ASIA PACIFIC UNIVERSITY OF TECHNOLOGY & INNOVATION

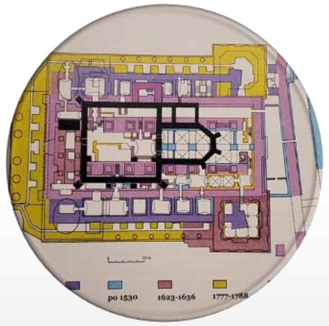
Introduction to Visual and Interactive Programming

CT803-4-0-OIVIP

Topic 2

Computational Thinking and Program Planning

Topic learning outcomes



Design

+ Problem solving using Computational Thinking

Define

+ Algorithm

Formulate

+ Steps to Solve Problems

Contents

Everyday Problems..

- **Scenario:** Your room is dirty, and cleaning it daily is a chore (you don’t want to do it!). You want to build a robot instead, that will do the cleaning
  - How will you tell it what to do?
  - Will your instructions be random?
  - Would you have a clear plan?
- Think like how a programmer would!

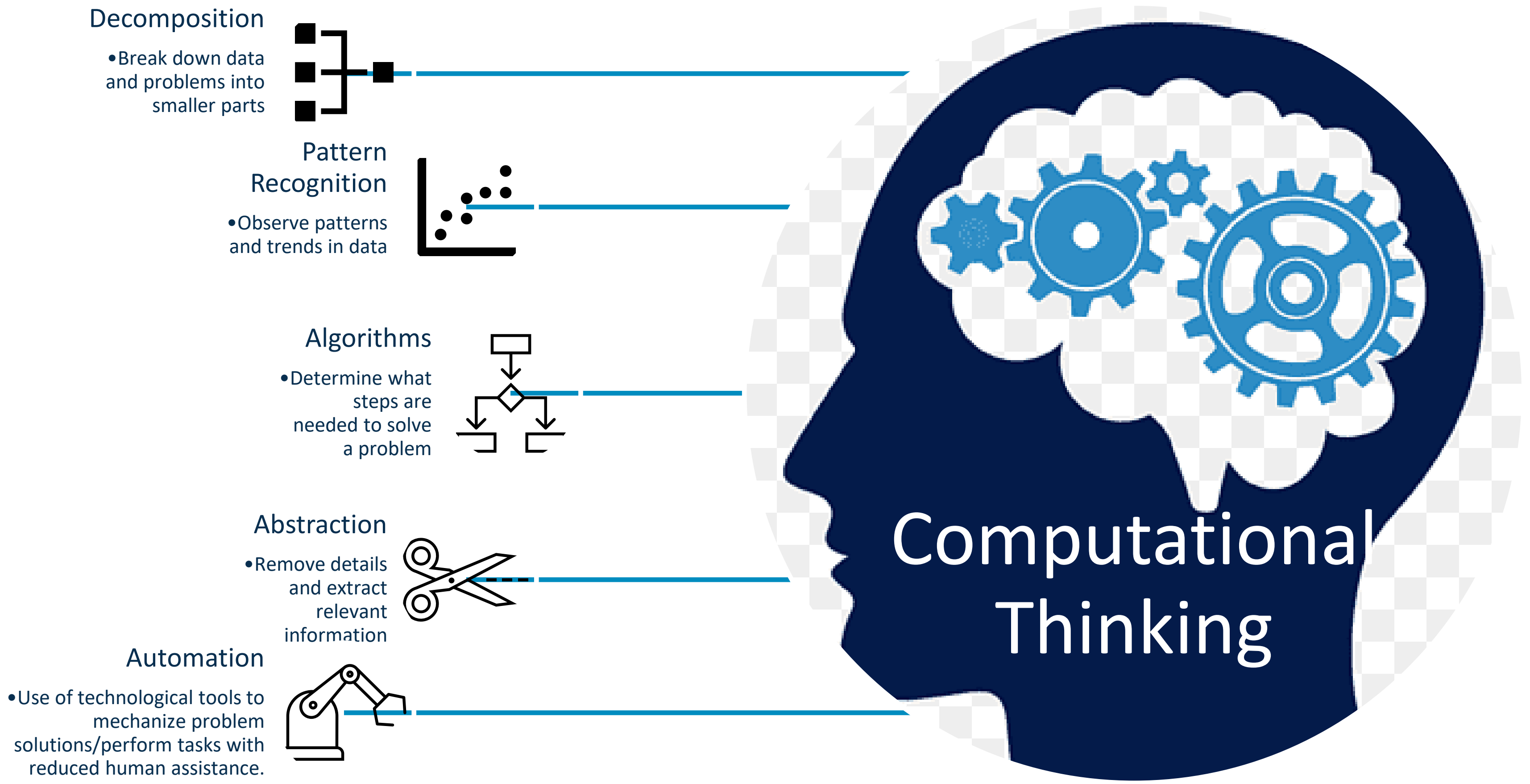
Computational Thinking

- Not necessarily about coding, computers, or robots to clean our room
- Way of approaching problems
  - Emphasizes logic, structure, and solving problems step-by-step
  - Toolbox of skills that can be applied to all sorts of challenges
- Principles of computational thinking:
  - Decomposition
  - Pattern recognition
  - Algorithms
  - Abstraction

  + Automation

Principles of Computational Thinking

- To think computationally through a difficult problem, task, or activity, you should be familiar with the following Five Principles of Computational Thinking


Decomposition
•Break down data and problems into smaller parts
Pattern Recognition
•Observe patterns and trends in data
Algorithms
•Determine what steps are needed to solve a problem
Abstraction
•Remove details and extract relevant information
Automation
•Use of technological tools to mechanize problem solutions/perform tasks with reduced human assistance.
Computational Thinking

Decomposition – break the problem into a smaller parts

This involves breaking down a complex task into smaller, and more manageable components.

This could be achieved through

Analyzing a problem: breaking a problem into parts

Synthesizing a problem: combining solutions to small subproblems to solve the large problem.

Parallelization: solving subproblems simultaneously

Sequential: solving subproblems in specific order.

Example: do a research on the different organs in order to understand how the human body digests food.

