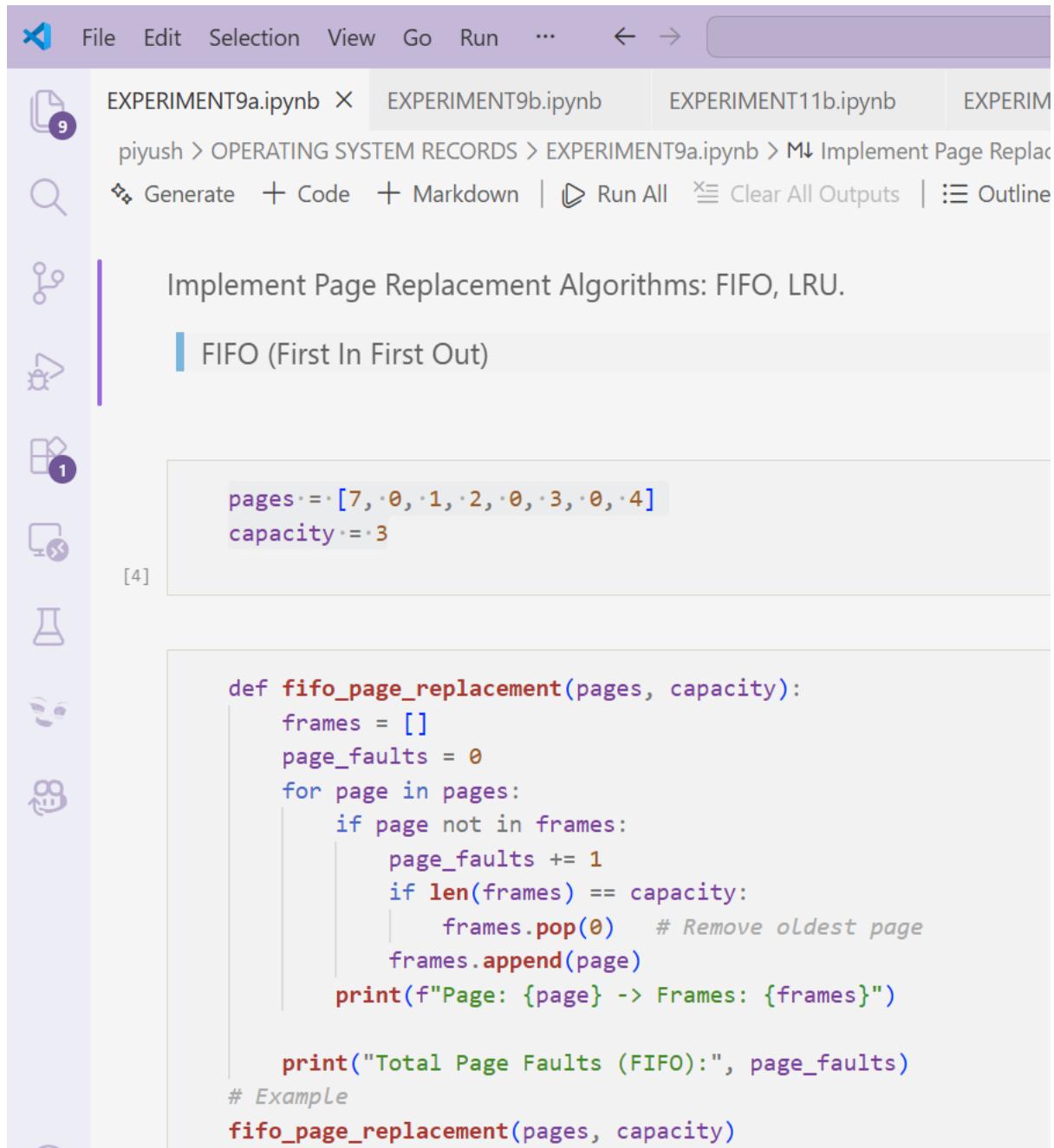


## EXPERIMENT 9

**AIM - Implement Page Replacement Algorithms: FIFO, LRU.**

**FIFO (First In First Out) - SOURCE CODE -**



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Toolbar:** EXPERIMENT9a.ipynb (active), EXPERIMENT9b.ipynb, EXPERIMENT11b.ipynb, EXPERIMENT11a.ipynb, piyush > OPERATING SYSTEM RECORDS > EXPERIMENT9a.ipynb > M4 Implement Page Replacement
- Search:** Generate, + Code, + Markdown, Run All, Clear All Outputs, Outline
- Left Sidebar:** Document icon (9), Cell icon (1), Cell icon with X, Cell icon with 1, Cell icon with X, Cell icon with 1, Cell icon with X, Cell icon with 1.
- Main Area:**
  - Title:** Implement Page Replacement Algorithms: FIFO, LRU.
  - Section:** FIFO (First In First Out)
  - Code Cell 1:** [4] pages = [7, 0, 1, 2, 0, 3, 0, 4]  
capacity = 3
  - Code Cell 2:** def fifo\_page\_replacement(pages, capacity):  
 frames = []  
 page\_faults = 0  
 for page in pages:  
 if page not in frames:  
 page\_faults += 1  
 if len(frames) == capacity:  
 frames.pop(0) # Remove oldest page  
 frames.append(page)  
 print(f"Page: {page} -> Frames: {frames}")  
  
 print("Total Page Faults (FIFO):", page\_faults)  
# Example  
fifo\_page\_replacement(pages, capacity)

## **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***

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ..., ⏪, ⏩, Search icon.
- Recent Notebooks:** EXPERIMENT9a.ipynb (9), EXPERIMENT9b.ipynb X, EXPERIMENT11b.ipynb, EXPERIMENT13.
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT9b.ipynb > ↻ def lru\_page\_replacement(pages, capacity):  
Generate + Code + Markdown | Run All | Restart | Clear All Outputs | J
- Text Area:** Implement Page Replacement Algorithms: FIFO, LRU.  
| LRU (Least Recently Used) Program
- Code Cell 1:** pages = [7, 0, 1, 2, 0, 3, 0, 4]  
capacity = 3  
[1] ✓ 0.0s
- Code Cell 2:** def lru\_page\_replacement(pages, capacity):  
 frames = []  
 page\_faults = 0  
 for page in pages:  
 if page not in frames:  
 page\_faults += 1  
 if len(frames) == capacity:  
 frames.pop(0) # Remove LRU page  
 else:  
 frames.remove(page) # Move page to recent position  
  
 frames.append(page)  
 print(f"Page: {page} -> Frames: {frames}")  
  
 print("Total Page Faults (LRU):", page\_faults)  
  
