1. OOPS: Method Overloading vs. Method Overriding

While both are forms of Polymorphism, they occur at different stages of the program lifecycle.

Comparison Table

Feature	Method Overloading	Method Overriding
Binding Time	Compile-time (Static)	Runtime (Dynamic)
Class	Occurs within the **same** class.	Occurs across **different** (Parent/Child) classes.
Parameters	Must be **different** (type or number).	Must be the **same**.
Inheritance	Not required.	**Mandatory** relationship.
Purpose	To provide multiple ways to do the same task.	To provide a specific implementation for a generic task.

2. OOPS Scenario: Decoupling Client Code

Problem: In a payment system, the "Client Code" (the part that processes the order) shouldn't care if the user is paying via Credit Card, Crypto, or Rewards Points.

Analysis:

1. **The Overriding Solution:** The base class `Payment` defines `execute_transaction()` . Every specific method (e.g., `CryptoPayment`) overrides it.
2. **Client Perspective:** The client code simply holds a reference to a `Payment` object and calls `payment.execute_transaction()` .
3. **The Benefit:** You can add a new payment method (e.g., `ApplePay`) simply by creating a new subclass and overriding the method. The existing "Client Code" remains untouched. This adheres to the **Open-Closed Principle** (Open for extension, closed for modification).

3. Programming: Linked List Traversal

A **Singly Linked List** is a linear data structure where elements are not stored in contiguous memory locations. Instead, elements are linked using pointers.

Structure of a Node

Each node contains:

1. **Data:** The value.
2. **Next:** A pointer/reference to the next node in the sequence.

Traversal Logic

To search or print elements, we must start at the `head` and follow the pointers until we reach `NULL`.

Algorithm:

1. Set `current = head`.
2. While `current` is not `NULL`:
   - If `current.data == target`: Return True (Element found).
   - Move to the next node: `current = current.next`.
3. Return False if the end of the list is reached.

Edge Cases:

- **Empty List:** `head` is `NULL`.
- **Single Node:** The first node's `next` is `NULL`.
- **Target at Tail:** Must traverse the entire list ($O(n)$).

4. SQL: Complex JOIN Conditions

Sometimes, a simple `table1.id = table2.id` isn't enough to filter the correct data. We can use multiple conditions within the `ON` clause or combine them with logical operators.

Scenario: Matching records within a date range

Suppose we want to join `Sales` and `Promotions`, but only if the sale happened while the promotion was active.

Syntax Example:

```
SELECT sales.item_id, promotions.promo_name
FROM sales
INNER JOIN promotions
    ON sales.product_id = promotions.product_id
    AND sales.sale_date BETWEEN promotions.start_date AND promotions.end_date
WHERE sales.region = 'North';
```

Why use multiple conditions in ON vs. WHERE?

- **Clarity:** Conditions in `ON` define *how* the tables relate. Conditions in `WHERE` define *which* results you want to see.
- **Performance:** In `OUTER JOINs`, putting conditions in the `ON` clause changes the result set significantly compared to putting them in the `WHERE` clause (which filters after the join).

Summary Table

Topic	Focus	Key Takeaway
OOPS	Overloading vs. Overriding	Overloading is about method signatures; Overriding is about inheritance logic.
DSA	Linked Lists	Unlike arrays, search is $O(n)$ because we cannot access elements by index.
SQL	Complex Joins	Use `AND` / `BETWEEN` in `ON` clauses to link tables based on multiple logical criteria.

Day 9 complete. You've transitioned from contiguous arrays to pointer-based structures!