Introduction to Algorithms

Building Confidence in Problem-Solving

Why Algorithms Matter

Solve Everyday Problems:

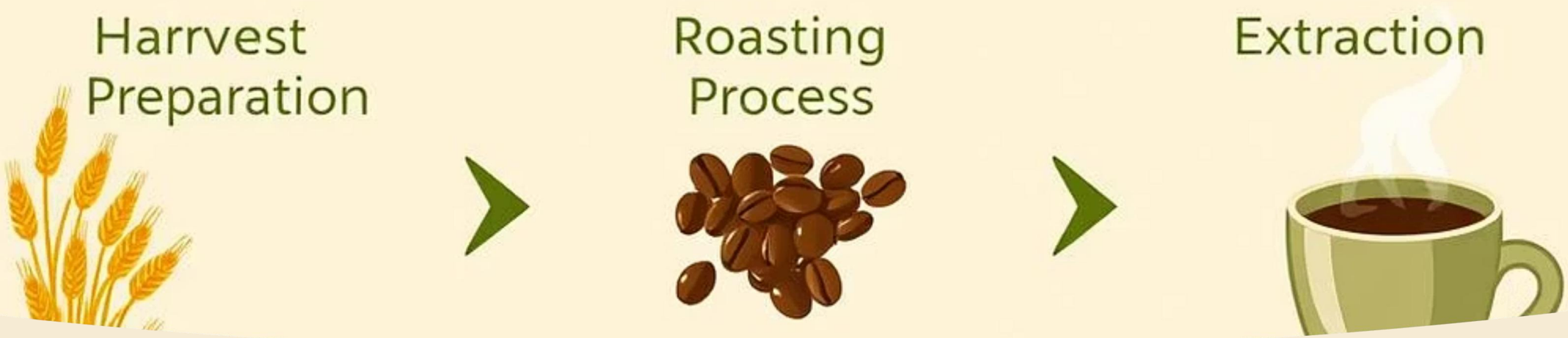
From following a recipe to planning your commute

Foundation for Programming:

Blueprint before you write code

Boosts Logical Thinking:

Breaks big tasks into manageable steps



What Is an Algorithm? (Basic)

A step-by-step recipe for solving a problem

Language-agnostic: works in any context

Finite:

ends in a result

Unambiguous:

each step is clear

Effective:

leads to solution

Algorithm vs Program (Intermediate)

Definition	Logical plan	Code implementation
Language	None (pseudocode/flowchart)	Python, Java, C++
Focus	What to do	How to do it
Reusability	Broadly reusable ideas	Requires porting to each language

Problem-Solving Steps (Basic ⇾ Advanced)

- Understand the problem
- Break it into smaller pieces
- Identify key operations
- Design flowchart or pseudocode
- Implement in code
- Test & Debug thoroughly

Advanced Tip: After step 2, ask “Can any sub-problem be reused?”

Visual Thinking with Flowcharts

Shapes & Meaning:

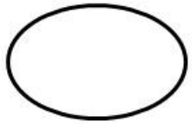
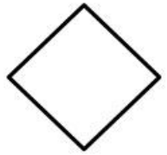
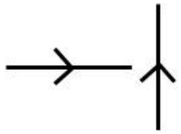
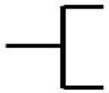
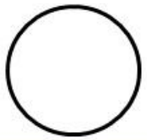
- Start / End: Ovals
- Process: Rectangles
- Decision: Diamonds
- Input/Output: Parallelograms
- Flow: Arrows

