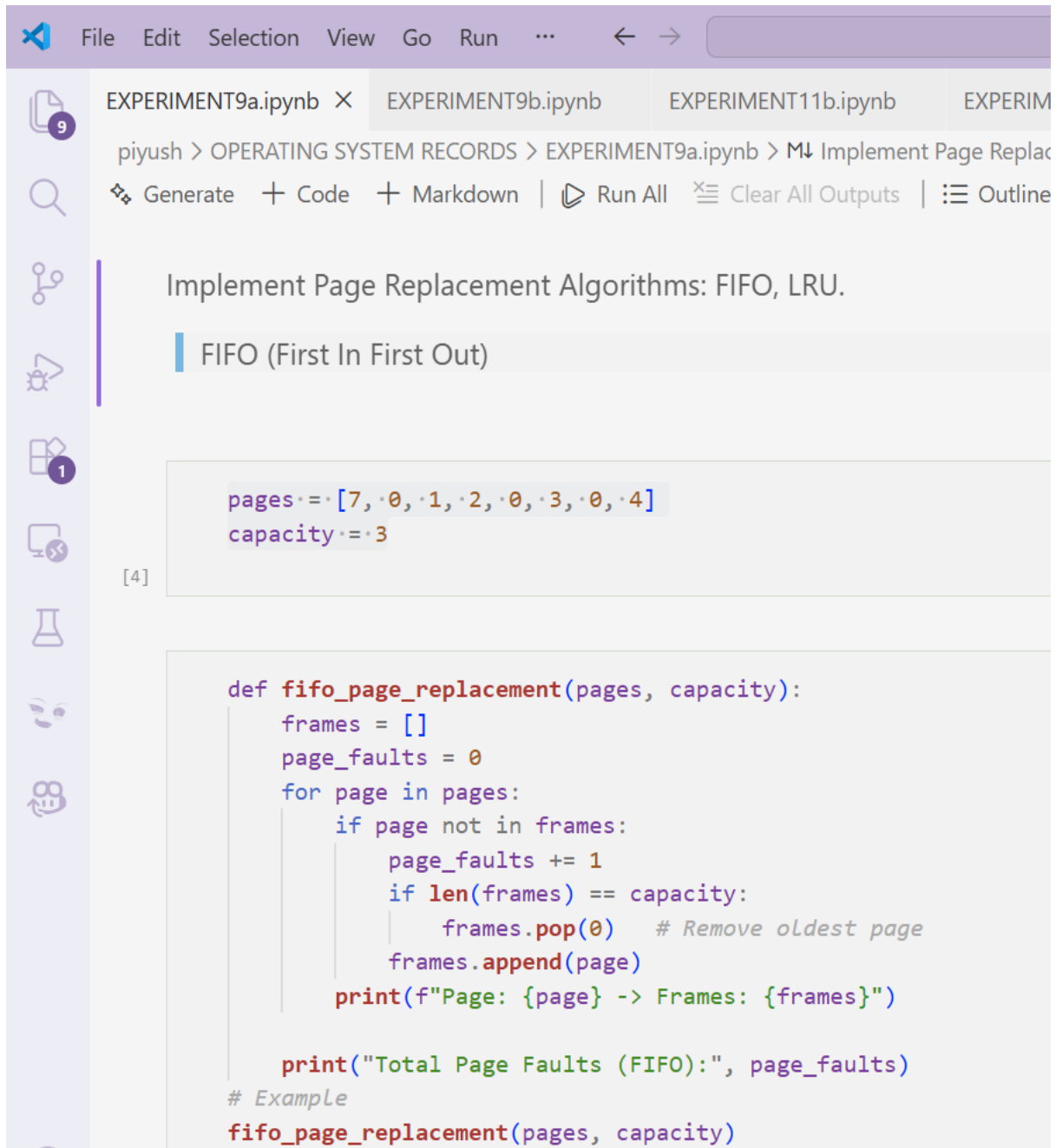


EXPERIMENT 9

AIM - Implement Page Replacement Algorithms: FIFO, LRU.

FIFO (First In First Out) - SOURCE CODE -



The screenshot shows a Jupyter Notebook with the following content:

```
File Edit Selection View Go Run ... < >
```

EXPERIMENT9a.ipynb X EXPERIMENT9b.ipynb EXPERIMENT11b.ipynb EXPERIM

piyush > OPERATING SYSTEM RECORDS > EXPERIMENT9a.ipynb > M↓ Implement Page Replac

Generate + Code + Markdown | Run All Clear All Outputs | Outline

Implement Page Replacement Algorithms: FIFO, LRU.


FIFO (First In First Out)

```
[4] pages = [7, 0, 1, 2, 0, 3, 0, 4]
     capacity = 3
```

```
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}")

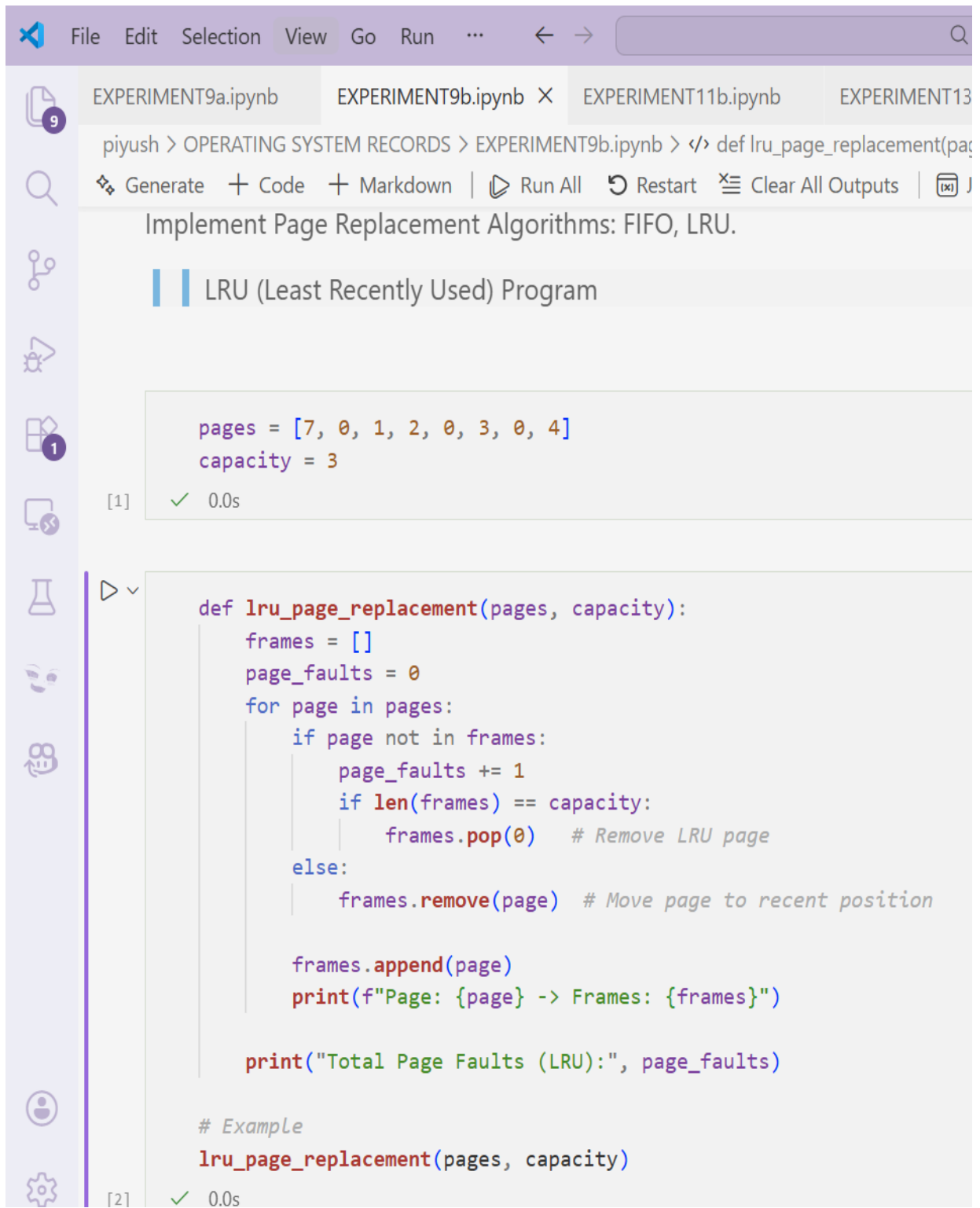
    print("Total Page Faults (FIFO):", page_faults)
# Example
fifo_page_replacement(pages, capacity)
```

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 displays a Jupyter Notebook with a purple-themed interface. The top bar includes menu items: File, Edit, Selection, View, Go, Run, and a search icon. Below the menu, several tabs are open: EXPERIMENT9a.ipynb, EXPERIMENT9b.ipynb (active), EXPERIMENT11b.ipynb, and EXPERIMENT13. The active notebook shows a file explorer on the left with a search icon and a list of files. The main area contains a title 'Implement Page Replacement Algorithms: FIFO, LRU.' and a subtitle 'LRU (Least Recently Used) Program'. The code is organized into two cells. The first cell, labeled [1], contains a list of pages and a capacity value. The second cell, labeled [2], contains the implementation of the LRU page replacement algorithm. The code defines a function `lru_page_replacement` that takes a list of pages and a capacity. It initializes a list of frames and a counter for page faults. It then iterates through the pages, checking if each page is already in the frames. If not, it increments the page fault counter and adds the page to the frames. If the frames are full, it removes the least recently used page (the first element in the list). If the page is already in the frames, it moves it to the most recent position (the end of the list). Finally, it prints the total page faults and the current state of the frames.

```
pages = [7, 0, 1, 2, 0, 3, 0, 4]
capacity = 3

[1] ✓ 0.0s
```

```
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 with the following elements:

- Top Bar:** File, Edit, Selection, View, Go, Run, and a search bar containing "MCA_".
- Tab Bar:** EXPERIMENT9b.ipynb, experiment10.ipynb, EXPERIMENT10.ipynb (active), EXPERIMENT11b.ipynb, and EX.
- Cell Header:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT10.ipynb > </> from collections import defaultdict
- Cell Actions:** Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, and Jupyter Variables.
- Cell Content:** A code cell titled "Extend LRU with a prediction model (predict next reference using past data)." containing the following Python code:

```
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

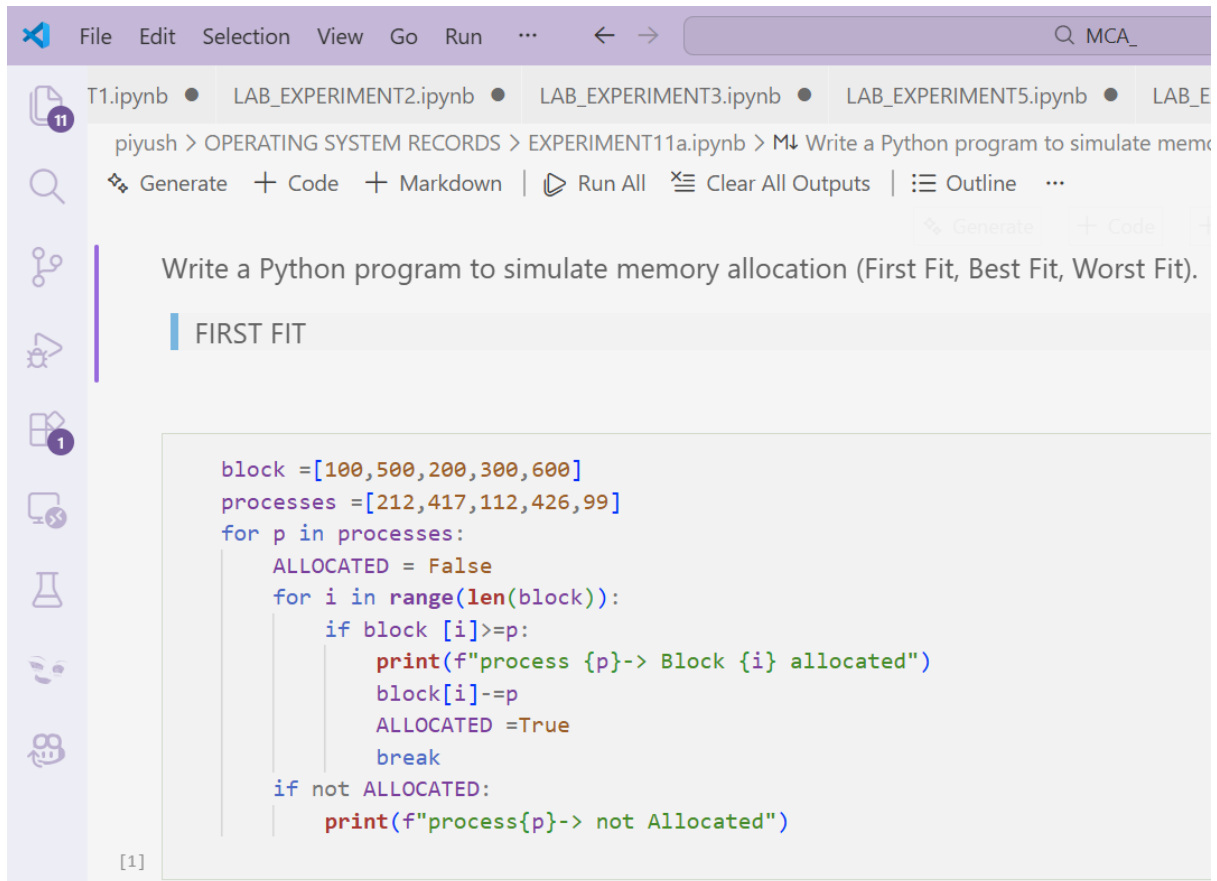


EXPERIMENT-11

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

➤ ***FIRST FIT***

SOURCE CODE-

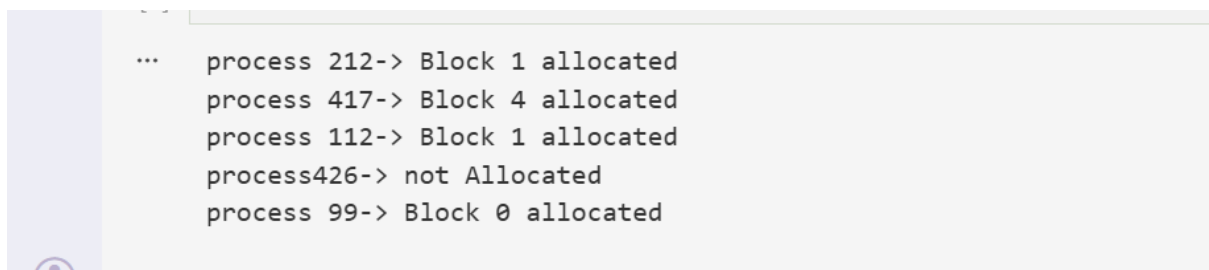


The screenshot shows a Jupyter Notebook window with a purple header bar. The title bar contains 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', and a search bar with 'MCA_'. The notebook has several tabs: 'T1.ipynb', 'LAB_EXPERIMENT2.ipynb', 'LAB_EXPERIMENT3.ipynb', 'LAB_EXPERIMENT5.ipynb', and 'LAB_E...'. The active tab is 'T1.ipynb', which shows the title 'piyush > OPERATING SYSTEM RECORDS > EXPERIMENT11a.ipynb > M↓ Write a Python program to simulate memo'. Below the title bar, there are buttons for 'Generate', '+ Code', '+ Markdown', 'Run All', 'Clear All Outputs', and 'Outline'. The main content area has a title 'Write a Python program to simulate memory allocation (First Fit, Best Fit, Worst Fit).'. Below the title, there is a section header 'FIRST FIT'. The code is written in a Python cell, which is currently in 'Code' view. The code is as follows:

```
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 is numbered [1] at the bottom left.

OUTPUT-

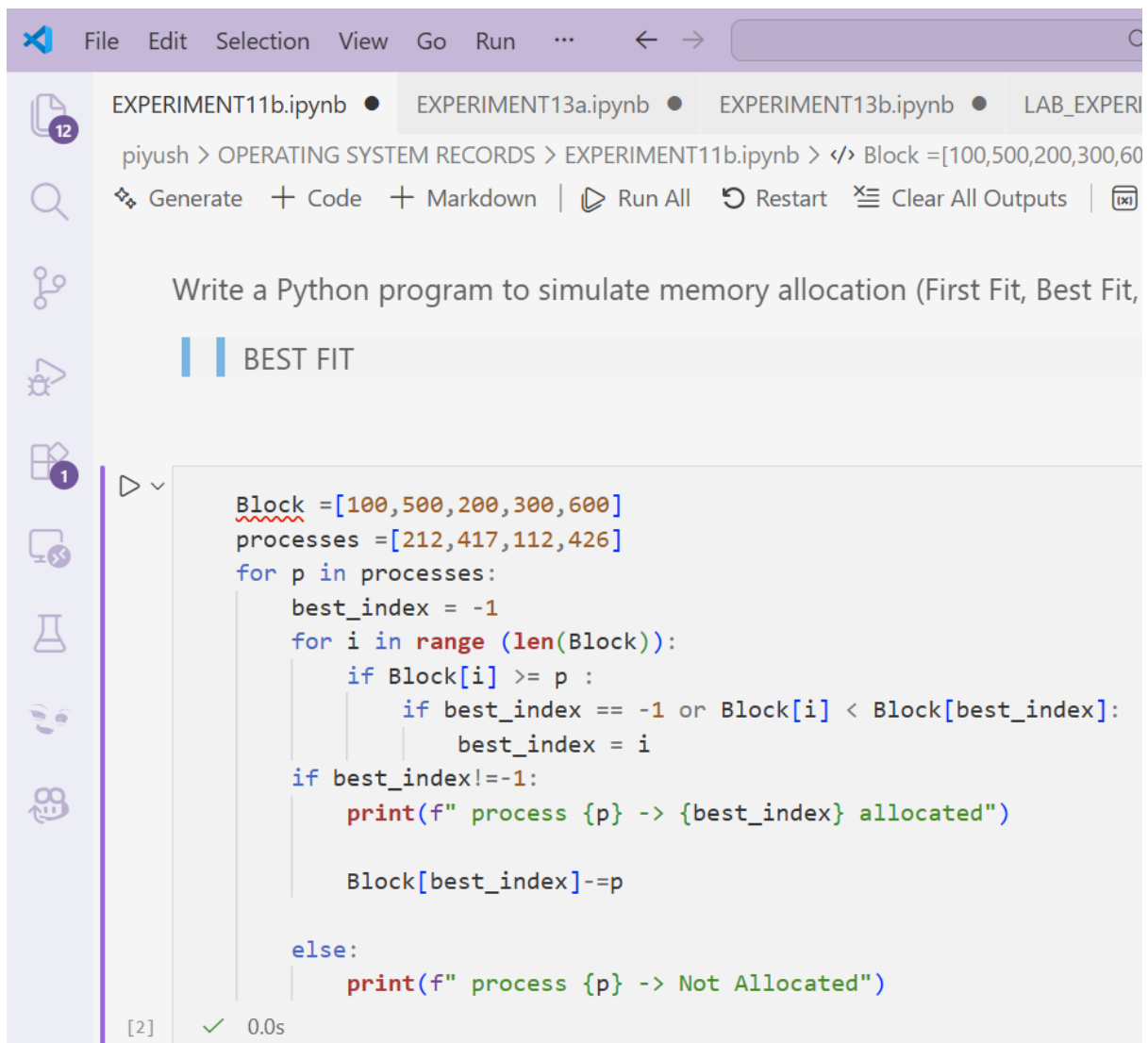


The screenshot shows the output of the Python code. The output is displayed in a cell with a light blue background. The output is as follows:

```
... 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 with the following components:

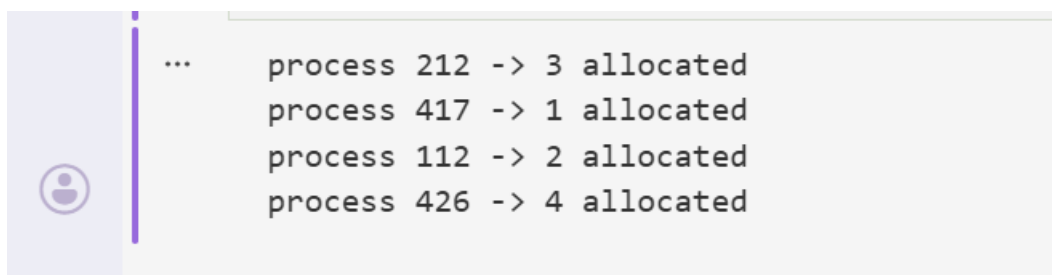
- Menu Bar:** File, Edit, Selection, View, Go, Run, ...
- Tab Bar:** EXPERIMENT11b.ipynb (active), EXPERIMENT13a.ipynb, EXPERIMENT13b.ipynb, LAB_EXPERI
- Header:** piyush > OPERATING SYSTEM RECORDS > EXPERIMENT11b.ipynb > </> Block =[100,500,200,300,60
- Toolbar:** Generate, + Code, + Markdown, Run All, Restart, Clear All Outputs, (x)
- Text:** Write a Python program to simulate memory allocation (First Fit, Best Fit, BEST FIT
- Code Cell:**

```
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:** [2] ✓ 0.0s

OUTPUT:

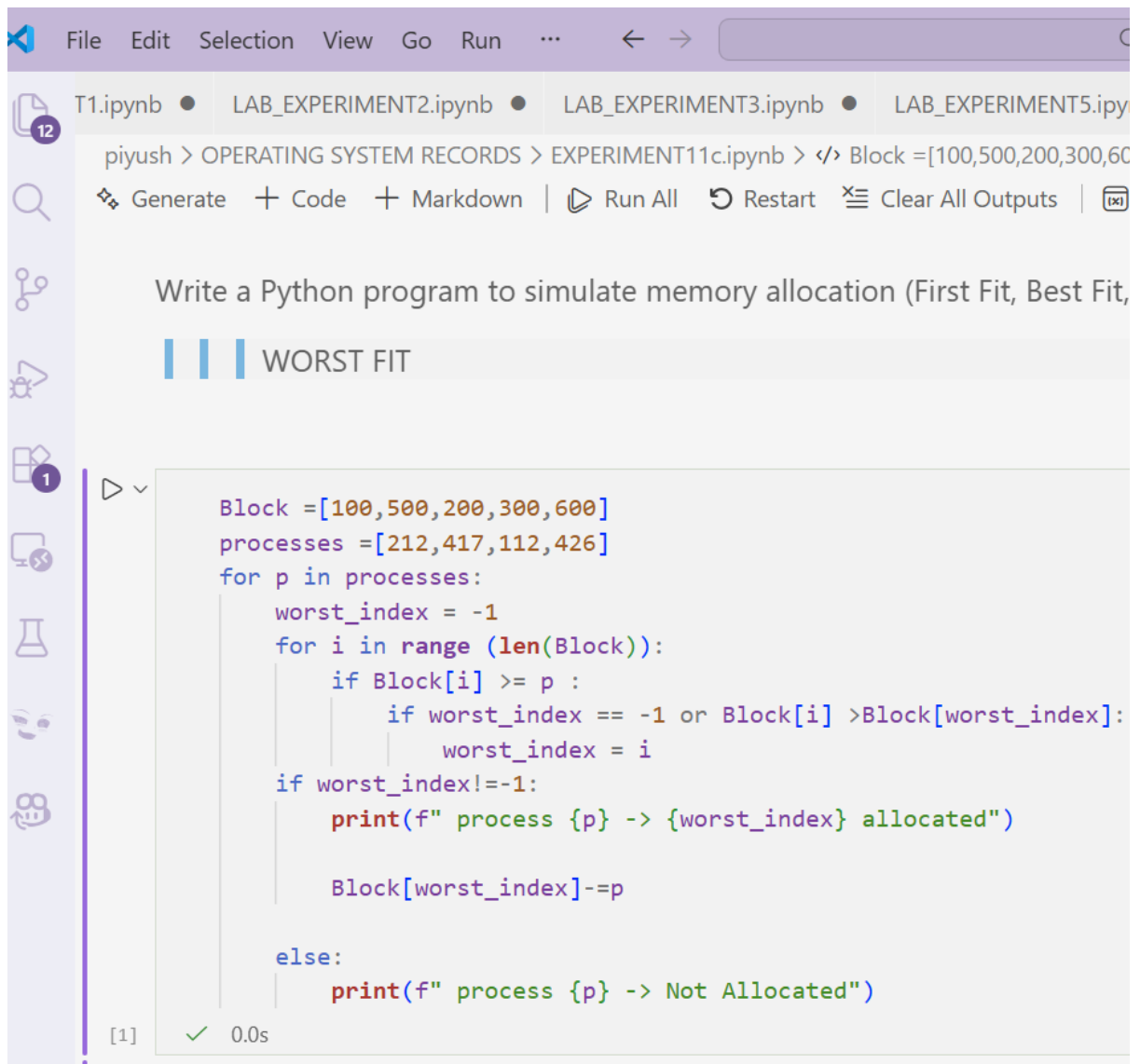


The screenshot shows the output of the code cell, which is a list of four lines indicating the allocation of memory to four processes:

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

➤ WORST FIT

SOURCE CODE –

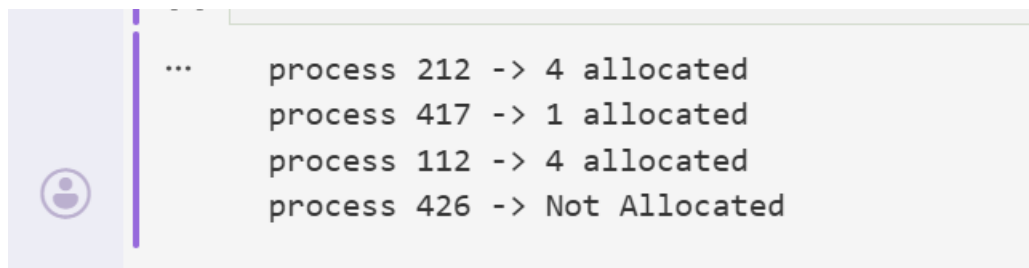


The screenshot shows a Jupyter Notebook with a purple header bar containing menu items: File, Edit, Selection, View, Go, Run, and navigation arrows. Below the header, several tabs are visible: T1.ipynb, LAB_EXPERIMENT2.ipynb, LAB_EXPERIMENT3.ipynb, and LAB_EXPERIMENT5.ipynb. The active tab is LAB_EXPERIMENT3.ipynb, which displays the text: "piyush > OPERATING SYSTEM RECORDS > EXPERIMENT11c.ipynb > </> Block =[100,500,200,300,600]". Below this, a prompt reads: "Write a Python program to simulate memory allocation (First Fit, Best Fit, WORST FIT)". The code cell contains the following Python code:

```
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")
```

Below the code, the output is shown: "[1] ✓ 0.0s".

OUTPUT -



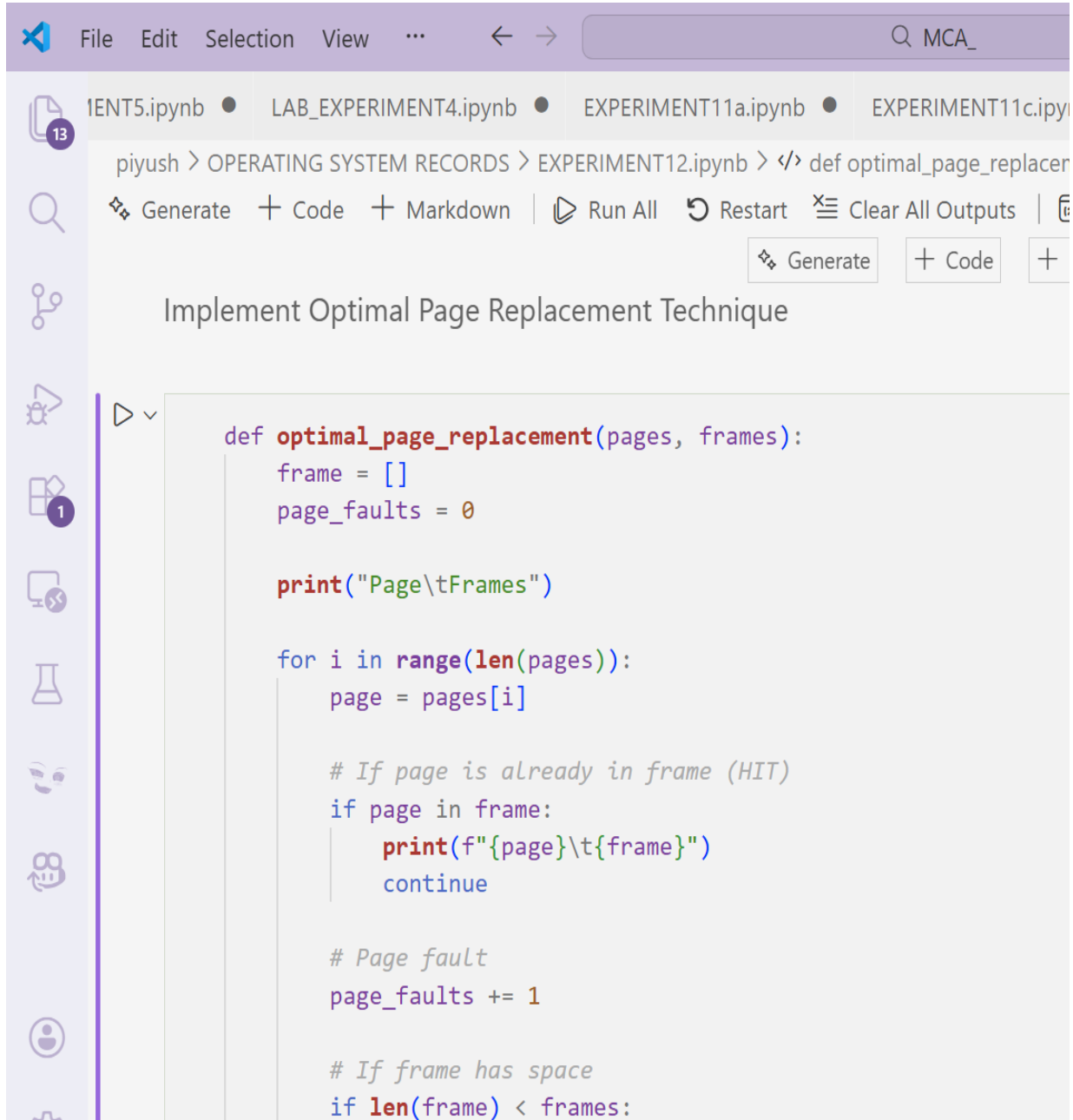
The screenshot shows the output of the Jupyter Notebook. It displays the following text:

```
... 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 a purple header bar. The top bar contains the file explorer, menu bar (File, Edit, Selection, View), and search bar (MCA_). The file explorer shows a list of files: IENT5.ipynb, LAB_EXPERIMENT4.ipynb, EXPERIMENT11a.ipynb, and EXPERIMENT11c.ipynb. The current file is EXPERIMENT12.ipynb, located in the directory piyush > OPERATING SYSTEM RECORDS. The notebook title is "Implement Optimal Page Replacement Technique". The code is written in Python and defines a function `optimal_page_replacement(pages, frames)` that implements the optimal page replacement algorithm. The code includes comments for each step: initializing the frame, counting page faults, checking for hits, and replacing the page with the one that will not be used for the longest time in the future.

```
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

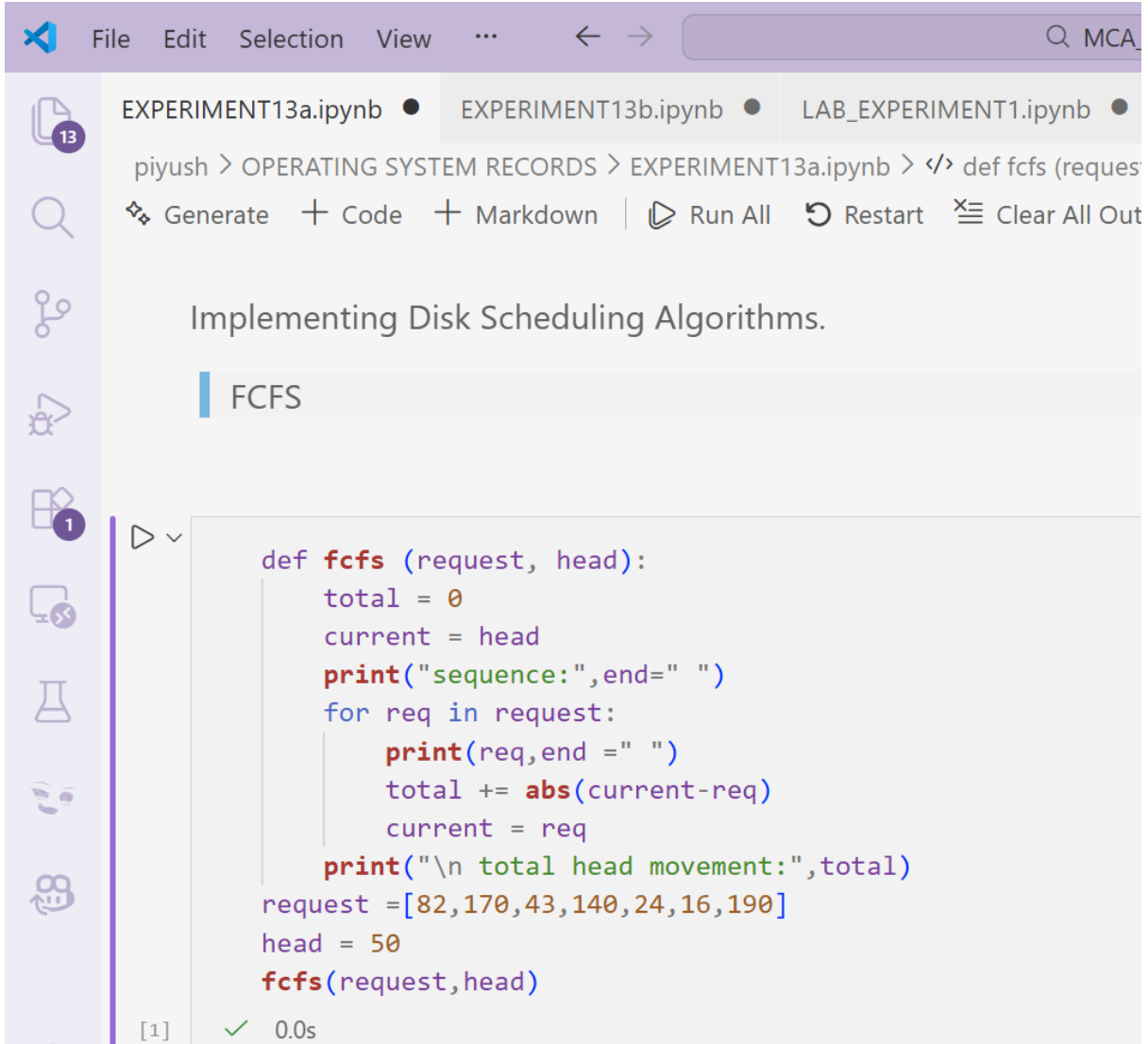
```

EXPERIMENT - 13

AIM: Implement Disk Scheduling Algorithms: FCFS, SSTF.

 **FCFS (First Come First Serve)**

SOURCE CODE –

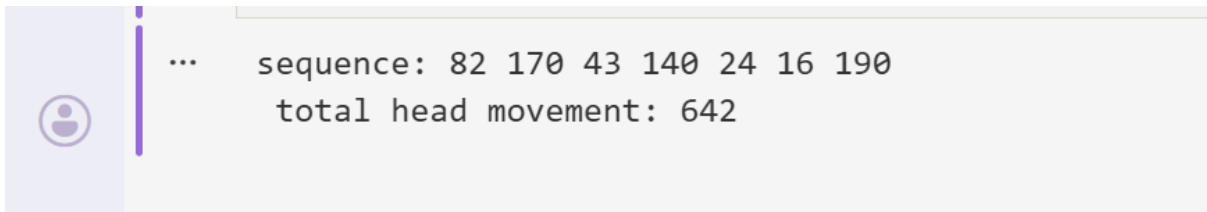


The screenshot shows a Jupyter Notebook with the following content:

- File Explorer: EXPERIMENT13a.ipynb (selected), EXPERIMENT13b.ipynb, LAB_EXPERIMENT1.ipynb
- Breadcrumbs: piyush > OPERATING SYSTEM RECORDS > EXPERIMENT13a.ipynb > </>
- Toolbar: Generate, Code, Markdown, Run All, Restart, Clear All Out
- Section Header: Implementing Disk Scheduling Algorithms.
- Section Subheader: FCFS
- Code Cell 1:

```
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: [1] ✓ 0.0s

Output:

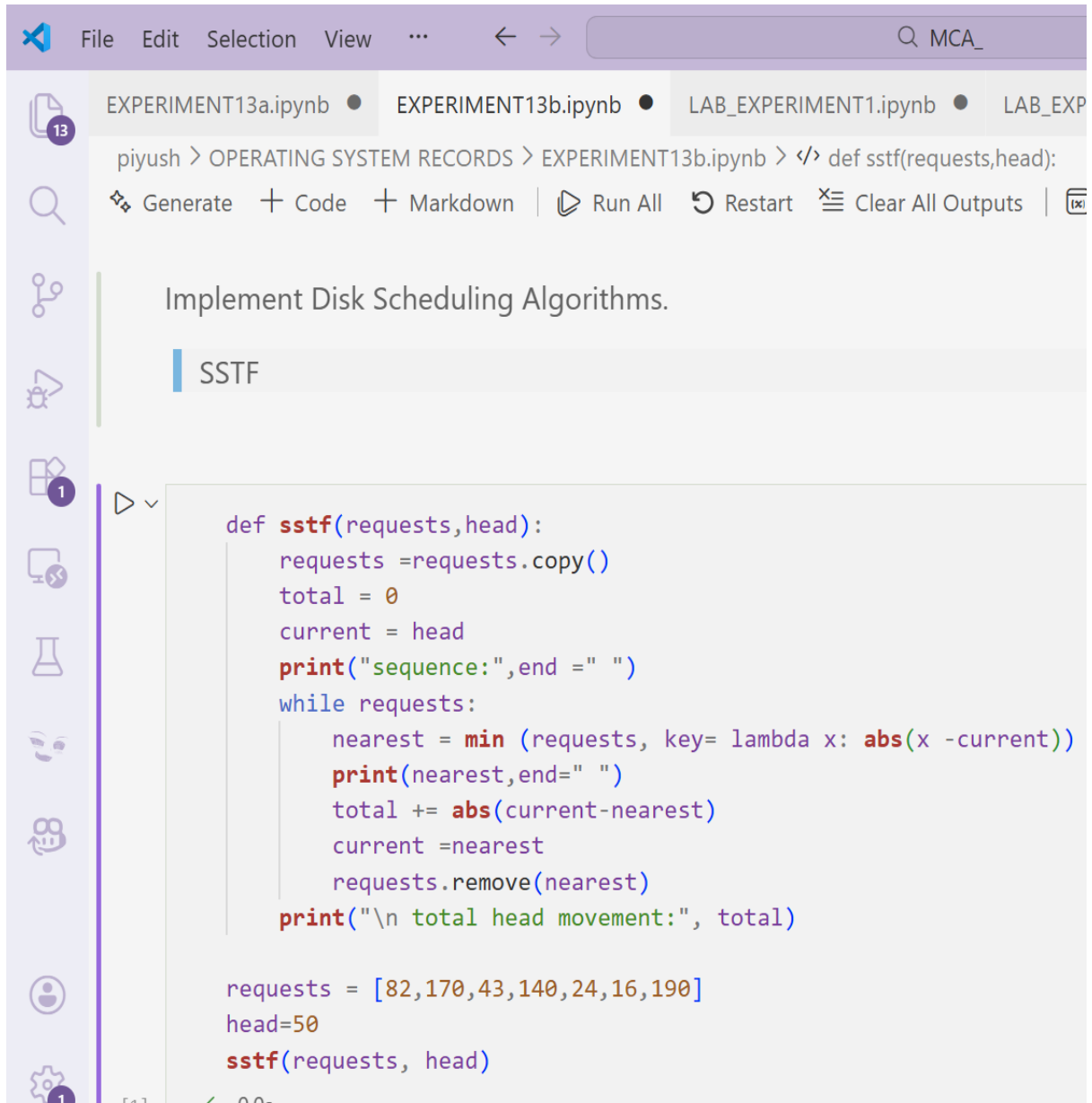


The output of the code cell is displayed as follows:

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

SSTF (Shortest Seek Time First)

SOURCE CODE –

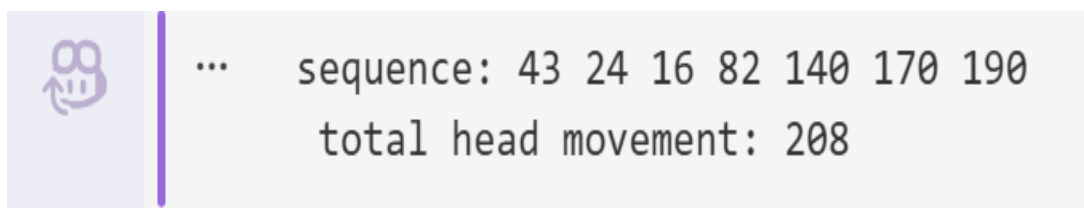


The screenshot shows a Jupyter Notebook window with the title bar "MCA_". The notebook has four tabs: "EXPERIMENT13a.ipynb", "EXPERIMENT13b.ipynb", "LAB_EXPERIMENT1.ipynb", and "LAB_EXP". The active tab is "EXPERIMENT13b.ipynb". The notebook content area shows the title "Implement Disk Scheduling Algorithms." and a sub-header "SSTF". The code cell contains the following Python code:

```
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:



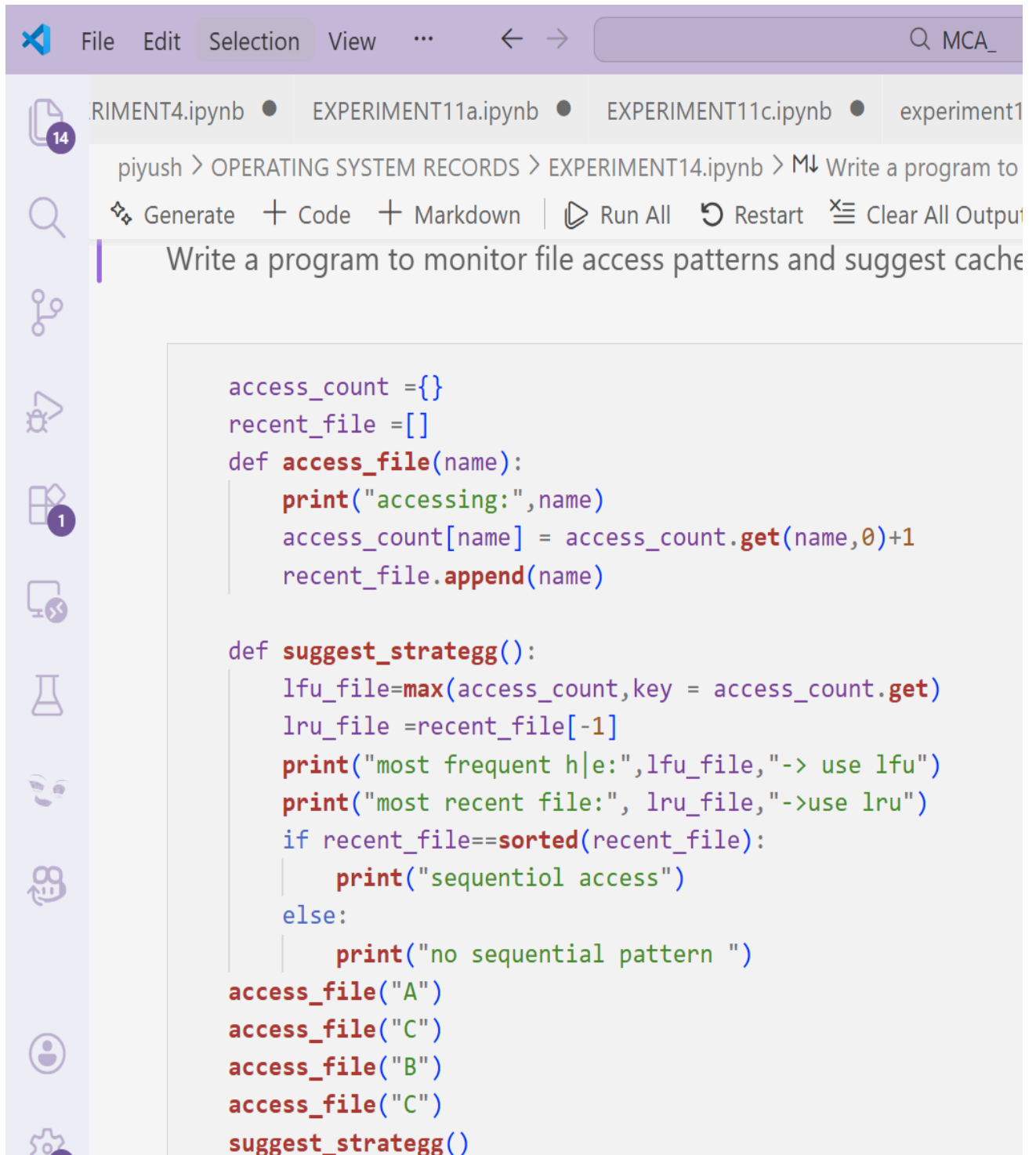
The output of the code is displayed in a cell with a light blue background. It shows the sequence of requests and the total head movement:

```
... sequence: 43 24 16 82 140 170 190
total head movement: 208
```

