Introduction to Computer Programming

•••

- *Eng. Sylvain Manirakiza* -

Sept 07, 2025

Meet your Instructor

Sylvain Manirakiza

- Msc. Big Data Analytics
- BSc: Telecommunication and computer network System
- Expert in IT, ML and AI,
- Consultant in Strategy and Operations & Learning and Development

- **Tel:** *+250795468149(whatsapp)*
- **Email:** *student.auca2023@gmail.com*

Devotion

"I can do all things through Christ who strengthens me."
— Philippians 4:13 (NIV)

Reflection

As we begin this journey into the world of programming, a subject that can sometimes feel challenging and unfamiliar, let this verse remind us of one powerful truth:

- You **are not alone** in your learning.
- With determination, support, and faith, **no concept is too hard**, and **no error is too big to fix**
- Christ is our **source of strength**, not just in spiritual matters, but in every area of life, including learning new skills.

>>>> Even when the code breaks, don't break. Learn, retry, and grow stronger.

Course Overview

Objectives 🎯

1. Understand the logical structures underlying programming.
2. Design and analyze algorithms to solve real-world problems.
3. Work with control structures, data structures, and modular programming.
4. Implement solutions using memory management concepts like pointers.
5. Handle data storage and retrieval through file systems.

Course Structure

Chap 1.

Introduction to Programming and Algorithms. (Week 1)

Chap 2.

Data Control Structures (Weeks 2 & 3)

Chap 3.

Linear Data Structures (Weeks 3 & 4)

Course Structure

Chap 4.

Modular Programming (Weeks 5 & 6)

Chap 5.

Pointers and Memory Management (Weeks 7 & 8)

Chap 6

File Handling (Week 9)

Day 1 - Agenda

Introduction to Computer Programming overview

- **Quick introduction**
  - *Your full name*
  - *Faculty/Department*
  - *What semester?*
  - *What did you do in High school?*
  - *One thing you expect to get from this course*

Classroom Rules and Guidelines

Introduction to Computer Programming

- **Participation and Respect**
  - *Be present and actively participate in discussions and activities.*
  - *Respect everyone's opinions and contributions during discussions.*
  - *Use appropriate language and tone in all communications.*
- **Time Management**
  - *Arrive on time for every session. 15 min before.* 15 min after 6pm, no one is allowed to join!
  - *Submit assignments and tasks by the given deadlines.*
  - *Follow the class schedule to avoid disruptions.*
- **Communication and Integrity**
  - *Raise your hand to ask questions or contribute during discussions.*
  - *Maintain academic honesty; no plagiarism or unauthorized sharing of materials.*
  - *Take responsibility for your own learning and ask for help when needed.*

Guidelines

Introduction to Computer Programming

- **Technology and Environment**
  - *Use technology (e.g., phones, laptops) only for class-related purposes.*
  - *Keep the classroom clean and organized.*
- **Collaboration and Feedback**
  - *Work collaboratively during group activities and respect assigned roles.*
  - *Provide constructive feedback when requested and take feedback positively.*
- **Engagement and Improvement**
  - *Actively contribute to group work and discussions.*
  - *Share concerns or suggestions with the instructor respectfully.*

Together, we will create a positive and productive learning environment!

