ICT @ Mzuni

Introduction to programming

Chimango Nyasulu, PhD

ICT @ Mzuni

Outline

- Describe Programming language
- Language Models
- Chronology of Programming
- Object Oriented Concepts
- Classes
- Encapsulation and Abstraction
- Messages and Operations
- Relationships
- Polymorphism

Describing Programming language

What is computer programming?

- is the process of designing and building executable computer software to accomplish a specific task.

Key components of computer programming include:

1. **Languages**: Programming languages (like C, Python, Java, C++, etc.) provide the syntax and semantics for writing code.

2. **Algorithms**: These are step-by-step procedures or formulas for solving a problem or performing a task.

3. **Logic**: Programming requires logical thinking and problem-solving skills to develop efficient solutions.

Describing Programming language

Key components of computer programming include:

4. **Development Tools**: Programmers use various tools and environments, such as integrated development environments (IDEs), text editors, and debuggers, to write, test, and maintain code.

5. **Testing and Debugging**: After writing code, programmers need to test their programs to find and fix bugs or errors to ensure that the software works as intended.

6. **Collaboration**: Many programming projects involve teamwork, where programmers collaborate with others, use version control systems (like Git), and adhere to coding standards and best practices.

Language Models

Programming language models, or programming language representations are formal abstractions that define the syntax, semantics, and behavior of programming languages.

Key components and characteristics of programming language models:

1. **Syntax**: This defines the set of rules that govern the structure of valid statements in a language.

2. **Semantics**: This refers to the meaning associated with the syntactic structures.

3. **Type System**: Many programming languages have a type system that defines how data types are handled.

4. **Execution Model**: This describes how programs are executed. It may include details about memory management, control flow, and how different constructs (like functions or objects) are invoked and managed during execution.

ICT @ Mzuni

Language Models

Key components and characteristics of programming language models:

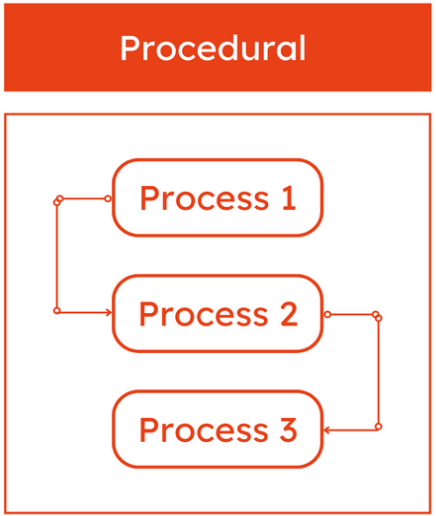
5. **Abstraction Levels**: Programming language models can vary in abstraction, from low-level languages (closer to machine code) to high-level languages (which are more abstract and user-friendly).

6. **Language Paradigms**: Programming language models can also encompass different paradigms, such as procedural, object-oriented, functional, and declarative programming

7. **Tooling and Ecosystem**: Models also consider the tools and environments that support programming languages, such as compilers, interpreters, debuggers, and integrated development environments (IDEs).

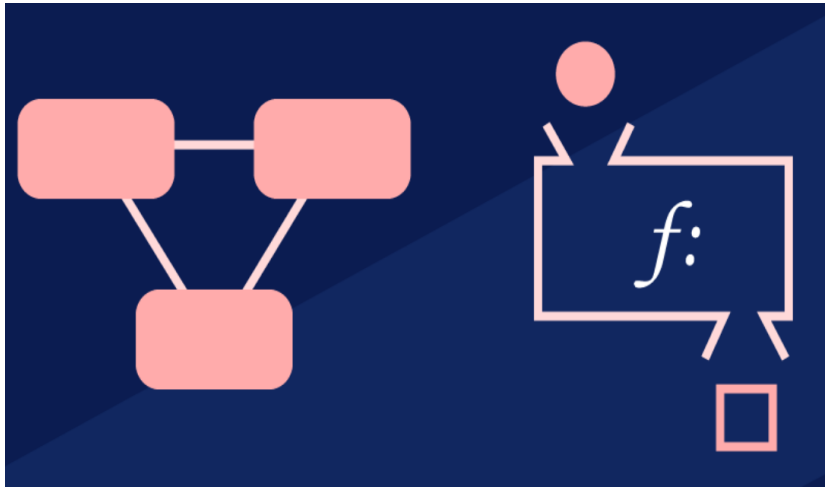
Types of language models

1. Procedural Languages: Focus on procedures or routines (e.g., C, Pascal)



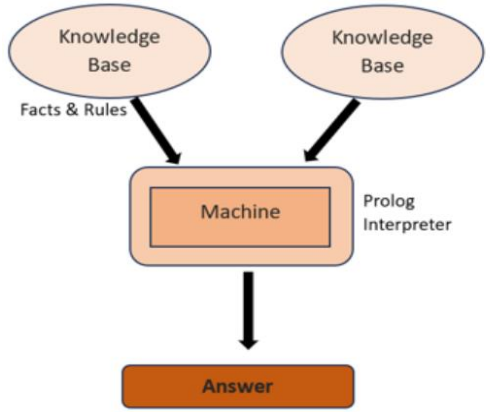
Types of language models

Functional Languages: Emphasize the application of functions (e.g., Haskell, Lisp)



