Problem Solving - Theory

Algorithm 🧮

The word "algorithm" has its roots in the name of the Persian mathematician Muhammad ibn Musa al-Khwarizmi, a scholar in the House of Wisdom in Baghdad, who lived from approximately 780 to 850 CE. The term is a Latinization of his name, specifically from the title of his seminal work "Algoritmi de numero Indorum," which can be translated as "Al-Khwarizmi on the numbers of the Indians."

An algorithm is a finite set of well-defined computer-implementable instructions to perform a computation, accomplish a specific task, or solve a specific problem.

That's too formal, right? Don't worry. Let's break it down:
- Finite: An algorithm must have clear start and end points. It consists of a finite number of steps.
- Well-Defined: Each step of an algorithm must be precisely defined; the actions to be performed must be rigorously and unambiguously specified.
- Computer-Implementable: The steps must be sufficiently detailed and clear so that they can be executed on a computer, even though the concept of an algorithm is not limited to computation by electronic computers.

Algorithms have input and output:
- Input refers to the data that are provided to be processed. In some cases, an algorithm might not require any external input to start its operation; it might use a predefined set of data or generate its own data internally.
- Output is the result produced by an algorithm after processing. It is often, but not always, derived from and influenced by the input. The output can manifest in various forms, such as a value, a decision, a set of instructions, or even a change in state or action performed by the algorithm.

Pseudocode 🤖

Pseudocode is a natural language-level description of algorithms using simple programming constructs. Developers mostly build the outline of algorithms during the initial stages of development in human-readable formats before translating them into code using programming languages. It enables developers to focus on algorithms without dealing with programming language constraints and syntax. It is informal and flexible. Therefore, it can be adapted to suit the needs of the developer.

The Process of Problem-Solving 🧪

Solving problems requires a very systematic approach:

1. Understand
    1. Identify the given and requested.
    2. Identify the constraints and requirements.
    3. Build examples.
2. Analyze
    1. Break down the problem into smaller pieces (if possible and efficient).
    2. Identify the variables and relationships
    3. Identify dependencies (if any).
3. Solve
    1. Build an algorithm.
    2. Express the algorithm in pseudocode.
    3. Convert the pseudocode into code.
    4. Test the code against examples and language-specific cases.

Approach problems with a clear understanding of what is given and what needs to be achieved, as well as the requirements and constraints that frame the problem. This understanding is crucial to define the problem's scope accurately. It's important to resist the urge to broaden the scope unnecessarily, which is a common pitfall for many new developers who aim to enhance the problem's complexity. While ambition is commendable, practical limitations such as time and resources must be considered to stay aligned with the original problem definition. As Voltaire said, "The best is the enemy of the good." Once the scope is established, construct examples to deepen your comprehension of the problem.

With a solid grasp of the problem, begin the analysis phase. Where possible and practical, decompose the problem into smaller, more manageable components that can