# Example  
lru\_page\_replacement(pages, capacity)  
[2] ✓ 0.0s

## **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 -

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Search Bar:** MCA\_
- Recent Notebooks:** EXPERIMENT9b.ipynb, experiment10.ipynb, EXPERIMENT10.ipynb (selected), EXPERIMENT11b.ipynb, EX
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT10.ipynb > </> from collections import defaultdict  
◆ Generate + Code + Markdown | ▶ Run All ⚡ Restart ✖ Clear All Outputs | 📈 Jupyter Variables
- Text Area:** Extend LRU with a prediction model (predict next reference using past data).
- Code Block:** The code implements a Predictive LRU algorithm. It uses a defaultdict to store transition counts and a list of frames to manage memory. The `predictive\_lru` function takes `pages` and `capacity` as input. It initializes `frames`, `page\_faults`, and `transition\_count`. It then iterates through `pages`, updating the transition history and predicting the next page. If a page is not in `frames`, it increments `page\_faults` and checks if `len(frames)` equals `capacity`. If so, it uses an eviction logic to remove pages that are not predicted. Finally, it prints the page, prediction, and frames, and outputs the total page faults.

```
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

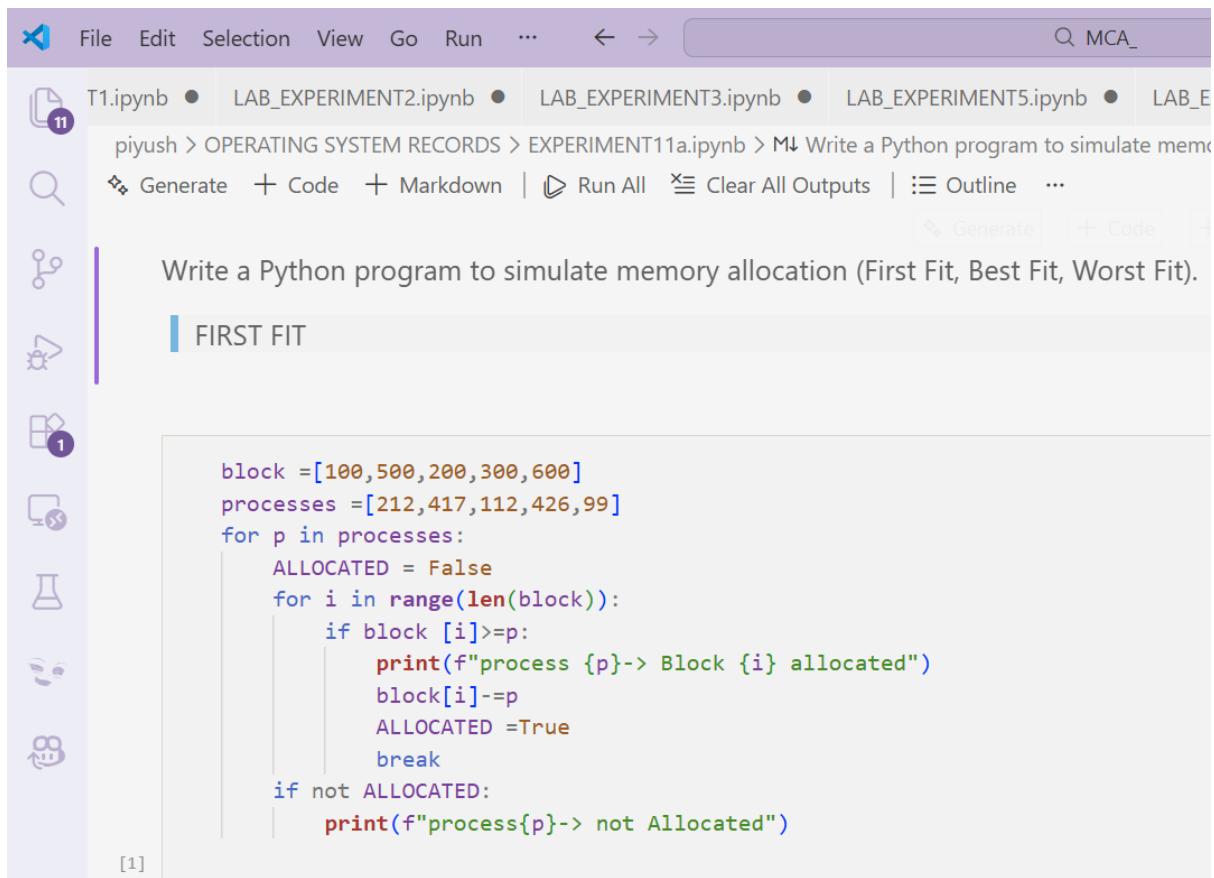


## **EXPERIMENT-11**

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

➤ **FIRST FIT**

**SOURCE CODE-**

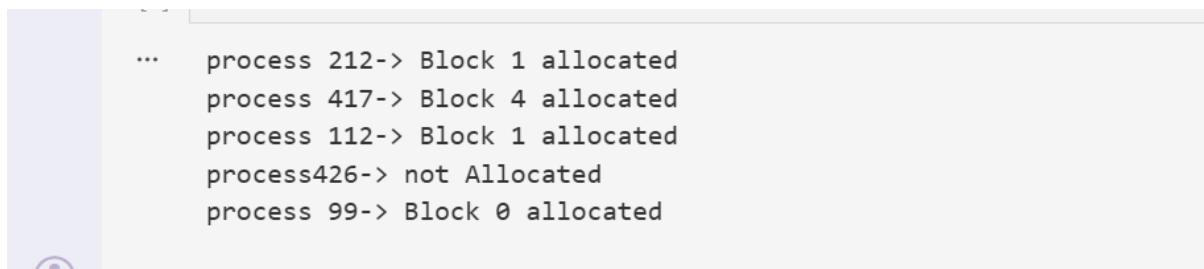


The screenshot shows a Jupyter Notebook interface. The top bar includes File, Edit, Selection, View, Go, Run, and other navigation options. A search bar on the right contains the text "MCA\_". Below the bar, a list of notebooks is visible: T1.ipynb, LAB\_EXPERIMENT2.ipynb, LAB\_EXPERIMENT3.ipynb, LAB\_EXPERIMENT5.ipynb, and LAB\_E. The main area displays a cell with the following Python code:

```
block =[100,500,200,300,600]
processes =[212,417,112,426,99]
for p in processes:
    ALLOCATED = False
    for i in range(len(block)):
        if block [i]>=p:
            print(f"process {p}-> Block {i} allocated")
            block[i]-=p
            ALLOCATED =True
            break
    if not ALLOCATED:
        print(f"process{p}-> not Allocated")
```

The cell has a status of [1].

**OUTPUT**



The output cell shows the results of the memory allocation process:

```
... process 212-> Block 1 allocated
process 417-> Block 4 allocated
process 112-> Block 1 allocated
process426-> not Allocated
process 99-> Block 0 allocated
```

➤ **BEST FIT**

## SOURCE CODE -

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ...
- Recent Notebooks:** EXPERIMENT11b.ipynb, EXPERIMENT13a.ipynb, EXPERIMENT13b.ipynb, LAB\_EXPERI
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT11b.ipynb > Block =[100,500,200,300,600]
- Cell Type:** Generate + Code + Markdown | Run All | Restart | Clear All Outputs |
- Text Area:** Write a Python program to simulate memory allocation (First Fit, Best Fit, Worst Fit)
- Section Header:** BEST FIT
- Code Cell:** Contains Python code for Best Fit memory allocation.
- Output Cell:** Shows the output of the code execution.

```
Block =[100,500,200,300,600]
processes =[212,417,112,426]
for p in processes:
    best_index = -1
    for i in range (len(Block)):
        if Block[i] >= p :
            if best_index == -1 or Block[i] < Block[best_index]:
                best_index = i
    if best_index!=-1:
        print(f" process {p} -> {best_index} allocated")
        Block[best_index]-=p
    else:
        print(f" process {p} -> Not Allocated")
```

## OUTPUT:

The output cell displays the results of the Best Fit algorithm:

```
...
process 212 -> 3 allocated
process 417 -> 1 allocated
process 112 -> 2 allocated
process 426 -> 4 allocated
```

➤ **WORST FIT**

## SOURCE CODE -

The screenshot shows a Jupyter Notebook interface. The top navigation bar includes File, Edit, Selection, View, Go, Run, ..., and a search bar. Below the bar, several open notebooks are listed: T1.ipynb, LAB\_EXPERIMENT2.ipynb, LAB\_EXPERIMENT3.ipynb, and LAB\_EXPERIMENT5.ipynb. A sidebar on the left contains icons for file operations like Open, Save, and Run Cell.

The main content area displays the following text:

```
Block =[100,500,200,300,600]
processes =[212,417,112,426]
for p in processes:
    worst_index = -1
    for i in range (len(Block)):
        if Block[i] >= p :
            if worst_index == -1 or Block[i] >Block[worst_index]:
                worst_index = i
    if worst_index!=-1:
        print(f" process {p} -> {worst_index} allocated")
        Block[worst_index]-=p
    else:
        print(f" process {p} -> Not Allocated")
```

The code cell has a status bar indicating [1], a green checkmark, and 0.0s.

## OUTPUT -

The output cell shows the results of the execution:

```
...     process 212 -> 4 allocated
process 417 -> 1 allocated
process 112 -> 4 allocated
process 426 -> Not Allocated
```

## EXPERIMENT - 12

**AIM:** Implement Optimal Page Replacement Technique

**SOURCE CODE-**

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, ..., <-, >, Q MCA\_
- Recent Files:** EXPERT5.ipynb, LAB\_EXPERT4.ipynb, EXPERIMENT11a.ipynb, EXPERIMENT11c.ipynb
- Toolbar:** Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, +
- Section Header:** Implement Optimal Page Replacement Technique
- Code Cell:** A code cell containing the Python implementation of the Optimal Page Replacement algorithm.

```
def optimal_page_replacement(pages, frames):
    frame = []
    page_faults = 0

    print("Page\tFrames")

    for i in range(len(pages)):
        page = pages[i]

        # If page is already in frame (HIT)
        if page in frame:
            print(f"{page}\t{frame}")
            continue

        # Page fault
        page_faults += 1

        # If frame has space
        if len(frame) < frames:
```

```
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 –**

The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, ..., back, forward, search bar containing "MCA".
- File List:** EXPERIMENT13a.ipynb, EXPERIMENT13b.ipynb, LAB\_EXPERIMENT1.ipynb.
- Toolbar:** Generate, Code, Markdown, Run All, Restart, Clear All Output.
- Section Header:** Implementing Disk Scheduling Algorithms.
- Code Cell:** A code cell titled "FCFS" containing the following Python code:

```
def fcfs (request, head):  
    total = 0  
    current = head  
    print("sequence:",end=" ")  
    for req in request:  
        print(req,end = " ")  
        total += abs(current-req)  
        current = req  
    print("\n total head movement:",total)  
request =[82,170,43,140,24,16,190]  
head = 50  
fcfs(request,head)
```
- Output:** The output shows the sequence of disk head movements and the total head movement distance.

```
sequence: 82 170 43 140 24 16 190  
total head movement: 642
```