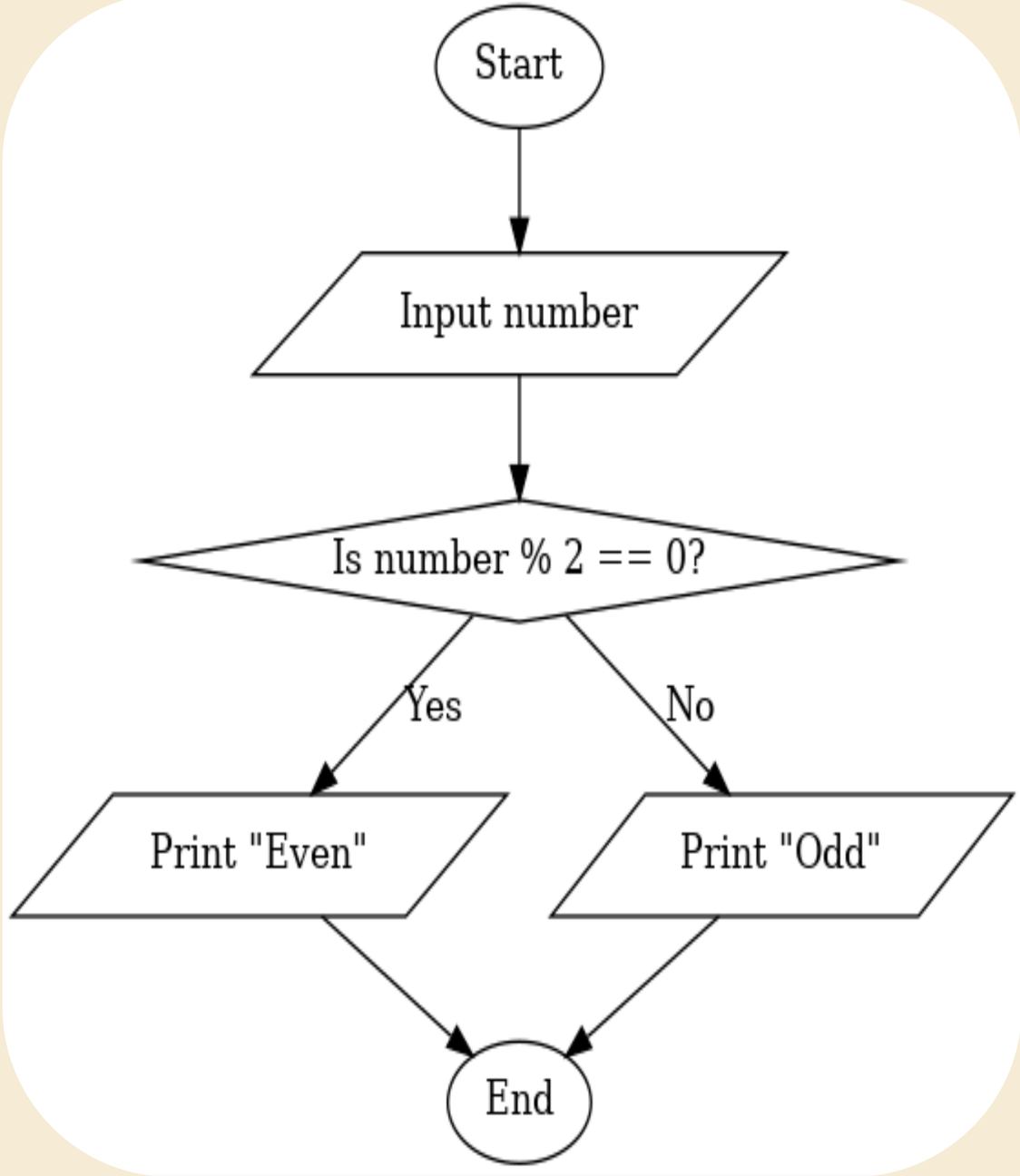
Demo: Flowchart for "Making Tea"

