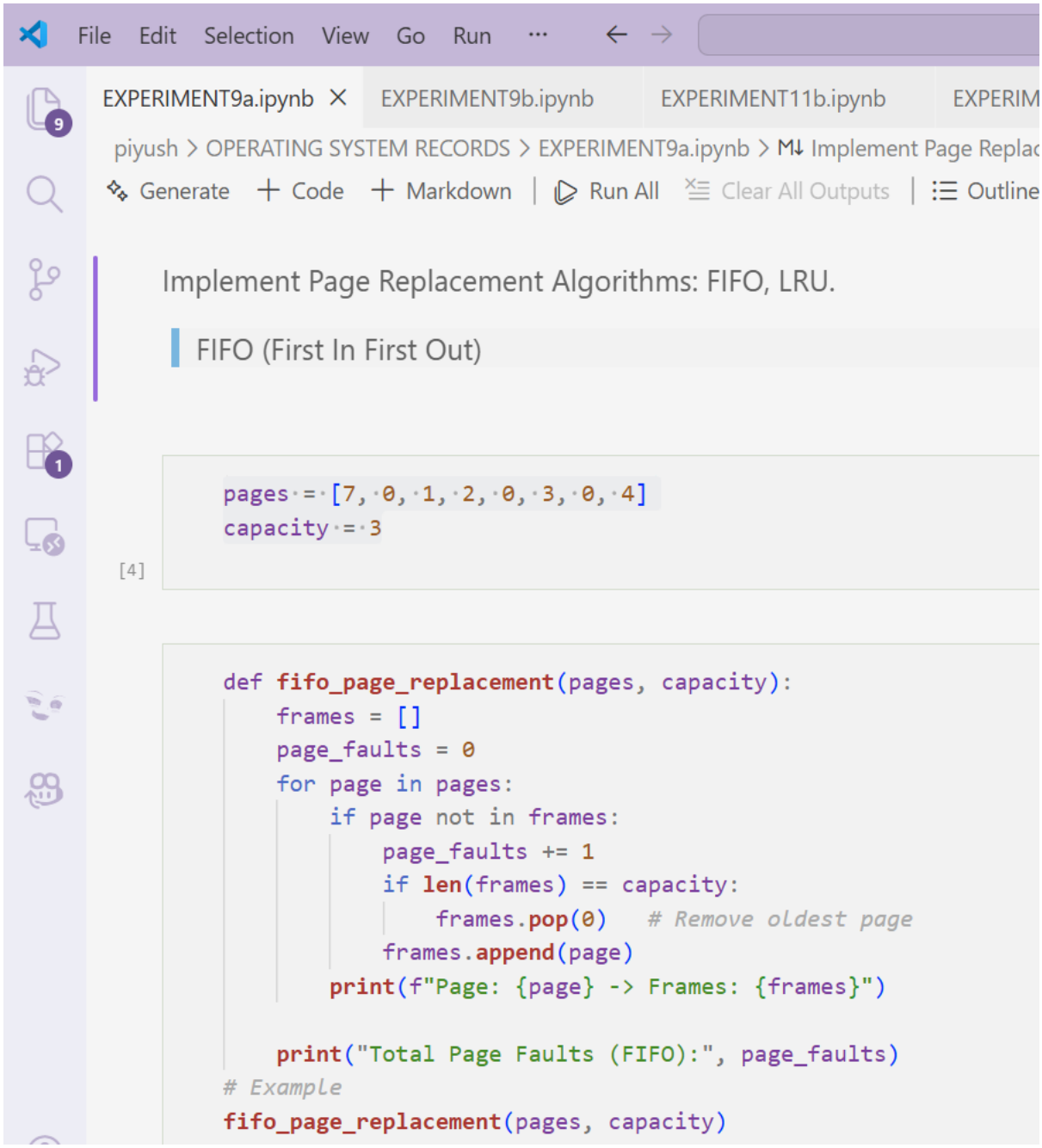
EXPERIMENT 9

AIM - Implement Page Replacement Algorithms: FIFO, LRU.

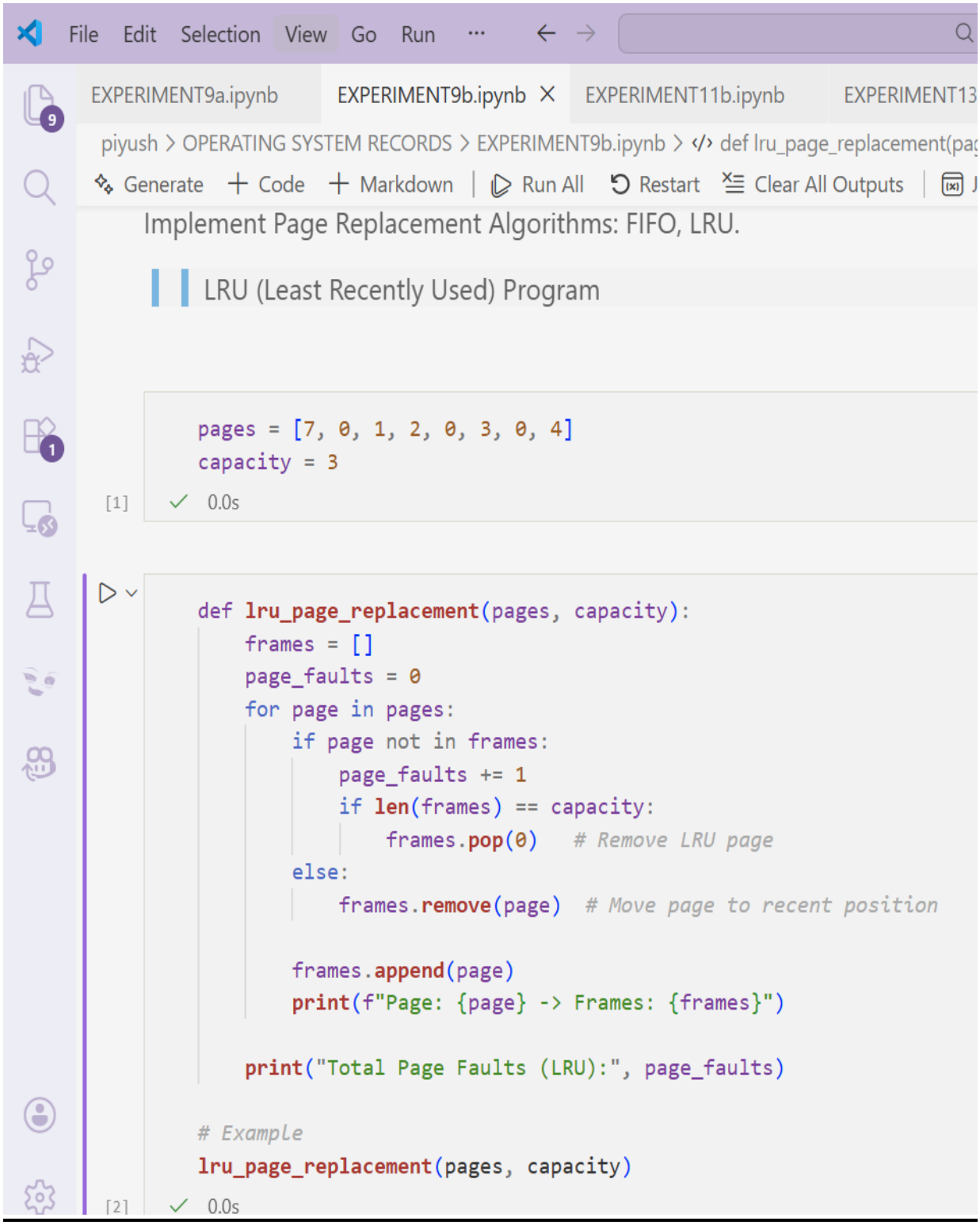
FIFO (First In First Out) - SOURCE CODE -



OUTPUT –

```
Page: 7 -> Frames: [7]
Page: 0 -> Frames: [7, 0]
Page: 1 -> Frames: [7, 0, 1]
Page: 2 -> Frames: [0, 1, 2]
Page: 0 -> Frames: [0, 1, 2]
Page: 3 -> Frames: [1, 2, 3]
Page: 0 -> Frames: [2, 3, 0]
Page: 4 -> Frames: [3, 0, 4]
Total Page Faults (FIFO): 7
```

LRU (Least Recently Used) - SOURCE CODE-



OUTPUT -

```
Page: 7 -> Frames: [7]
Page: 0 -> Frames: [7, 0]
Page: 1 -> Frames: [7, 0, 1]
Page: 2 -> Frames: [0, 1, 2]
Page: 0 -> Frames: [1, 2, 0]
Page: 3 -> Frames: [2, 0, 3]
Page: 0 -> Frames: [2, 3, 0]
Page: 4 -> Frames: [3, 0, 4]
Total Page Faults (LRU): 6
```

EXPERIMENT - 10

AIM - Extend LRU with a prediction model (predict next reference using past data).