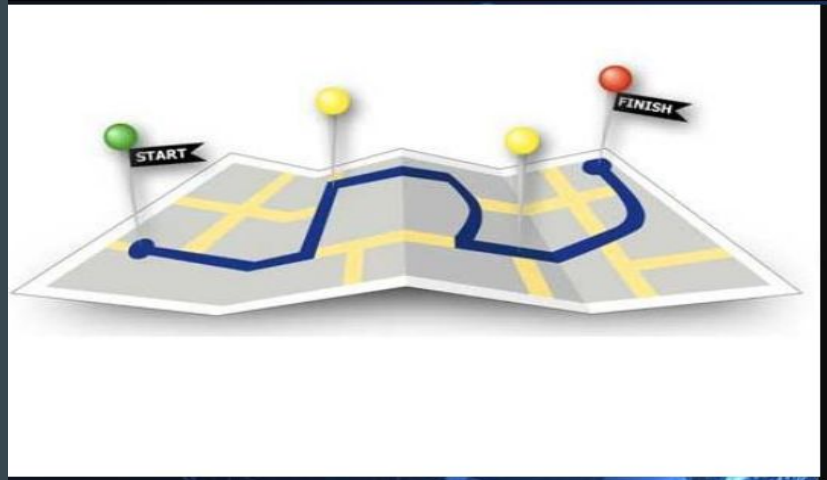
Objectives

Objective 1	• Understand What is programming
Objective 2	• Categories of programming.
Objective 3	• Introduction to pseudocode: Designing algorithms without syntax constraints.

CHAP I:

PROGRAMMING LOGICAL STRUCTURE

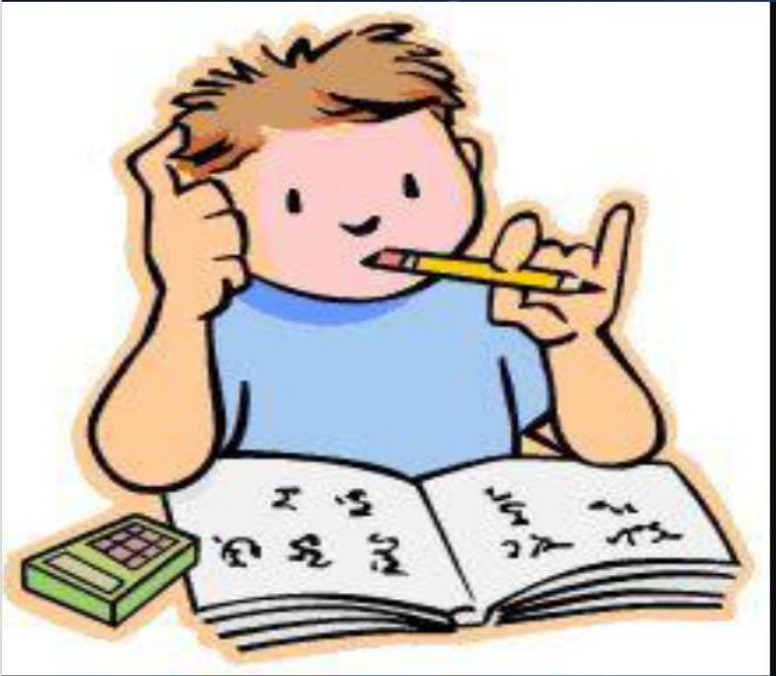
What is an Algorithm ?



- An algorithm is a series of instructions, once executed correctly, leads to a given result .
- If the algorithm is correct, the result is the desired one
- **“The Algorithm Solve the problem”**

- An algorithm must have: Start instruction.
- Each instruction must be precise.
- Each instruction must be unambiguous.
- Each instruction must be executed in finite time.
- An algorithm must have stop instruction.

