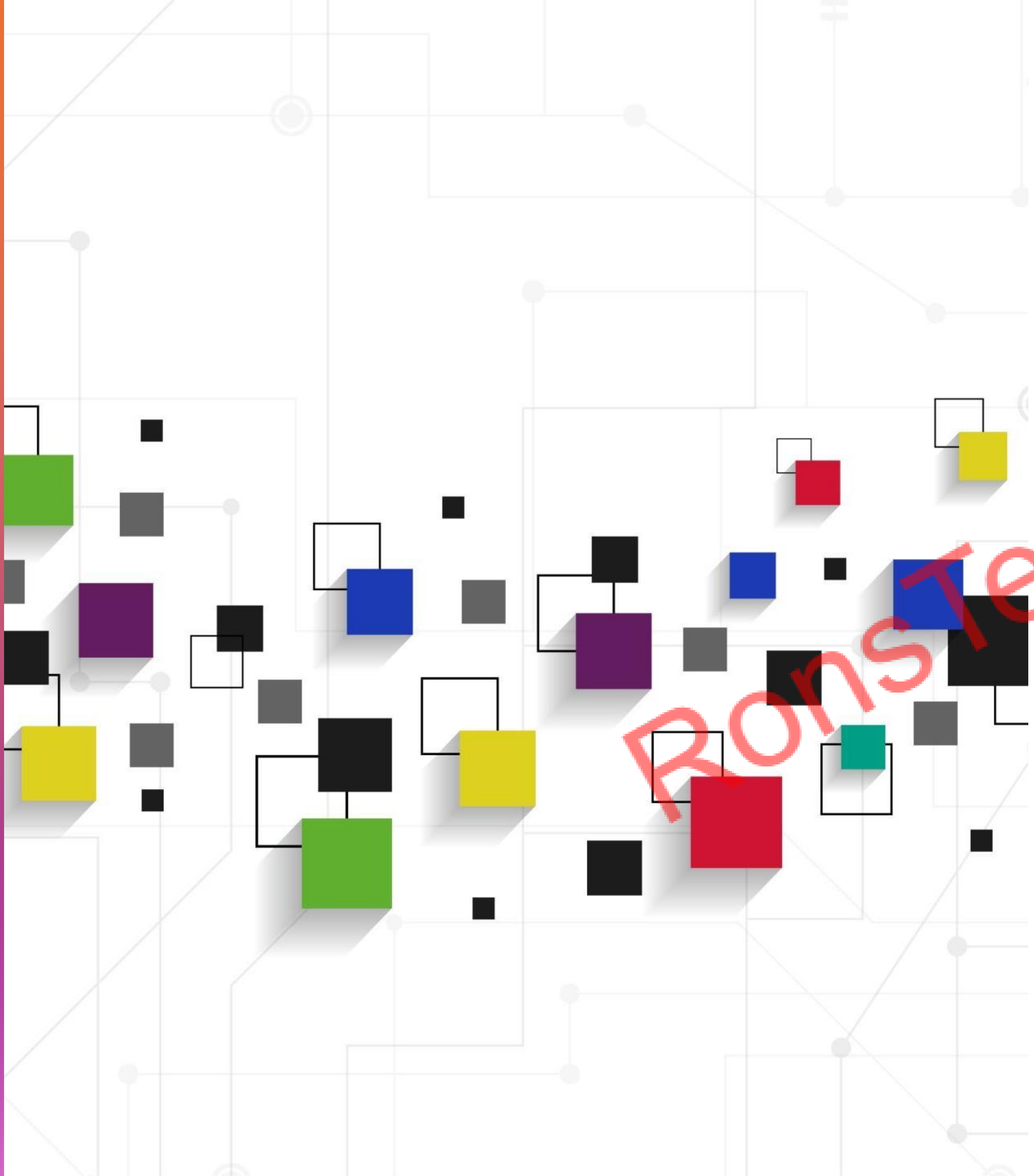
BTEC Level 3 Computing

Unit 1 - Principles of Computer Science

Computational Thinking

Unit Information

- The four main categories:
- Computational thinking.
- Standard methods and techniques used to develop algorithms.
- Programming paradigms.
- Types of programming and mark-up languages.