Pattern Recognition – Observe patterns and trends in data

This involves identifying and defining trends or patterns within a problem.

Patterns make it easier to see the relationships between different parts of a larger problem and to decide what actions can and must be taken to address it.

Example: classify animal based on their characteristics and articulate common characteristics for the groupings.

Algorithm Design - Determine what steps are needed to solve a problem

This involves the development of step-by-step rules or instructions for solving a problem or completing a task.

The instruction can be used again to answer similar problems.

The instructions must be precise and unambiguous.

Examples

Following a recipe or direction

Getting dressed.

Abstraction: Remove details and extract relevant information

This involves identification of particular similarities and differences between comparable problems to work towards a solution.

It aids in the development of problem-related models that can handle large amounts and ranges of data.

Examples :

Learning about physics using a ball and ramp	Experimenting and graphing results in an acceleration lab.	Developing laws and theorems by looking at similar formulas and equations.

Abstraction – Remove details and extract relevant information

Question: Is it possible to learn to drive a car without knowing how all the components work?

Example:

- Draw a cat.
- The image must be representative of all types of cats:
  - Ears, eyes and tail
  - Relevant characteristics
- Optional details:
  - Fur, nose, whiskers
- Unnecessary details
  - Sound, favourite food
  - Can be **filtered out**

Automation - Use of technological tools to mechanize problem solutions

This involves the use of technological tools to mechanize problem solutions/perform tasks with reduced human assistance.

The principle of automation can be applied to tasks that are:

Repetitive: This include tedious tasks like copy and paste, data entry, and switching between tabs

Fragile: This are tasks or activities that involve a high level of human error such as typos, forgotten checklists or coordinated changes steps; and

Timely: This include recurring tasks or activities such as reminders and automatic/instant responses.

