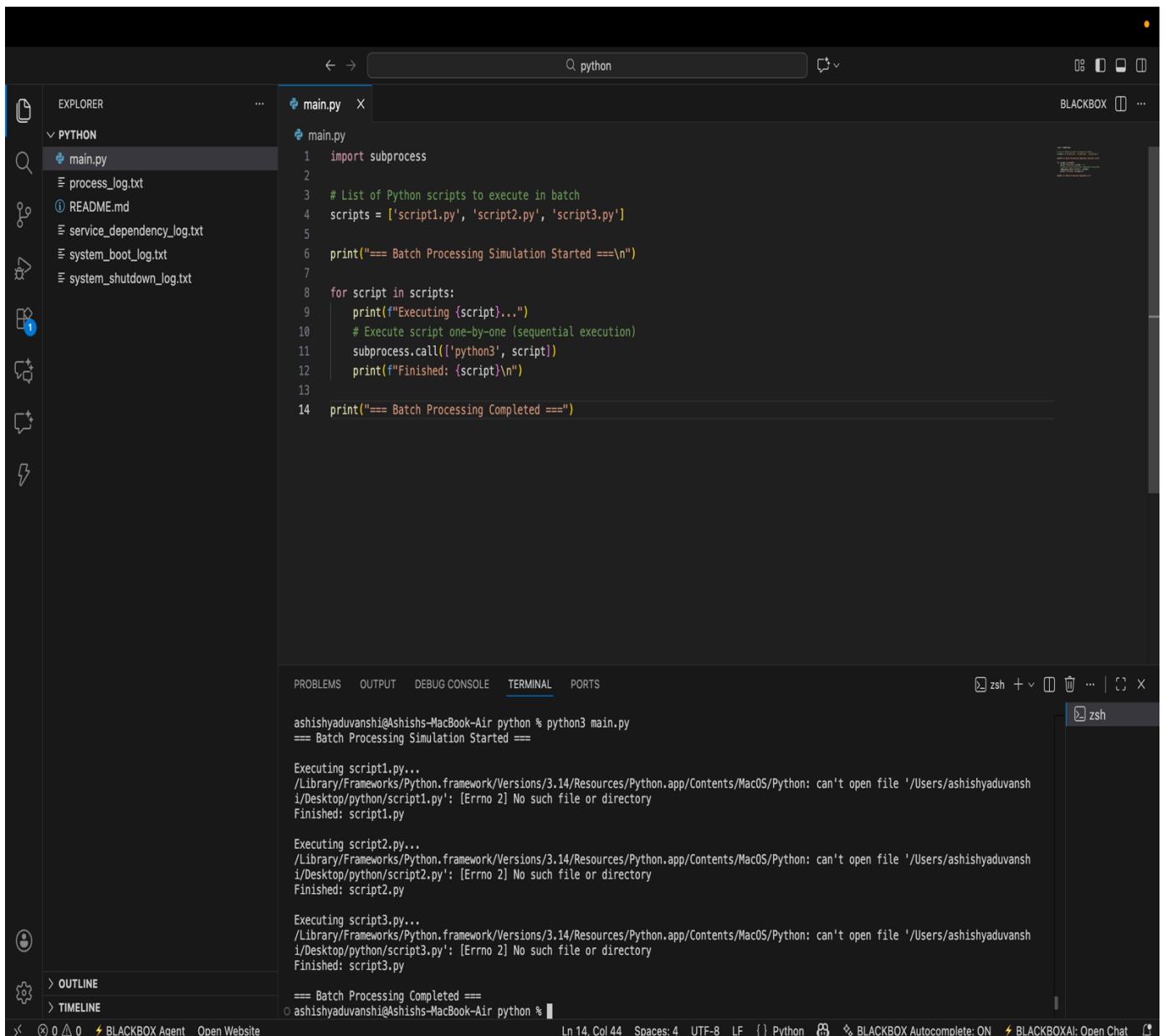
```
import subprocess
```

```
scripts = ['script1.py', 'script2.py', 'script3.py']
```

```
for script in scripts:
```

```
    print(f'Executing {script}...')
```

```
    subprocess.call(['python3', script])
```



