What is Pseudocode?

Pseudocode is a description of an algorithm using everyday wording, but molded to appear similar to a simplified programming language. It bridges the gap between natural language and actual code, allowing programmers to express logic without worrying about specific syntax rules.

Think of pseudocode as a rough draft for your program. Just like we might outline an essay before writing it, pseudocode helps us outline our program's logic before implementing it in a specific programming language.

Key characteristics of pseudocode

- Uses plain English mixed with programming-like structure
- Focuses on logic rather than syntax
- Language-independent (can be converted to any programming language)
- Easy to read and understand by both programmers and non-programmers
- Uses common programming constructs like loops, conditionals, and variables

Example of pseudocode

Here's a simple example of pseudocode for finding the largest number in a list:

```
BEGIN

SET largest_number to 0

FOR each number in the list:

IF number is greater than largest_number:

SET largest_number to number

DISPLAY largest_number

END
```

How to write pseudocode

Writing effective pseudocode follows several best practices:

- **Start with clear structure**: Begin with **BEGIN** and end with **END** to define algorithm boundaries. Use indentation to show operation hierarchy, similar to actual programming.

- **Use standard keywords**: Include common terms like **SET** or **ASSIGN** for variables, **IF/THEN/ELSE** for conditions, **WHILE** or **FOR** for loops, **INPUT** for user data, **OUTPUT** or **DISPLAY** for results, and **CALL** for functions.
- **Keep it simple**: Write as if explaining to someone who understands basic programming but not specific syntax. Avoid implementation details.
- **Focus on logic flow**: Concentrate on operation sequences and decision points rather than specific function names or syntax. Capture the algorithm's essence, not implementation specifics.

When and why to use pseudocode

Pseudocode is invaluable during the planning phase, team communication, documentation, and complex problem-solving. It helps you think through problems systematically, create implementation roadmaps, and serve as a universal language for teams regardless of their preferred programming language.

What is a flowchart?

A **flowchart** is a visual representation of an algorithm or process that uses standardized symbols connected by arrows to show the flow of execution. Using a pseudocode and flowchart approach together provides both textual and visual ways to understand and communicate how a program works.

Flowcharts are particularly powerful because they transcend language barriers and technical expertise levels. A well-designed flowchart can be understood by programmers, managers, and stakeholders alike.

Common flowchart symbols

Flowcharts have some standard symbols that allow them to be read and understood by a wider group of people. These are some of the most commonly-used symbols:

Terminal

The terminal is an oval that indicates the beginning and end of a program. It usually contains the words Start or End.

Flowline

The flowline is a line from one symbol pointing towards another to show the process's order of operation. This displays the flow of execution in a program.

Input/Output

Input/output is represented by a rhomboid and indicates the input or output of data. This is similar to setting a value to a variable.

Process

A process, represented by a rectangle, is an operation that manipulates data. Think of this as changing the value of a number-based variable using an operator such as +.

Decision

Decisions are represented by a rhombus and show a conditional operation that will determine which path a program should take. This is similar to conditional statements which control the flow of execution in a program.

How to use flowcharts in programming

- **Start with the big picture**: Identify main algorithm components - what inputs are needed, what processes must occur, what decisions need to be made, and what outputs should be produced.

- **Map the flow**: Connect components using flowlines to show program movement from step to step. Pay special attention to decision points where the flow can branch in different directions.
- **Handle loops and iterations**: Use flowlines that loop back to earlier steps for repetitive operations. This visually represents while loops, for loops, and other iterative constructs.
- **Test the logic**: Walk through your flowchart with sample data to ensure logic flow makes sense and handles all possible scenarios, including edge cases.

When and why to use flowcharts in programming

Flowcharts excel for visual problem-solving, team collaboration, complex algorithms with multiple decision points, and documentation. They're particularly useful when you need to visualize intricate logic or train new team members on existing processes.

Pseudocode and flowchart comparison

Understanding when to use each tool helps maximize their effectiveness:

Aspect	Pseudocode	Flowchart
Format	Text-based	Visual diagrams
Best for	Detailed logic	Complex decision trees
Learning curve	Lower	Moderate
Team communication	Technical audiences	Mixed audiences
Space efficiency	Compact	Requires more space

Practical example of pseudocode and flowchart

Let's demonstrate how pseudocode and flowchart work together by solving a real-world problem: creating a password validator that checks if a password is at least 8 characters long and contains at least one number.

The problem

Passwords are everywhere, and we create them all the time to access various services. However, it can be helpful to guide users to create stronger passwords by imposing some restrictions on what passwords are considered valid.