SOURCE CODE -

File Edit Selection View Go Run ··· MCA_

EXPERIMENT9b.ipynb | experiment10.ipynb | EXPERIMENT10.ipynb ● | EXPERIMENT11b.ipynb | EX

piyush > OPERATING SYSTEM RECORDS > EXPERIMENT10.ipynb > from collections import defaultdict

Generate + Code + Markdown | Run All Restart Clear All Outputs | Jupyter Variabl

Generate + Code +

Extend LRU with a prediction model (predict next reference using past data).

```python
from collections import defaultdict

def predictive_lru(pages, capacity):
    frames = []
    page_faults = 0
    transition_count = defaultdict(lambda: defaultdict(int))

    prev_page = None

    def predict_next(page):
        if page not in transition_count:
            return None
        return max(transition_count[page], key=transition_count[page].get)

    for page in pages:
        # Update transition history
        if prev_page is not None:
            transition_count[prev_page][page] += 1

        prediction = predict_next(prev_page)

        if page not in frames:
            page_faults += 1
            if len(frames) == capacity:
                # Eviction Logic
                for p in frames:
                    if p != prediction:
                        frames.remove(p)
                        break
            frames.append(page)
        else:
            frames.remove(page)
            frames.append(page)  # Move to most recent

        print(f"Page: {page}, Prediction: {prediction}, Frames: {frames}")
        prev_page = page

    print("Total Page Faults (Predictive LRU):", page_faults)


# Example
pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3]
capacity = 3
predictive_lru(pages, capacity)
```

OUTPUT:

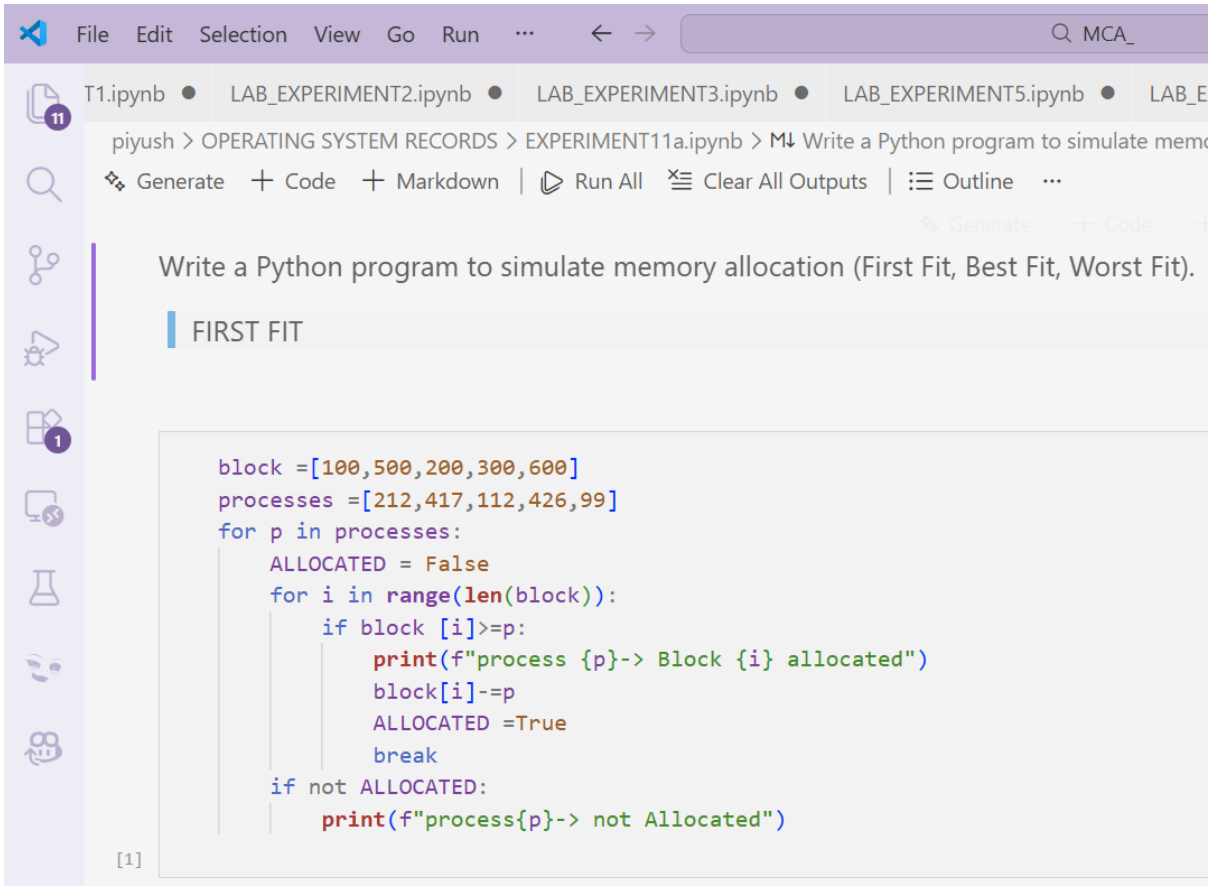
```
Page: 7, Prediction: None, Frames: [7]
Page: 0, Prediction: 0, Frames: [7, 0]
Page: 1, Prediction: 1, Frames: [7, 0, 1]
Page: 2, Prediction: 2, Frames: [0, 1, 2]
Page: 0, Prediction: 0, Frames: [1, 2, 0]
Page: 3, Prediction: 1, Frames: [1, 0, 3]
Page: 0, Prediction: 0, Frames: [1, 3, 0]
Page: 4, Prediction: 1, Frames: [1, 0, 4]
Page: 2, Prediction: 2, Frames: [0, 4, 2]
Page: 3, Prediction: 0, Frames: [0, 2, 3]
Total Page Faults (Predictive LRU): 8
```

EXPERIMENT-11

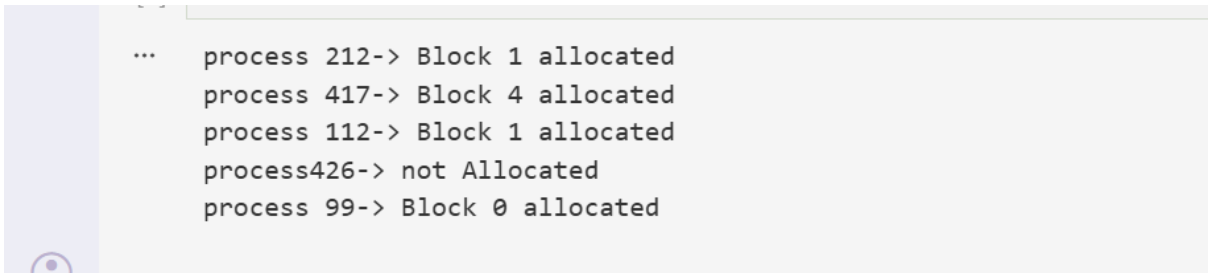
AIM: Write a Python program to simulate memory allocation (First Fit, Best Fit, Worst Fit).