The screenshot shows the VS Code interface with the following details:

- EXPLORER:** Shows files: main.py, process_log.txt, README.md, service_dependency_log.txt, system_boot_log.txt, and system_shutdown_log.txt.
- MAIN PANE:** Displays the content of `main.py` (shown above).
- TERMINAL:** Shows the output of running the script:

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
== Batch Processing Simulation Started ==

Executing script1...
/Library/Frameworks/Python.framework/Versions/3.14/Resources/Python.app/Contents/MacOS/Python: can't open file '/Users/ashishyaduvanshi/Desktop/python/script1.py': [Errno 2] No such file or directory
Finished: script1.py

Executing script2...
/Library/Frameworks/Python.framework/Versions/3.14/Resources/Python.app/Contents/MacOS/Python: can't open file '/Users/ashishyaduvanshi/Desktop/python/script2.py': [Errno 2] No such file or directory
Finished: script2.py

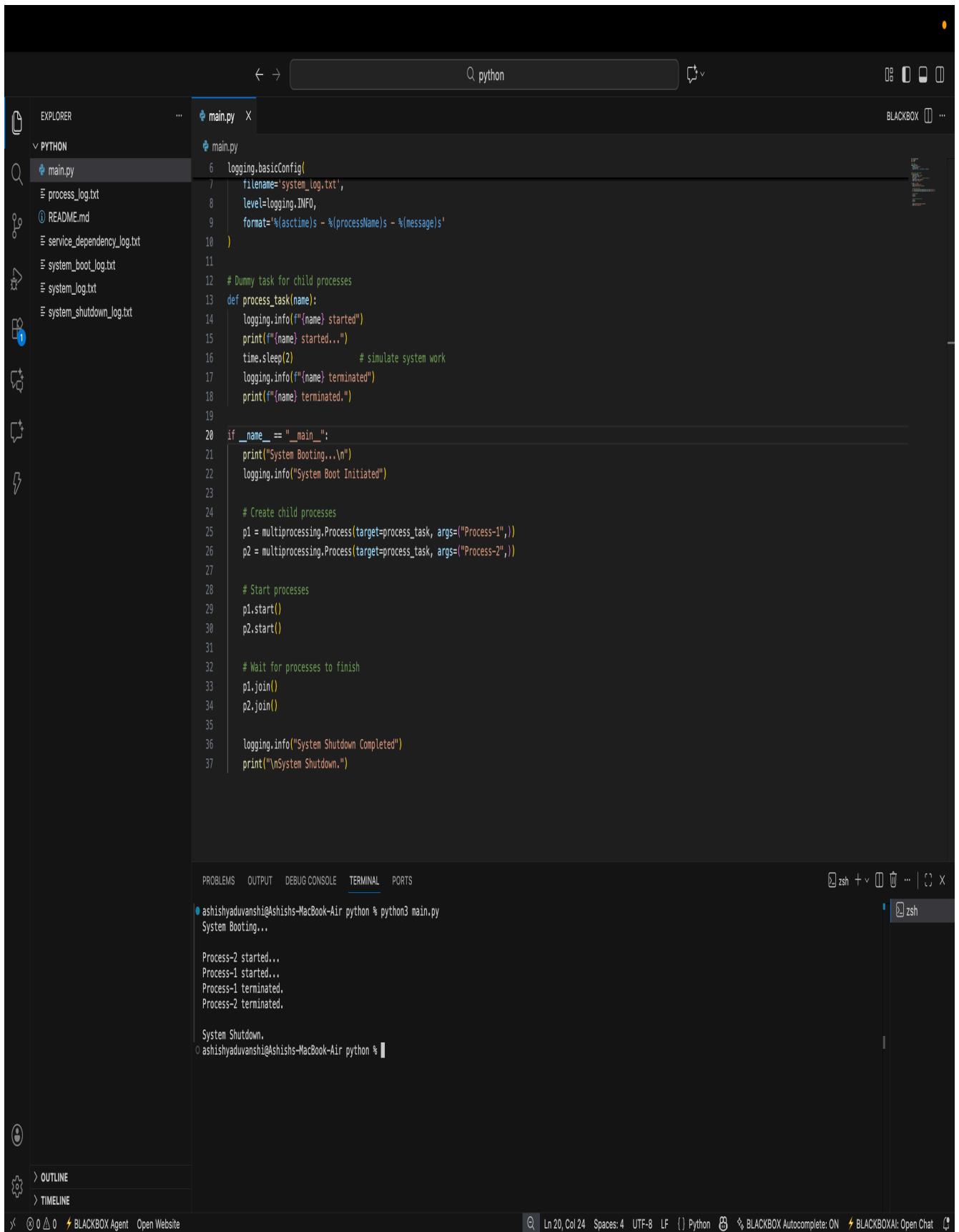
Executing script3...
/Library/Frameworks/Python.framework/Versions/3.14/Resources/Python.app/Contents/MacOS/Python: can't open file '/Users/ashishyaduvanshi/Desktop/python/script3.py': [Errno 2] No such file or directory
Finished: script3.py

