Lesson 9: Algorithms & Python Programming

1. The "Thinking" Phase

Problem Solving & Design

- **Problem Solving Process:** Understand Problem → Plan Solution → Implement → Evaluate.
- **Methodologies:**
  - **Modularization:** Breaking a big problem into smaller, manageable sub-problems (modules).
  - **Top-down design & Stepwise Refinement:** Starting with a general solution and gradually adding details in steps.
- **Structure Charts:** Diagrams that show the hierarchical breakdown of a system into its modules.

Algorithms

- **Definition:** A finite sequence of well-defined instructions to solve a problem.
- **Core Constructs:** Sequence, Selection, Repetition (Definite/Indefinite Iteration).
- **Representations:**
  - **Flowcharts:** Diagrams. Symbols: Terminator (Oval), Process (Rectangle), I/O (Parallelogram), Decision (Diamond).
  - **Pseudo-code:** Human-readable description. Keywords: `BEGIN`, `READ`, `PRINT`, `IF`, `ELSE`, `WHILE`.
- **Verification:** A **Hand Trace** table is used to manually simulate an algorithm and track variable values to check for logic errors.

2. The "Setup" Phase

Programming Paradigms

- **Imperative:** Describes *how* to get a result (e.g., C, Python).
- **Declarative:** Describes *what* result you want (e.g., SQL).

Program Translation

- **Source Code** (human) vs. **Object Code** (machine).
- **Language Generations:** 1GL (Machine) → 2GL (Assembly) → 3GL (High-Level) → 4GL (Domain-Specific).
- **Translators:** Compiler (translates all at once), Interpreter (translates line-by-line), Hybrid (Python's approach).
- **Linker:** Combines object files/libraries into one executable.

IDE (Integrated Development Environment)

- A software suite combining an **Editor**, **Compiler/Interpreter**, and a **Debugger**.

3. The "Coding" Phase: Python Core

Basics & I/O

- **Comments:** `#`. **Indentation:** Defines code blocks.
- **Data Types:** `int`, `float`, `str`, `bool`.
- **Type Casting:** `input()` always returns a string. Use `int(input())` for numbers.

Operators & Control Structures

Operator Categories

- **Arithmetic:** `+, -, *, /, %, //, **`
- **Relational:** `==, !=, >, <, >=, <=`
- **Logical:** `and, or, not`
- **Bitwise:** `& (AND), | (OR), ( XOR)`

- **Selection:** `if...elif...else`.
- **Repetition:** `for` (definite) & `while` (indefinite).
- **Loop Control:** `break` (exits loop), `continue` (skips to next iteration).
- **Nesting:** Control structures can be placed inside one another.

4. The "Advanced" Phase: Python

Sub-programs (Functions)

- **Built-in:** e.g., `print()`, `len()`. **User-defined:** Uses `def`.
- **Scope: Global** (outside fn) vs. **Local** (inside fn).

Core Data Structures

- **List [ ]:** Ordered, **mutable**. Methods: `.append()`, `.remove()`.
- **Tuple ( ):** Ordered, **immutable**.
- **Dictionary { }:** Unordered, mutable `key:value` pairs.

File Handling with Code

File I/O Example (Recommended Way)

```
# "w" for write, "r" for read, "a" for append
# The 'with' statement handles closing the file
with open("data.txt", "w") as f:
    f.write("Some data")
```

Database Connectivity (MySQL)

Basic Python & MySQL Workflow

```
import mysql.connector
# 1. Connect (requires mysql-connector library)
db = mysql.connector.connect(host="...", user="...")
# 2. Create cursor
cursor = db.cursor()
# 3. Execute SQL query
cursor.execute("INSERT INTO ...")
# 4. Commit changes
db.commit()
# 5. Close connection
db.close()
```

Basic Algorithms

- **Sequential Search:** Checks each item one-by-one until a match is found.
- **Bubble Sort:** Repeatedly steps through the list, compares adjacent items, and swaps them if they are in the wrong order.