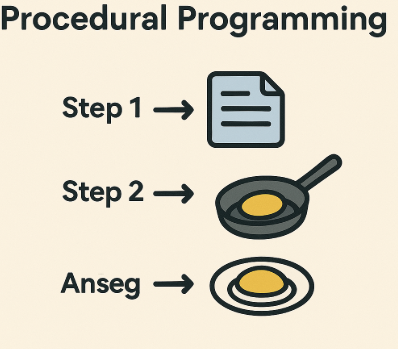
Programming in general

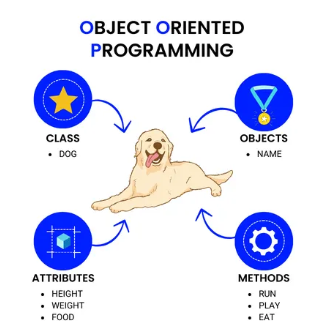
Programming Paradigms:

Is difference ways of thinking and writing a computer program and there is many ways to do that but the most common wans is the following:

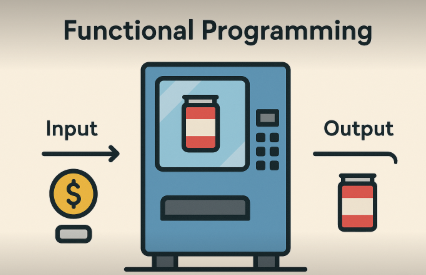
1. Procedural program : it is a way that uses a basic style. Moreover the program is listed step by step. For example, think like making a cheese sandwich. First you get the bread then you cut it to Two pieces then you but the chess then you close it .



1. Objects -Oriented Programing (OOP) : this style you creat objects like a cat so you have 2 common thighs one is the data like : color , type , color of eyes then we have the actions like : jump , run , say thing . you can make many cats but with difference thing

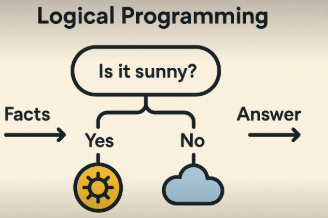


1. Functions programing : it is like a simple toy care press forwarded it goes straight pres back it goes back give input get output no changes in the car



1. Logic programing : only give facts and rules and the computer should find the answer like

All dogs can run Alxes is a dog. Can Alxes run?? Answer yes .



| Mmthod | Cons |
| --- | --- |
| Procedural program | Each step is done in order.  Easy to follow and understand.  Good for small tasks. |
| Objects -Oriented Programing | Good for big systems  Makes code organized  Helps reuse code |
| Functions programing | Clean and safe  Easy to test  Good for data and math |
| Logic Programing | You don’t give steps — just logic  Good for smart systems (like AI) |

Algorithm programming components :

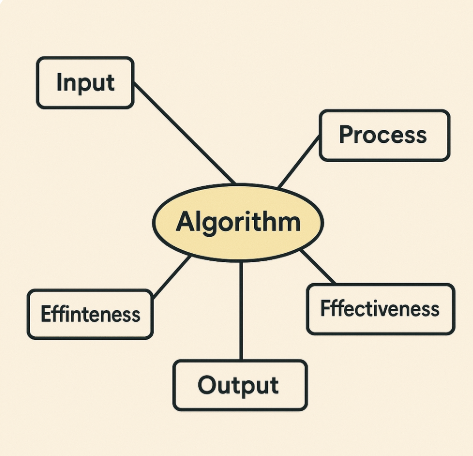
**Algorithm** : is a set of steps to solve a problem or complete a task like making a tea recipe

An **Algorithm** created of 3 main things :

1. **Input** : the data that is given for the algorithm .
2. **Process** : the steps or the actions of the algorithm .
3. **Output** : is the result of the algorithm .

**Algorithm** has also some important concepts like :

1. **Definiteness** : Each step must be clear and exact.
2. **Finiteness** : The algorithm must end after a number of steps. ❌ Bad: Keeps going forever
3. **Effectiveness**: Each step must be simple and can be done by a person or computer.



## 

## **Types of Programming Languages**

### **1. Low-Level Languages**

These talk **directly to the machine**. Fast, but hard to understand.

#### **a. Machine Language (Binary)**

* 0s and 1s (example: 101010)
* Directly runs on CPU
* Very fast, but hard for humans

#### **b. Assembly Language**

* Uses short words (called mnemonics), like MOV, ADD
* Still close to hardware
* Needs an **assembler** to convert it to machine code

**Used for:** Hardware control, embedded systems

### **2. High-Level Languages**

These are **easy for humans to read**. Slower than low-level, but easy to write and understand.

#### **Examples: Python, C++, Java, C#, JavaScript**

* Use normal English-like words (e.g., print("Hello"))
* Need to be **translated** before running

## 

## 

## **Based on Execution**

## **A. Compiled Languages**

* The program is **translated all at once** before running
* Uses a **compiler**
* Runs fast

#### **Examples: C, C++, Rust, Go**

Good for: Performance

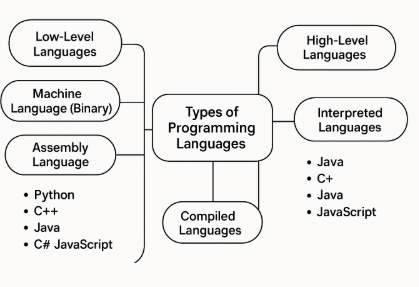
### **B. Interpreted Languages**

* The program is **translated line by line while running**
* Uses an **interpreter**
* Slower but flexible

#### **Examples: Python, JavaScript, Ruby**

Good for: Quick testing, scripts

### **C. Hybrid Languages**

* Use **both compiling and interpreting**
* Code is first compiled to an intermediate form, then interpreted or run in a virtual machine (VM)  
  

#### **Examples: Java, C#, Kotlin**

Good for: Portability across systems

## **Compiler vs Interpreter**

| Feature | Compiler | Interpreter |
| --- | --- | --- |
| Working Style | Translates whole code at once | Translates one line at a time |
| Speed | Fast after compiling | Slower (runs line by line) |
| Error Checking | Shows all errors after compiling | Shows error one at a time |
| Output | Creates an executable file (.exe) | Does not create a separate file |
| Example Languages | C, C++, Java (partially), Go | Python, JavaScript, Ruby |
| Usage | Good for big programs needing speed | Good for testing and quick scripts |
| Need to Re-Translate? | No, once compiled | Yes, every time you run it |

## **Example:**

* **Compiler:** Like writing your full essay, then printing it once at the end.
* **Interpreter:** Like reading your essay out loud, **line by line** to someone.

