Module 7: Algorithms and Computational Thinking

Prof. Dr. Florian Haselbeck
Memoona Shakeel

Weihenstephan-Triesdorf University of Applied Sciences

Learning outcomes

Understand how to think like a programmer, break down problems, spot patterns, plan step-by-step solutions, and express them in pseudocode.

Key Topics

- Computational Thinking
- Decomposition
- Pattern Recognition
- Abstraction
- Algorithms
- Flowcharts & Pseudocode

Computational Thinking (4 ideas that help you solve)

What it is: Computational thinking is a way of approaching problems so they are easier to solve, with a computer or even by hand.

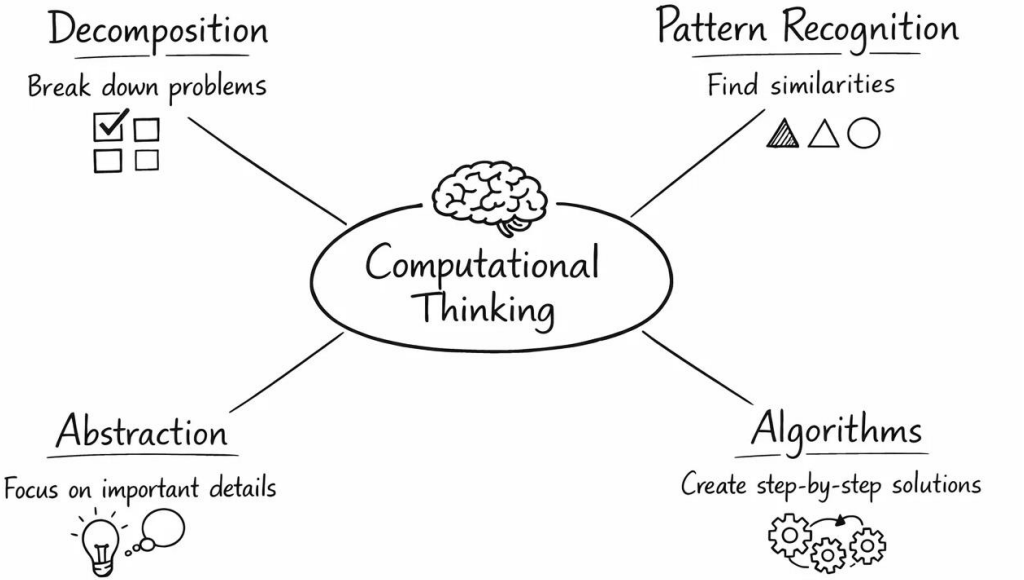
Not Just for Coding

These ideas help with any complex Task like cooking, planning a farm, etc.

Every Python script uses decomposition (functions) and algorithms (if/for logic).

The goal

Turn a vague problem into clear steps a computer can follow exactly.



Computational Thinking

Decomposition

Break a big problem into smaller, manageable parts.

Example: Planning a party

1. Choose a date
2. Make a guest list
3. Send invitations
4. Plan games
5. Prepare food
6. Decorate the venue

Big Problem: Plan a party → Date, Guests, Invites, Games, Food, Decorations

Pattern Recognition

Look for similarities, trends, or patterns in data to make better predictions.

Example: Number pattern

2, 4, 6, 8, 10, ?

→ The numbers go up by 2 each time.

→ Next number will be 12.

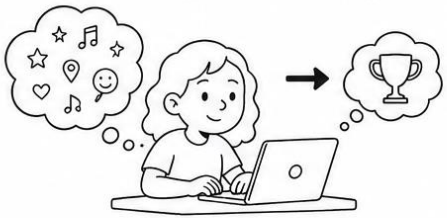
Rule: Add 2

2	4	6	8	10
3	6	9	12	15
5	10	15	20	25
1	3	5	7	

Real-life example:
Weather patterns help predict if it will rain tomorrow.

Abstraction

Focus on the important information and ignore the unnecessary details.



Real-life example: When reading a book, you focus on the main idea instead of every single word.

Example:
You are writing a story.

All details:
- character hair color
- wall color
- shoes they wore
- time of day
- what they had for lunch
- how they felt

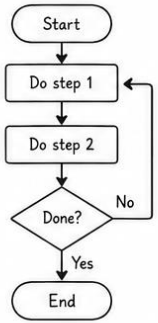
Important details:
- character
- problem
- main events
- solution

Algorithm Design

Create a clear, step-by-step plan (or set of rules) to solve a problem.

