PUSH AND POP ASSIGNMENT

Q1: How does this show the LIFO nature of stacks (MTN MoMo app example)?

- In the MoMo app, when you input payment details (step 1, step 2, step 3…), each new step is *pushed* onto the stack.
- If you press the back button, the **last step you entered is the first one removed** — just like in a stack, where the **Last In = First Out (LIFO)**.
- So, if the last thing you did was enter an amount, pressing back will remove that before touching the earlier steps (like recipient's details).

Q2: Why is this action similar to popping from a stack (UR Canvas example)?

- In Canvas, each page/module you navigate to is added on top of the navigation stack.
- When you press *back*, the most recent page (on top) is removed — this is exactly a **pop operation**.
- You don't skip directly to the first page; you must remove (pop) the top one first.

- **Push** = adding a new step/page.
- **Pop** = removing the most recent step/page.
- Both examples show the **LIFO rule** of stacks in real life

Download MTN MoMo stack diagram

the **MoMo app steps** stack up.
When you press back, the **top item (Step 3: Enter Amount)** is popped out first — showing the LIFO nature.

Download UR Canvas stack diagram

Here's the **UR Canvas navigation stack** when you press back, the top page (*Page 3: Assignment*) is popped off first, leaving you at *Page 2: Module*.

Q3. How could a stack enable the undo function when correcting mistakes (BK Mobile Banking example)?

- In mobile banking, each transaction you perform is **pushed** onto the history stack (latest action on top).
- If you make a mistake (e.g., wrong amount typed), the **undo** function simply *pops* the top action off the stack.
- This way, the last mistake you made is the first one corrected → exactly the **LIFO principle** in action.

Stacks make undo easy because they keep the most recent actions right at the top, ready to remove.

Q4. How can stacks ensure forms are correctly balanced (Irembo registration forms example)?

- Think of **form fields like brackets**: for every opening field (e.g., "Start of Name section"), there must be a matching closing field (e.g., "End of Name section").
- A stack checks this by:
    1. **Push** every opening field onto the stack.
    2. **Pop** when the matching closing field is found.
- If at the end, the stack is empty ✅ → the form is correctly matched.
- If something is left in the stack ❌ → fields are incomplete/mismatched.

Just like in code where parentheses () or {} must be balanced, forms also need to "open and close" properly, and stacks help validate this automatically.

Q5: Task sequence (Push/Pop example)

Operations:

1. Push (“CBE notes”) → Stack = [CBE notes]
2. Push (“Math revision”) → Stack = [CBE notes, Math revision]
3. Push(“Debate”) → Stack = [CBE notes, Math revision, Debate]
4. Pop () → removes “Debate” → Stack = [CBE notes, Math revision]
5. Push (“Group assignment”) → Stack = [CBE notes, Math revision, Group assignment]

Top of stack = “Group assignment”

Q6 Undo with multiple Pops (ICT exam example)

- Imagine each answer/action you do is pushed onto the stack.
- If the student undoes **3 recent actions**, that means **Pop () x3**.
- The 3 latest answers will be removed, leaving only the **earlier answers still in the stack**.

So, the stack keeps **all answers except the last 3**.

Q7: Pop to backtrack in RwandAir booking

- **Idea:** A stack remembers the pages or steps you’ve visited.
- **How it works:**
    1. Each step (e.g., “Choose flight,” “Enter passenger info,” “Payment”) is **pushed** onto the stack.
    2. When the passenger clicks **“Back”**, the app **pops** the top step from the stack, returning to the previous step.
- ✅ This lets the passenger retrace exactly in **reverse order** of steps visited.

Q8: Push/Pop to reverse “Umwana ni umutware”

- **Algorithm using stack:**
    1. Start with an empty stack.

2. Push each word:
    - Push("Umwana") → ["Umwana"]
    - Push("ni") → ["Umwana", "ni"]
    - Push("umutware") → ["Umwana", "ni", "umutware"]
3. Pop words one by one and write them in order popped:
    - Pop → "umutware"
    - Pop → "ni"
    - Pop → "Umwana"

- ✅ **Reversed proverb:** "umutware ni Umwana"

Q9: DFS using a stack in Kigali Public Library

- **Why a stack suits DFS:**
    - DFS (Depth-First Search) explores as far as possible along each branch before backtracking.
    - Stack **remembers the path** and allows backtracking to the last unexplored shelf.
    - Queue (FIFO) would explore level by level (breadth-first), which is slower if you want to go deep into shelves first.

