B 1.1.2 to B1.1.4

Lesson Overview

Computational thinking is a way of approaching problems so that they can be solved efficiently using logic, patterns, and algorithms.
It's not about coding — it's about **how to think** before you start coding.

This lesson explores **four key concepts**:

1. **Abstraction**
2. **Algorithmic Design**
3. **Decomposition**
4. **Pattern Recognition**

You'll also learn how to represent logical steps visually through **flowcharts**, which are used to design algorithms before writing actual programs.

1. Abstraction

Explanation

Abstraction means focusing on what's important and ignoring unnecessary details.
It helps programmers and problem-solvers simplify reality so they can build clear, efficient systems.

Think of abstraction as **zooming out** — you hide the inner complexity so you can focus on the bigger picture.

Example in Daily Life

When you use Google Maps, you don't see every tree or building — just the key roads, routes, and landmarks you need. That's abstraction in action.

Example in Computing

When you call a built-in function like `print()` in Python or `System.out.println()` in Java, you don't worry about how it works internally — you only care about the result.
This is called **functional abstraction** because the complex details are hidden.

Why It Matters

- Reduces unnecessary complexity in programs
- Makes systems easier to understand and maintain
- Saves time by focusing only on relevant components

Reflection

Abstraction isn't just technical — it's a thinking skill.
It's what allows scientists to model weather, economists to predict markets, or doctors to visualize body systems without getting lost in microscopic details.

2. Algorithmic Design

Explanation

An **algorithm** is a clear, step-by-step plan for solving a problem.
Good algorithmic design means breaking a task into logical, ordered steps that can be followed by a human or a computer.

Before writing any program, a programmer must design the algorithm that defines **what the program should do**, not just **how** it should do it.

Example

Problem: You want to find the average of two numbers.

Algorithm:

1. Ask the user to enter the first number.
2. Ask the user to enter the second number.
3. Add both numbers.
4. Divide the result by 2.
5. Display the average.

Key Qualities of a Good Algorithm

- **Finite:** It must complete in a reasonable time.
- **Unambiguous:** Each step must be clear and specific.
- **Logical:** The steps must be in the correct order.
- **Efficient:** It should use minimal resources.

Real-World Example

Think of a recipe. It gives exact ingredients, order of steps, and expected outcomes. If you skip steps or do them in the wrong order, the dish fails.
That's the same with algorithms — **the order matters**.

3. Decomposition

Explanation

Decomposition means breaking a large, complex problem into smaller, manageable sub-problems.
This makes the task less overwhelming and easier to solve systematically.

Example in Real Life

If your goal is to organize a school festival, you might split it into smaller tasks:

- Stage setup
- Food stalls
- Ticket management
- Security
  Each team handles one part, and when combined, they form the full event.

Example in Programming

When creating a video game, the problem can be decomposed into:

- Player movement
- Enemy behavior
- Score system
- Graphics rendering
  Each of these parts can be programmed separately and then integrated.

Why It Matters

- Enables **collaboration** (different programmers can work on different modules)
- Makes **debugging and testing** easier
- Encourages **reusability** (modules can be used in other projects)

Common Misconception

Decomposition is about **breaking down the problem**, not the program.
You apply decomposition **before** coding, during the planning phase.

4. Pattern Recognition

Explanation

Pattern recognition is the ability to identify recurring trends, similarities, or structures in data or problems.
By recognizing these patterns, we can **reuse solutions**, predict outcomes, and design smarter systems.

Example in Real Life

When you recognize that the route to school takes longer on rainy days, you adjust your departure time. That's pattern recognition — identifying a relationship between weather and travel time.

Example in Computing

When programmers notice that several problems follow a similar logic (like sorting, searching, or counting), they can reuse tested code instead of starting from scratch.

Applications

- **Data analysis:** spotting customer trends
- **Machine learning:** teaching computers to identify images or voices
- **Cybersecurity:** detecting repeated attack patterns

Why It Matters

Pattern recognition builds efficiency. It allows both humans and computers to handle complexity by focusing on what repeats or stands out.

5. Computational Thinking in Action

Computational thinking connects all four concepts — abstraction, algorithmic design, decomposition, and pattern recognition — into one unified approach.

Example: Designing an Online Food Ordering App

1. **Decomposition:** Split the project into ordering, payment, delivery tracking, and feedback modules.
2. **Abstraction:** Focus only on essential customer interactions, not internal code.
3. **Pattern recognition:** Notice that “order” and “payment” use similar data validation logic.
4. **Algorithmic design:** Create clear, ordered steps for each process.

By following this approach, you create an efficient, logical, and user-friendly system.

6. Flowcharts

Explanation

A **flowchart** is a diagram that shows the logical flow of an algorithm using standardized symbols.
It’s often the bridge between a **problem description** and **actual code**.

Common Flowchart Symbols

- **Oval:** Start/End
- **Parallelogram:** Input or Output
- **Rectangle:** Process (calculation or assignment)
- **Diamond:** Decision (Yes/No or True/False)
- **Arrow:** Flow of control

https://www.geeksforgeeks.org/competitive-programming/an-introduction-to-flowcharts/

Example 1 – Find the Sum of Two Numbers

Steps:

1. Start
2. Input A and B
3. `Sum = A + B`
4. Output “Sum =”, Sum
5. End

Logic Type: Sequential

Example 2 – Check if a Number is Even or Odd

Steps:

1. Start
2. Input `num`
3. If `num MOD 2 = 0` → Output “Even”

4. Else → Output “Odd”
5. End

Logic Type: Selection

Example 3 – Find the Largest of Two Numbers

Steps:

1. Start
2. Input A and B
3. If `A > B` → Output “A is larger”
4. Else → Output “B is larger”
5. End

Logic Type: Conditional decision

Example 4 – Calculate Factorial of a Number

Steps:

1. Start
2. Input `n`
3. Initialize `fact = 1`, `i = 1`
4. While `i ≤ n` → `fact = fact × i`, `i = i + 1`
5. Output “Factorial =”, fact
6. End

Logic Type: Iteration

7. Reflection and TOK Connection

“A map is not the territory.”
Just as a map simplifies reality to guide travelers, abstraction simplifies problems so programmers can solve them.

TOK Discussion:
If abstraction simplifies truth, does that mean we lose part of the truth?
How do we decide what details are worth keeping and which to ignore?

Connection to Knowledge:
Computational thinking teaches us that **understanding comes from simplifying complexity**, not memorizing details.

8. Practice and Application

Activity

Create a flowchart for this problem:

“Ask the user for five numbers. Add all positive numbers to a total. Count how many negative numbers were entered. Display both totals.”

Then test it using sample input values.

9. IB-Style Practice Questions

Define

1. Define the term **computational thinking**.
2. Define **abstraction** in the context of computer science.
3. Define **decomposition** with an example.

Outline

4. Outline an algorithm for making a cup of tea.
5. Outline how pattern recognition can help in machine learning.
6. Outline how algorithmic thinking ensures program efficiency.

Explain

7. Explain how abstraction reduces complexity in large projects.
8. Explain how decomposition promotes teamwork in software development.
9. Explain the purpose of using flowcharts before programming.

Discuss

10. Discuss how computational thinking is used in designing an online shopping system.
11. Discuss how pattern recognition and algorithmic thinking are applied in AI systems that recommend products.
12. Discuss the ethical implications of abstraction when used to model human behavior in AI systems.