Computational Thinking

Involves **THREE (3)** Key Dimensions

- Computational Concepts
- Computational Practices
- Computational Perspectives

Computational Concepts

Concepts	Description
Data	Storing, retrieving and updating
Operators	Support for mathematical and logical expressions
Events	One thing causing another thing to happen
Parallelism	Making things happen at the same time
Sequence	Identifying a series of steps for a task.
Conditionals	Making decisions based on conditions
Loops	Running the same sequence multiple times

Computational Practices

Practices	Description
Experimenting and Iterating	Developing a little bit, then trying it out, then developing more. https://player.vimeo.com/video/106046863
Testing and debugging	Making sure things work, finding and solving problems when they arise. https://player.vimeo.com/video/106046866
Reusing and remixing	Making something by building on existing projects or ideas. https://vimeo.com/106433597
Abstracting and modularizing	Exploring connection between the whole and the parts. https://vimeo.com/106438541

Computational Perspectives

Perspectives	Description
Expressing	Realizing that computation is a medium of creation, "I can create."
Connecting	Recognizing the power of creating with and for others, "I can do different things when I have access to others."
Questioning	Feeling empowered to ask questions about the world. "I can (use computation to) ask questions to make sense of (computational things in) the world."
Exploring	Understanding how one thing can be done in many ways "I know how I like to do {this}, how do other people like to do it? Why is that better for them?"

Why is it important?

- It moves students beyond technology literacy
- It creates problem solvers instead of software technicians
- It emphasizes creating knowledge rather than using information
- It presents endless possibilities for creatively solving problems
- It enhances the problem-solving techniques you already have

Algorithm

- Steps to solve a problem (a recipe)
- Key components:
    - Input
    - Process
    - Output
- Let’s look at a recipe of making a sandwich

Input: what is going to be put inside (ingredients)

- **Bread:** Two slices of your favorite bread
  - White, wheat, sourdough, multigrain - the choice is yours!
- **Spread:** The delicious foundation of flavor
  - Butter, peanut butter, hummus, mayo, mustard, the possibilities are endless!
- **Fillings:** This is where things get exciting!
  - Sliced cheese, ham, turkey, avocado, tomato, cucumber, sprouts - let your imagination (and fridge) run wild!

Process: what are the steps to produce the output

- **Prepare the canvas**: Place one bread slice on your plate. This is your flavor stage!
- **Spread the love**: Using your knife, evenly coat the bread with your chosen spread. Think of it as priming the flavor pump.
- **Layering the goodness**: Add your desired fillings. Go for a classic combo or invent your own masterpiece!
- **Top it off**: Place the second bread slice gently on top, pressing down lightly to seal in all that deliciousness.
- **The final cut (optional)**: If you're feeling fancy, slice your sandwich in half diagonally or into triangles. Presentation matters!

Output: what will the user experience at the end

- Of course, a **magnificent sandwich**!

But wait! Outputs may differ:

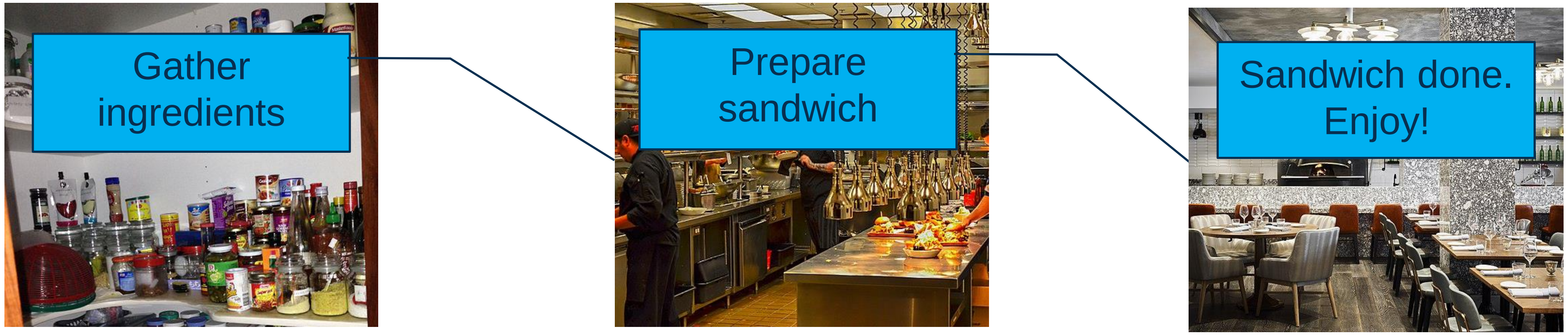
- Control structure
- Conditionals:

```
if spread = peanut butter
        if filling = none        → Output would be a peanut butter sandwich!
```