== Batch Processing Completed ==
```
- STATUS BAR:** Shows the terminal mode, file path `ashishyaduvanshi@Ashishs-MacBook-Air python %`, and status indicators: Line 14, Col 44, Spaces: 4, UTF-8, LF, Python, and BLACKBOX Autocomplete: ON.

Task 2: System Startup and Logging

Simulate system startup using Python by creating multiple processes and logging their start and end into a log file.

```
import multiprocessing
import logging
import time
logging.basicConfig(filename='system_log.txt', level=logging.INFO,
                    format='%(asctime)s - %(processName)s - %(message)s')
def process_task(name):
    logging.info(f'{name} started')
    time.sleep(2)
    logging.info(f'{name} terminated')
if __name__ == '__main__':
    print("System Booting...")
    p1 = multiprocessing.Process(target=process_task, args=("Process-1",))
    p2 = multiprocessing.Process(target=process_task, args=("Process-2",))
    p1.start()
    p2.start()
    p1.join()
    p2.join()
    print("System Shutdown.")
```



```
main.py
6  logging.basicConfig(
7      filename='system_log.txt',
8      level=logging.INFO,
9      format='%(asctime)s - %(processName)s - %(message)s'
10 )
11
12 # Dummy task for child processes
13 def process_task(name):
14     logging.info(f"{name} started")
15     print(f"{name} started...")
16     time.sleep(2)           # simulate system work
17     logging.info(f"{name} terminated")
18     print(f"{name} terminated.")
19
20 if __name__ == "__main__":
21     print("System Booting...")
22     logging.info("System Boot Initiated")
23
24 # Create child processes
25 p1 = multiprocessing.Process(target=process_task, args=("Process-1",))
26 p2 = multiprocessing.Process(target=process_task, args=("Process-2",))
27
28 # Start processes
29 p1.start()
30 p2.start()
31
32 # Wait for processes to finish
33 p1.join()
34 p2.join()
35
36 logging.info("System Shutdown Completed")
37 print("\nSystem Shutdown.")
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
System Booting...

Process-2 started...
Process-1 started...
Process-1 terminated.
Process-2 terminated.

System Shutdown.
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

OUTLINE TIMELINE

BLACKBOX Agent Open Website

Ln 20, Col 24 Spaces: 4 UTF-8 LF Python % BLACKBOX Autocomplete: ON BLACKBOXAI: Open Chat

Task 3: System Calls and IPC (Python - fork, exec, pipe)

Use system calls (fork(), exec(), wait()) and implement basic Inter-Process Communication using pipes in C or Python.

```
import os

r, w = os.pipe()

pid = os.fork()

if pid > 0:

    os.close(r)

    os.write(w, b"Hello from parent")

    os.close(w)

    os.wait()

else:

    os.close(w)

    message = os.read(r, 1024)

    print("Child received:", message.decode())

    os.close(r)
```