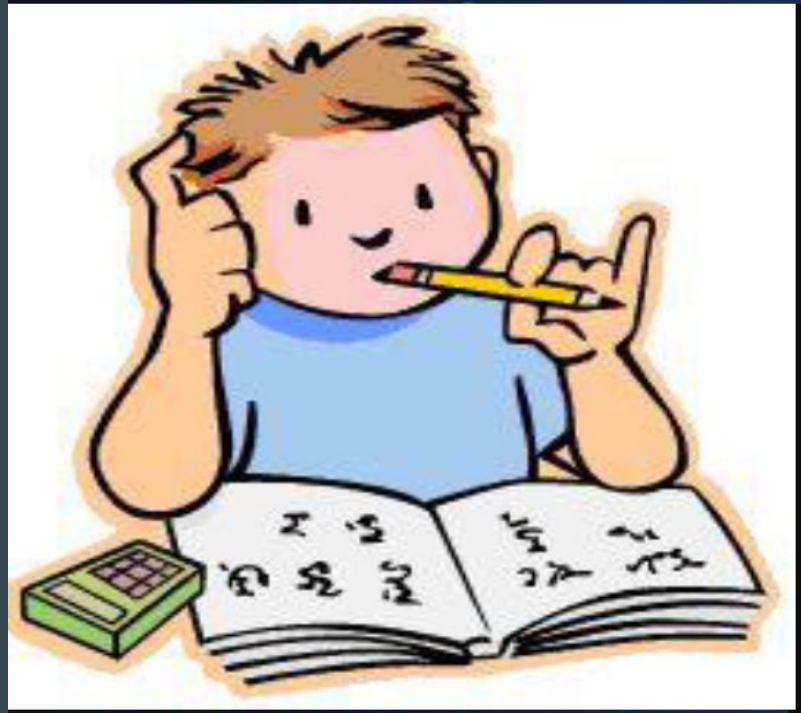
Do you have to be mathematical to be good at algorithmic?



- Do you have to be "good at math" to correctly explain your way to someone? I'll let you judge.
- Mastery of algorithms requires two qualities:

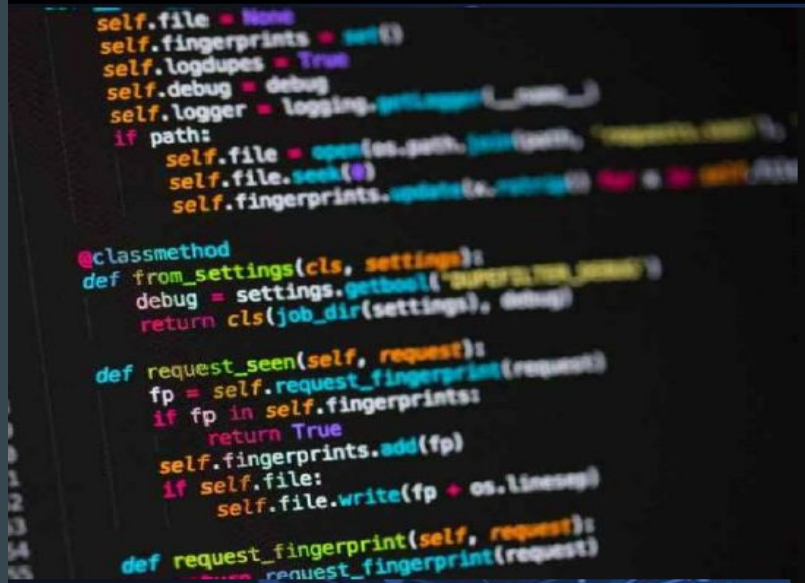
intuition: To know a priori which instructions will achieve the desired result. This is repeated reasoning that begins laborious and ends up becoming "spontaneous".

Contd….



methodical and rigorous: Each time we write a series of instructions that we believe to be correct, we must systematically put ourselves in the place of the machine that will execute them, armed with paper and pencil, in order to check if the result obtained is the one we wanted

Algorithm and Programing



- Why Study algorithms to learn how to program?
- The algorithmic expresses the instructions solving a given problem independently of the particularities of this or that language.
- To learn algorithmic is to learn to handle the logical structure of a computer program.

1. What is Computer Programming ?

- Programming is about solving problems and automating tasks through the creation of algorithms and code.
- It involves breaking down complex tasks into smaller, manageable steps that a computer can follow.
- Programming languages provide **a way for humans to communicate with computers** , enabling them to carry out tasks efficiently and accurately.
- Programmers use programming languages to write code that instructs the computer on what actions to take and how to perform them.

1. What is Computer Programming ?

- This code can encompass a wide range of activities, from basic arithmetic calculations to complex simulations and data analysis.
- By using programming languages and following programming paradigms, developers can create software that serves various purposes and meets specific user needs.