- Loop:

```
for i = 1 to 100
        spread mayo              → Output would be a really sloppy sandwich!
```

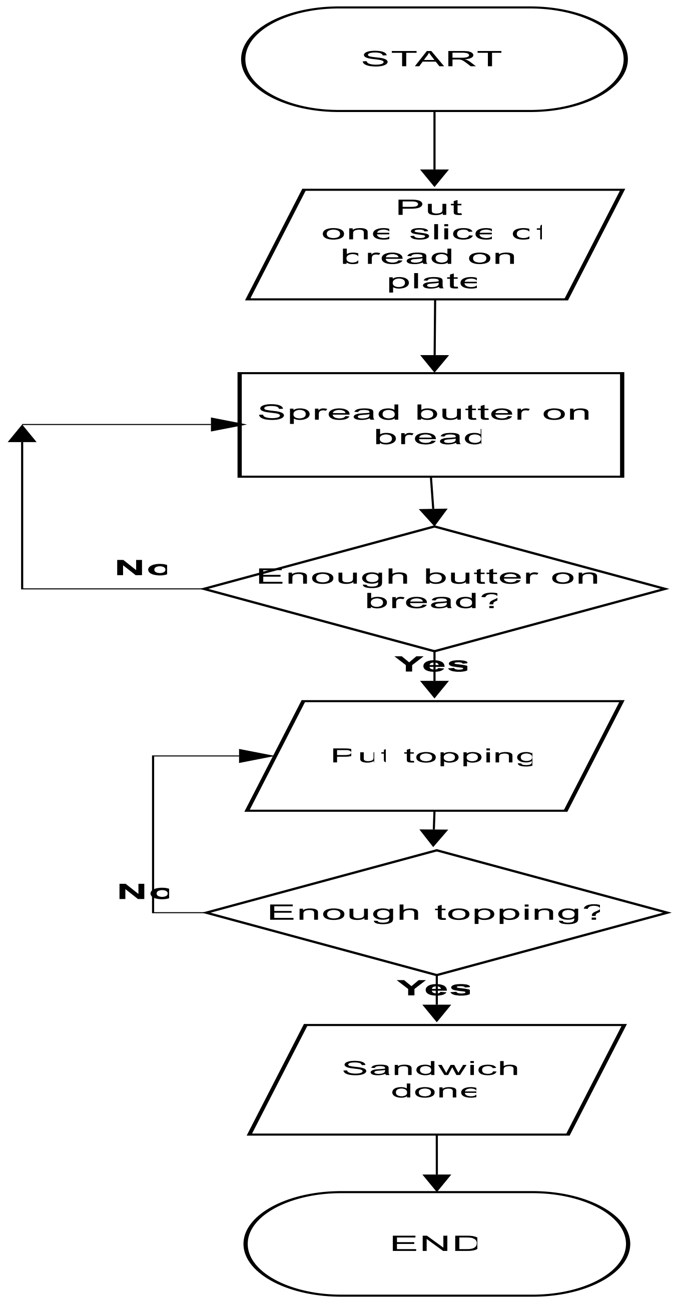
Wireframe: quick sketch of main steps (summary)



Pseudocode: detailed recipe

- Place one slice of bread on plate
- Spread butter evenly on slice of bread
- Place topping on buttered bread until satisfied
- Place one slice of bread on top of topping with
- Cut sandwich diagonally in half