The screenshot shows a dark-themed Python development environment in VS Code. The top bar has a search field containing "python". The Explorer sidebar on the left shows files: main.py, process_log.txt, README.md, service_dependency_log.txt, system_boot_log.txt, system_log.txt, and system_shutdown_log.txt. The main editor window displays the following Python code in "main.py":

```
1 import multiprocessing
2 import os
3
4 def child_process(pipe_conn):
5     """Child reads data sent by Parent."""
6     message = pipe_conn.recv()      # Read message from parent
7     print(f"[Child] PID = {os.getpid()} received: {message}")
8     pipe_conn.close()
9
10
11 def parent_child_pipe():
12     parent_conn, child_conn = multiprocessing.Pipe() # Create a pipe
13
14     print("== PIPE IPC Simulation Started ==")
15
16     # Create child process
17     child = multiprocessing.Process(target=child_process, args=(child_conn,))
18
19     child.start()
20
21     # Parent sending message
22     msg = "Hello Child, this is Parent!"
23     print(f"[Parent] PID = {os.getpid()} sending: {msg}")
24     parent_conn.send(msg)
25
26     child.join()
27
28     print("\n== PIPE IPC Simulation Completed ==")
29
30
31 if __name__ == "__main__":
32     parent_child_pipe()
```

The bottom section shows the terminal output:

```
ashishyaduvanshi@Ashishs-MacBook-Air python % python3 main.py
== PIPE IPC Simulation Started ==

[Parent] PID = 34538 sending: Hello Child, this is Parent!
[Child] PID = 34540 received: Hello Child, this is Parent!

== PIPE IPC Simulation Completed ==
ashishyaduvanshi@Ashishs-MacBook-Air python %
```

The status bar at the bottom shows: Ln 31, Col 24 Spaces: 4 UTF-8 LF () Python ⚡ BLACKBOX Autocomplete: ON ⚡ BLACKBOXAI: Open Chat

Task 4: VM Detection and Shell Interaction

Create a shell script to print system details and a Python script to detect if the system is running inside a virtual machine.

```
#!/bin/bash

echo "Kernel Version:"

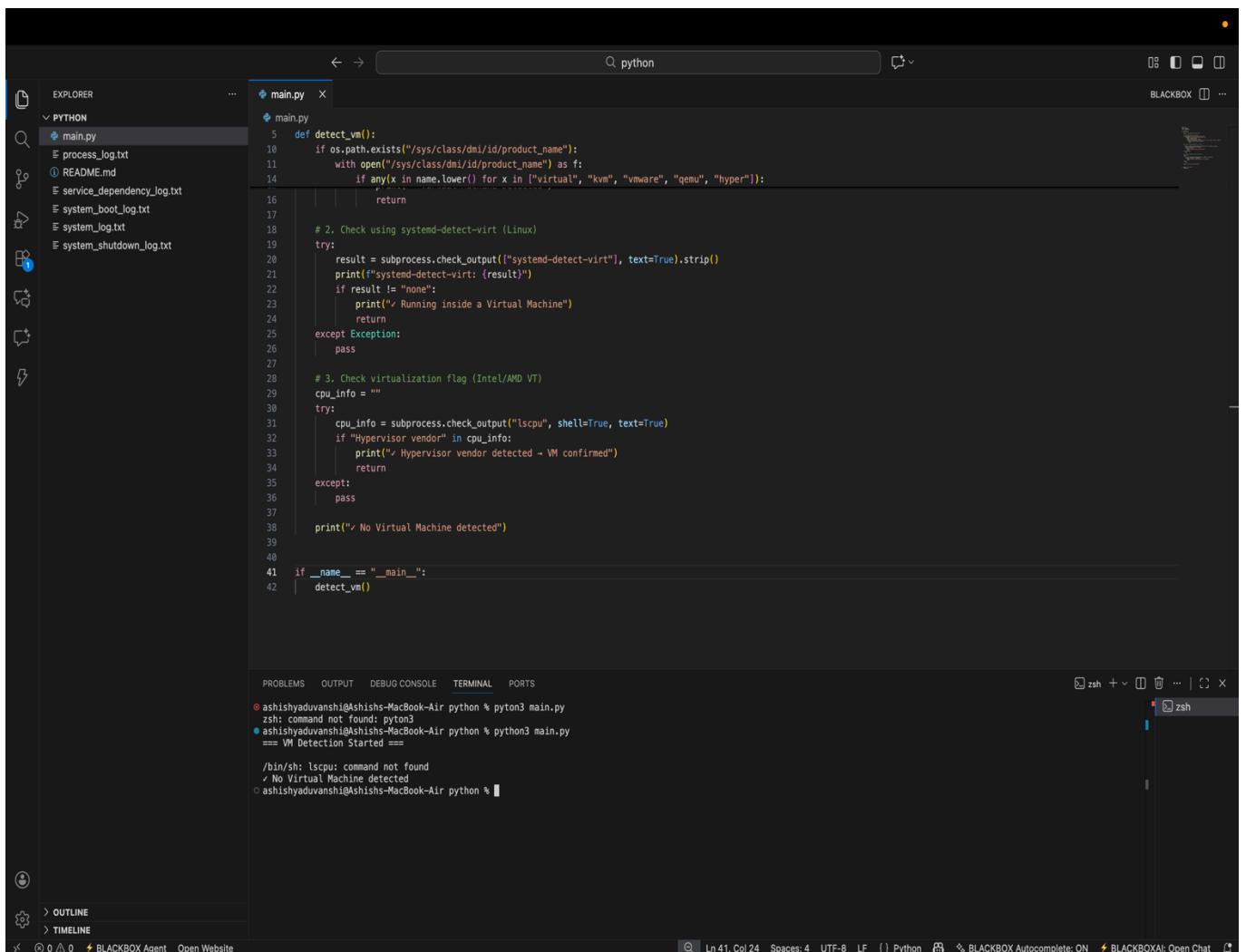
uname -r

echo "User:"

whoami

echo "Hardware Info:"

lscpu | grep 'Virtualization'
```