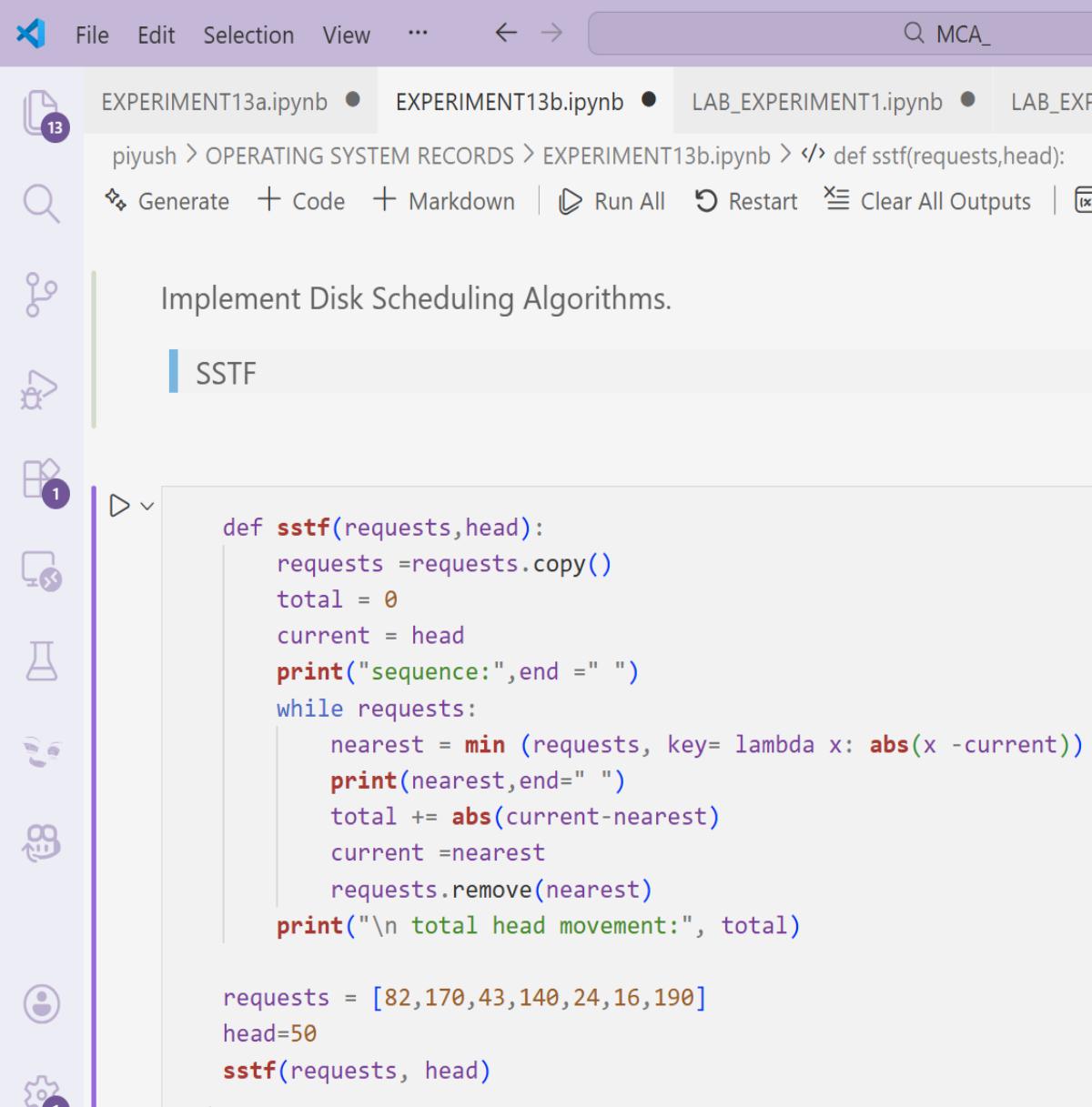
**Output:**

The output cell displays the results of the FCFS algorithm execution:

```
sequence: 82 170 43 140 24 16 190  
total head movement: 642
```

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

### **SOURCE CODE –**



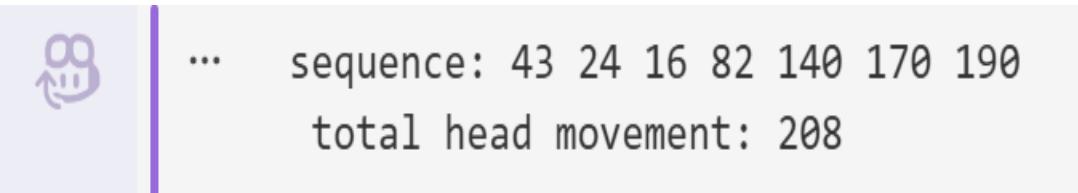
The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, ..., back, forward, search bar containing "MCA\_".
- File List:** EXPERIMENT13a.ipynb, EXPERIMENT13b.ipynb, LAB\_EXPERIMENT1.ipynb, LAB\_EXP.
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT13b.ipynb > def sstf(requests,head):  
Generate + Code + Markdown | Run All | Restart | Clear All Outputs |
- Text Area:** Implement Disk Scheduling Algorithms.  
**SSTF**
- Code Block:** A code cell containing the SSTF algorithm implementation. The code defines a function `sstf` that takes `requests` and `head` as parameters. It initializes `total` to 0, sets `current` to `head`, prints the sequence of requests, and then enters a loop where it finds the nearest request, calculates the total movement, updates `current`, removes the request from the list, and prints the total head movement. Finally, it prints the total head movement.

```
def sstf(requests,head):
    requests = requests.copy()
    total = 0
    current = head
    print("sequence:", end = " ")
    while requests:
        nearest = min (requests, key= lambda x: abs(x -current))
        print(nearest,end=" ")
        total += abs(current-nearest)
        current =nearest
        requests.remove(nearest)
    print("\n total head movement:", total)

requests = [82,170,43,140,24,16,190]
head=50
sstf(requests, head)
```

### **OUTPUT:**



The screenshot shows the output of the SSTF algorithm execution. The output is displayed in a light gray box and includes:

- A small icon of a smiling face with a gear.
- The text "... sequence: 43 24 16 82 140 170 190"
- The text "total head movement: 208"

## EXPERIMENT - 14

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

### **SOURCE CODE-**

The screenshot shows a Jupyter Notebook interface with the following details:

- Title Bar:** Shows "File Edit Selection View ..." and a search bar containing "MCA\_".
- File List:** Displays "RIMENT4.ipynb", "EXPERIMENT11a.ipynb", "EXPERIMENT11c.ipynb", and "experiment1.ipynb".
- Toolbar:** Includes icons for "Generate", "Code", "Markdown", "Run All", "Restart", and "Clear All Output".
- Code Cell:** Contains the following Python code:

```
access_count = {}
recent_file = []
def access_file(name):
    print("accessing:", name)
    access_count[name] = access_count.get(name, 0) + 1
    recent_file.append(name)

def suggest_strategy():
    lfu_file = max(access_count, key=access_count.get)
    lru_file = recent_file[-1]
    print("most frequent file:", lfu_file, "-> use lfu")
    print("most recent file:", lru_file, "-> use lru")
    if recent_file == sorted(recent_file):
        print("sequential access")
    else:
        print("no sequential pattern")
access_file("A")
access_file("C")
access_file("B")
access_file("C")
suggest_strategy()
```
- Left Sidebar:** Features a vertical toolbar with various icons, some with numerical badges (e.g., 14).

