Name: DUSABIMANA THEOPHILLE

REG NO: 224010076

LEVEL 2

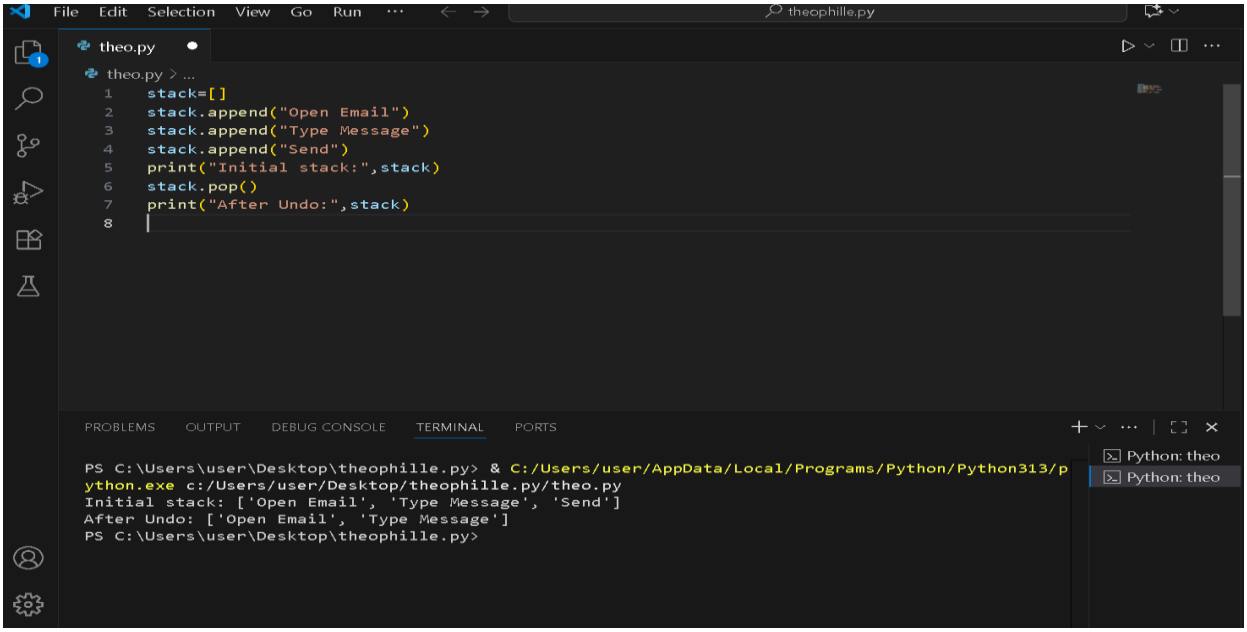
DATA STRUCTURE - BIT -EXERCISE NO:4

PROJECT 16

STACK QUESTIONS:

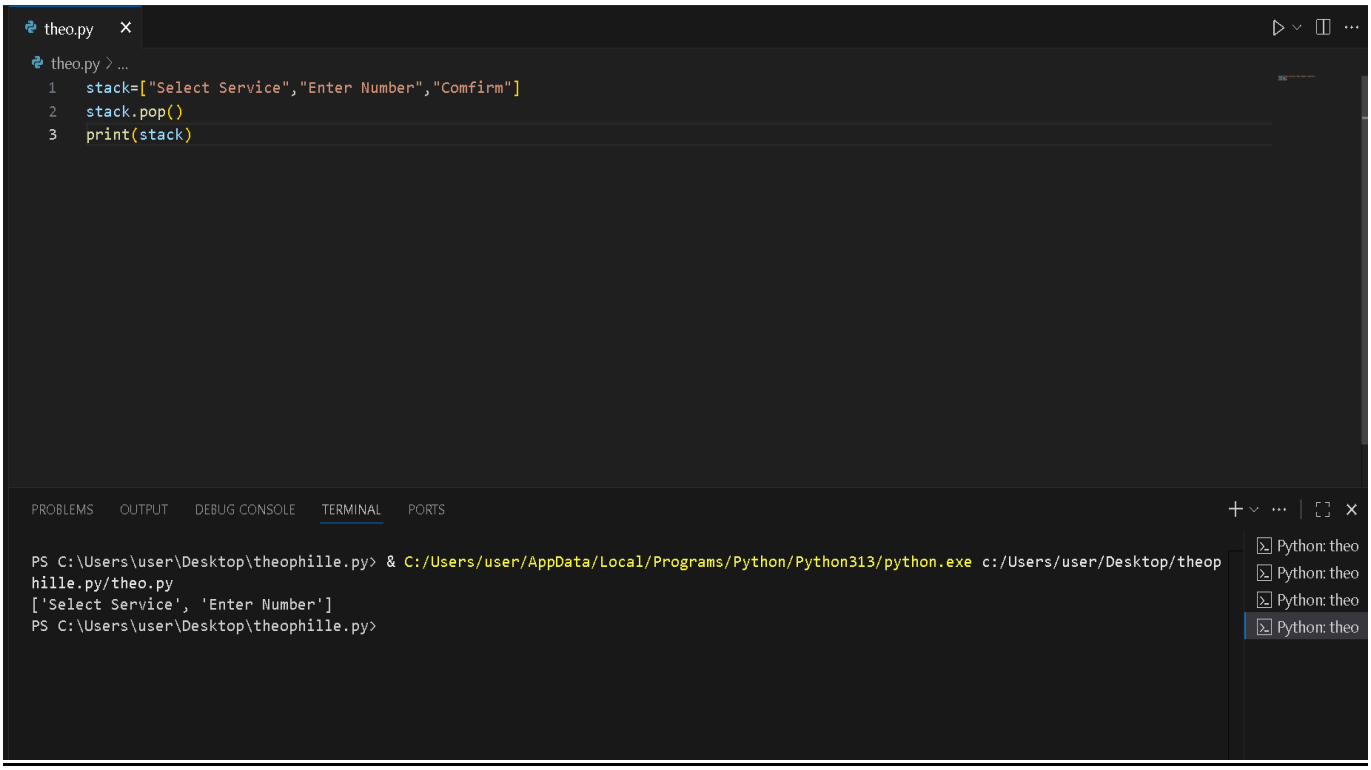
1. **Practical: UR** student pushes [“open Email”,” Type Message” “Send”] Undo last , the left one is [“Open Email” , “ Type Message”]

Practical



Q2. Practical: MoMo stacks stores [“Select Service” , “Enter Number” ,”Confirm”].Pop one . the remans is [“Select Service”,“Enter Number”]

Practical by Code



Challenge: Show how stack reverses the word "DATA"

Algorithmic Steps:

- Initialize an empty stack.
    - We'll use a Python list as the stack.
- Push each character of the word onto the stack.
    - This builds the stack in the order: D, A, T, A (bottom to top).
    - Initialize an empty string for the reversed result.
- Pop each character from the stack and append to the result string.
    - Since a stack is LIFO, popping reverses the order.
    - Output the reversed word.

Corresponding code lines

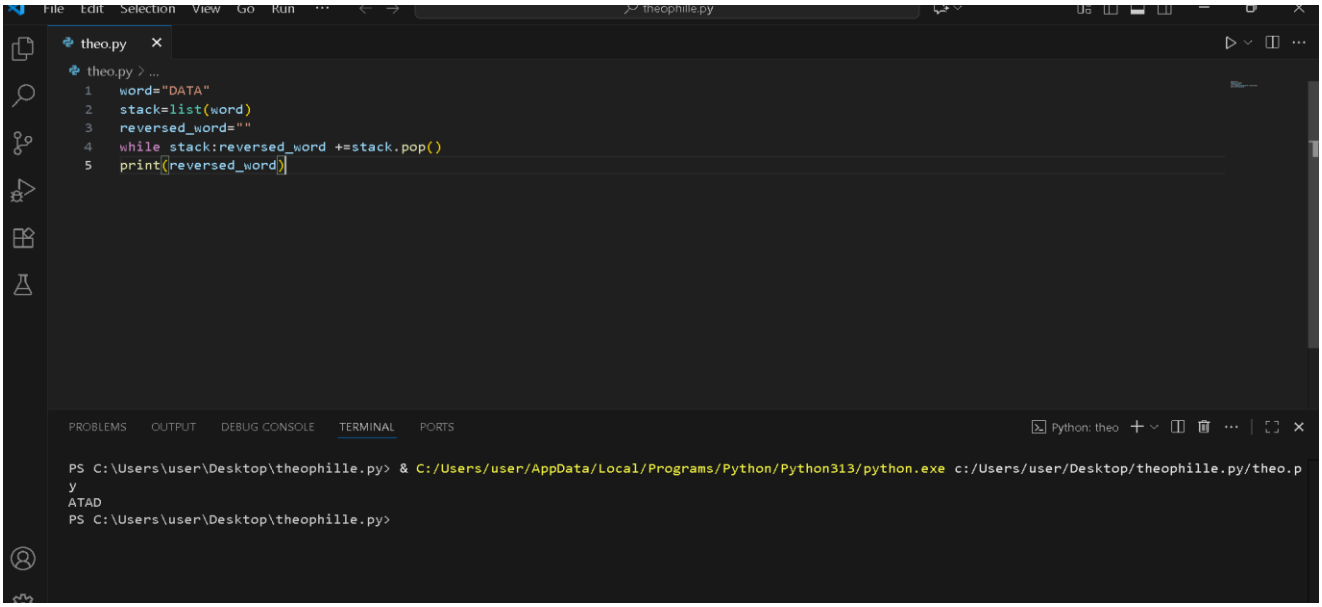
Word="DATA"

