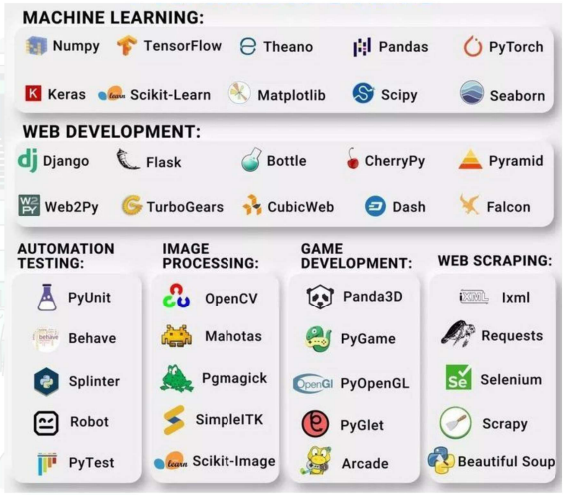
Introduction to Problem Solving with Python

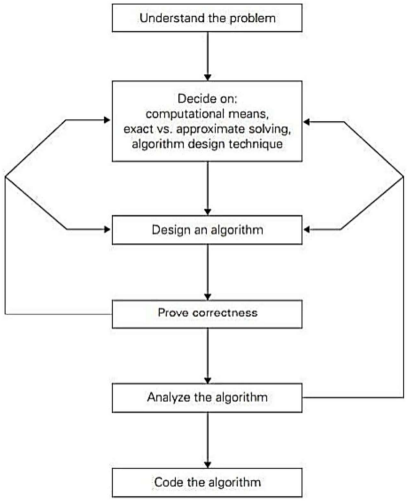
Why Python?

- Top programming language according to IEEE Spectrum
- **Very easy** to learn. Does not need any pre-requisites. Linear learning curve.
- **Versatile** language used in multiple domains: data science, machine learning, web development.
- **Rich libraries and frameworks** to do various tasks easily.
- Constantly **evolving**.
- All the **top tier companies** use it.

Python Libraries



Problem Solving



Understanding the Problem

- What are the expected inputs?
- What are the expected outputs?
- What is the format of the input and output?
- Are there any inputs that need specific handling?

Decisions to be made

- Computational Means: What device ?
- How accurate should the solution be?
  - exact algorithm
  - approximate algorithm
- What data structure?

Algorithm Design Technique

- *A general approach to solving problems algorithmically that is applicable to a variety of problems from different areas of computing.*
- How do we create an organized way to solve problems?
- Methodology – Pseudocode, Flowchart, etc.

Algorithm Correctness

- Making sure that the algorithm yields a required result
  - for every legitimate input
  - in a finite amount of time.
- Use mathematical induction because an algorithm's iterations provide a natural sequence of steps needed for such proofs.
  - Math and Algorithms go hand in hand.

Analyze the Algorithm

- Efficiency
  - Time – How fast does the algorithm execute ??
  - Space – How much memory does it take ??
- Simplicity
  - Simpler algorithms are easier to use and understand
  - Few bugs

Coding the Algorithm

- Ultimately algorithms must be implemented as computer programs.
- Choice of language depends on the user, or domain, or project involved.

Algorithms

Introduction

- What is an algorithm ?
  - Step-by-step instructions to solve a problem, or complete a task.
- Can be expressed in
  - natural language
  - psuedocode
  - programming languages

Components

- Inputs
- Outputs
- Variables
- Statements and Control Structures
  - sequential statements
  - selection statements
  - iterative loops
- Termination Condition
  - when to stop and exit?

Properties

- Well defined and unambiguous:
  - Every step of the algorithm is precisely defined
  - There is no room for interpretation or ambiguity
  - There is only one possible way to execute each step of the algorithm
  - The algorithm's behavior is completely determined by its inputs
- Effectively computable:
  - The algorithm can be executed using a finite number of steps, in finite amount of time.

Expected Qualities

- Time
  - Must end in the stipulated time.
- Memory
  - Must consume stipulated amount of memory
- Accuracy
  - Must fall within certain acceptable error range

Problem Analysis Chart

- A structured representation of a problem.
- Helps you organize your thoughts
- Defines:
  - Given Data (Input)
  - Required Results (Output)
  - Processing (Steps/Logic)
  - Solution Alternatives (Optional)
  - Conditions/Constraints (Optional)

Example 1

- "Write an algorithm to compute the average of a set of numbers."
- Given Data: set of numbers
- Required Results: average of the numbers
- Processing: sum all the numbers; divide by the count of numbers

Example 2

- “Searching for a number in a list”

Given Data	**Required Results**
A list of numbers, and a target number	Whether the target is in the list
Processing Required	**Solution Alternatives**
Scan the list and compare each element with the target	- Linear search - Binary search

Problem Solving Approaches

- **Top-down**
  - First focus on the big picture. Gradually refine the details.
  - Break down the problem from a high-level overview into smaller sub-problems.
- **Bottom-up**
  - Start from the simplest subproblems and build up the solution by combining answers of smaller parts.
- **Divide and Conquer**
  - Split the problem into smaller independent subproblems, solve them recursively, and then combine the results.
- **Backtracking**
  - Explore all possible solutions by trying out options and undoing (backtracking) if a choice leads to a dead end.