EXPERIMENT - 14

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

SOURCE 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_strategg():
    lfu_file = max(access_count, key = access_count.get)
    lru_file = recent_file[-1]
    print("most frequent h|e:", 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_strategg()
```

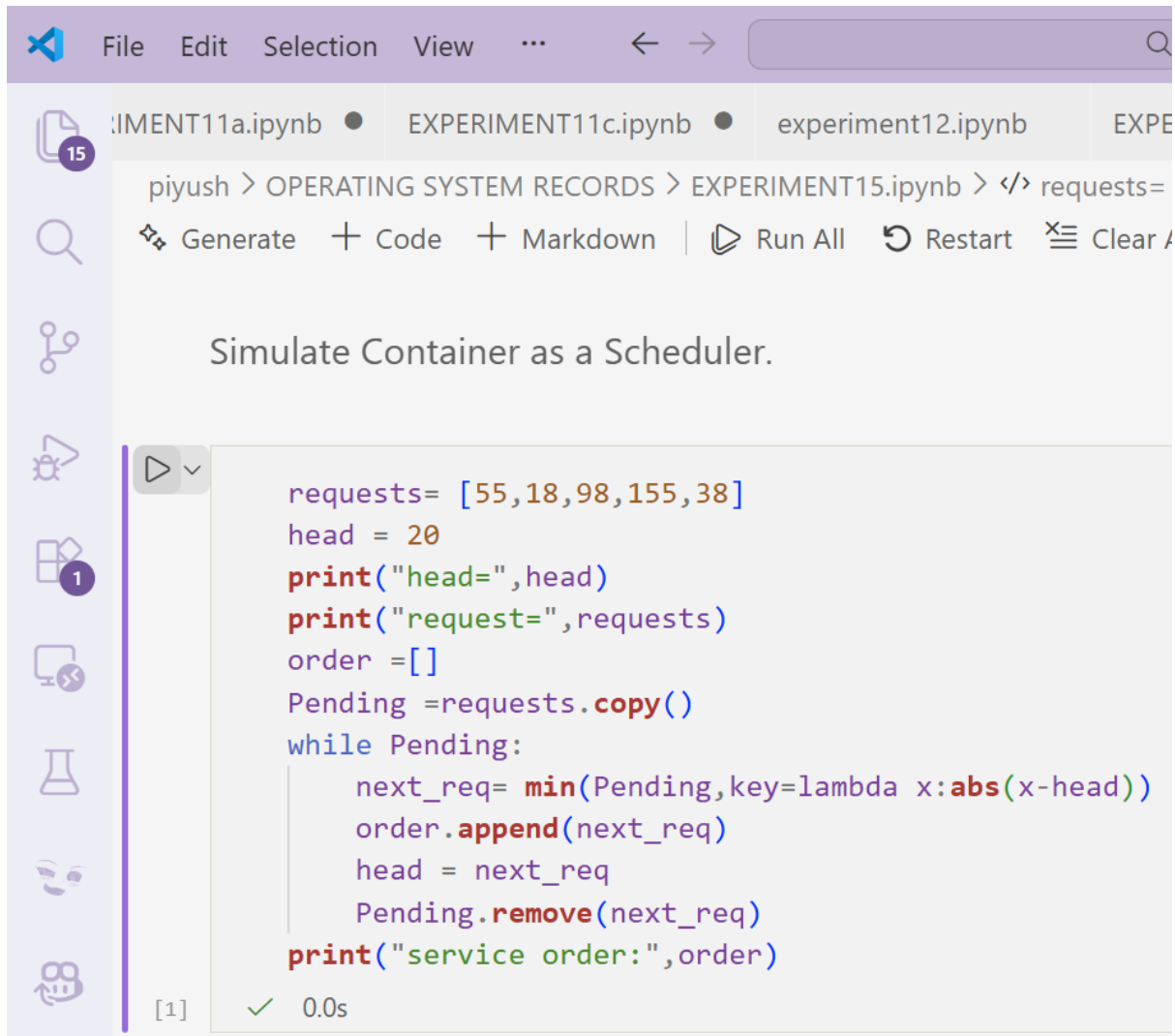
OUTPUT -

	[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-

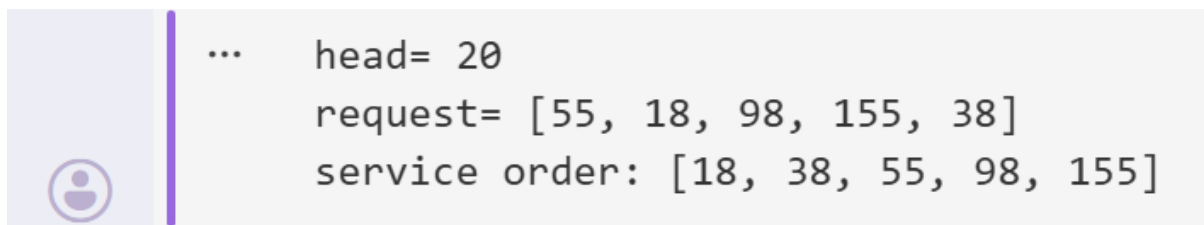


The screenshot shows a Jupyter Notebook interface with a purple header bar containing 'File', 'Edit', 'Selection', 'View', and navigation arrows. Below the header, there are tabs for 'EXPERIMENT11a.ipynb', 'EXPERIMENT11c.ipynb', 'experiment12.ipynb', and 'EXPERIMENT15.ipynb'. The active tab is 'EXPERIMENT15.ipynb', which shows the title 'Simulate Container as a Scheduler.' and a code cell. The code cell contains 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)
```

Below the code cell, there is a status bar showing '[1]' with a green checkmark and '0.0s'.

OUTPUT:



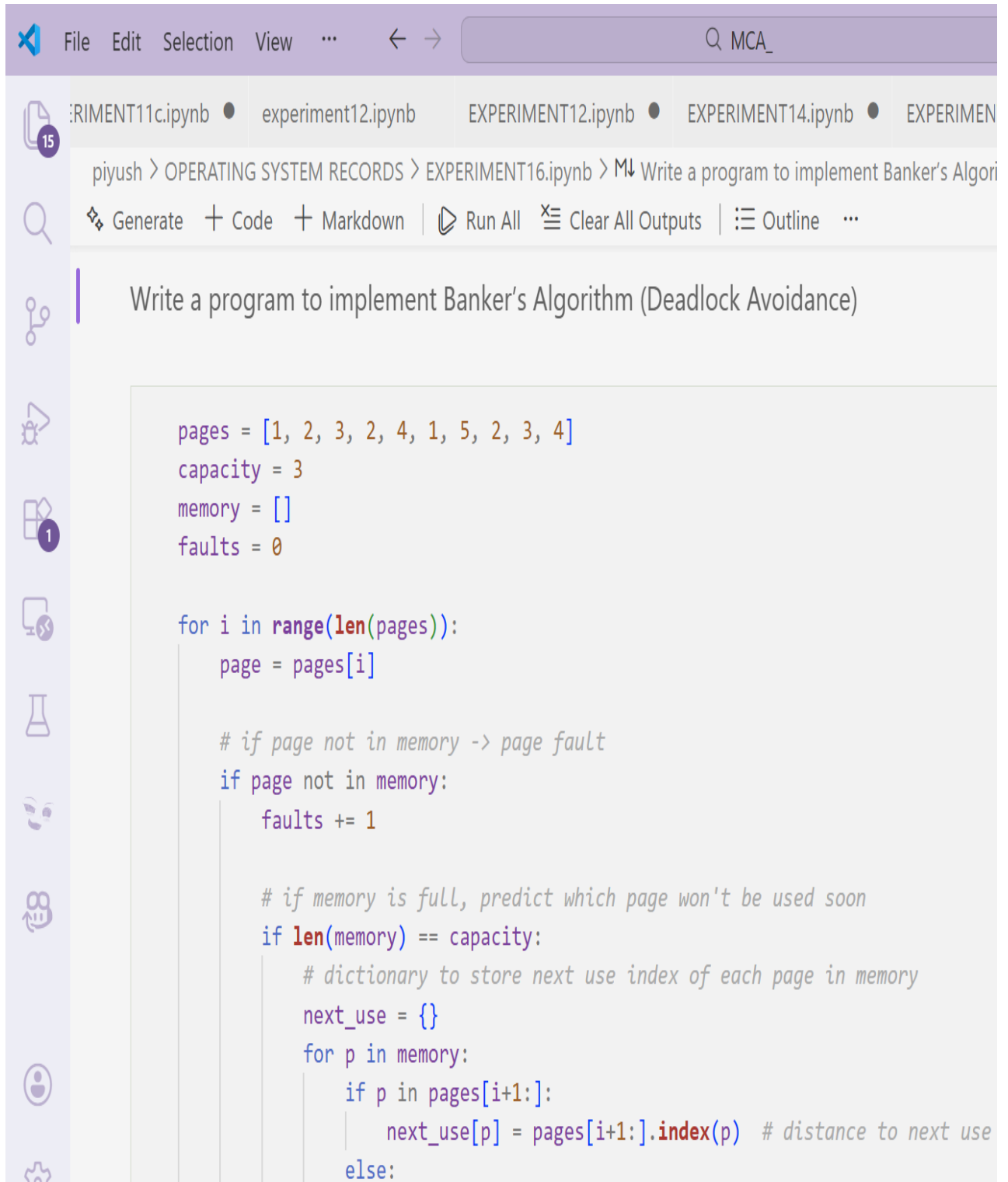
The screenshot shows the output of the Jupyter Notebook code cell. It displays the following text:

```
... 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 window with a purple header bar containing the VS Code logo and menu items: File, Edit, Selection, View, and a search bar with the text 'MCA_'. Below the header, there are tabs for several notebooks: 'EXPERIMENT11c.ipynb', 'experiment12.ipynb', 'EXPERIMENT12.ipynb', 'EXPERIMENT14.ipynb', and 'EXPERIMENT16.ipynb'. The active notebook is 'EXPERIMENT16.ipynb', which is open to a cell titled 'Write a program to implement Banker's Algorithm (Deadlock Avoidance)'. The cell contains the following Python code:

```
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 -

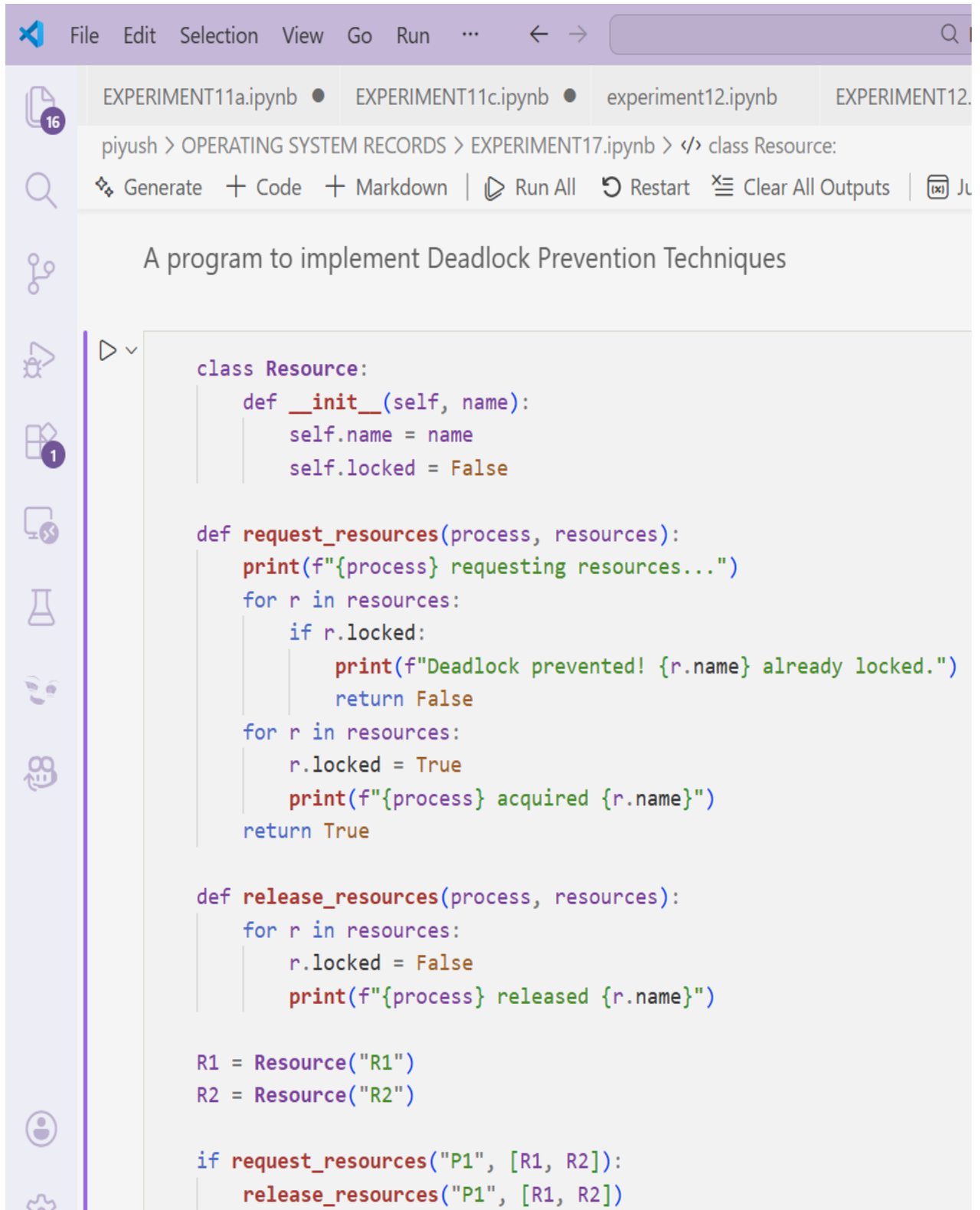
```
... Step 1: Page=1 | Memory=[1]
Step 2: Page=2 | Memory=[1, 2]
Step 3: Page=3 | Memory=[1, 2, 3]
Step 4: Page=2 | Memory=[1, 3, 2]
Step 5: Page=4 | Memory=[1, 2, 4]
Step 6: Page=1 | Memory=[2, 4, 1]
Step 7: Page=5 | Memory=[2, 4, 5]
Step 8: Page=2 | Memory=[4, 5, 2]
Step 9: Page=3 | Memory=[4, 2, 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 window with a purple header bar containing menu items: File, Edit, Selection, View, Go, Run, and search icons. Below the header, several tabs are visible: EXPERIMENT11a.ipynb, EXPERIMENT11c.ipynb, experiment12.ipynb, and EXPERIMENT12. The active tab is EXPERIMENT17.ipynb, showing the path: piyush > OPERATING SYSTEM RECORDS > EXPERIMENT17.ipynb. The notebook interface includes a toolbar with icons for search, generate, code, markdown, run all, restart, clear all outputs, and a Ju icon. The title of the notebook is "A program to implement Deadlock Prevention Techniques". The code is written in Python and defines a Resource class and functions for requesting and releasing resources.

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

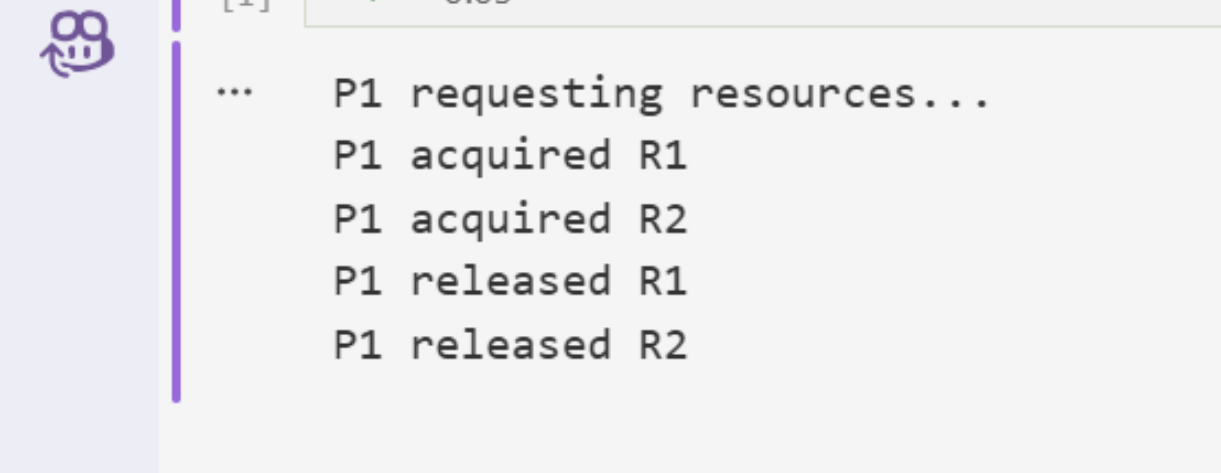
def request_resources(process, resources):
    print(f"{process} requesting resources...")
    for r in resources:
        if r.locked:
            print(f"Deadlock prevented! {r.name} already locked.")
            return False
    for r in resources:
        r.locked = True
        print(f"{process}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

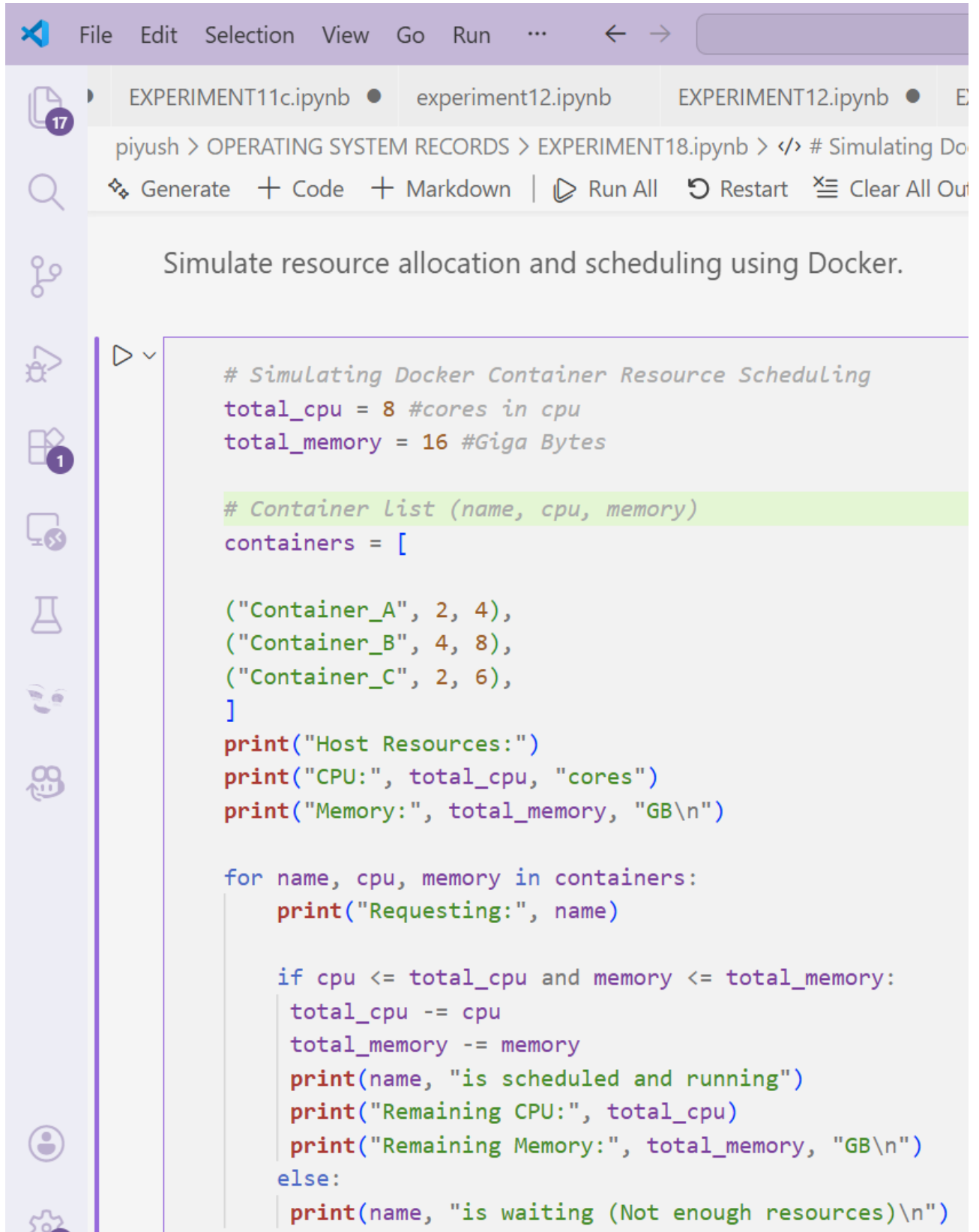