Example: Making a sandwich

1. Take two slices of bread.
2. Add your favorite filling.
3. Add any toppings.
4. Close the sandwich.
5. Enjoy!



Real-life example:
Following a recipe is an algorithm!

Decomposition (break the big problem into small pieces)

What it is: Split one large problem into smaller pieces that are each easy to solve on their own.

Why it Helps

A big problem feels overwhelming.

Small pieces feel manageable.

Solve one at a time.

Each small piece become one function.

In Python

Each sub problem = one function

```
def read_sensors()
def ceeck_conditions()
```

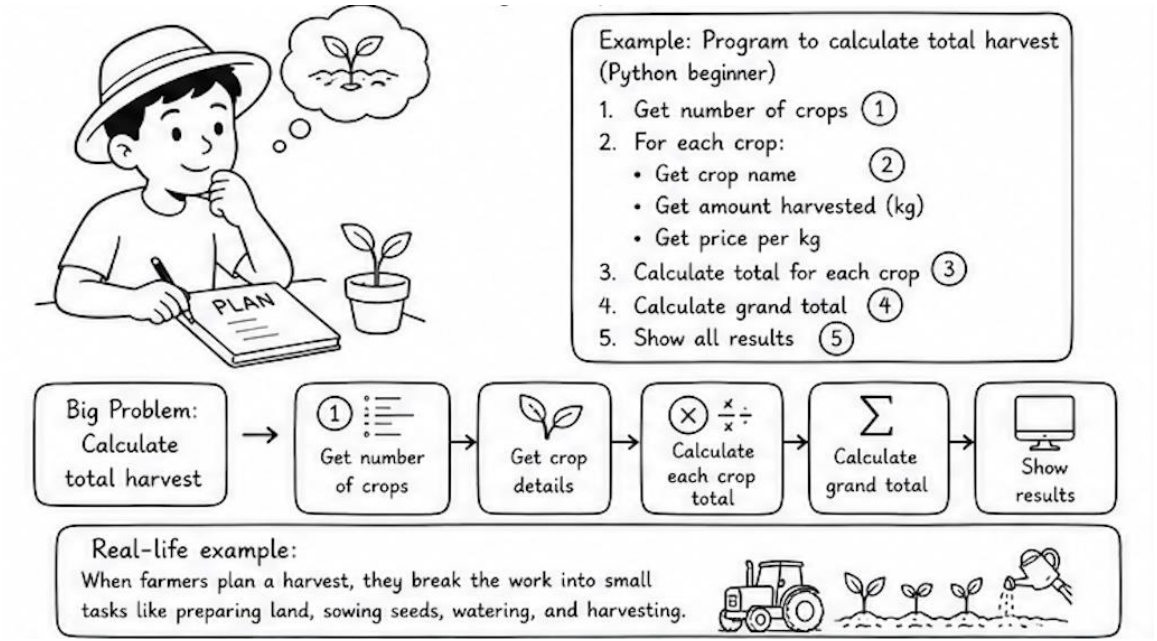
Farm example

'Build an irrigation system'

→ read sensors

→ check conditions

→ open valve → log

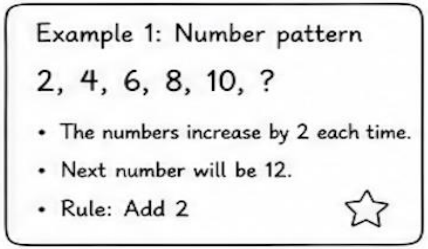


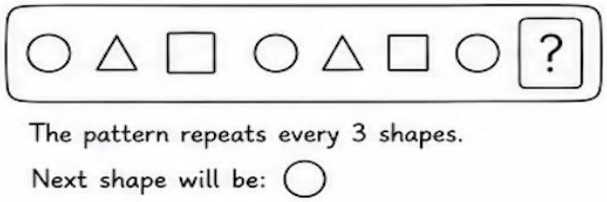
Pattern Recognition

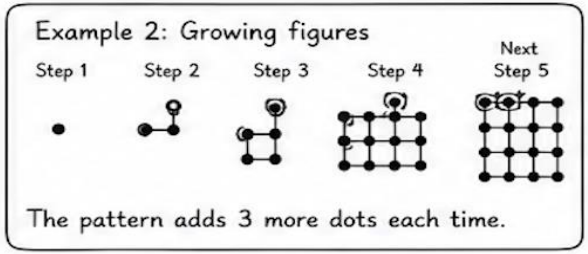
Pattern Recognition (spot what repeats)
Find things that look similar.
If 3 fields need the same check, write ONE function, call it 3 times.
Don't repeat code, spot the pattern, reuse the solution.