Stack=list(word)

Reversed_word=""

While stack:reversed_word += stack.pop()

Print(reversed_word)



<u>Reflection questions:</u> Why is stack good for temporary undo but not queues?

- Stacks are good for temporary undo because they let you retrace your steps backwards in the correct order (last step undone first). Queues are not suitable, since they process from the oldest action, which doesn't match how undo is supposed to work.

Queue Questions:

- **Practical**

At Airtel shop, 7 clients join for SIM swap . After 3 are served, who is next?

The next is Client 4

```python
from collections import deque
queue=deque(["Client1","Client2","Client3","Client4","Client5","Client6","Client7"])
print("Initial Queue:",list(queue))
for _ in range(3):queue.popleft()
print("Next to be served:",queue[0])
```

```
PS C:\Users\user\Desktop\theophille.py> & C:/Users/user/AppData/Local/Programs/Python/Python313/python.exe c:/Users/user/Desktop/theophille.py/theo.p
y
Initial Queue: ['Client1', 'Client2', 'Client3', 'Client4', 'Client5', 'Client6', 'Client7']
Next to be served: Client4
PS C:\Users\user\Desktop\theophille.py>
```

- Practical (Rwanda): In Kigali restaurant, 5 customers queue for orders. Who is served last? , The person who served last is last customer 5

Practical

```python
queue = ["Cust1","Cust2","Cust3","Cust4","Cust5"]

print("Queue:", queue)
print("Served Last:", queue[-1])
```

```
PS C:\Users\user\Desktop\theophille.py> & C:/Users/user/AppData/Local/Programs/Python/Python313/python.exe c:/Users/user/Desktop/theophille.py/theo.p
y
Queue: ['Cust1', 'Cust2', 'Cust3', 'Cust4', 'Cust5']
Served Last: Cust5
PS C:\Users\user\Desktop\theophille.py>
```

Challenge Questions: Challenge: Implement a priority queue for CHUK emergency patients. Why is this better than simple FIFO?

Algorithm Steps:

1. Assign priority: Critical > Serious > Normal.

2. Insert patients with priority, name.
3. Always serve the patient with highest priority first, not just the one who came earlier.

With Code

```python
import heapq

# (priority, patient)
# Lower number = higher priority
patients = []
heapq.heappush(patients, (1,"Patient A - Critical"))
heapq.heappush(patients, (2,"Patient B - Serious"))
heapq.heappush(patients, (3,"Patient C - Normal"))

print("Serving order:")
while patients:
    print(heapq.heappop(patients))
```

```
PS C:\Users\user\Desktop\theophille.py> & C:/Users/user/AppData/Local/Programs/Python/Python313/python.exe c:/Users/user/Desktop/theophille.py/theo.p
y
Serving order:
(1, 'Patient A - Critical')
(2, 'Patient B - Serious')
(3, 'Patient C - Normal')
PS C:\Users\user\Desktop\theophille.py>
```

Reflection: Why is FIFO essential for service fairness?

- FIFO ensures fairness because people are served in the order they arrived.
- It avoids favoritism, skipping, or discrimination.
- Real-life services like banks, restaurants, or ticketing must be fair, otherwise customers lose trust.
  That's why queues stick to FIFO.