**OUTPUT -**

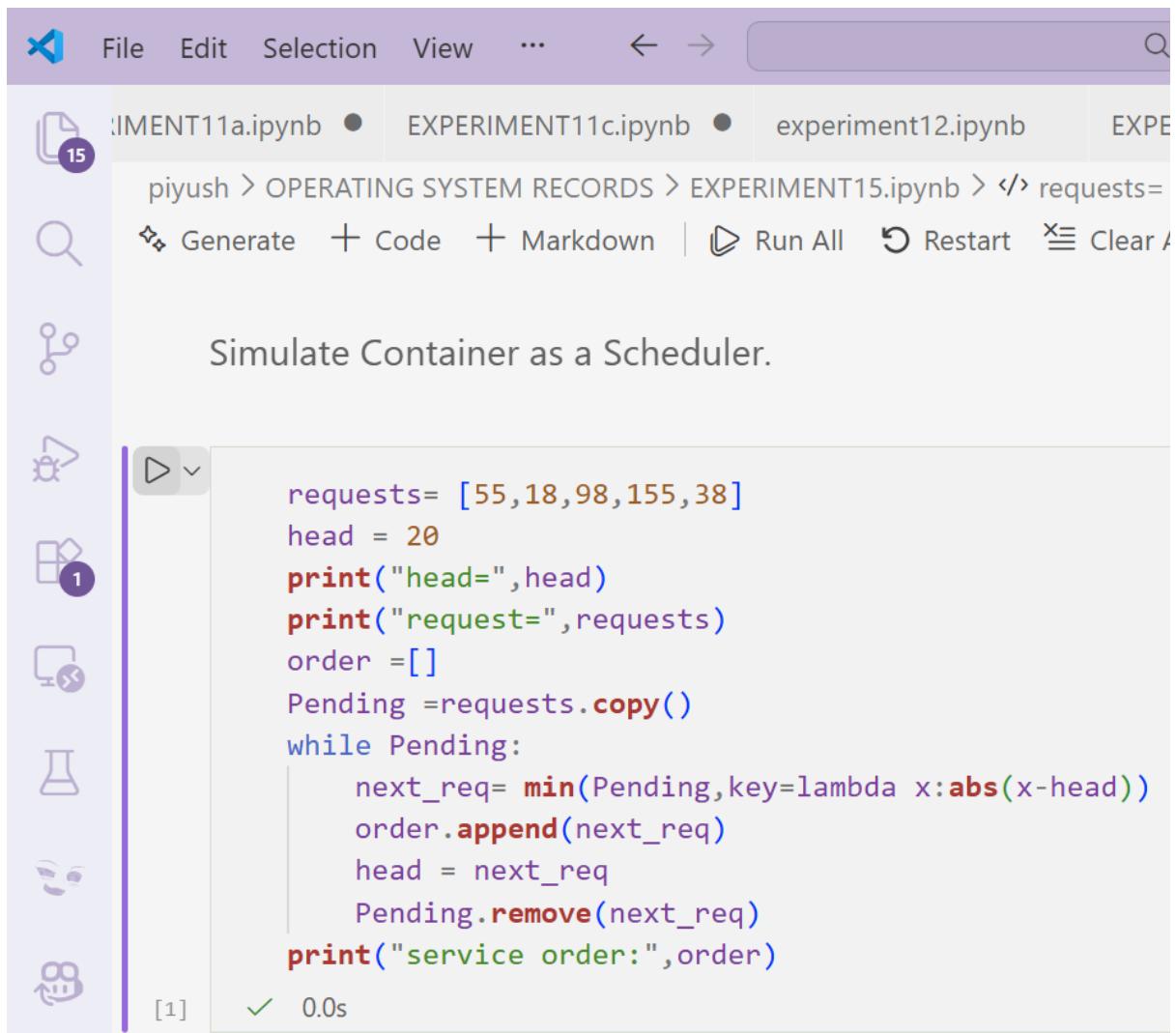
```
[1] ✓ 0.0s

... 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-**

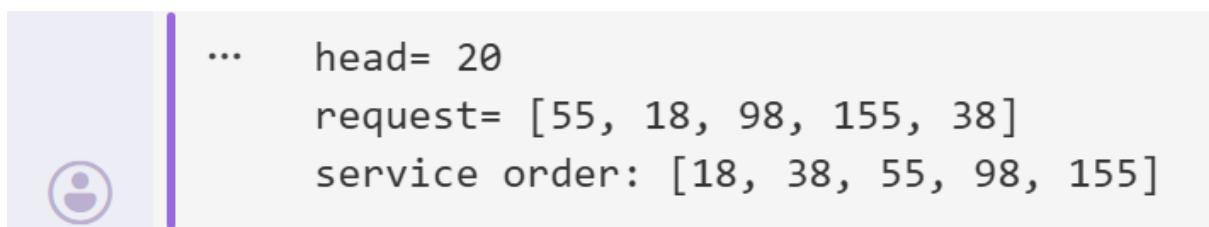


The screenshot shows a Jupyter Notebook interface. The top bar includes File, Edit, Selection, View, ..., back, forward, and search icons. Below the bar, several notebook files are listed: EXPERIMENT11a.ipynb, EXPERIMENT11c.ipynb, experiment12.ipynb, and EXPERIMENT15.ipynb (which is currently active). A sidebar on the left contains icons for file operations like Generate, Code, Markdown, Run All, Restart, and Clear All. The main area displays the following Python code:

```
requests= [55,18,98,155,38]
head = 20
print("head=",head)
print("request=",requests)
order =[]
Pending =requests.copy()
while Pending:
    next_req= min(Pending,key=lambda x:abs(x-head))
    order.append(next_req)
    head = next_req
    Pending.remove(next_req)
print("service order:",order)
```

The code defines a list of requests and initializes a head value of 20. It then prints the head and request lists. A while loop processes the Pending requests by finding the one closest to the current head (using the absolute difference), appending it to the service order, updating the head, and removing it from the Pending list. Finally, it prints the resulting service order.

**OUTPUT:**



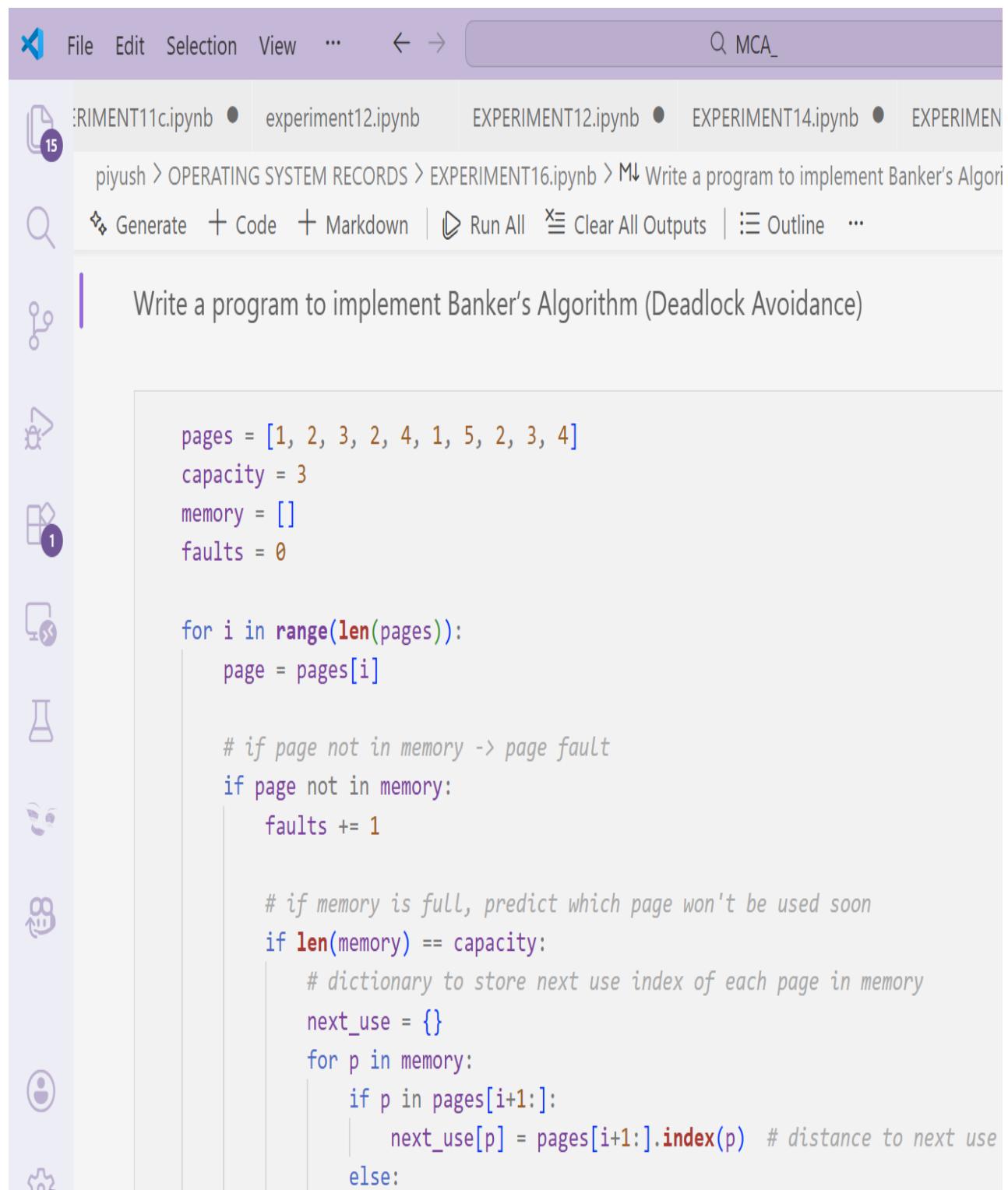
The output cell shows the results of running the provided code. The output is:

```
... head= 20
request= [55, 18, 98, 155, 38]
service order: [18, 38, 55, 98, 155]
```

## EXPERIMENT - 16

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

**SOURCE CODE-**



The screenshot shows a Jupyter Notebook interface with the following details:

- Toolbar:** File, Edit, Selection, View, ..., back, forward, search bar containing "MCA\_".
- File List:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT16.ipynb > M↓ Write a program to implement Banker's Algorithm
- Toolbar Buttons:** Generate, Code, Markdown, Run All, Clear All Outputs, Outline, ...
- Code Cell:** A cell containing Python code to implement the Banker's Algorithm for Deadlock Avoidance.

```
pages = [1, 2, 3, 2, 4, 1, 5, 2, 3, 4]
capacity = 3
memory = []
faults = 0

for i in range(len(pages)):
    page = pages[i]

    # if page not in memory -> page fault
    if page not in memory:
        faults += 1

    # if memory is full, predict which page won't be used soon
    if len(memory) == capacity:
        # dictionary to store next use index of each page in memory
        next_use = {}
        for p in memory:
            if p in pages[i+1:]:
                next_use[p] = pages[i+1:].index(p) # distance to next use
            else:
```