For our example, we want passwords that:

- Are at least 8 characters long
- Contain at least one number

Valid passwords would include:

- supers3cure
- james1510
- meandmy2dogs

Invalid passwords would include:

- password (no number)
- dog (too short, no number)
- hunter2 (too short)

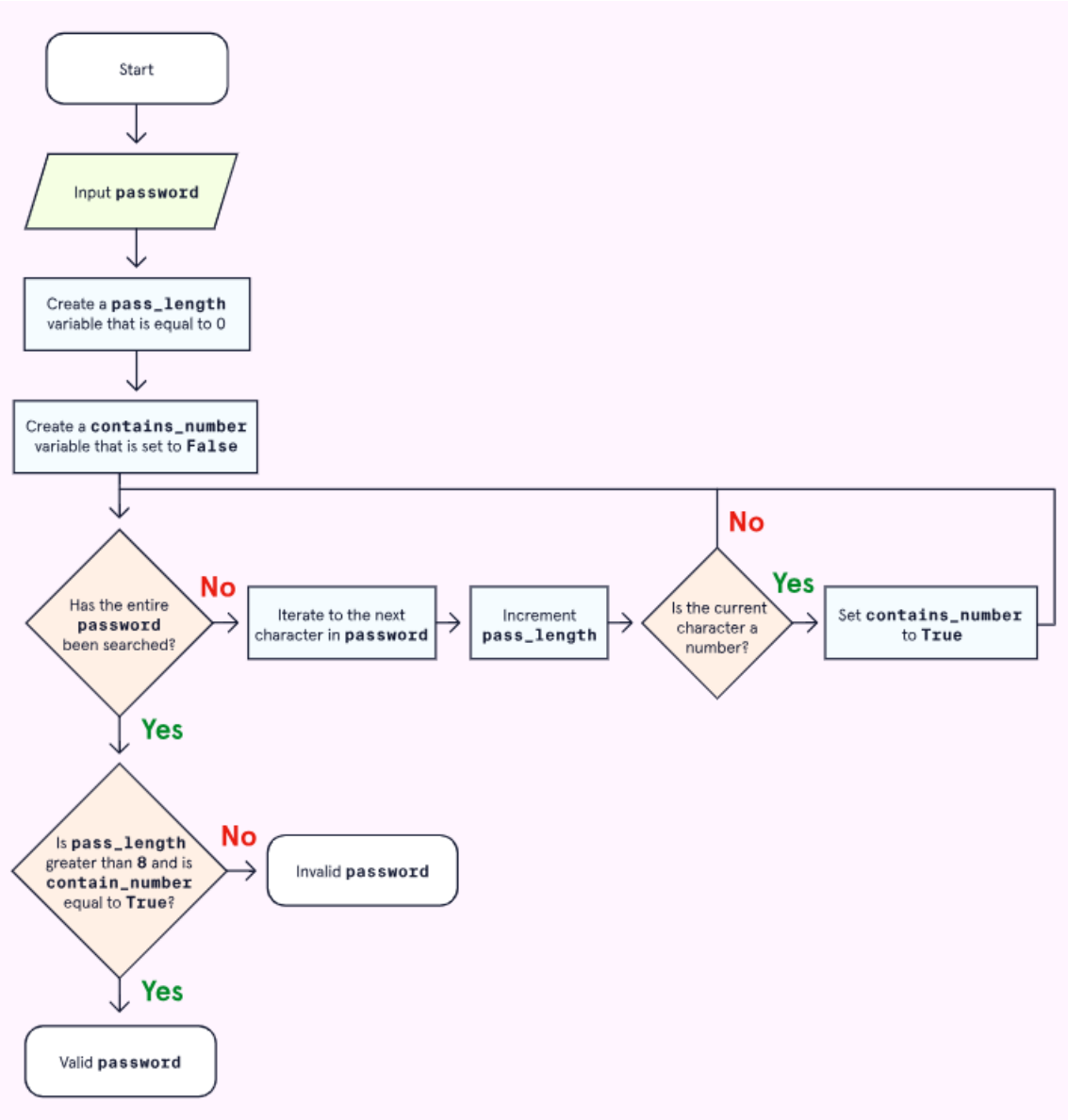
The solution: step-by-step approach

Let's break down our password validation algorithm into clear steps that we can later convert to pseudocode and flowcharts.

```
BEGIN Password Validator

INPUT password

SET pass_length to 0

SET contains_number to False

WHILE pass_length is less than length of password:

SET current_character to password at position pass_length

INCREMENT pass_length by 1

IF current_character is a digit:

SET contains_number to True

IF pass_length is greater than 8 AND contains_number is True:

OUTPUT "Valid Password!"

ELSE:

OUTPUT "Invalid Password"

END
```

Flowchart implementation

Here's how our password validator looks as a flowchart:



From pseudocode to real code

With our pseudocode complete, we can implement it in any programming language. Here's the Python version:

```
# Password Validator Implementation

password = "c0decademy"

# Initialize tracking variables

pass_length = 0

contains_number = False

# Check each character in the password

while pass_length < len(password):

current_character = password[pass_length]

pass_length += 1

# Check if current character is a digit

if current_character.isdigit():

contains_number = True

# Validate password based on our criteria

if pass_length > 8 and contains_number:

print("Valid Password!")

else:

print("Invalid Password")
```

p