2. Categories of Computer Programming

- Programming can be categorized into several different domains or categories based on the type of applications being developed, the platforms they target, and the specific goals of the programming task. Here are some common categories of programming:

1. Application Development:

- ✔ Desktop Applications: Creating software that runs on personal computers, such as word processors, photo editors, and video players.
- ✔ Mobile Applications: Developing apps for smartphones and tablets using platforms like Android and iOS.
- ✔ Web Applications: Building interactive applications that run in web browsers, often using technologies like HTML, CSS, and JavaScript.

2. Systems Programming:

- Operating Systems: Developing the core software that manages hardware resources and provides services to other software.
- Device Drivers: Creating software that enables communication between hardware devices and the operating system.
- Embedded Systems: Writing software for embedded devices like microcontrollers, used in products such as appliances, automobiles, and industrial machinery.

- **Game Development:**
  - ✔ Game Engines: Designing game engines that power video games and interactive simulations.
  - ✔ Game Logic: Implementing the gameplay mechanics, graphics, sound, and user interactions in video games.
- **Scripting and Automation:**
  - ✔ **Scripting Languages:** Writing scripts to automate tasks, manage files, and manipulate data. Examples include Python, Ruby, and PowerShell.
  - ✔ **Automation Tools:** Creating scripts or programs that perform repetitive tasks, such as batch processing or system maintenance.

- **Scientific and Data Analysis:**
  - ✔ Data Science: Using programming to analyze and interpret large datasets, draw insights, and make predictions.
  - ✔ Scientific Computing: Developing software for simulations, modeling, and scientific research in fields like physics, biology, and engineering.
- **Artificial Intelligence (AI) and Machine Learning:**
  - ✔ Machine Learning Algorithms: Implementing algorithms that allow computers to learn from data and make predictions or decisions.
  - ✔ Natural Language Processing (NLP): Developing software that enables computers to understand and generate human language.

- **Web Development:**
  - ✔ Front-End Development: Creating the user interface and user experience for web applications using HTML, CSS, and JavaScript.
  - ✔ Back-End Development: Building the server-side logic, databases, and APIs that power web applications.
- **Security and Ethical Hacking:**
  - ✔ Security Tools: Writing software to detect vulnerabilities, secure networks, and protect against cyber threats.
  - ✔ Ethical Hacking: Simulating cyberattacks to identify and fix security weaknesses in software and systems.

- **Networking and Network Programming:**
  - ✔ Network Applications: Developing software that facilitates communication and data exchange between computers over a network.
- **Database Programming:**
  - ✔ Database Management: Writing code to create, query, and manage databases, ensuring data integrity and efficiency.

3. What is Programming Language ?

- A programming language is a formalized set of rules and syntax that allows humans to communicate with computers and give them instructions to perform specific tasks.
- It serves as an intermediary between human-readable instructions and machine-executable code.
- Programming languages are used to write software applications, scripts, algorithms, and more, enabling developers to create a wide range of computational solutions.
- Programming languages provide a structured way to express algorithms and logic, making it easier for programmers to convey their intentions to computers.

- They come with a defined set of keywords, syntax rules, and conventions that must be followed to create valid and functional code.

4. Categories Of Programming Language

- Programming languages can be broadly categorized into several types:
  - ✔ **High-Level Languages:** These languages are designed to be more human-readable and abstracted from the underlying hardware. They provide a level of abstraction that makes programming more intuitive and efficient. Examples include Python, Java, C++, and Ruby.
  - ✔ **Low-Level Languages:** These languages are closer to the machine code that computers understand, and they provide finer control over hardware resources. They are less human-readable and require more detailed knowledge of computer architecture. Examples include Assembly languages.
  - ✔ **Scripting Languages:** These are interpreted languages often used for automating tasks, quick prototyping, and developing web applications. Examples include JavaScript, Python, and Ruby.