- ***FIRST FIT***

SOURCE CODE-

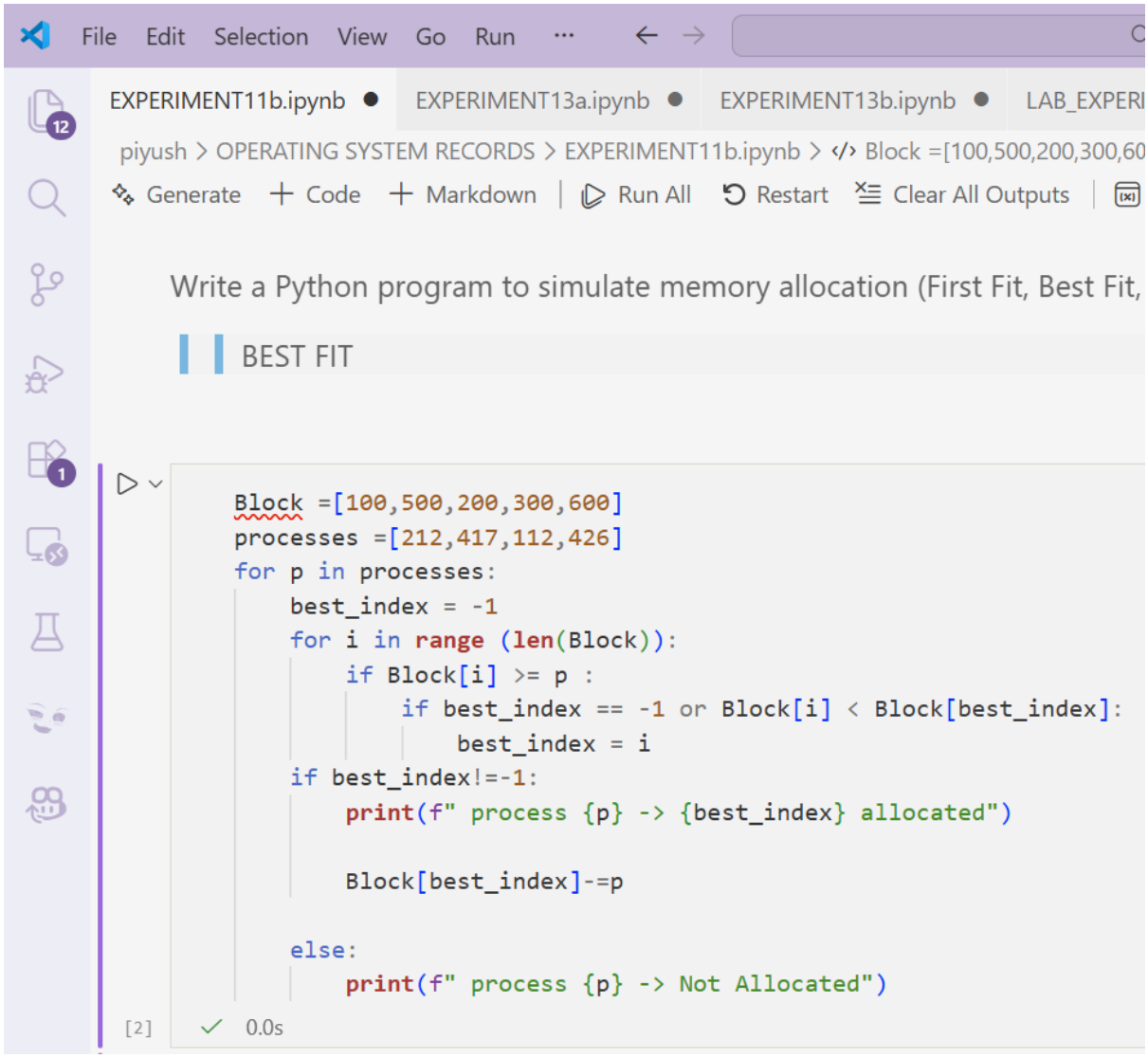


OUTPUT-

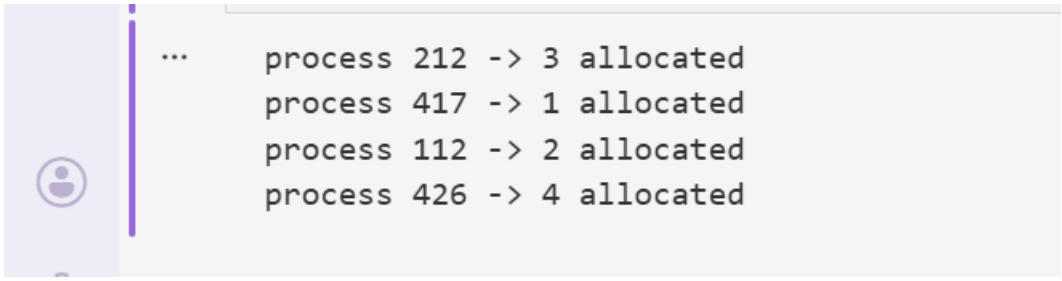


- ***BEST FIT***

SOURCE CODE -

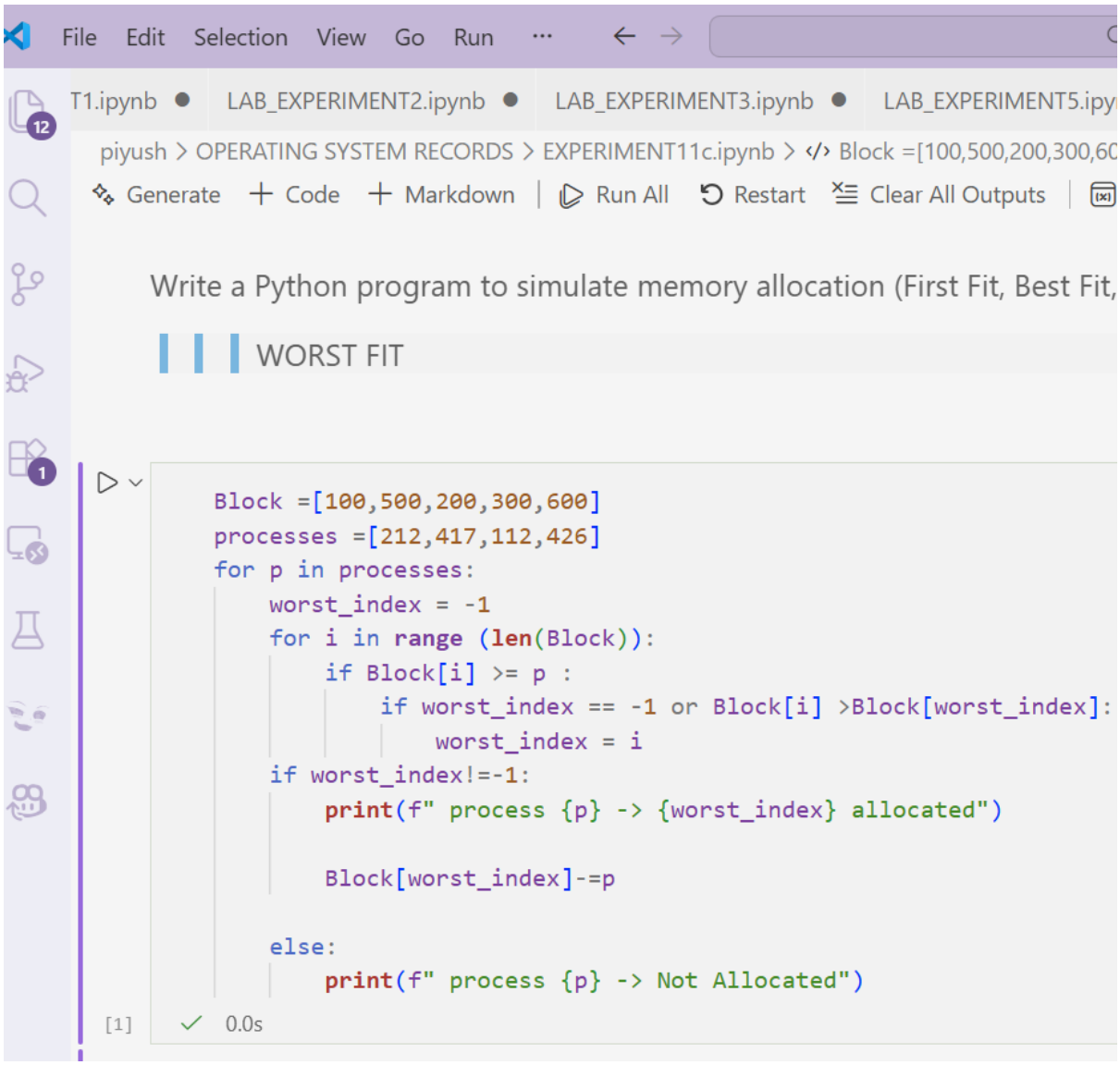


OUTPUT:

