

1. Curriculum Orientation

- **Learning Objectives:**

- Understand what decomposition is and why it is used in programming.
- Learn how to decompose a problem into sub-problems.
- Learn how to predict the outcome of an algorithm and test if predictions are correct.
- Learn how iterative development is used to write programs.
- Create test plans for specific programs that use a range of data (normal, invalid, boundary).
- Understand why a range of data is needed when testing programs.

2. Classroom Discussion: Core Theory & Concepts

Stage A: Translating Checkpoints (Core Definitions)

- **Decomposition:** Breaking a complex problem into smaller sub-problems.
- **Sub-problem:** A small problem that we combine with other small problems to solve a complex / large problem. Sub-problems can be broken down into steps.
- **Iterative Development:** Programs are written in stages one sub-problem at a time, testing that it works, and then moving to the next sub-problem (repetative development cycle).
- **Predicting Outcomes:** Saying or guessing what will happen (outputs or changes to data) when a program runs with specific inputs.
- **Test Data Types:**
 - **Normal Test Data:** Data that is accepted, or within bounds.
 - **Invalid Test Data:** Data that is not accepted, or outside bounds.
 - **Boundary Test Data:** Data that may or may not be accepted, but is on the edge of what is or isn't allowed.
- **Test Plan:** A structured table used to record and test a program systematically. Key headings include *Test data type*, *Description*, *Example inputs*, *Expected output*, and *Result*.

Test Plan for a Program That Adds Two Numbers

Test Data Type	Description	Example Input(s)	Expected Output	Result
Normal	Typical values entered	5, 3	8	Pass
Boundary	Lowest valid values	0, 0	0	Pass
Boundary	Highest valid values	100, 100	200	Pass
Abnormal / Invalid	Letter entered instead of a number	A, 5	Error message shown	Pass
Abnormal / Invalid	Blank input	(blank), 5	Error message shown	Pass

Stage B: Contextualizing Scenarios (Why We Need These Tools)

Why do we decompose problems?

1. **Natural for humans:** Breaking down large tasks (e.g., cleaning the whole home room-by-room or by chore) gives smaller problems to tackle one at a time.
2. **Team collaboration:** Hundreds of programmers cannot work on the same file at once. Decomposing allows different people or smaller teams to work independently on specific sub-problems simultaneously, writing the program faster.
3. **Specialist utilization:** Specialists in graphics, or user interfaces can focus on their specific areas, improving overall program quality.
4. **Reusability and sub-routines:** Repeating elements (e.g., cleaning windows in different rooms) can be written as reusable sub-routines, saving time so programmers do not write the same code multiple times.
5. **Easier debugging:** It is easier to change a small sub-problem than a large problem because there are fewer statements to test and less can go wrong.

Why do we test with a range of data?

Testing a program with only one piece of data or only normal data does not prove it works in every condition. Without boundary data, edge logic errors (such as using `<` instead of `<=`) will go unnoticed.

Stage C: Troubleshooting Walkthrough (Diagnosing the Friction)

Look at this age-check algorithm for a club allowing children **16 or younger** to join:

```
age = int(input("Enter your age: "))
if age < 16:
    print("You are allowed to join")
else:
    print("Sorry, you are too old to join")
```

- **Test 1 (Normal Data: 10):** 10 is less than 16 → Output: "You are allowed to join" (Pass).
- **Test 2 (Invalid Data: 20):** 20 is not less than 16 → Output: "Sorry, you are too old to join" (Pass).
- **Test 3 (Boundary Data: 16):** 16 is not less than 16 → Output: "Sorry, you are too old to join" (**FAILED: 16-year-olds should be allowed to join!**)
- **Diagnosis and Fix:** The condition used `<` (less than), but it should be `<=` (less than or equal to).

3. Question Bank and Solutions

A. Short Question and Answers

Q1. Define decomposition in computer science.

Answer: Decomposition means breaking down a large, complex problem into smaller sub-problems. These smaller sub-problems are solved individually and then put together to solve the overall problem.

Q2. What is the difference between a sub-problem and an individual step?

Answer: A sub-problem can be broken down further into multiple steps to follow, whereas an individual step is a single instruction that cannot be broken down any further.

Q3. Explain why there is no single "correct" way to decompose a program.

Answer: There are many valid ways to break down any large task. Different programmers can decompose the same problem into different sets of sub-problems, and all of them can still work properly to solve the overall problem.