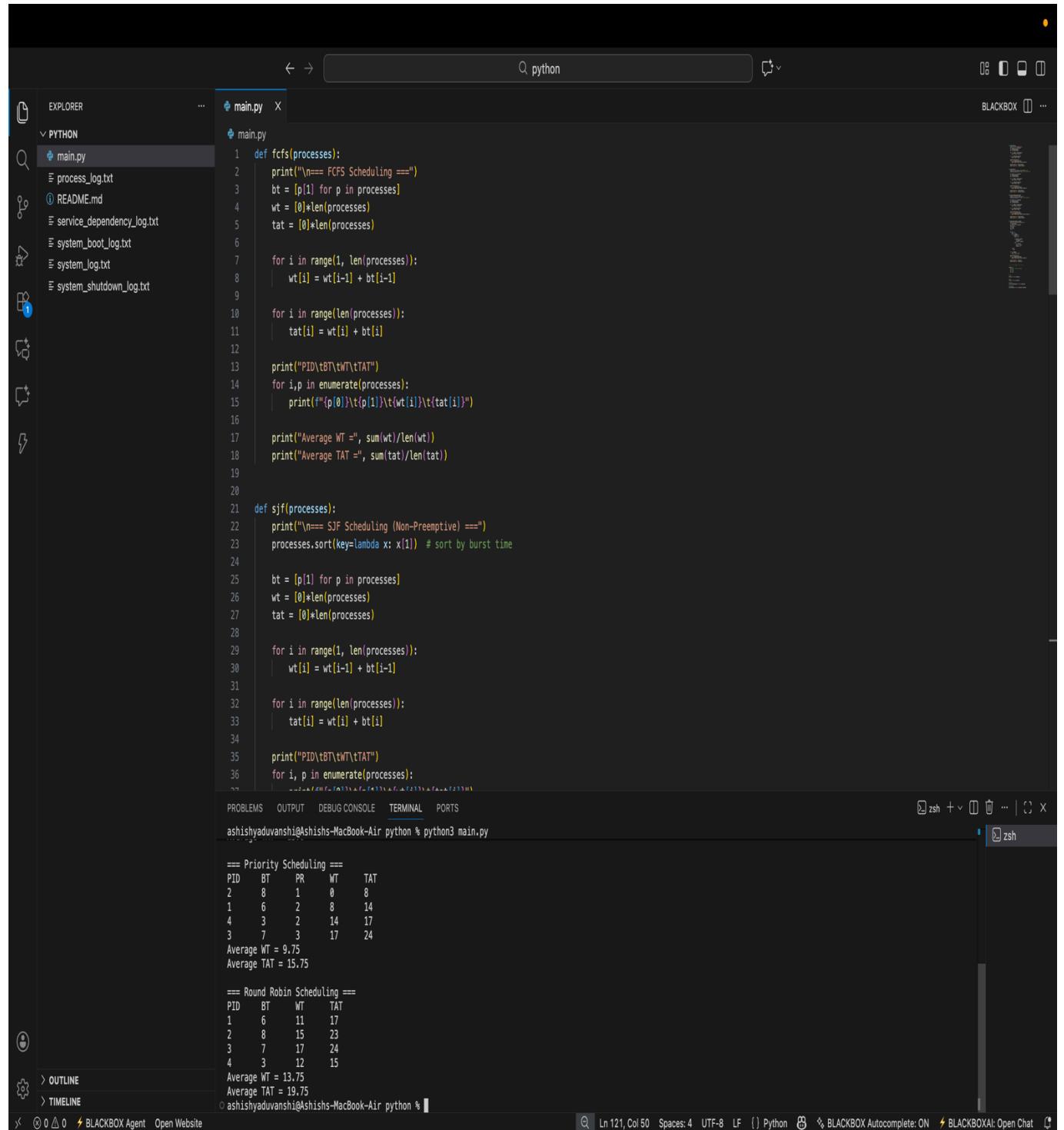
The screenshot shows a dark-themed IDE interface with the following details:

- EXPLORER:** Shows a folder structure under "PYTHON" containing "main.py", "process_log.txt", "README.md", "service_dependency_log.txt", "system_boot_log.txt", "system_log.txt", and "system_shutdown_log.txt".
- MAIN.PY:** The code for the Python script is displayed in the main editor window. It includes logic to check for the presence of "/sys/class/dmi/id/product_name", use "systemd-detect-virt" (Linux), and check for Intel/AMD VT virtualization flags using "lscpu".
- TERMINAL:** The terminal window shows the following session:
 - ashishyaduvanshi@Ashishs-MacBook-Air: ~
 - python % python3 main.py
 - zsh: command not found: python3
 - ashishyaduvanshi@Ashishs-MacBook-Air: ~
 - python % python3 main.py
 - == VM Detection Started ==
 - /bin/sh: lscpu: command not found
 - > No Virtual Machine detected
 - ashishyaduvanshi@Ashishs-MacBook-Air: ~
 - python %
- STATUS BAR:** Shows "Ln 41, Col 24" and "Python" in the status bar.

Task 5: CPU Scheduling Algorithms

Implement FCFS, SJF, Round Robin, and Priority Scheduling algorithms in Python to calculate WT and TAT.

Use existing Round Robin, FCFS, SJF, Priority scheduling Python codes from Lab 3)



The screenshot shows a code editor interface with the following details:

- File Explorer:** Shows a folder structure under 'PYTHON' containing 'main.py', 'process_log.txt', 'README.md', 'service_dependency_log.txt', 'system_boot_log.txt', 'system_log.txt', and 'system_shutdown_log.txt'. 'main.py' is the active file.
- Code Editor:** The content of 'main.py' is displayed, containing Python code for implementing FCFS, SJF, and Round Robin scheduling algorithms. The code calculates Waiting Time (WT) and Turnaround Time (TAT) for processes.
- Terminal:** The terminal window shows the command 'ashishyaduvanshi@ashishs-MacBook-Air python % python3 main.py' being run, followed by the output of the scheduling algorithms.
- Output:** The terminal output shows the results for Priority Scheduling and Round Robin Scheduling.

```
ashishyaduvanshi@ashishs-MacBook-Air python % python3 main.py

    == Priority Scheduling ==
    PID  BT    PR    WT    TAT
    2    8     1     0     8
    1    6     2     8     14
    4    3     2     14    17
    3    7     3     17    24
    Average WT = 9.75
    Average TAT = 15.75

    == Round Robin Scheduling ==
    PID  BT    WT    TAT
    1    6     11    17
    2    8     15    23
    3    7     17    24
    4    3     12    15
    Average WT = 13.75
    Average TAT = 19.75
```