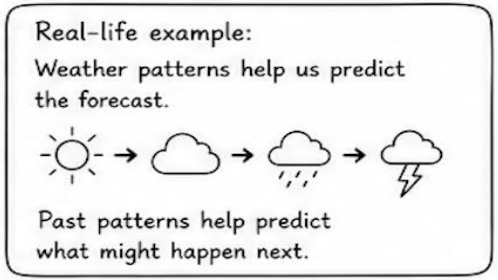
Pattern e.g.

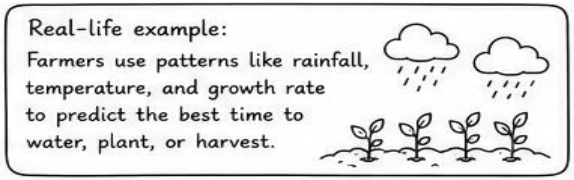
```
for field in fields:
    check_field(field)
```











Abstraction

Abstraction (focus on what matters, hide the rest)

Remove unnecessary detail.

Use function names that say what they do, not how.

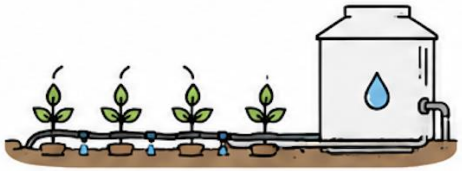
For example, functions len(), print(), sorted() hide all their internal code.

Abstraction e.g.

```
sorted([3,1,2])
# no idea how, just works
```

Example: Water Plants

We create a function water_plants().
It handles all the steps.
We just call the function.



Inside the function (details hidden)

```
def water_plants(duration):
    pump_on()          # turn on pump
    open_valve()       # open valve
    wait(duration)     # water for some time
    close_valve()      # close valve
    pump_off()         # turn off pump
    print("Plants watered!")
```

Using the function (simple)

```
# We just call the function
water_plants(30)

print("Done!")
```

We don't need to know how pump_on(), open_valve(), wait(), close_valve(), or pump_off() work.
We just use water_plants() and it does the job!

Tip for Beginners: Use abstraction (functions) to hide complex steps.
It makes your code easy to read, reuse, and maintain.

Algorithms (a clear step-by-step plan)

What it is: An algorithm is an ordered set of clear steps that solves a specific problem. Same input → same output, every time.

Clear
Every step has one meaning.
No ambiguity, no guessing.

Finite
It's always stops.
Never runs forever.

Correct
Gives the right answer for valid input

Example: Calculate total fertilizer cost (Python)

1. Ask user for amount of fertilizer (kg)
2. Ask user for price per kg
3. Multiply amount by price
4. Show total cost

Example Python code:

```
amount = float(input("Enter amount of fertilizer (kg): "))
price = float(input("Enter price per kg: "))
total = amount * price
print("Total cost is Rs.", total)
```

What it does:

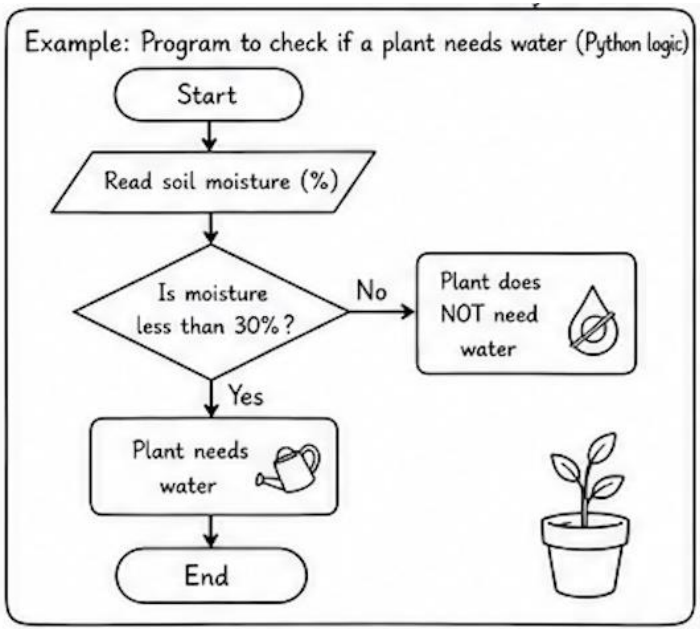
- Takes input from the user
- Multiplies amount and price
- Shows the total cost

Real-life example:
Many farm tasks follow a fixed sequence, like:

★ Tip for beginners: Read the steps first, then try to write the code.

Flowchart (draw the steps before you code them)

What it is: A diagram using boxes and arrows to show the order of steps and decisions in an algorithm. Draw it BEFORE writing code.



Example Python code:

```
moisture = float(input("Enter soil moisture (%): "))
if moisture < 30:
    print("Plant needs water")
else:
    print("Plant does NOT need water")
```

Real-life example:

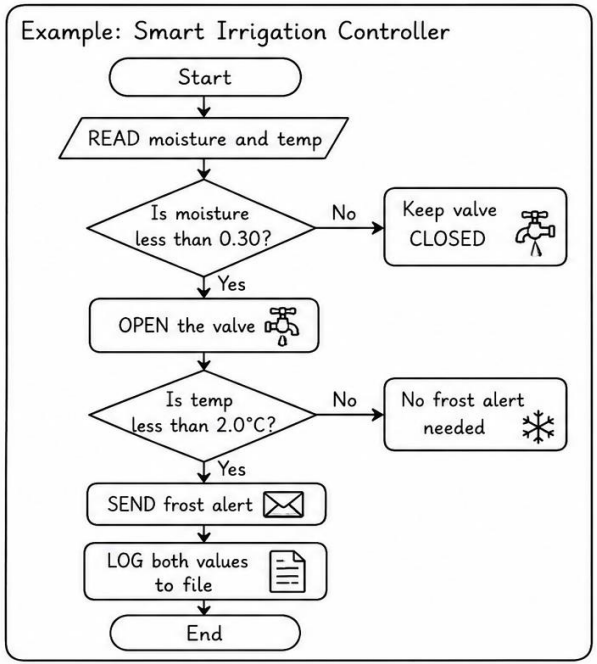
You can use flowcharts for many farm decisions like:

- ✓ When to water plants
- ✓ When to apply fertilizer
- ✓ When to harvest crops

Tip for beginners: Drawing the flowchart first helps you think clearly and avoid mistakes in your code.

Pseudocode (plan in plain English, then translate to Python)

What it is: Pseudocode uses plain English to describe the steps of an algorithm. No Python syntax needed, just clear numbered steps



Pseudocode (plain English)

- READ moisture and temp
- IF moisture < 0.30 THEN OPEN the valve
- IF temp < 2.0 THEN SEND frost alert
- LOG both values to file

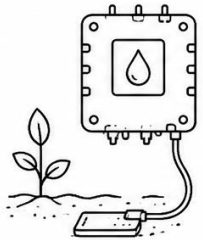
Python code

```
moisture = read_sensor('soil')
temp = read_sensor('temp')

if moisture < 0.30:
    open_valve()

if temp < 2.0:
    send_frost_alert()

log_reading(moisture, temp)
```



Tip for beginners: Planning with pseudocode helps you think through the steps before you write any code. It makes your program clearer and easier to debug!

Overview of key content

- Computational Thinking = Decomposition · Pattern Recognition · Abstraction · Algorithms
- Decompose first, split big problems into small functions
- An algorithm is a clear, finite, step-by-step plan that always gives the same result
- Pattern Recognition + Abstraction = write less, do more

🎉 Congratulations 🎉

You finished IT Basics!

Thank you!

Contact Detail

Prof. Dr. Florian Haselbeck

florian.haselbeck@hswt.de

Raum D1.230a

For any questions, please feel free to use the Moodle-Course Forum!