Flowchart: steps in diagram form (like a map)



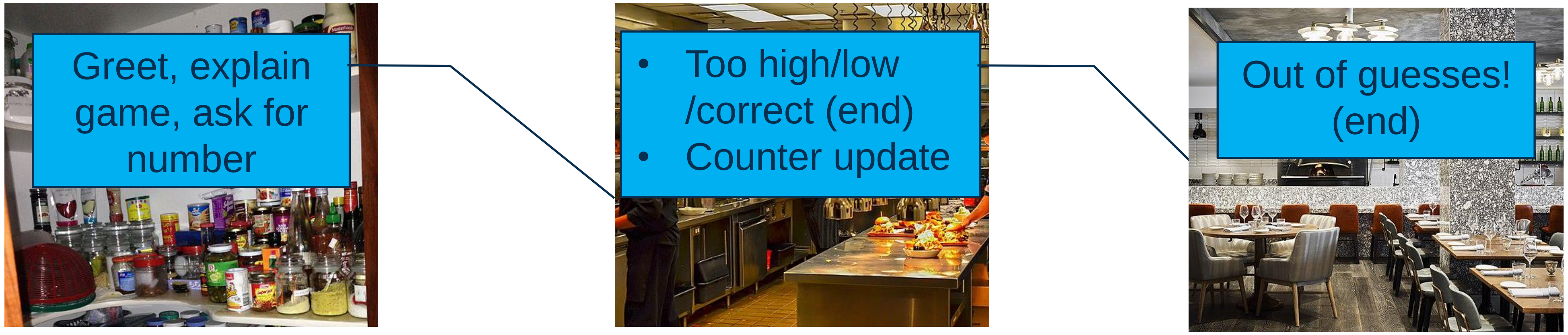
Storyboard: steps in pictures (like a comic book)

- **Panel 1:** Two excited kids gather ingredients.
- **Panel 2:** They carefully spread butter on one bread slice.
- **Panel 3:** They unleash their creativity with toppings like chopped onions, beef brisket, and pickles.
- **Panel 4:** They carefully assemble the sandwich masterpiece.
- **Panel 5:** Big smiles and thumbs up as they take the first bite!

Recap: Week 2 Exercise

- **Create a guessing game:** the sprite will greet the user and asks the user to guess a random number between 1 and 100, and the program gives feedback on whether the guess is too high, too low, or correct. The user can only guess __ number of times (decided by the programmer) so the program needs to keep track of how many guesses the user has made. The game ends when the user guesses the correct number or runs out of guesses.

Wireframe: quick sketch of main steps (summary)