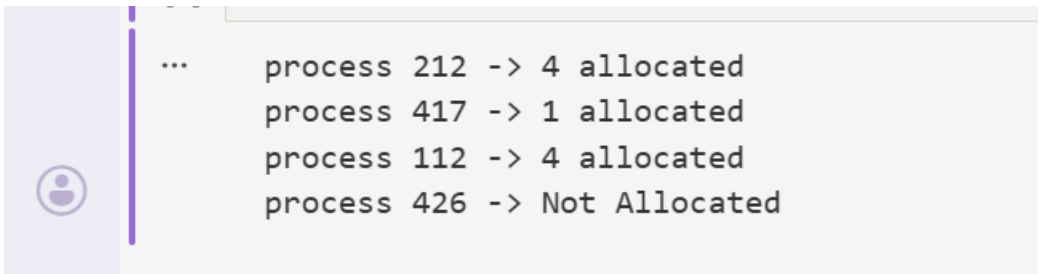


- ***WORST FIT***

SOURCE CODE –



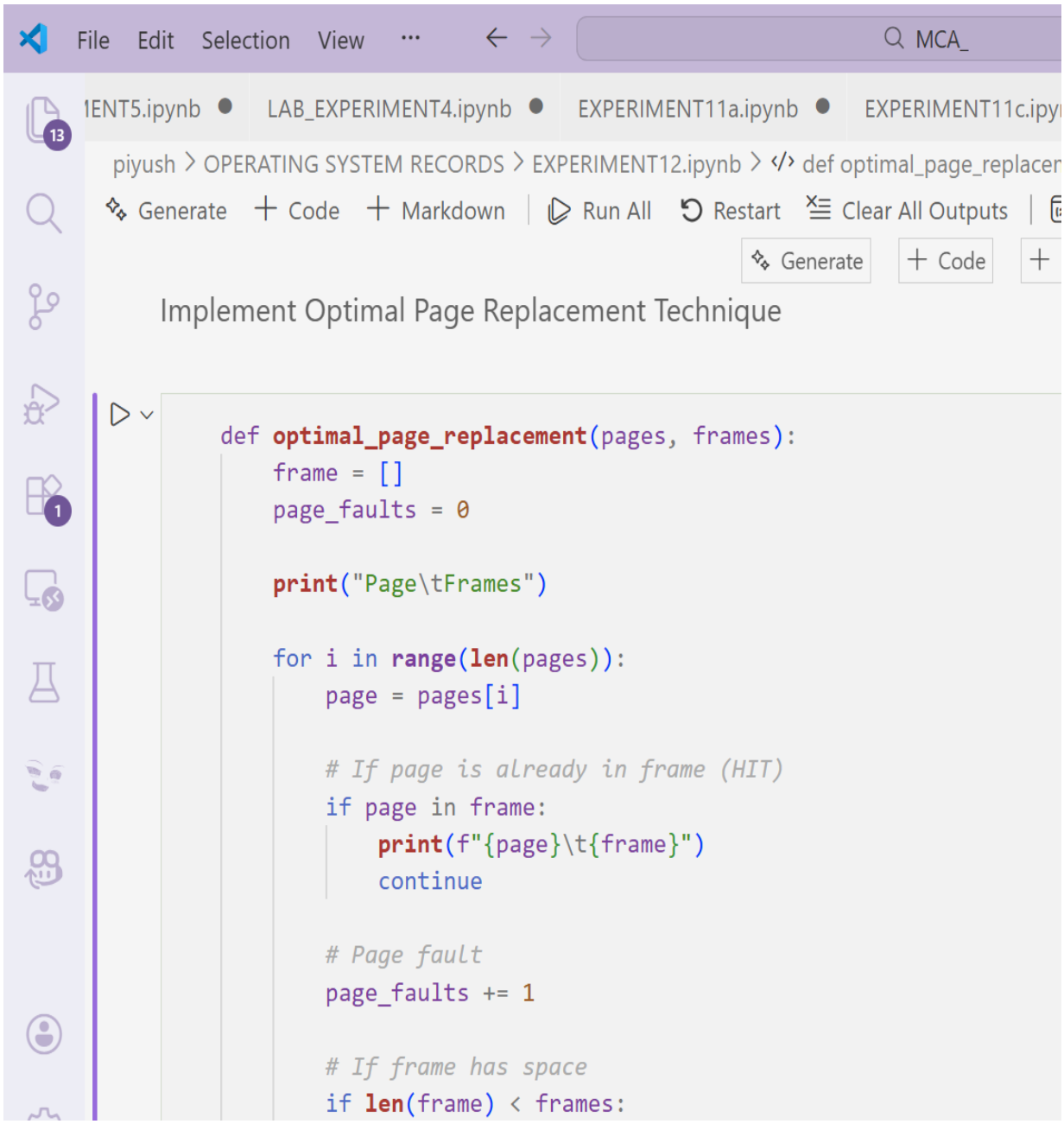
OUTPUT -



EXPERIMENT - 12

AIM: Implement Optimal Page Replacement Technique

SOURCE CODE-



```
                frame.append(page)
            else:
                # Find page with farthest future use
                future_use = {}

                for f in frame:
                    if f in pages[i+1:]:
                        future_use[f] = pages[i+1:].index(f)
                    else:
                        future_use[f] = float('inf')

                # Replace page with maximum future index
                replace_page = max(future_use, key=future_use.get)
                frame[frame.index(replace_page)] = page

            print(f"{page}\t{frame}")

        print("\nTotal Page Faults:", page_faults)


    # -------- Example --------
    pages = [7, 0, 1, 2, 0, 3, 0, 4, 2, 3, 0, 3, 2]
    frames = 3
    optimal_page_replacement(pages, frames)
```

OUTPUT-

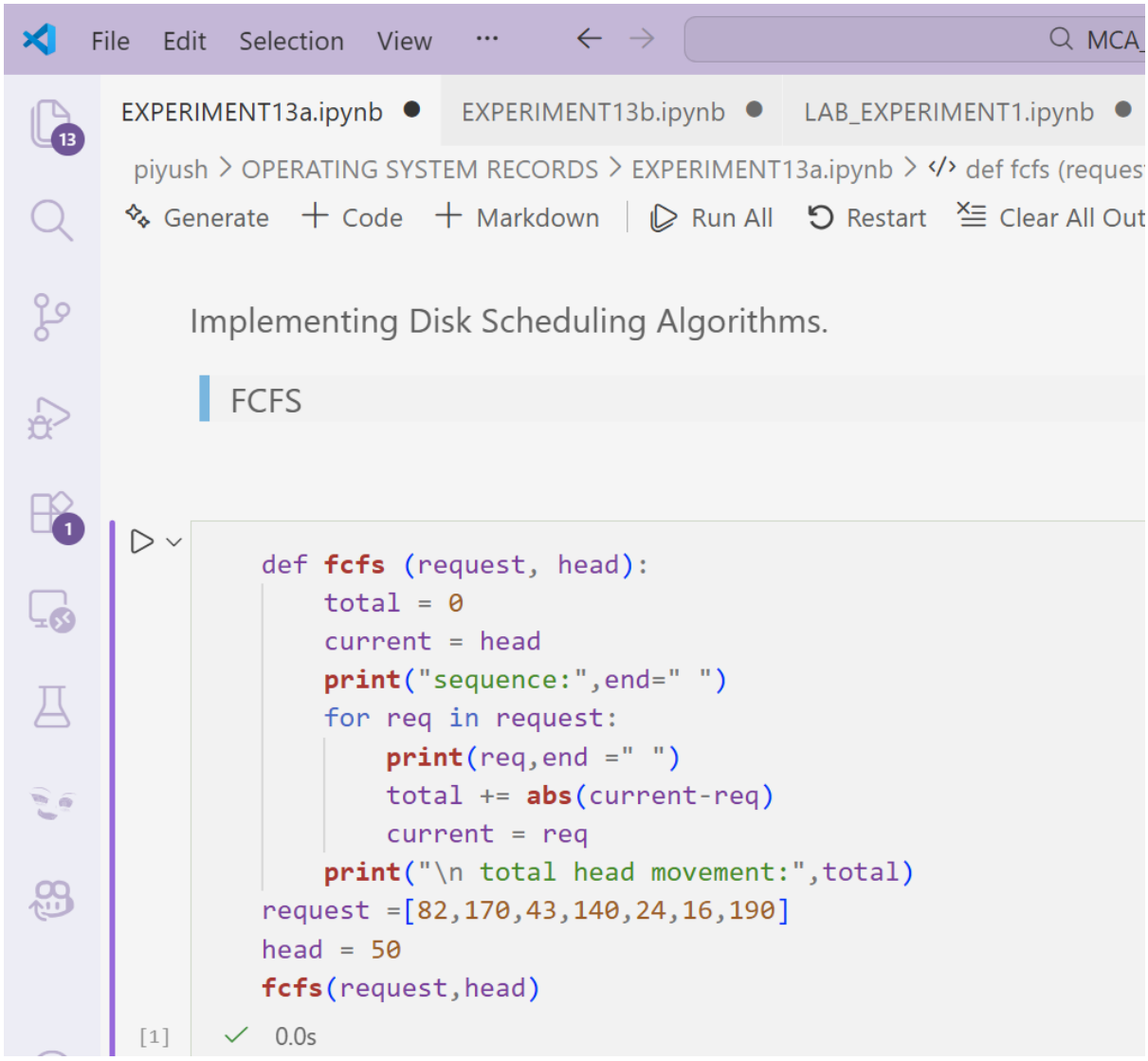
```
...     Page    Frames
        7       [7]
        0       [7, 0]
        1       [7, 0, 1]
        2       [2, 0, 1]
        0       [2, 0, 1]
        3       [2, 0, 3]
        0       [2, 0, 3]
        4       [2, 4, 3]
        2       [2, 4, 3]
        3       [2, 4, 3]
        0       [2, 0, 3]
        3       [2, 0, 3]
        2       [2, 0, 3]

        Total Page Faults: 7
```

EXPERIMENT - 13

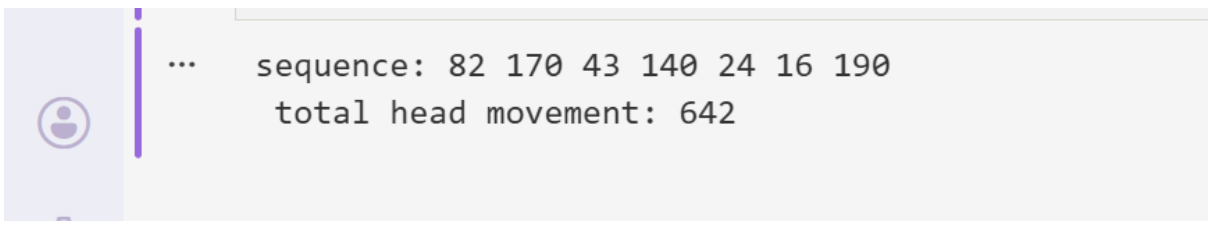
AIM: Implement Disk Scheduling Algorithms: FCFS, SSTF.