Types of language models

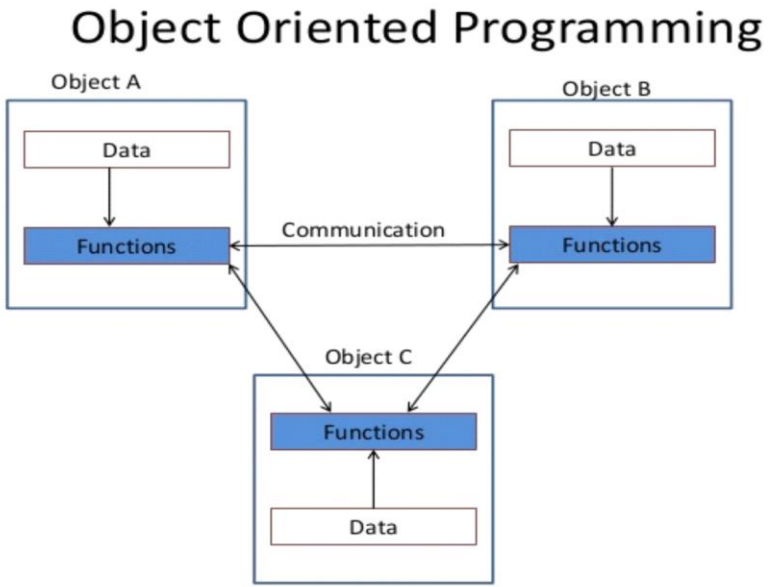
Logic Languages: Based on formal logic (e.g., Prolog)



Logical Programming

Types of language models

Object-Oriented Languages: Organize code into objects (e.g., Java, C++)



ICT @ Mzuni

Chronology of Programming Languages

- **1940s-1950s:** Early assembly languages and FORTRAN
- **1960s:** COBOL and ALGOL, introduction of structured programming
- **1970s:** C language development and the rise of UNIX
- **1980s:** Object-oriented programming begins with C++
- **1990s:** Java and the rise of the web
- **2000s-Present:** Python, JavaScript, and modern languages (e.g., Rust, Go)

Object-Oriented Concepts

What is Object-Oriented Programming (OOP)?

- is a programming paradigm centered around the concept of "objects," which are instances of classes.
- software design around data or objects rather than functions and logic.
- helps create applications that are easier to understand, maintain, and extend.
- common OOP languages include Java, C++, Python, and C#.

A class is a template for creating objects; it defines data (attributes) and behaviors (functions) that operate on that data.

An object represents a specific instance of a class (attributes, behaviour, identity).

ICT @ Mzuni

Object-Oriented Concepts

Key principles of OOP include:

1. **Encapsulation**: involves bundling the data (attributes) and methods (functions) that operate on the data into a single unit, known as a class.

2. **Abstraction**: model classes based on the essential properties and behaviors an object should have

3. **Inheritance**: allows a new class (subclass or derived class) to inherit properties and methods from an existing class (superclass or base class).

4. **Polymorphism**: allows methods to be defined in a base class and overridden in derived classes.

ICT @ Mzuni

Messages and Operations

- A "message" refers to a communication mechanism used to invoke methods or functions on objects.
  - When one object wants to request an action from another object, it sends a message to that object (calling a method).
  - E.g. from object car, you might send a message startEngine(parameter list).

Relationships in OOP

- In OOP, a relationship refers to how different classes or objects interact with each other.
    - These relationships help to structure the code and define how objects collaborate to achieve functionality.

Types of relationships in OOP:

1. **Association:** This is a general relationship where one class uses/interacts with another class.
    - It can be one-to-one, one-to-many, or many-to-many.
    - E.g. a Teacher class might be associated with a Student class, where a teacher can teach multiple students, and a student can have multiple teachers.

ICT @ Mzuni

Relationships in OOP

Types of relationships in OOP:

2. Aggregation: This is a special form of association that represents a "whole-part" relationship but with a weaker bond.

- The part can exist independently of the whole.
- E.g. a Library class may have an aggregation relationship with a Book class; books can exist without being part of a library.

3. Composition: This is a stronger form of aggregation where the part cannot exist independently of the whole.

- E.g. a Car class may have a composition relationship with an Engine class; if the car is destroyed, the engine is also destroyed.

ICT @ Mzuni

Relationships in OOP

Types of relationships in OOP:

4. Inheritance: This is a relationship where a class (the subclass or derived class) inherits properties and behaviors (methods) from another class (the superclass or base class).

- E.g. a Dog class might inherit from an Animal class, meaning Dog will have all the attributes and behaviors of Animal plus any additional features specific to Dog.

5. Dependency: This describes a relationship where one class depends on another class to function. If one class changes, it may affect the other class.

- For instance, a Driver class may depend on a Car class; if the Car class changes, it may impact how the Driver class operates.