```
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 -

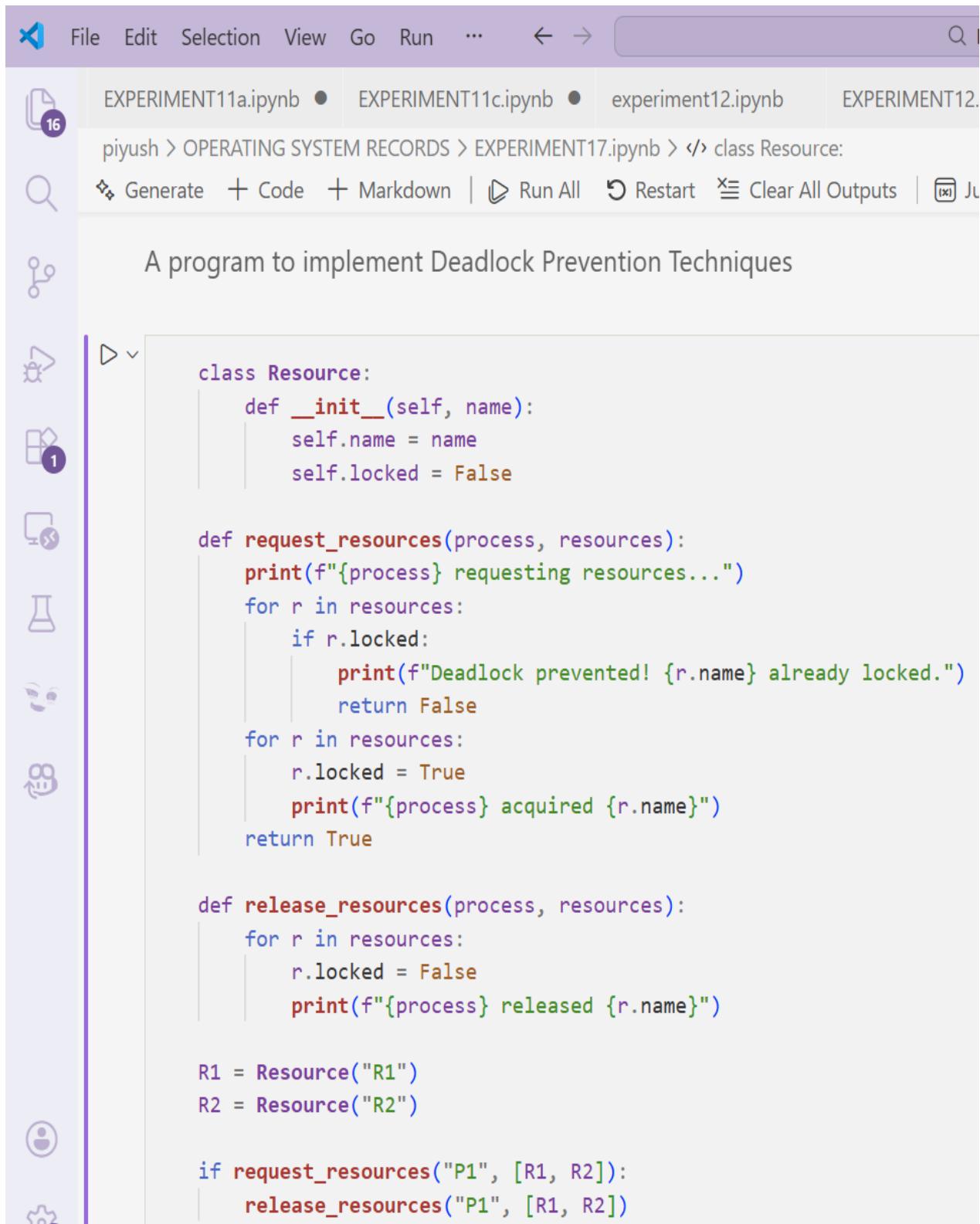
```
... Step 1: Page=1 | Memory=[1]
Step 2: Page=2 | Memory=[1, 2]
Step 3: Page=3 | Memory=[1, 2, 3]
Step 4: Page=2 | Memory=[1, 3, 2]
Step 5: Page=4 | Memory=[1, 2, 4]
Step 6: Page=1 | Memory=[2, 4, 1]
Step 7: Page=5 | Memory=[2, 4, 5]
Step 8: Page=2 | Memory=[4, 5, 2]
Step 9: Page=3 | Memory=[4, 2,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



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ..., back, forward, search.
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT17.ipynb > class Resource:  
Generate, Code, Markdown, Run All, Restart, Clear All Outputs, Kernel.
- Text Area:** A program to implement Deadlock Prevention Techniques
- Code Content:**

```
class Resource:
    def __init__(self, name):
        self.name = name
        self.locked = False

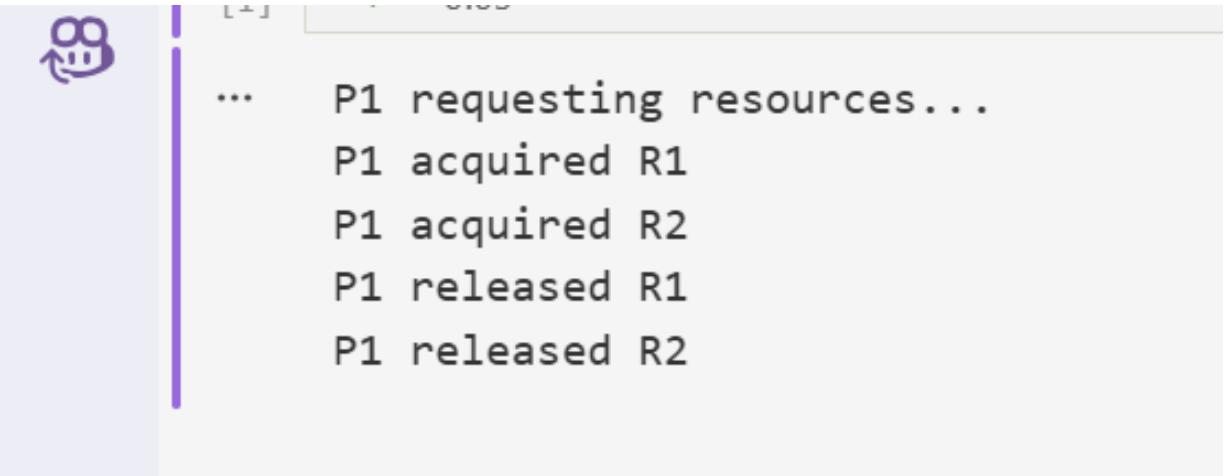
    def request_resources(process, resources):
        print(f"{process} requesting resources...")
        for r in resources:
            if r.locked:
                print(f"Deadlock prevented! {r.name} already locked.")
                return False
        for r in resources:
            r.locked = True
        print(f"{process} acquired {r.name}")
        return True

    def release_resources(process, resources):
        for r in resources:
            r.locked = False
        print(f"{process} released {r.name}")

R1 = Resource("R1")
R2 = Resource("R2")

if request_resources("P1", [R1, R2]):
    release_resources("P1", [R1, R2])
```

## ***OUTPUT -***



A screenshot of a terminal window titled "Terminal". The window contains the following text:

```
... 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 –**



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ..., back, forward.
- File List:** EXPERIMENT11c.ipynb, experiment12.ipynb, EXPERIMENT12.ipynb, EXPERIMENT18.ipynb (active).
- Toolbar:** Generate, Code, Markdown, Run All, Restart, Clear All Output.
- Text Cell Content:**

```
# Simulating Docker Container Resource Scheduling
total_cpu = 8 #cores in cpu
total_memory = 16 #Giga Bytes

# Container List (name, cpu, memory)
containers = [
    ("Container_A", 2, 4),
    ("Container_B", 4, 8),
    ("Container_C", 2, 6),
]
print("Host Resources:")
print("CPU:", total_cpu, "cores")
print("Memory:", total_memory, "GB\n")

for name, cpu, memory in containers:
    print("Requesting:", name)

    if cpu <= total_cpu and memory <= total_memory:
        total_cpu -= cpu
        total_memory -= memory
        print(name, "is scheduled and running")
        print("Remaining CPU:", total_cpu)
        print("Remaining Memory:", total_memory, "GB\n")
    else:
        print(name, "is waiting (Not enough resources)\n")
```

## **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



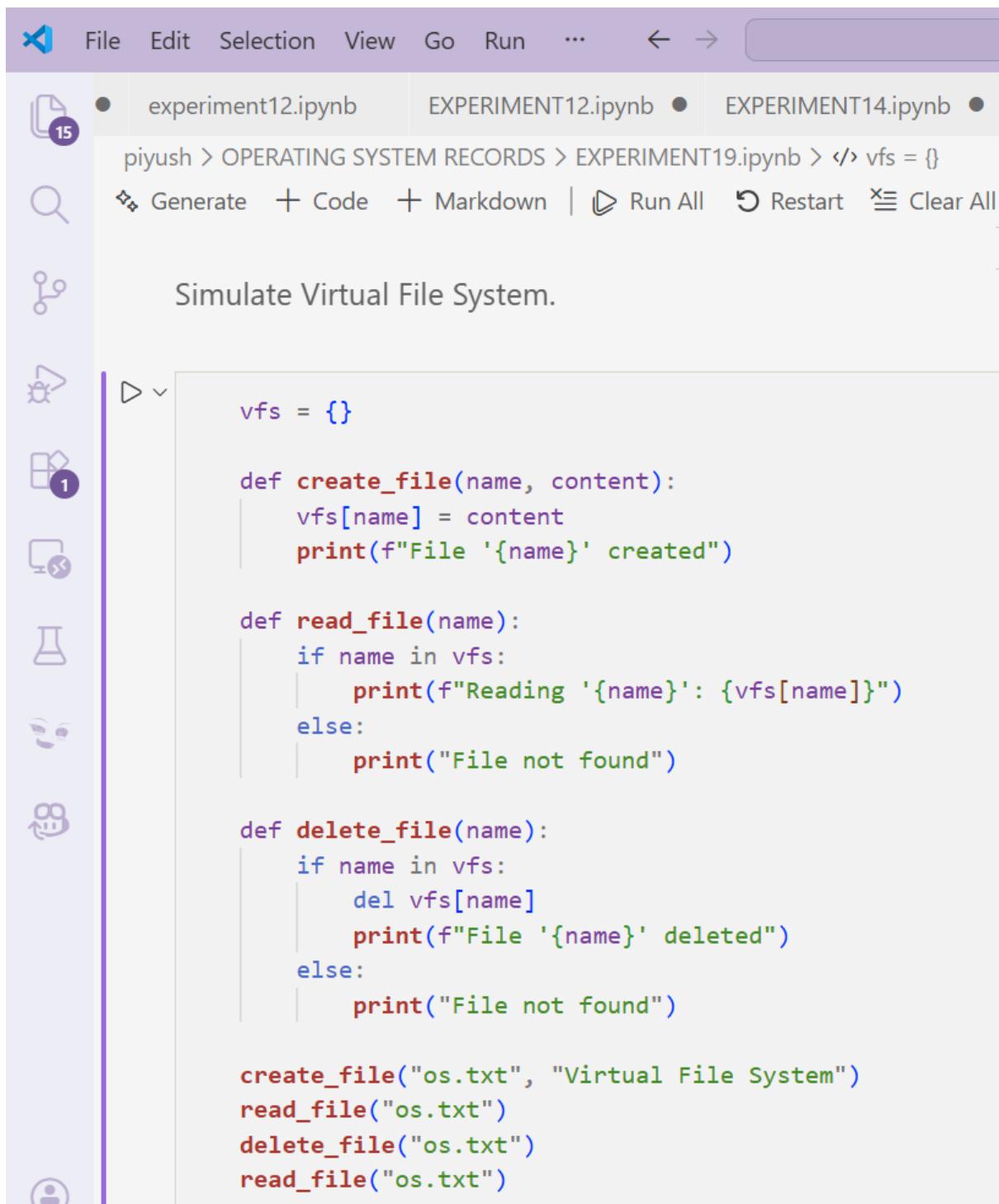
Requesting: Container\_C

Container\_C is waiting (Not enough resources)

## EXPERIMENT - 19

**AIM:** Simulate Virtual File System.

**SOURCE CODE-**



The screenshot shows a Jupyter Notebook interface with the following details:

- File Bar:** File, Edit, Selection, View, Go, Run, ..., back, forward.
- File List:** experiment12.ipynb, EXPERIMENT12.ipynb, EXPERIMENT14.ipynb.
- Toolbar:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT19.ipynb > </> vfs = {}  
Generate + Code + Markdown | Run All Restart Clear All
- Section Header:** Simulate Virtual File System.
- Code Cell:** Contains the following Python code:

```
vfss = {}

def create_file(name, content):
    vfss[name] = content
    print(f"File '{name}' created")

def read_file(name):
    if name in vfss:
        print(f"Reading '{name}': {vfss[name]}")
    else:
        print("File not found")

def delete_file(name):
    if name in vfss:
        del vfss[name]
        print(f"File '{name}' deleted")
    else:
        print("File not found")

create_file("os.txt", "Virtual File System")
read_file("os.txt")
delete_file("os.txt")
read_file("os.txt")
```

**OUTPUT:**

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