Q4. State two advantages of dividing a programming task among a team of programmers.

Answer: It allows programmers to work independently at the same time, which lets the program be written more quickly. It also allows specialists in areas like graphics, or interfaces to work on the parts they do best.

Q5. What difficulty arises when many programmers write code for the same project?

Answer: The programmers must coordinate and set rules so their separate parts work together properly. If they do not communicate, they will encounter problems when attempting to combine their individual programs.

Q6. What is iterative development?

Answer: Iterative development is the process of writing a program in small stages. A developer writes a section of code, tests it, and then either fixes it or moves on to write the next section, repeating this cycle until the entire program is complete.

Q7. What does it mean to predict the outcome of a program?

Answer: Predicting the outcome means looking at the program code and inputs to guess what will happen or what will be output when it runs. It involves reading the overall logic without needing to execute every single statement line by line.

Q8. Define normal test data and provide an example from an input range of 1 to 100.

Answer: Normal test data is data that is accepted because it lies comfortably within the allowed boundaries of the program. For an allowed range of 1 to 100, examples of normal test data are 2 or 99.

Q9. Define boundary test data and provide an example from an age limit of "16 or younger".

Answer: Boundary test data is data that lies right on the edge of what is or is not allowed by a condition. For an age limit of 16 or younger, the boundary test data values are 16 (the highest accepted age) and 17 (the lowest rejected age).

Q10. Why are boundary test data values not possible when testing text answers like "NIHOC"?

Answer: Boundary tests only apply when inputs exist on a scale or continuous range. Because text answers are either strictly correct or incorrect, there are no edge values that are close to being accepted or rejected.

B. Second-Order Thinking Skills Questions

Q11 Examine the following morning routine steps and categorize them into Sofia's sub-problems:

Steps: (i) Turn off alarm, (ii) Put notebooks in the pouch, (iii) Chew toast, (iv) Put on school uniform, (v) Brush teeth.

Answer:

- (i) *Turn off alarm* belongs to **Getting up**.
- (ii) *Put notebooks in the pouch* belongs to **Packing bag**.
- (iii) *Chew toast* belongs to **Having breakfast**.
- (iv) *Put on school uniform* belongs to **Getting dressed**.
- (v) *Brush teeth* belongs to **Getting washed**.

Q12 A developer tests an algorithm that accepts ages 16 or younger using only the test inputs 8 and 12. Evaluate whether this program has been tested thoroughly.

Answer: The program has not been tested thoroughly. The developer only used normal test data and omitted invalid data (such as 20) to confirm older applicants are rejected, as well as boundary data (16 and 17) to verify edge logic. Without boundary testing, off-by-one errors (such as using < instead of <=) cannot be caught.

Q13 In a snakes and ladders computer game, Group A decomposed the startup into "Loading and starting a new game", while Group B decomposed it into "Select number of players" and "Set players in first position". Analyze why both approaches are valid.

Answer: Both decompositions are valid because there is no single mandatory way to break down a problem. Group A created a broader sub-problem containing multiple setup steps, while Group B broke the initial setup down into two smaller, more focused sub-problems. Both achieve the exact same operational goal.

C. Higher-Order Thinking Skills (HOTS) Questions

Q14 Design a complete test plan table for a game score input program that accepts scores from 1 to 100 inclusive. Your plan must include normal, invalid, and boundary test data types.

Answer:

Test Data Type	Description	Example Inputs	Expected Output
Normal	Input from 1 to 100 allowed	50	Accepted
Normal	Input from 1 to 100 allowed	2	Accepted
Invalid	Value below lowest allowed number	0	Not accepted
Invalid	Value above highest allowed number	101	Not accepted
Boundary	Lower allowed boundary edge	1	Accepted
Boundary	Upper allowed boundary edge	100	Accepted

Q16 Compare the "all-at-once" approach to writing code with the "iterative development" method for the *Fire, Water, Earth* game. Why is iterative development safer?

Answer: In the all-at-once approach, a programmer writes the entire program code from beginning to end before testing any part of it. If an error occurs, finding the bug is difficult because many statements could be at fault.

In iterative development, the programmer divides the game into sub-problems (e.g., inputting player names first, testing it; then coding element selections, testing it). If code fails, only a few statements need checking, making bugs easier to locate, diagnose, and repair.