Computational Thinking

- This lesson will cover Computational thinking and its four sub categories.
- Decomposition.
- Pattern Recognition.
- Pattern Generalisation & Abstraction.
- Algorithm Design.

Computational Thinking

What is Computational Thinking?

A problem-solving approach used in computer science.

Helps break down complex problems into manageable parts.

Essential skill for programming and algorithm development.

Applicable to many fields beyond computing.

Core Components of Computational Thinking

- Decomposition.
- Pattern Recognition.
- Pattern Generalization & Abstraction.
- Algorithm Design.

Decomposition

Decomposition

- Identifying and describing problems and processes.
- Breaking down problems and processes into distinct steps.
- Describing problems and processes as a set of structured steps.
- Communicating the key features of problems and processes to others.

Decomposition Example: Jerk Chicken

- Work.
- Get paid.
- Buy chicken.
- Clean meat.
- Season meat.
- Cook meat.
- Eat food.
- Sleep.

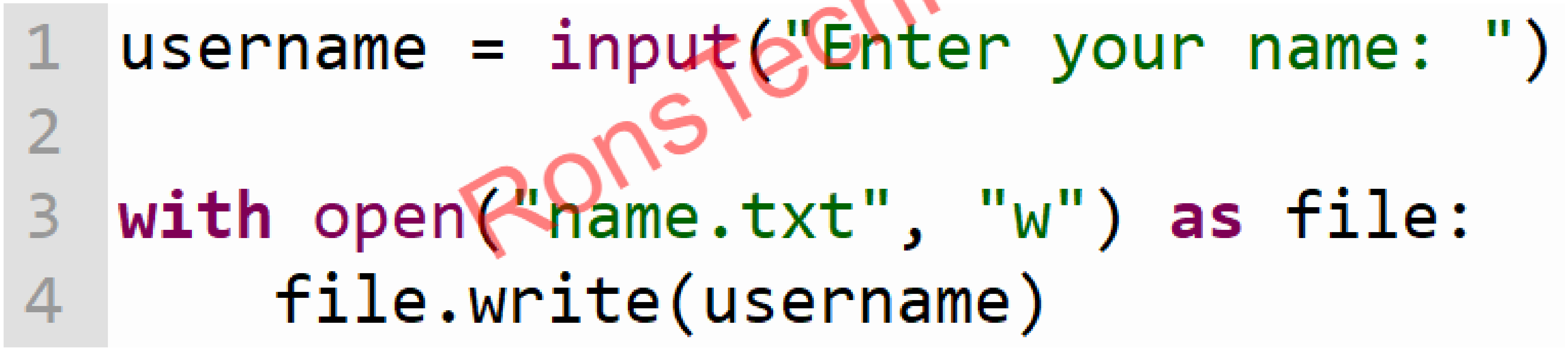
Decomposition Example: Get Name Save to File

- **Get Name:**
- Prompt the user to enter their name.
- Store the entered name in a variable.

- **Create File:**
- Create a new text file.
- Specify the file name (e.g., "name.txt").
- Open the file in write mode.

- **Write Name to File:**
- Write the stored name to the opened file.
- Close the file to save the changes.

Decomposition Example Code, "Python"



```python
username = input("Enter your name: ")

with open("name.txt", "w") as file:
    file.write(username)
```

Pattern Recognition

What is Pattern Recognition?

- Identifying common elements or features in problems or systems.
- Identifying and interpreting common differences between processes or problems.
- Identifying individual elements within problems.
- Describing patterns that have been identified.
- Making predictions based on identified patterns.

Why do Pattern Recognition?

- Helps us spot trends and make predictions.
- Enables us to generalize knowledge and apply it to new situations.
- Examples:
- Recognizing patterns in a data set.
- Identifying recurring themes in a story.

Pattern Recognition Example: Pause Menu

- The Fifa games are the same EVERy year.
- They do not need to recreate the game each time.
- They take what they have and make small tweaks.
- For example, the Fifa start menu, might have been the same code for years because it barely changes.

Pattern generalisation and abstraction

What is Abstraction?

1

Identifying the essential features of a problem and representing them in a simplified form.

2

Filtering out unnecessary details to focus on essential elements.

What is Abstraction?