Q10: Push/Pop for navigation in BK Mobile app

- **Suggested stack feature:**
    - **Transaction undo/redo:**
        1. Every transaction or page visited is **pushed** to a stack.
        2. User clicks **"Back"** → pop to see previous transaction details.
        3. Optional **redo stack**: popped transactions are pushed to a second stack if user wants to go forward again.

- ✅ This makes moving through history smooth and reversible.

A. Basics

Q1: Restaurant serving in order → FIFO behavior

- **FIFO:** First In, First Out.
- **How it works:**
    - Customers arrive and join the **rear of the queue** (enqueue).
    - The customer who came first is **served first** (dequeue).
- ✅ This shows FIFO because no one who arrived later can skip ahead

Q2: YouTube playlist → like a dequeue operation

- **Explanation:**
    - Videos are lined up to play in order.
    - The next video in the playlist (front of the queue) plays automatically → **dequeue** operation.
- ✅ The playlist automatically removes the front video from "waiting" and plays it, exactly like a queue.

B. Application

Q3: RRA office line → real-life queue

- People **join the line at the rear** when submitting tax payments → **enqueue**.
- People **served one by one from the front** → **dequeue**.
- ✅ This is exactly a **real-life queue** system: first come, first served.

Q4: MTN/Airtel SIM replacement → improving customer service

- **Queues ensure:**
    1. Fairness – first come, first served.
    2. Orderly processing – reduces confusion or crowding.

3. Efficiency – staff can serve systematically without skipping anyone.

- ✅ This makes the service faster, smoother, and more customer-friendly.

LOGICAL

Q5: Equity Bank operations
Operations:
Enqueue("Alice") → Enqueue("Eric") → Enqueue("Chantal") → Dequeue() → Enqueue("Jean")

- Step by step:
  1. Enqueue Alice → [Alice]
  2. Enqueue Eric → [Alice, Eric]
  3. Enqueue Chantal → [Alice, Eric, Chantal]
  4. Dequeue → removes front → Alice removed → [Eric, Chantal]
  5. Enqueue Jean → [Eric, Chantal, Jean]

  ✅ **Front of the queue: Eric**

Q6: RSSB pension applications → fairness using queue

- **Explanation:**
  - Applications are handled **in the order received** (FIFO).
  - First applicant is processed first; the last applicant waits at the end.
- ✅ Ensures **fairness**, no one jumps the line.

Q7: Different queue types in Rwandan life

Queue type	Real-life example	Explanation
Linear queue	People at a wedding buffet	Serve one by one, first come, first served.
Circular queue	Buses looping at Nyabugogo	After the last bus, the next goes back to the start — continuous loop.

Queue type	Real-life example	Explanation
Deque (double-ended queue)	Boarding a bus from front/rear	People can enter/exit from **both ends.**

Q8: Kigali restaurant orders → queue modeling

- **Process:**
    1. Customers place orders → **enqueue** in "waiting to be served" list.
    2. When food is ready → **dequeue** orders in the same sequence.
- ✅ Queues ensure **orders are served in correct order**, avoiding mix-ups.

Q9: CHUK hospital emergencies → priority queue

- **Why priority queue, not normal queue:**
    - Normal queue = FIFO → first come, first served.
    - Priority queue → **patients with higher urgency (emergency cases) jump ahead**, even if they arrived later.
- ✅ Ensures **critical patients get immediate attention**

Q10: Moto/e-bike taxi app → fair matching using queue

- **Implementation idea:**
    1. Riders **enqueue** as they become available.
    2. Students (passengers) arrive → app **dequeues** the first available driver.
    3. Optional: separate **priority queues** for special cases (e.g., VIP passengers, shared rides).
- ✅ This ensures **drivers are matched fairly** in the order they became available.

ENDDD