1. OOPS: super() & Constructor Chaining

What is super()?

The `super()` function is a built-in method that returns a temporary object of the parent class. It allows you to call methods and constructors from the parent class inside a child class.

- **Constructor Chaining:** This is the process of calling one constructor from another. In inheritance, when you create a child object, you typically want the parent's initialization logic to run first to ensure base attributes (like an `id` or `timestamp`) are correctly set.

Key Rules:

1. **Parent First:** Usually, `super().__init__()` is called as the first line in the child constructor.
2. **Reuse:** It prevents code duplication by letting the parent handle common attributes.

2. OOPS Scenario: Extending Account Behavior

Problem: You have a base `Account` class. How do you efficiently create `SavingsAccount` and `LoanAccount` without rewriting the balance and account number logic?

Analysis:

1. **The Base Class (`Account`):** Handles `account_number` and `balance`.
2. **The Extension (`SavingsAccount`):** - Needs everything the base has PLUS an `interest_rate`.
   - **Code Logic:** In the constructor, call `super().__init__(account_number, balance)` then set `self.interest_rate = rate`.
3. **The Benefit:** If the way we validate `account_number` changes, we only change it in the `Account` class. Both child classes will automatically benefit from the fix. This is "DRY" (Don't Repeat Yourself) programming at its best.

3. Programming: Reverse a Singly Linked List

This is a classic "must-know" problem that tests your understanding of pointer manipulation.

Problem: Given the `head` of a linked list, reverse it in-place.

The Three-Pointer Approach

To reverse the list, we need to keep track of the **previous**, **current**, and **next** nodes.

Algorithm:

1. Initialize `prev = NULL`, `curr = head`, and `next = NULL`.
2. While `curr` is not `NULL`:

- `next = curr.next` (Save the next node).
- `curr.next = prev` (Reverse the current node's pointer).
- `prev = curr` (Move `prev` forward).
- `curr = next` (Move `curr` forward).

3. `head = prev` (The new head is the last non-NULL node).

Complexity:

- **Time:** $O(n)$ since we visit each node once.
- **Space:** $O(1)$ because we only use three pointer variables regardless of list size.

4. SQL: FULL JOIN (Interview Perspective)

A `FULL JOIN` returns all records when there is a match in either the left or the right table.

Interview FAQ: "Why use FULL JOIN over LEFT or RIGHT?"

- **Data Reconciliation:** Use it to find discrepancies between two systems. For example, comparing a "Legacy Database" and a "New Database" to see which records exist in only one or in both.
- **Complete Mapping:** When you need a full list of both entities (e.g., All Books and All Authors) regardless of whether they are linked.

Handling NULLs

In an interview, you might be asked to find rows that exist *only* in Table A or *only* in Table B using a FULL JOIN.

```
SELECT a.id, b.id
FROM table_a a
FULL JOIN table_b b ON a.id = b.id
WHERE a.id IS NULL OR b.id IS NULL;
```

FULL JOIN vs. UNION

- **Join:** Combines columns from different tables side-by-side.
- **Union:** Appends rows from one table beneath another (requires the same number of columns).

Summary Table

Topic	Focus	Key Takeaway
OOPS	`super()`	Ensures the parent's "foundation" is built before the child adds "rooms."

DSA	Reverse List	$O(n)$ time with $O(1)$ space using the three-pointer swap.
SQL	FULL JOIN	The "Union of Joins"; best for data migration and audit reports.

10 days down, 50 to go! You've officially mastered the core mechanics of object initialization and