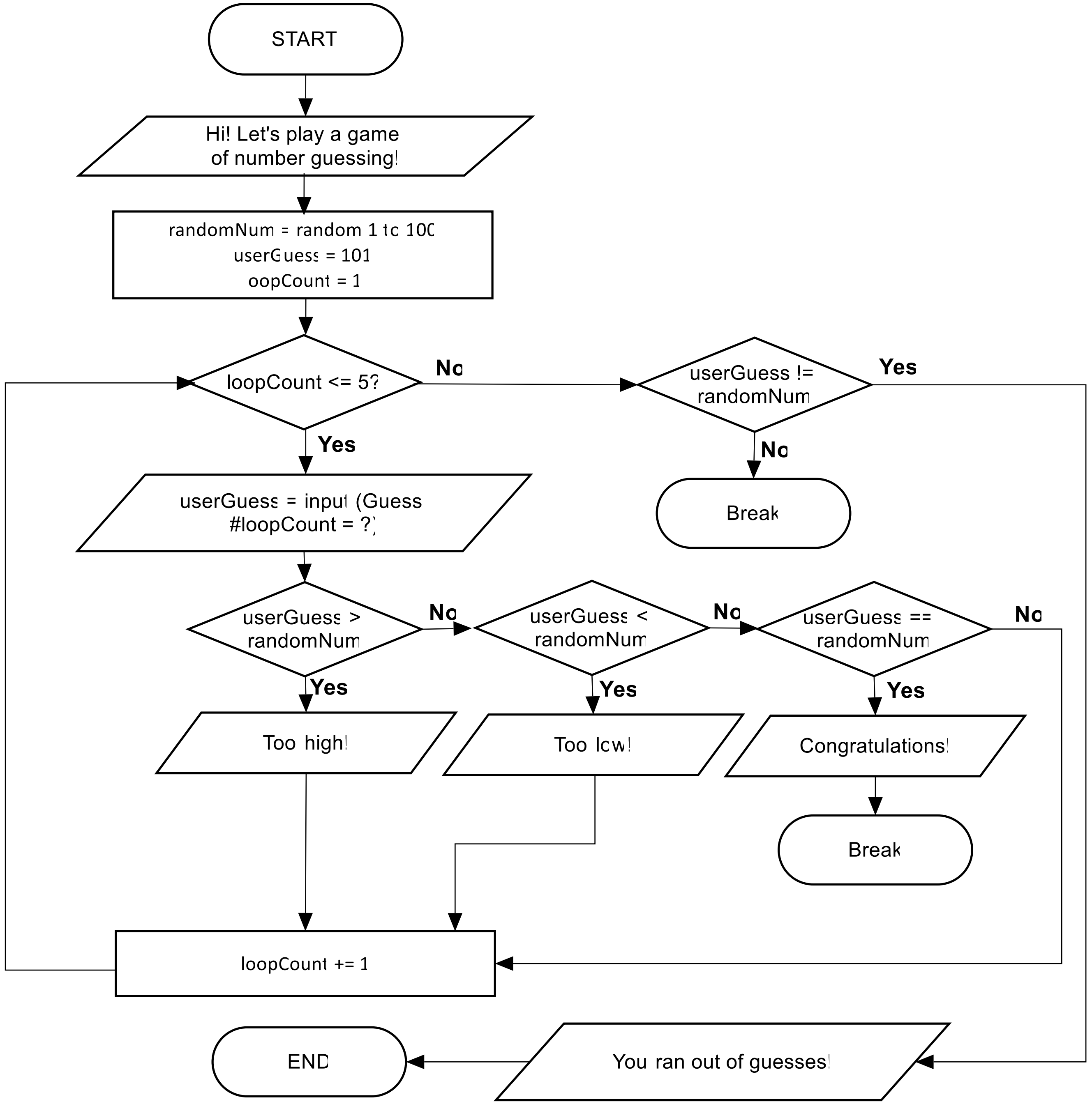

Greet, explain game, ask for number
• Too high/low /correct (end)
• Counter update
Out of guesses! (end)

Pseudocode: detailed recipe

When green flag clicked

```
Greet the user and explain the game
randomNum = random number between 1 and 100
userGuess = 101; loopCount = 1
while loopCount <= 5:          # user can only guess up to 5 times
    Ask for guess no. loopCount and store answer in userGuess
    if userGuess > randomNum:
        Display "Too high!" message
    if userGuess < randomNum :
        Display "Too low!" message
    if userGuess == randomNum :
        Display "Congratulations! You guessed the number!" and end the game
    increase loopCount by 1
if userGuess != randomNum: Display "Sorry, you ran out of guesses!"
End the program
```

Flowchart: steps in diagram form (like a map)



Storyboard: steps in pictures (like a comic book)

Scene	Description
1	A sprite pops up and says, "Welcome! Guess a number between 1 and 100!"
2	The user types a number into a text box.
3	The sprite analyzes the guess.
4a	The sprite frowns and says, "Too high! Go lower."
4b	The sprite looks surprised and says, "Too low! Aim higher."
4c	The sprite cheers and says, "You got it! Play again?"
5	The counter displays how many guesses the user has left.
6	The sprite shakes its head and says, "Out of chances! Play again?"

Exercise!

- Your pseudocode may (or may not) be similar to the instructor's (mine!). Create a flowchart to represent your pseudocode in a logical diagram. Then, create a storyboard for your number guessing program