- **FCFS (First Come First Serve)**

SOURCE CODE –

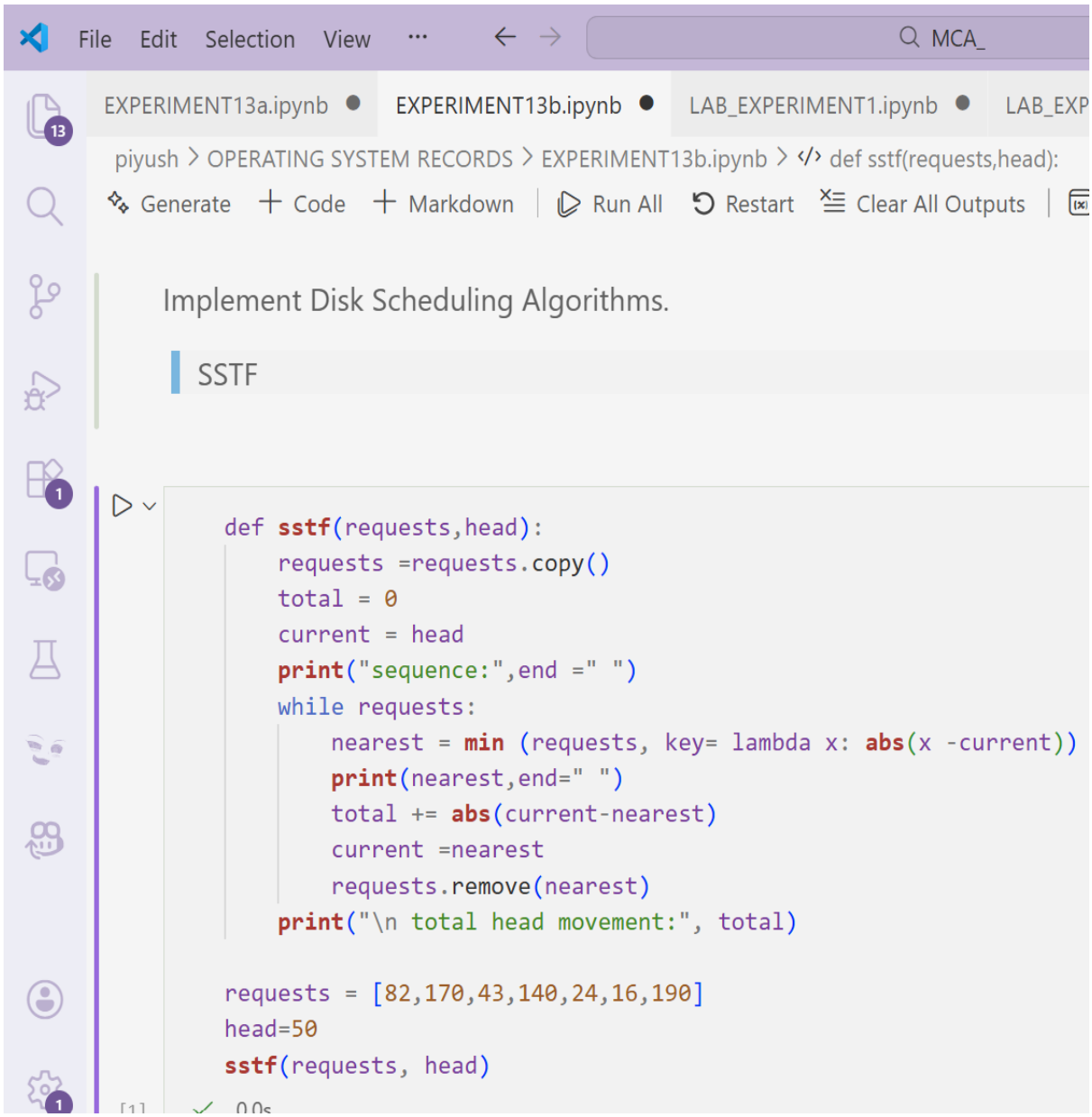


Output:

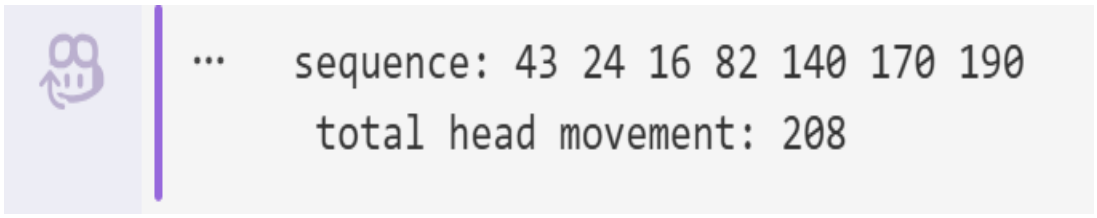


- ***SSTF (Shortest Seek Time First)***

SOURCE CODE –



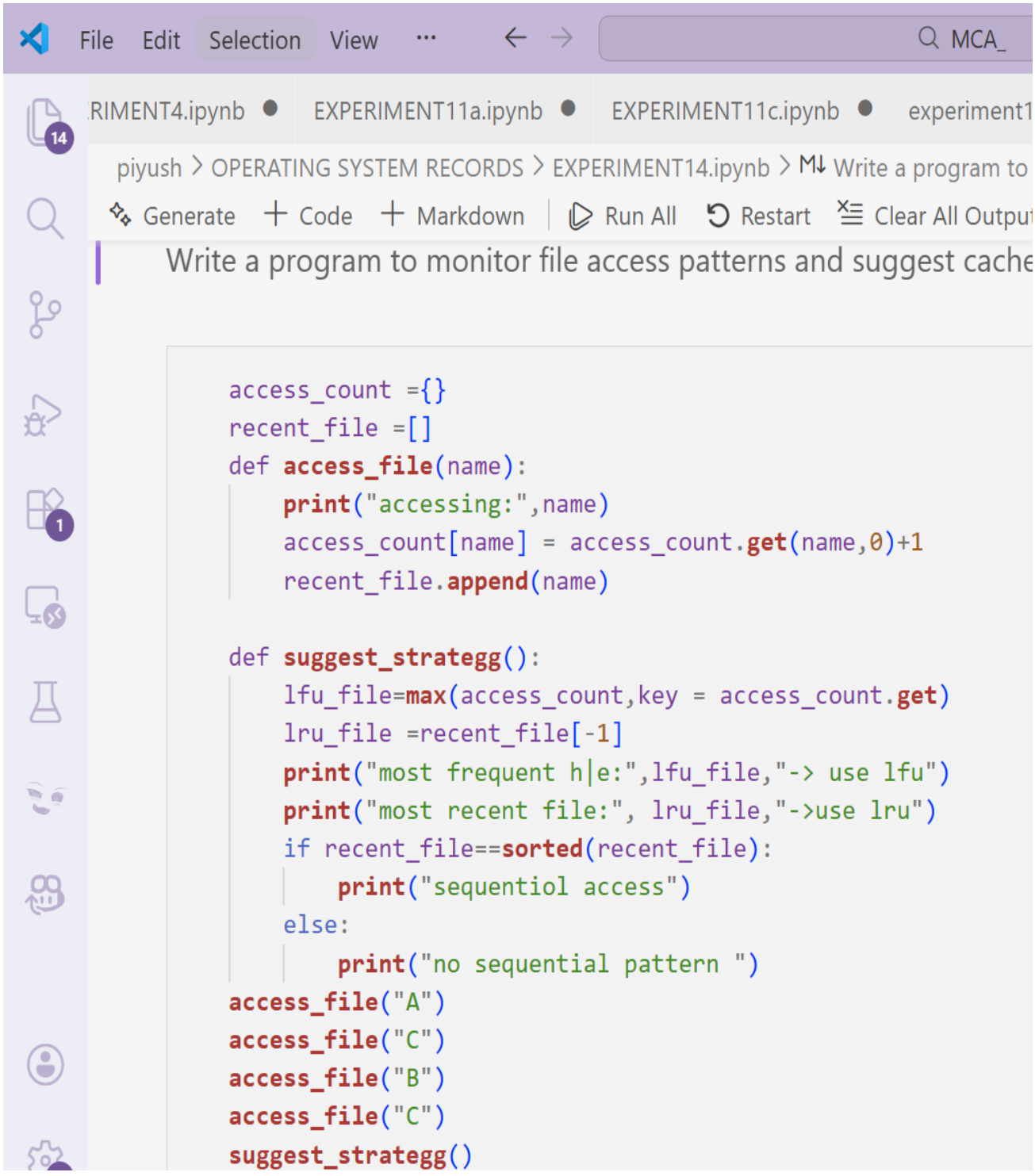
OUTPUT:



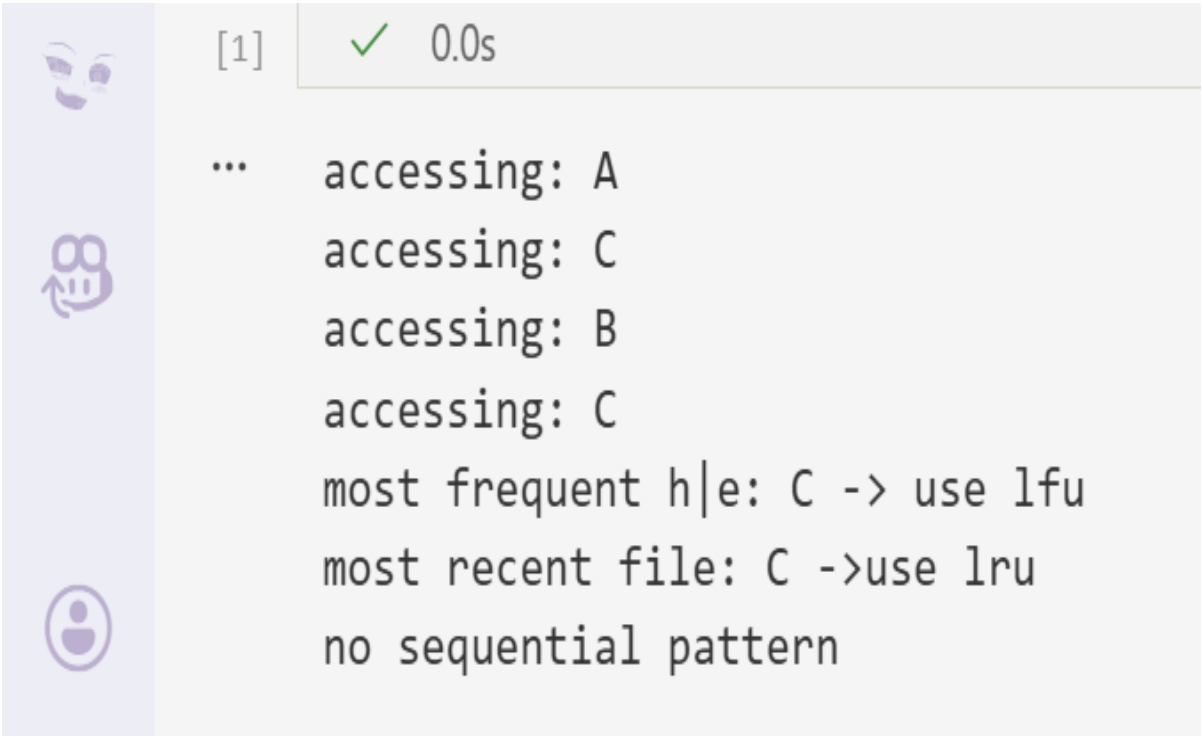
EXPERIMENT - 14

AIM: Write a program to monitor file access patterns and suggest cache strategy.

SOURCE CODE-



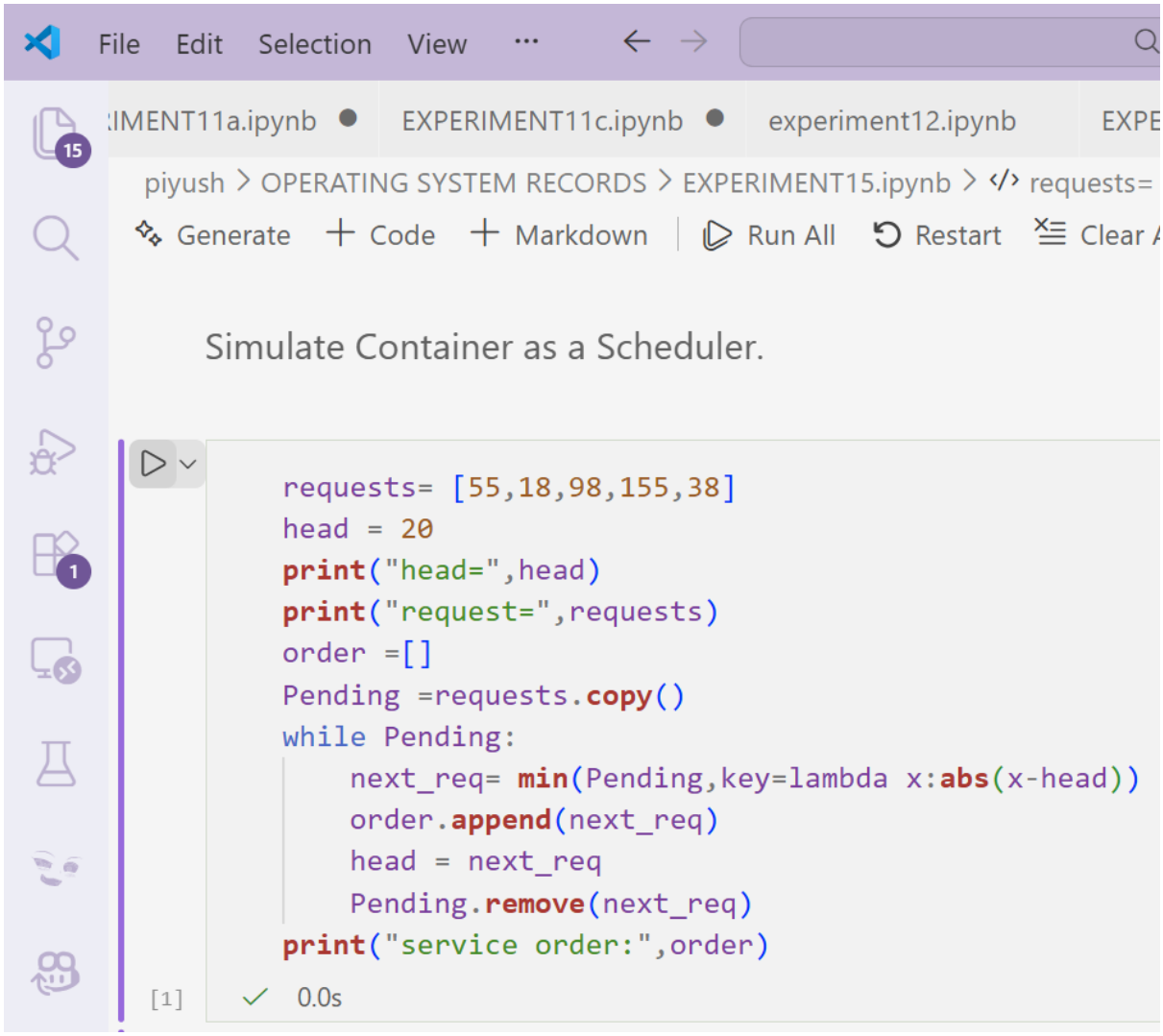
OUTPUT -


[1]
0.0s
accessing: A
accessing: C
accessing: B
accessing: C
most frequent h|e: C -> use lfu
most recent file: C ->use lru
no sequential pattern

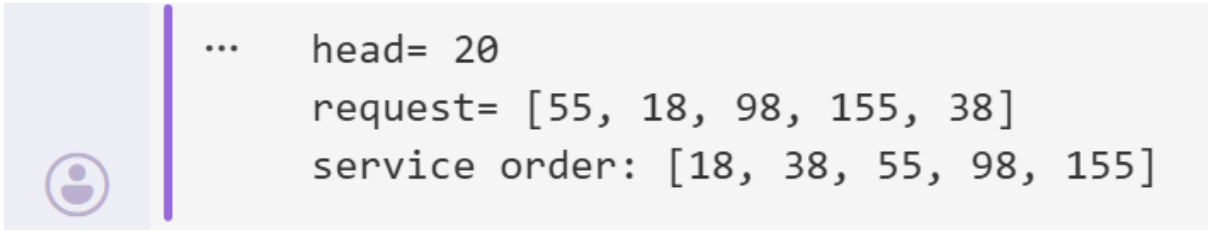
EXPERIMENT - 15

AIM: Simulate Container as a Scheduler.