- Representing parts of a problem or system in general terms by identifying:
- variables
- constants
- key processes
- repeated processes
- inputs
- outputs.

Why is abstraction important?

Simpler code: Easier to read, write, and understand.

Reusability: You can use the same code in different places.

Problem-solving: It helps break down big problems into smaller ones.

Teamwork: Different people can work on different parts of a project.

Abstraction example

Imagine you're driving a car.

You don't need to know exactly how the engine works or how the brakes stop the car.

You just need to know how to use the steering wheel, gas pedal, and brake pedal.

This is abstraction in action.

Abstraction Example

- Google Maps.
- We do not need to know all the streets we go past.
- We only need to know the directions to the location we are going to.

Compartmentalise

VARIABLES.

CONSTANTS.

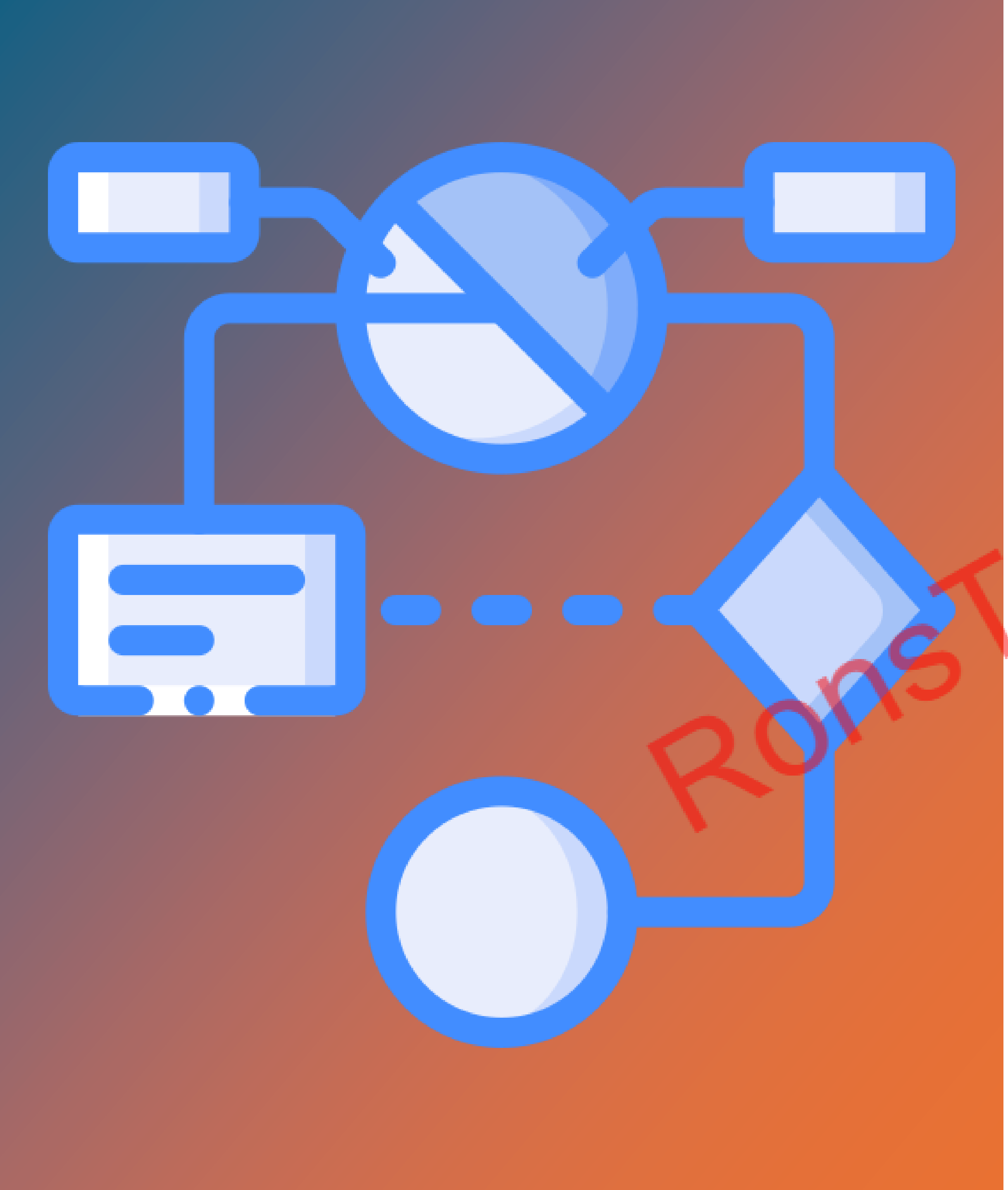
KEY PROCESSES.

