BIT - Data Structure Exercise: Stack and Queue

Part I — STACK

A. Basics

Q1 — MTN MoMo: How does pressing back show LIFO?
Each step in the payment flow (e.g., Account → Amount → Confirm → PIN) is pushed onto the stack as you progress. Pressing Back removes the most recent step — that is the top of the stack — so the last step you entered is the first one removed. That is exactly LIFO.

Python demo (simple):

```
steps = []
steps.append('Account')   # push
steps.append('Amount')
steps.append('Confirm')
steps.append('PIN')
print('Before Back:', steps)
last = steps.pop()
print('Back removed:', last)
print('Now top:', steps[-1])
```

Q2 — UR Canvas back = popping: Why similar?
Navigating pages/modules is the same idea. Each visited module is pushed; pressing Back pops the last visited module and returns you to the previous one. Programmatically this is stack.pop().

B. Application

Q3 — BK Mobile: How could stacks enable undo when correcting mistakes?
Two common stack-based undo approaches:
1. Action-stack (command pattern): Push every user action (or inverse action) onto an undo stack. undo() pops the last action and applies the inverse.
2. Snapshot-stack (state history): Push snapshots of the state (or diffs). Pop to restore the previous snapshot.

Important caveat: For financial transactions you cannot blindly pop() confirmed transactions — you must create compensating transactions (a new transaction that reverses an effect) and log them. Undo is safe only before final commit or for UI edits.

Python sketch (simple undo for UI edits):

```
history = []
current = {'recipient': 'A', 'amount': 100, 'note': 'Rent'}
```

```
history.append(current.copy())
current['amount'] = 150
history.append(current.copy())

current = history.pop()      # revert to previous snapshot
```

Q4 — Irembo forms: How do stacks ensure forms are balanced?
For matching opening/closing tokens (parentheses, brackets, or even matching field start/end tags), push each opening token onto a stack; when you see a closing token, pop and check if it matches the corresponding opening. If a mismatch or extra closing occurs → invalid. At the end, stack must be empty.

Python example:

```
def is_balanced(s):
    stack = []
    pairs = {')':'(', '}':'{', ']':'['}
    for ch in s:
        if ch in '({[':
            stack.append(ch)
        elif ch in ')}]':
            if not stack or stack.pop() != pairs[ch]:
                return False
    return not stack

print(is_balanced('{[()]}'))  # True
print(is_balanced('{[(])}'))  # False
```

C. Logical

Q5 — Push/Pop sequence
Operations: Push('CBE notes'), Push('Math revision'), Push('Debate'), Pop(), Push('Group assignment')

Step-by-step stack contents:
1. ['CBE notes']
2. ['CBE notes', 'Math revision']
3. ['CBE notes', 'Math revision', 'Debate']
4. Pop() removes 'Debate' → ['CBE notes', 'Math revision']
5. Push 'Group assignment' → ['CBE notes', 'Math revision', 'Group assignment']

Answer: Top (next) = Group assignment.

Q6 — Undo with 3 Pops (ICT exam)
Suppose the answers stack: ['Q1', 'Q2', 'Q3', 'Q4', 'Q5'] (Q5 is top). Undo 3 times removes Q5, Q4, Q3. Remaining: ['Q1', 'Q2'].

D. Advanced Thinking

Q7 — RwandAir booking backtracking: How stack enables retracing?
Each booking step can be pushed as a state or page. Popping returns you to the previous state with its data. Because stack stores the path, repeated pop() retraces step-by-step backwards.
UI note: Use a pair of stacks (back_stack, forward_stack) to implement Back/Forward navigation.

Q8 — Reverse sentence using a stack
Sentence: 'Umwana ni umutware'
Algorithm: Split into words, push each onto stack, then pop all to reverse order.

Python:
```
def reverse_words(s):
    stack = []
    for w in s.split():
        stack.append(w)
    rev = []
    while stack:
        rev.append(stack.pop())
    return ' '.join(rev)

print(reverse_words('Umwana ni umutware'))  # 'umutware ni Umwana'
```

Q9 — DFS using a stack: Why stack is better than queue for deep search?
DFS explores as deep as possible before backtracking. A stack naturally supports this.
Memory: O(depth) vs BFS's O(b^d).

Q10 — Suggest a stack feature for BK Mobile transaction navigation
Feature: History Back/Forward + Undo draft. Maintain two stacks (back_stack, forward_stack). Undo stack stores draft edits.

Part II — QUEUE

A. Basics

Q1 — Restaurant queue FIFO:
Customers enter line and are served in order. First in → first out.

Python demo:
```
from collections import deque
q = deque()
q.append('Amina')
q.append('Brian')
q.append('Chloe')
print(q.popleft())  # Amina
```

Q2 — YouTube playlist as dequeue:
Playlist plays first item, removes it, then goes to next — same as queue dequeue.

B. Application

Q3 — RRA tax line as queue:
People waiting to pay taxes form a queue. First ticket called = served first.

Q4 — How queues improve customer service (MTN/Airtel centers):
• Enforce fairness
• Reduce conflicts
• Allow measurement of wait times
• Enable priority lanes

C. Logical

Q5 — Equity Bank sequence
Operations: Enqueue('Alice'), Enqueue('Eric'), Enqueue('Chantal'), Dequeue(), Enqueue('Jean')

Result: ['Eric','Chantal','Jean'] → Front = Eric

Q6 — FIFO message handling (RSSB pensions):
By processing in arrival order, no later arrival overtakes earlier ones — ensures fairness.

D. Advanced Thinking

Q7 — Map queue types to Rwandan life:
• Linear queue: Wedding buffet line
• Circular queue: Buses looping Nyabugogo
• Deque: Boarding bus front/rear

Q8 — Restaurant orders: Enqueue orders, dequeue when ready.

Python:
```
orders = deque()
orders.append(('Table1','Fish'))
orders.append(('Table2','Beef'))
served = orders.popleft()
print('Serve:', served)
```

Q9 — Priority queue at CHUK: Why not normal queue?
In triage, critical cases must be treated first → use priority queue.

Python with heapq:
```
import heapq
pq = []
heapq.heappush(pq, (2,'Patient: broken arm'))
heapq.heappush(pq, (1,'Patient: heart attack'))
print(heapq.heappop(pq))
```

Q10 — Matching riders and students (moto/e-bike app) — fair queue design: Maintain driver and rider queues. Match in FIFO order with fairness + geographic optimization.