SOURCE CODE-



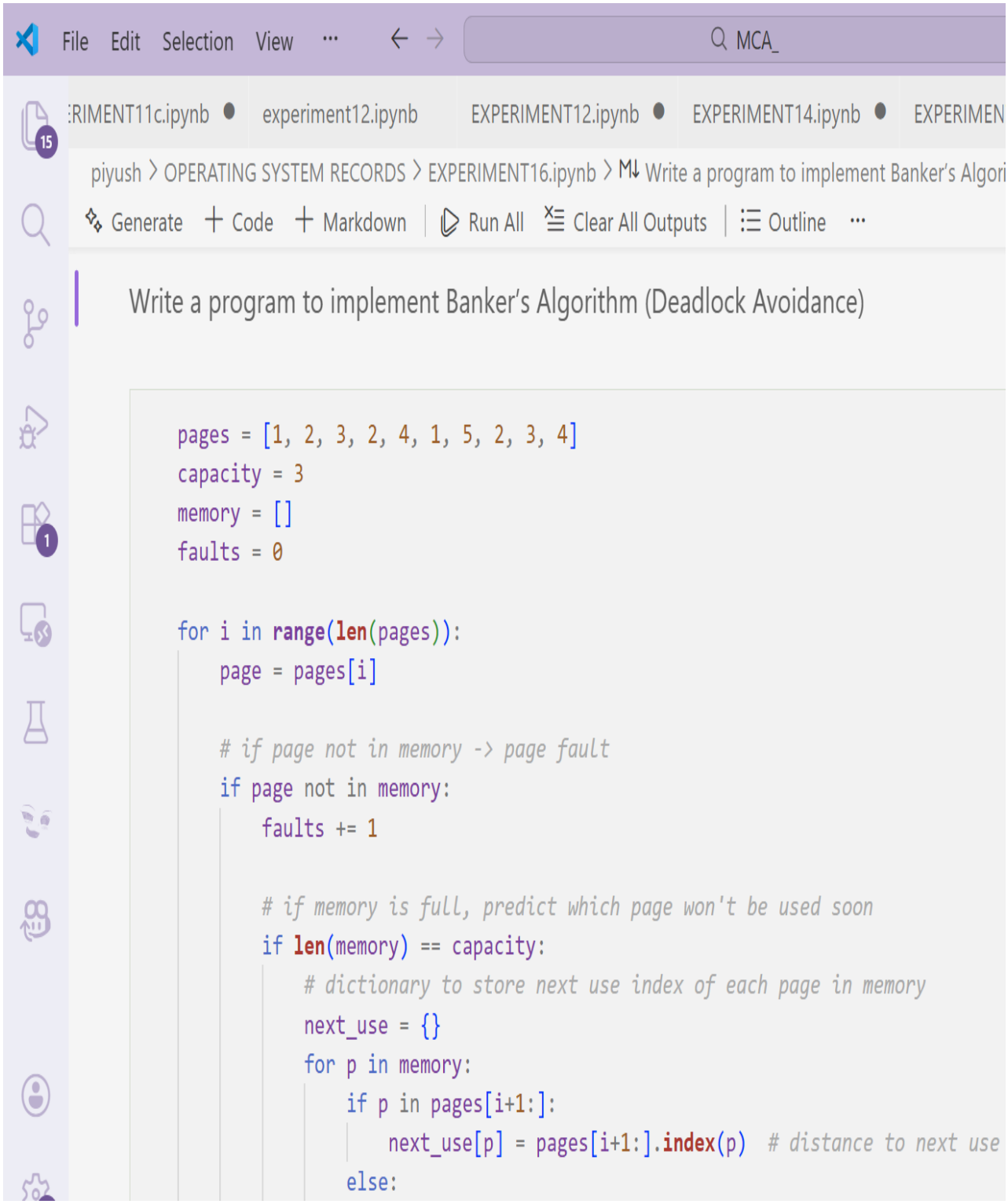
OUTPUT:



EXPERIMENT - 16

AIM: I Write a program to implement Banker's Algorithm (Deadlock Avoidance)

SOURCE CODE-



```
                    next_use[p] = float('inf')  # not used again

                # page with the farthest next use or never used again
                page_to_remove = max(next_use, key=next_use.get)
                memory.remove(page_to_remove)

            # add the new page
            memory.append(page)

        else:
            # page hit -> move it to end (most recently used)
            memory.remove(page)
            memory.append(page)

        print(f"Step {i+1}: Page={page} | Memory={memory}")

    print("\nTotal Page Faults:", faults)
```

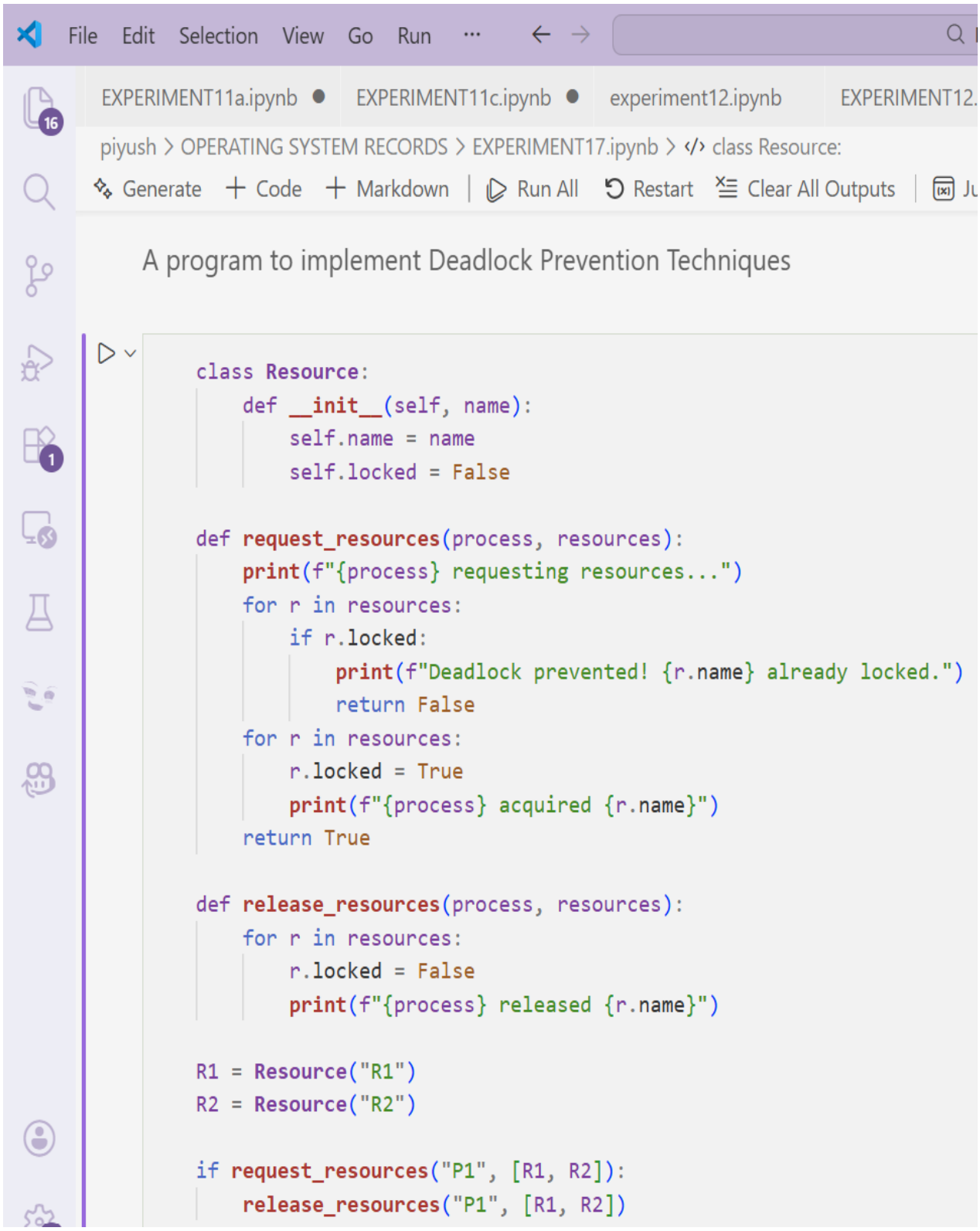
OUTPUT -

```
Step 1: Page=1 | Memory=[1]
Step 2: Page=2 | Memory=[1, 2]
Step 3: Page=3 | Memory=[1, 2, 3]
Step 4: Page=2 | Memory=[1, 3, 2]
Step 5: Page=4 | Memory=[1, 2, 4]
Step 6: Page=1 | Memory=[2, 4, 1]
Step 7: Page=5 | Memory=[2, 4, 5]
Step 8: Page=2 | Memory=[4, 5, 2]
Step 9: Page=3 | Memory=[4, 2, 3]
Step 10: Page=4 | Memory=[2, 3, 4]

Total Page Faults: 6
```