```
[1] ...  
... P1 requesting resources...  
    P1 acquired R1  
    P1 acquired R2  
    P1 released R1  
    P1 released R2
```

EXPERIMENT - 18

AIM: Simulate resource allocation and scheduling using Docker.

SOURCE CODE –



The screenshot displays a Jupyter Notebook environment. The top menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', and a search icon. Below the menu, the file explorer shows three open files: 'EXPERIMENT11c.ipynb', 'experiment12.ipynb', and 'EXPERIMENT12.ipynb'. The active file is 'EXPERIMENT18.ipynb', located within the 'piyush > OPERATING SYSTEM RECORDS' directory. The notebook's title bar reads 'Simulate resource allocation and scheduling using Docker.' The code area contains a Python script for simulating Docker container resource scheduling. The script defines total CPU and memory, lists containers with their requested resources, and simulates the allocation process, printing the status of each container.

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

# Container List (name, cpu, memory)
containers = [

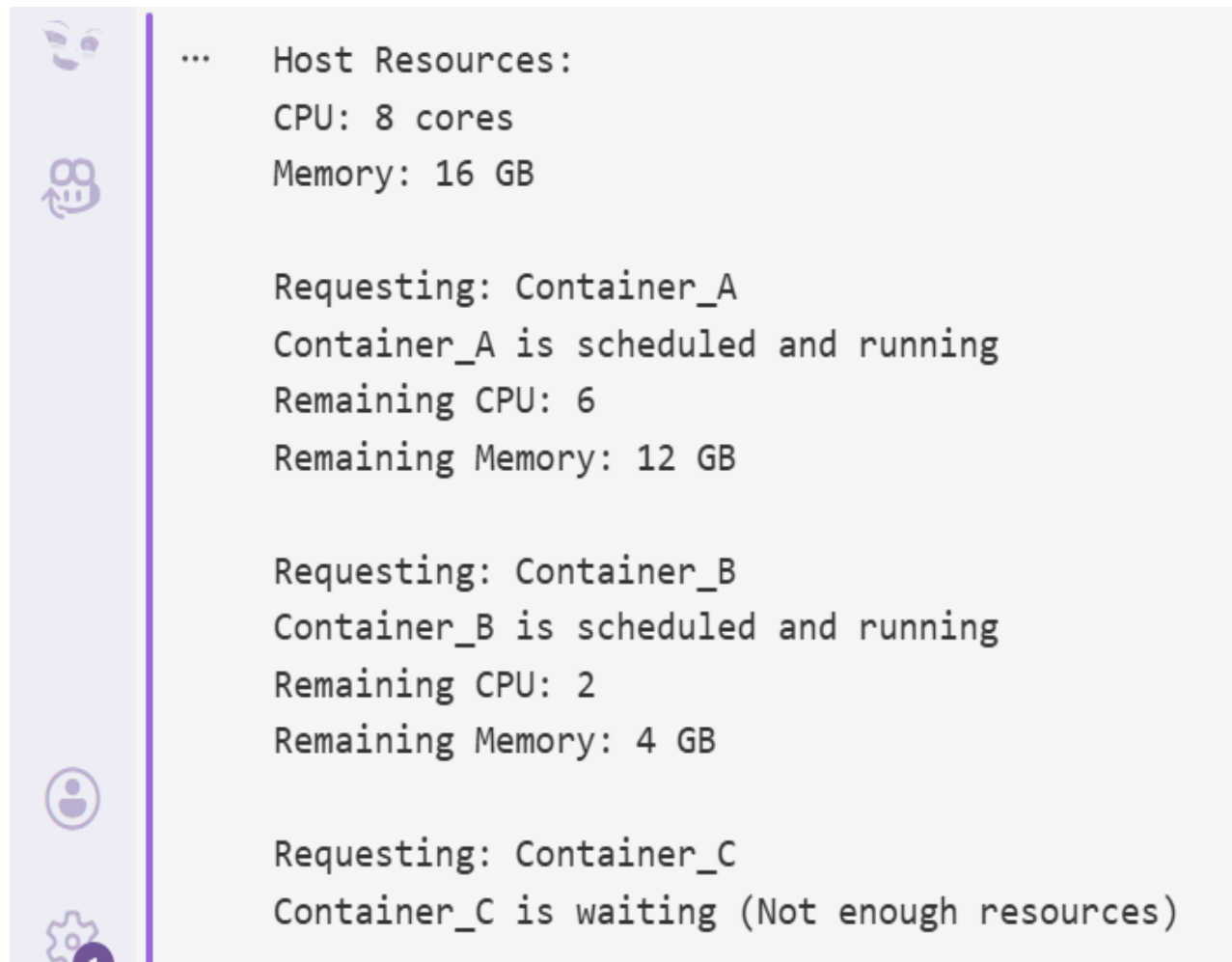
    ("Container_A", 2, 4),
    ("Container_B", 4, 8),
    ("Container_C", 2, 6),
]

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 -

A terminal window with a light gray background. On the left side, there is a vertical sidebar with four icons: a face with eyes, a face with a hand, a person in a circle, and a gear. The terminal text is as follows:

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

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

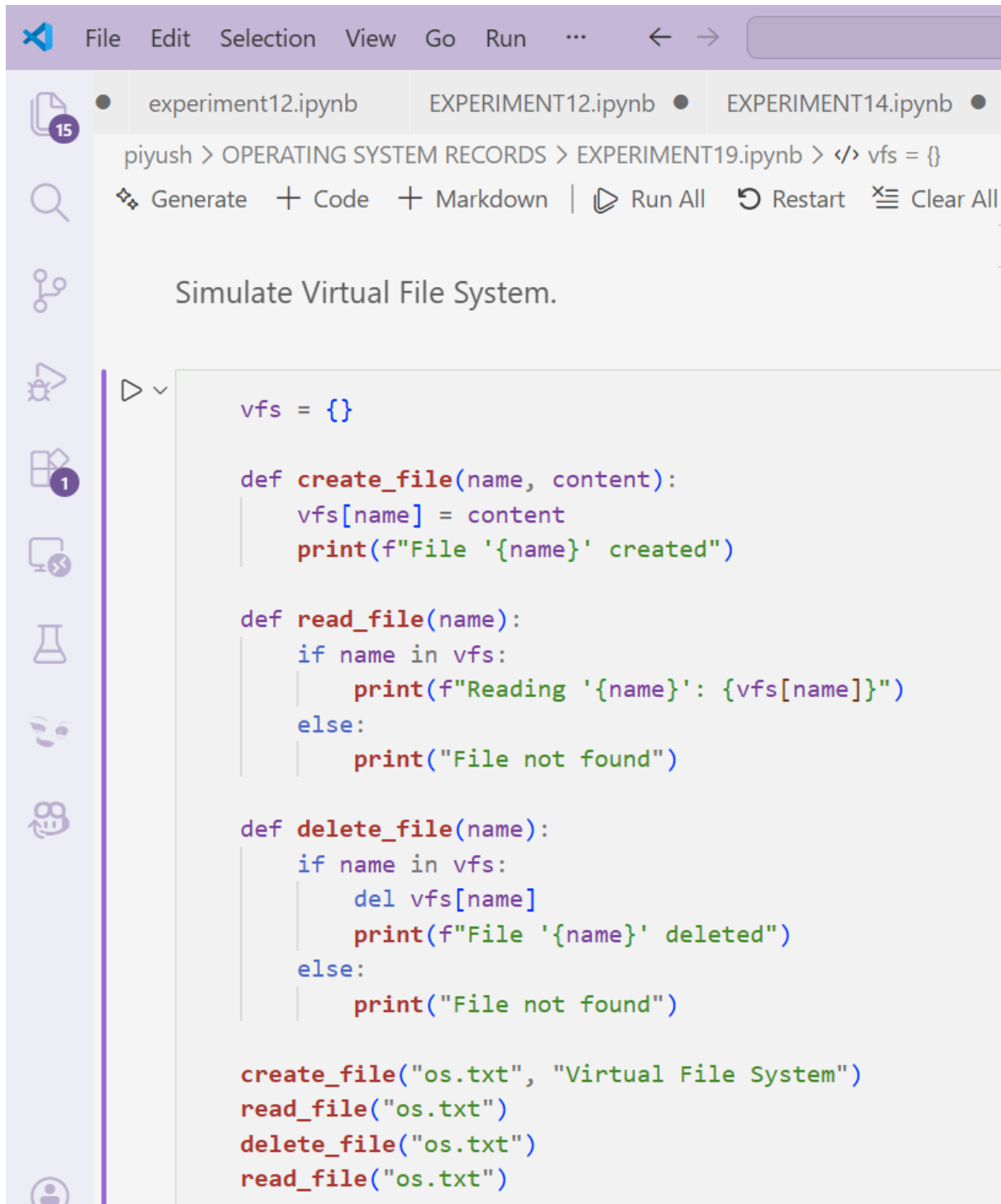
    Requesting: Container_B
    Container_B is scheduled and running
    Remaining CPU: 2
    Remaining Memory: 4 GB

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

EXPERIMENT - 19

AIM: Simulate Virtual File System.

SOURCE CODE-



The screenshot displays a Jupyter Notebook environment. The top menu bar includes 'File', 'Edit', 'Selection', 'View', 'Go', 'Run', and a search icon. The breadcrumb trail shows the path: 'piyush > OPERATING SYSTEM RECORDS > EXPERIMENT19.ipynb > </> vfs = {}'. The notebook title bar shows three open files: 'experiment12.ipynb', 'EXPERIMENT12.ipynb', and 'EXPERIMENT14.ipynb'. The left sidebar contains various icons for file management and execution. The main code area is titled 'Simulate Virtual File System.' and contains the following Python code:

```
vfs = {}

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

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

def delete_file(name):
    if name in vfs:
        del vfs[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
```