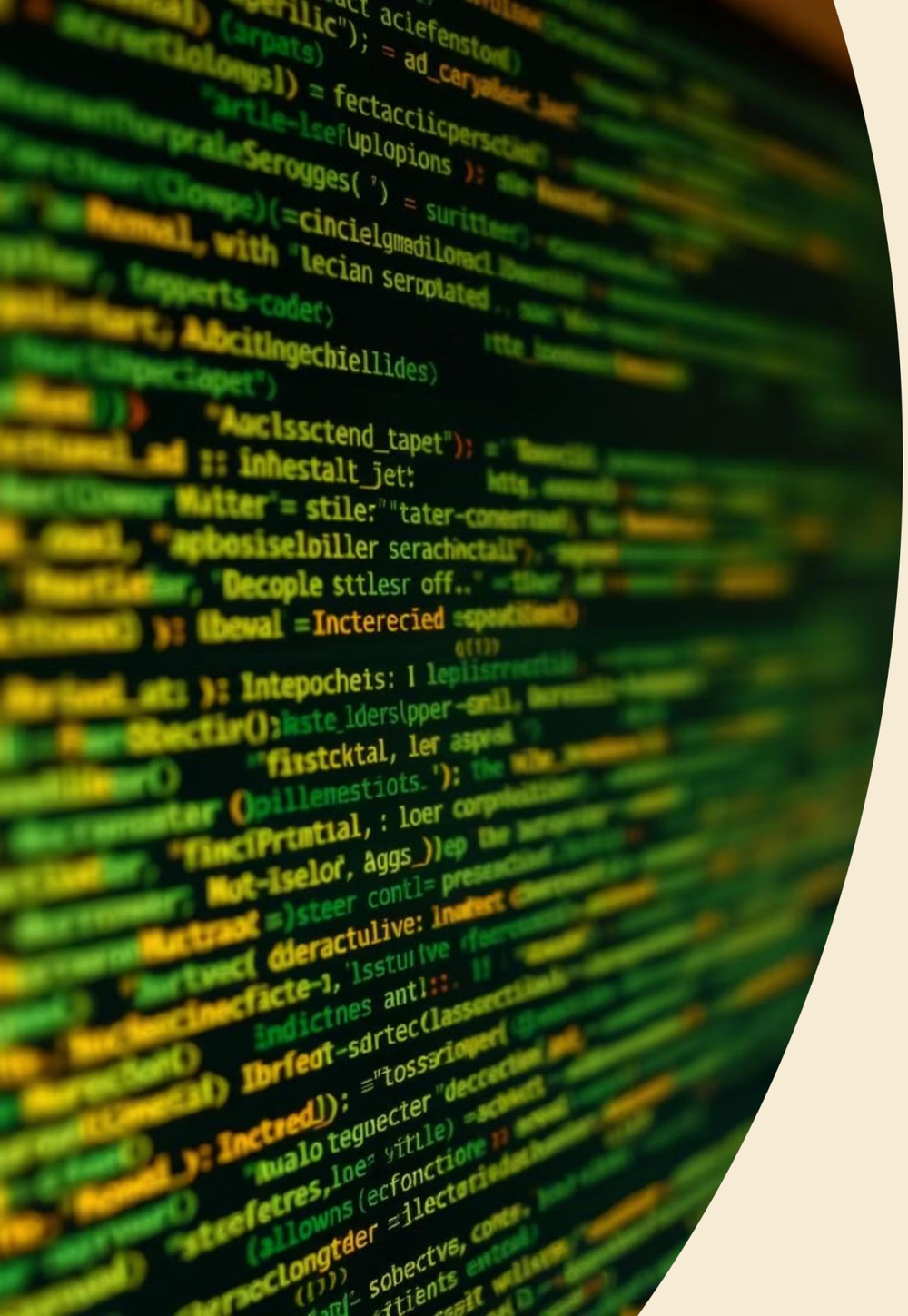
Start ↣ Boil water ↣ Have tea bag? ↣ Yes ↣ Steep ↣ Pour ↣ End

teachoo

Symbols used in a flowchart

Symbol	Purpose
	Start/Stop
	Input/Output
	Processing
	Decision box
	Flow line
	Annotation
	Connectors





Pseudocode Examples

```
PROCEDURE FindMax(A, B, C)

IF A ≥ B AND A ≥ C THEN

RETURN A

ELSE IF B ≥ C THEN

RETURN B

ELS

ERETURN C

END IF

END PROCEDURE
```

Why pseudocode?

- Focus on logic, not syntax
- Easy to convert into any language

Simple Algorithm Example: Linear Search

Goal: Find value X in list L

Pseudocode:

```
FOR each element E in L DO

IF E == X THEN

RETURN index of E

END IF

END FOR

RETURN "Not Found"
```

Complexity: O(n) time, O(1) space

Complexity Basics (Simplified)

Time Complexity:

- O(1): constant (access item)
- O(n): linear (scan)
- $O(n^2)$: quadratic (nested loops)

Everyday Analogy: Sorting 5 books vs. 50 books

Space Complexity:

- Extra memory used beyond input

Slide 10: Advanced Complexity Insight (Optional)

- **Big-O (O):** Worst-case upper bound
- **Big-Ō (Theta):** Average-case tight bound
- **Big-Ω (Omega):** Best-case lower bound

Why it matters: Choose the right algorithm for better performance

Slide 11: Debugging Approach

- **Step-by-step:** Isolate issues
- **Print/log intermediate values**
- **Compare expected vs. actual results**
- **Techniques:**
  - Rubber-duck debugging
  - Peer walkthroughs