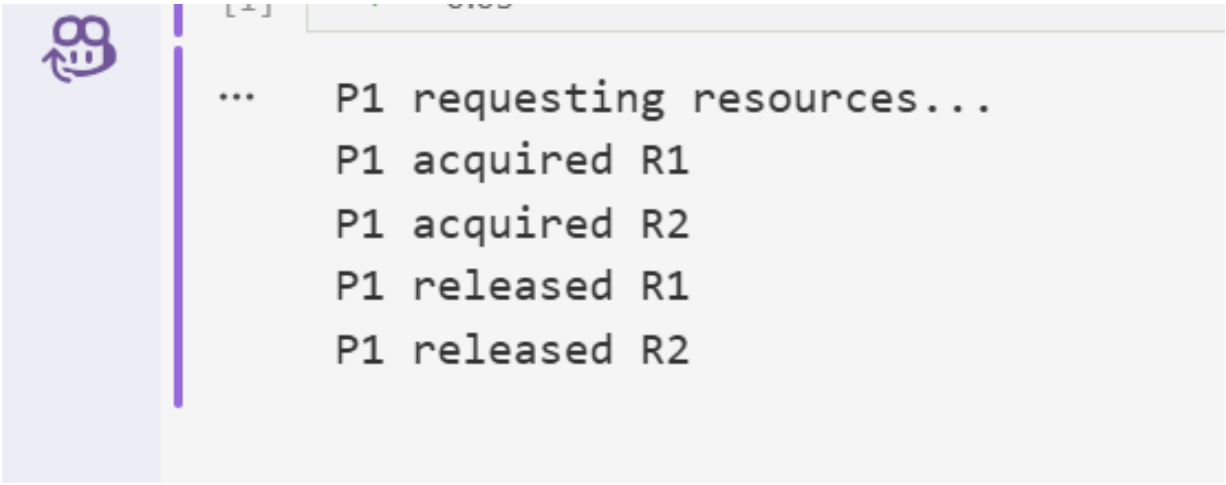
EXPERIMENT - 17

AIM: Write a program to implement Deadlock Prevention Techniques

SOURCE CODE-



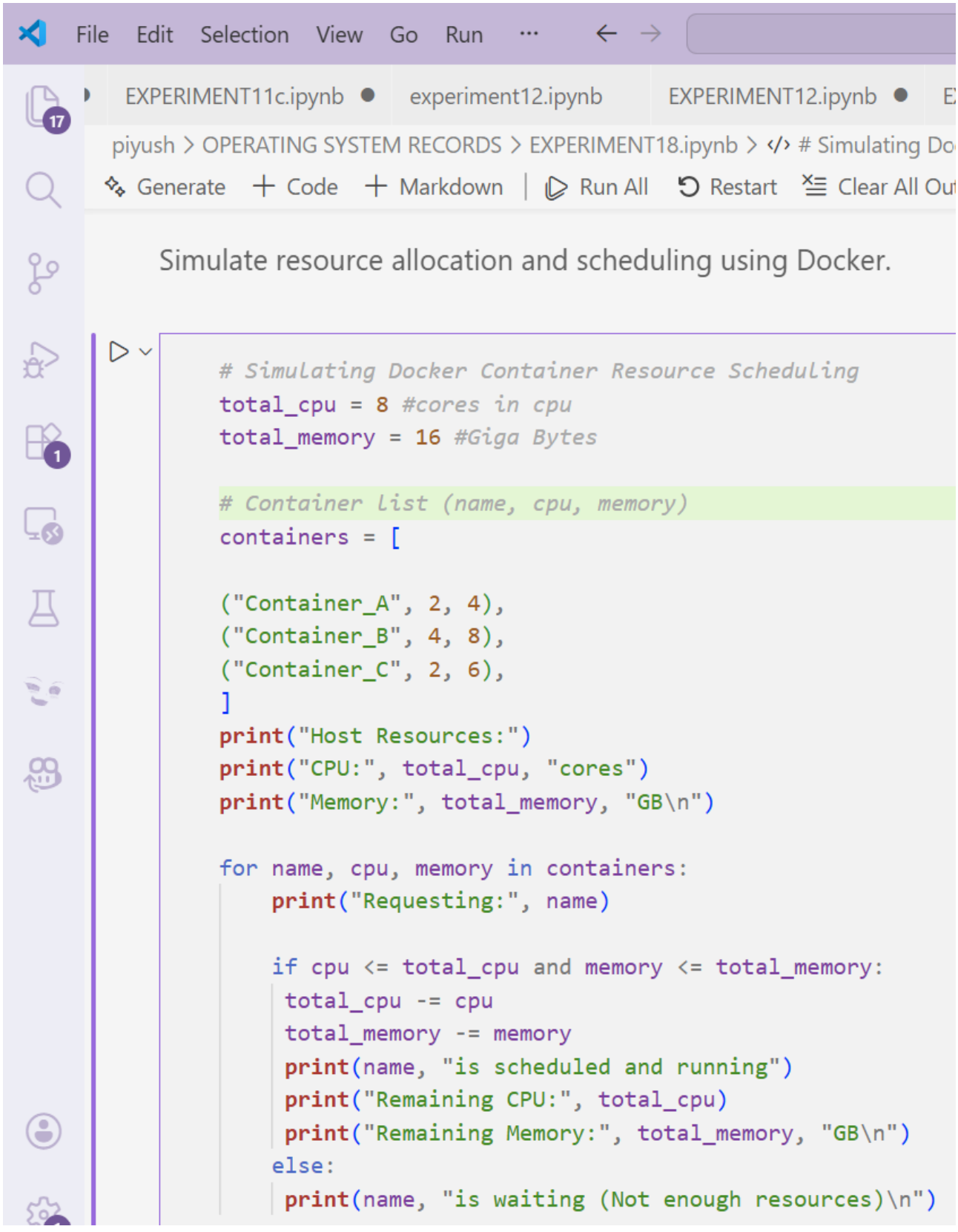
OUTPUT -


P1 requesting resources...
P1 acquired R1
P1 acquired R2
P1 released R1
P1 released R2

EXPERIMENT - 18

AIM: Simulate resource allocation and scheduling using Docker.

SOURCE CODE –



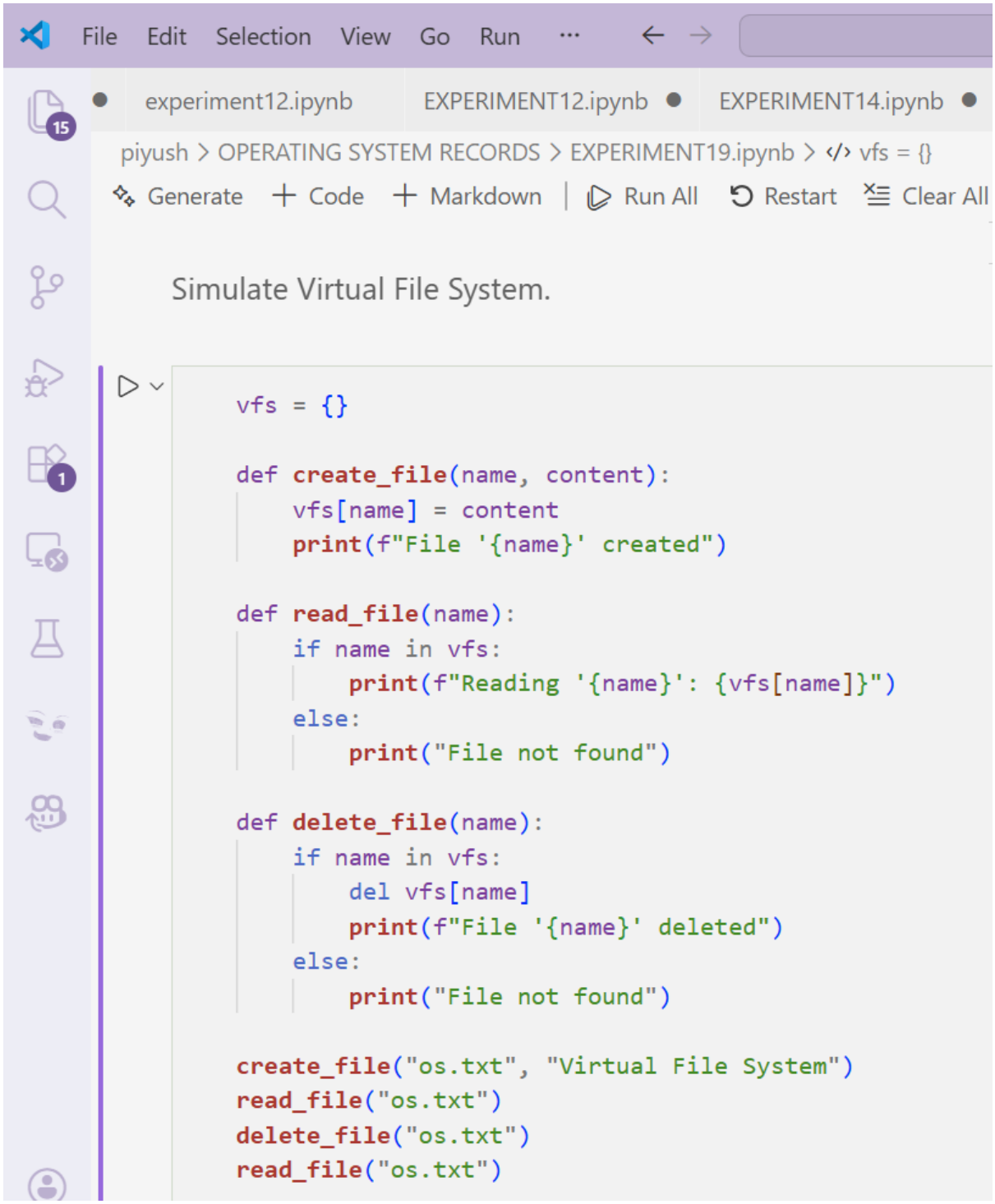
OUTPUT -

```
Host Resources:
CPU: 8 cores
Memory: 16 GB

Requesting: Container_A
Container_A is scheduled and running
Remaining CPU: 6
Remaining Memory: 12 GB

Requesting: Container_B
Container_B is scheduled and running
Remaining CPU: 2
Remaining Memory: 4 GB

Requesting: Container_C
Container_C is waiting (Not enough resources)
```

EXPERIMENT - 19

AIM: Simulate Virtual File System.

SOURCE CODE-



OUTPUT:

```
File 'os.txt' created
Reading 'os.txt': Virtual File System
File 'os.txt' deleted
File not found
```