- ✔ **Compiled Languages:** In these languages, code is transformed into machine-readable binary code before execution. This can lead to faster execution but requires an additional compilation step. Examples include C, C++, and Swift.
- ✔ **Interpreted Languages:** In these languages, code is executed directly by an interpreter, which converts the code to machine instructions on the fly. Examples include Python, Ruby, and JavaScript.
- ✔ **Domain-Specific Languages (DSLs):** These languages are tailored for specific tasks or industries, such as SQL for database queries, HTML/CSS for web development, and LaTeX for typesetting documents.

- ✔ **Functional Languages:** These languages emphasize the use of functions as the primary building blocks of programs. Examples include Haskell, Lisp, and Erlang.
- ✔ **Object-Oriented Languages:** These languages focus on representing data and behavior as objects, allowing for better organization and modularity. Examples include Java, C++, and Python.
- ✔ **Procedural Languages:** These languages focus on specifying procedures or routines that manipulate data step by step. Examples include C, Pascal, and Fortran.

- ✔ **Functional Languages:** These languages emphasize the use of functions as the primary building blocks of programs. Examples include Haskell, Lisp, and Erlang.
- ✔ **Object-Oriented Languages:** These languages focus on representing data and behavior as objects, allowing for better organization and modularity. Examples include Java, C++, and Python.
- ✔ **Procedural Languages:** These languages focus on specifying procedures or routines that manipulate data step by step. Examples include C, Pascal, and Fortran.

5. Programming with Pseudocodes

- Pseudocode is a way to plan and outline a program's logic and structure using a mixture of human-readable language and simplified programming constructs.
- It's **not an actual programming language** , but rather a tool for designing algorithms and expressing ideas before implementing them in a specific programming language.
- Pseudocode makes it easier to understand and communicate the algorithm's flow without getting bogged down in the syntax details of a particular programming language.

5. Programming with Pseudocodes

Ways to Write an Algorithm

Algorithms can be represented in different forms depending on the audience and purpose. Here are the five most common ways, with simple examples:

- **1. Natural Language**: Written as plain instructions.
  Example: 'Step 1: Take two numbers. Step 2: Add them together. Step 3: Display the result.'
- **2. Pseudocode**: Structured but not tied to a programming language.
  Example:
  ```
  INPUT number1, number2
  sum = number1 + number2
  OUTPUT sum
  ```
- **3. Flowcharts**: Graphical representation using symbols and arrows.
  Example: Start → Input → Process (Add) → Output → End
- **4. Programming Code**: Writing the algorithm in an actual language.
  Example (Python):
  ```
  number1 = int(input('Enter first number: '))
  number2 = int(input('Enter second number: '))
  print('Sum:', number1 + number2)
  ```
- **5. Mathematical Notation**: Expressed as formulas or relations.
  Example: Binary Search repeatedly halves the interval until the element is found.

- Keep in mind that pseudocode is not standardized, and you can adapt it to your needs and preferences.
- The goal is to communicate the algorithm's logic clearly without being constrained by the syntax rules of a specific programming language.
- When using pseudocode, focus on expressing the overall logic, control structures (if-else, loops), and the sequence of steps involved in solving the problem.
- Once you have a well-defined pseudocode algorithm, you can then translate it into a specific programming language of your choice.

Homework

Assignment #1 – Introduction to Computer Programming & Algorithms

Instructions:
Building on our discussion in class, you are required to conduct independent research and write a short paper (maximum **one page**) covering the following points:

1. **What is Computer Programming?**
    - Define and explain computer programming in your own words.
2. **Programming vs. Algorithm**
    - Explain the difference between a computer program and an algorithm.
3. **Real-World Example**
    - Provide one real-world example where an algorithm is applied (it can be in daily life, technology, business, science, etc.).

Format Guidelines:

- Maximum: **1 page** (typed, single-spaced or 1.5 spacing).
- Use clear headings for each section.
- Cite at least one source of information (book, article, or reliable website).

Submission Deadline:

- **Thursday,11th, before the end of the day(11:49pm)**.
- Submit via this Google form.