Project 77 — Stack and Queue Questions

Stack Questions

Q1 — Practical (MoMo undo)

Question: Push ['Enter Code', 'Enter PIN', 'Confirm'] onto a stack. Undo one. Which remains?

Python code (complete):

```
stack = []
stack.append('Enter Code')
stack.append('Enter PIN')
stack.append('Confirm')
stack.pop()
print('After undo (pop):', stack)
```

Answer: After undo, the stack contains ['Enter Code', 'Enter PIN'].

Q2 — Practical (UR lessons stack)

Question: UR pushes ['Lesson1', 'Lesson2', 'Lesson3'] then pops one. Which is top?

Python code (complete):

```
stack = ['Lesson1', 'Lesson2', 'Lesson3']
stack.pop()
print('After pop:', stack)
print('Top is:', stack[-1])
```

Answer: The top element after popping is 'Lesson2'.

Q3 — Challenge (Reverse 'RWANDALIFE' using a stack)

```
word = 'RWANDALIFE'
Original: RWANDALIFE
Reversed: EFILADNAWR
stack = []
stack.append('R')
stack.append('W')
stack.append('A')
stack.append('N')
stack.append('D')
stack.append('A')
stack.append('L')
stack.append('I')
stack.append('F')
stack.append('E')
reversed_word = ''
```

```
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
reversed_word += stack.pop()
print('Reversed (no-loops):', reversed_word)
Reversed: EFILADNAWR
```

Q4 — Reflection (Why stacks model undo actions well?)

Stacks match undo behavior because they use Last-In, First-Out (LIFO) ordering. The most recent action is placed on top and is therefore the first to be removed when performing an undo. This naturally mirrors how users expect undo to work: revert the most recent change first. The structure is simple, predictable, and easy to implement in software (edit history, command stacks, etc.).

Queue Questions

Q5 — Practical (RRA queue)

Question: Six citizens queue ['Citizen1'..'Citizen6']. After 3 are served, who is next?

Python code :

Output:

```
Initial queue: ['Citizen1', 'Citizen2', 'Citizen3', 'Citizen4',
'Citizen5', 'Citizen6']
Queue after serving 3: ['Citizen4', 'Citizen5', 'Citizen6']
Next to be served: Citizen4
```

Answer: Citizen 4 is next to be served.

Q6 — Practical (Airtel queue)

Question: Nine clients queue ['Client1'..'Client9']. Who is last?

Output:

```
Initial queue: ['Client1', 'Client2', 'Client3', 'Client4', 'Client5',
'Client6', 'Client7', 'Client8', 'Client9']
Last client in queue: Client9
```

Answer: The last client is Client9.

Q7 — Challenge (Queue vs Stack for event tickets)

Algorithmic reasoning and recommendation:

1. People who arrive earlier should receive tickets earlier — that's the natural expectation.
2. A queue implements First-In, First-Out (FIFO), so the earliest arrival is served first.
3. A stack implements Last-In, First-Out (LIFO), which would give advantage to the latest arrival.
4. Therefore, a queue is the correct structure for ticket distribution to ensure fairness and predictability.

Q8 — Reflection (Why FIFO ensures fairness in tickets?)

FIFO ensures fairness because it preserves the arrival order: whoever joins the line first is the first to be served. This prevents later arrivals from cutting ahead and creates a clear, objective rule for service order. For public-facing services like ticketing, that transparency is important for trust and smooth operations.