Designing a Solution

As mentioned previously, there are three ways to design solutions for computational problems. They are:

- Algorithm
- Flowchart
- Pseudocode

For explaining each of these designing methods, let's take an example problem:

> Find the sum of first `n` natural numbers

Algorithm

Definition: A finite sequence of well defined instructions to solve a problem or perform a task. The charecteristics of an algorithm are:

- **Input**: It defines theinput of the problem clearly.
- **Output**: It defines the output of the problem.
- **Finiteness**: The solution is finite and ends after a certain instruction.
- **Definite**: Each instruction is clear, and precise.
- **Effective**: Steps must be easy to understand and execute by the computer.
- **Simplicity**: Steps are simple, easy to understand, and implement.

> Now taking the example:
>
> 1. Get input `n` from user
> 2. Initiate `sum` to 0
> 3. For `i` from 1 to `n`
>   a. add `i` to `sum`
> 4. Output `sum`

Pseudocode

Definition: A high-level, informal description of an algorithm that uses both natural language and programming-like constructs to explain the solution.

It focusses on logic more than syntax. Pseudocodes are useful because:

- It bridges the gap between the problem statement and the programming solution.
- Helps in designing algorithms before programming.

To write a good pseudocode, there should be:

- **Clarity**:
  - Use simple and natural language
  - Keep instructions simple and short
- **Structure**:
  - Make sure to indent code blocks (statements under `IF`, `FOR` etc.)
- **Consistency**:
  - Language, words and structure used must be uniform throughout the pseudocode
- **Abstraction**:
  - Focus on logic, not syntax
  - Any unnessacery instructions must be avoided
- **Numbering**:
  - For short pseodocodes, numbering steps helps in discussions and referencing.

Now taking this example:

1. ALGORITHM SUM (n)
2. INPUT n
3. sum ← 0
4. FOR i ← 1 TO n DO
5. sum ← sum + i
6. OUTPUT sum
7. END

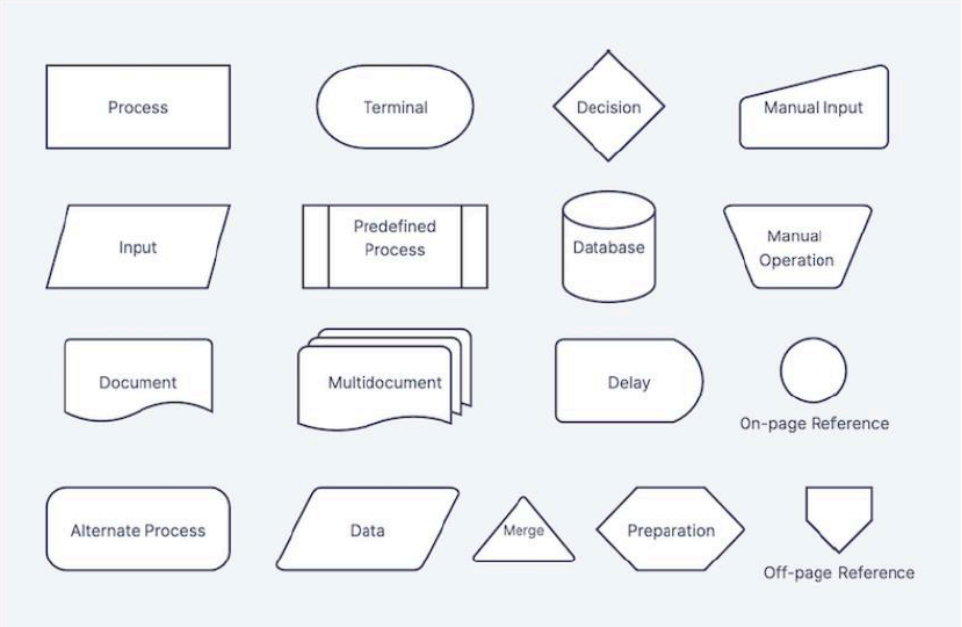
Flowchart

Definition: Visual representation of an algorithm or process using symbols, shapes and arrows showing the flow of control.

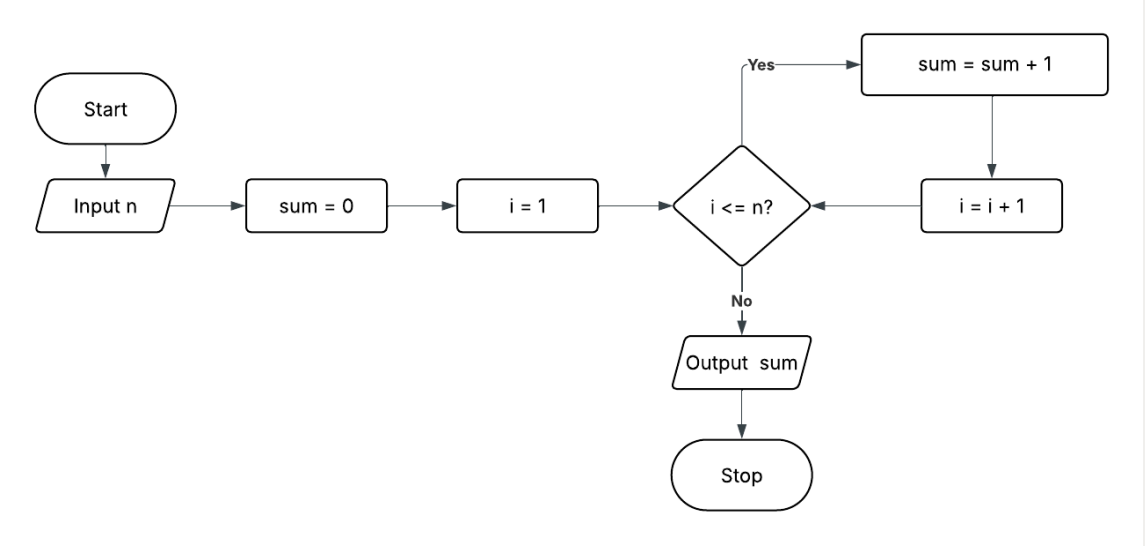
It can be useful because:

- It helps in communication between programmers and non-programmers
- Visualizes and algorithm, making it easier to understand the flow
- Verifying becomes easier

To make flowcharts, it is important to keep in mind that usage of shapes and symbols must be consistend and not random. The image shows the list of shapes and their purpuse.



Now taking the example:



Summary

- Designing solutions commonly uses three representations: **Algorithm**, **Pseudocode**, and **Flowchart**.
- An **Algorithm** is a finite, definite, effective, and simple sequence of steps that transforms clearly defined inputs into outputs.

- **Pseudocode** expresses the algorithm's logic in structured, language-agnostic steps. Good pseudocode is clear, consistently formatted, indented for structure, and focused on logic over syntax.
- A **Flowchart** visualizes control flow with standard symbols and arrows, improving communication and verification of the process.
- Example used: compute the sum of the first `n` natural numbers by initializing `sum ← 0`, iterating `i = 1..n`, and accumulating into `sum`, then outputting the result.
- Takeaways: start from precise problem definition and inputs, choose a clear representation, maintain consistency, and verify logic before coding.