REPEATED PROCESSES.

INPUTS.

OUTPUTS.

Algorithm Design



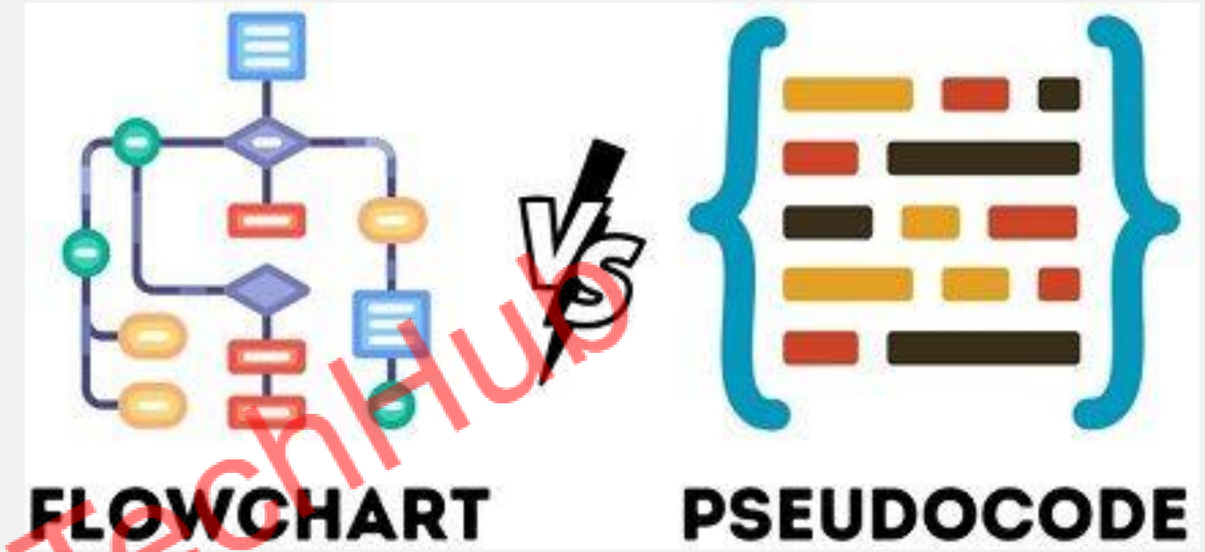
What is Algorithm Design?

- Describing a step-by-step strategy to solve a problem.

Why is algorithm design important?

- Speed: Good algorithms can make computers work faster.
- Efficiency: They can save resources like memory and energy.
- Correctness: They ensure the computer does the right thing every time.
- Problem-solving: They help us break down big problems into smaller ones.

Algorithm Design Examples



- Pseudocode.
- Flowcharts.

Next Time

Standard methods and techniques used to develop algorithms.

Page 21 on the specification.

Resources

- Specification: https://qualifications.pearson.com/content/dam/pdf/BTEC-Nationals/computing/2016/specification-and-sample-assessments/btec-nat-l3-ext-dip-in-computing-spec.pdf
- Decomposition YouTube Search - https://www.youtube.com/results?search_query=decomposition+computing
- Decomposition - https://www.youtube.com/watch?v=kXDtRKN7qmE
- Decomposition - https://www.youtube.com/watch?v=8HFqzzZxV9k
- BBC Decomposition - https://www.bbc.co.uk/bitesize/guides/zqqfyrd/revision/1
- BBC Decomposition - https://www.bbc.co.uk/bitesize/articles/z8ngr82#zg73r2p
- BBC Decomposition - https://www.bbc.co.uk/bitesize/articles/zxxdqfr#znn26g8
- BBC Decomposition Practice - https://www.bbc.co.uk/bitesize/guides/zqqfyrd/revision/2