Python_Notes Week_01!

Computer: Electronic Device for storing and processing data, typically in binary form, according to given instructions and reduces humans' efforts.

Program: Set of instruction that a computer follows in order to perform a particular task.

Computer Programing: Act to write computer program or the process of writing instructions for a computer to follow. These instructions are called programs.

Languages: A structure system of communication that consists of grammar and vocabulary. It is primary means by which humans convey meaning, both in spoken and signed forms, and may also conveyed through writing.

Translator: A translator is a computer program that converts the programing instruction in binary language and source code that computer understand and process.

High level language: A programing language that is designed to make it easier for humans to understand and write.

Assembly Language: A programing language that translate high level language into machine language.

Compiler based language: A compiler-based language is a programming language that is converted into machine code using a compiler. This machine code is then executed by the processor. Programs are compiled into machine code before execution (e.g., C, C++)

Interpreter Based Language: An interpreted language is a programming language that executes instructions directly, without first compiling the code into machine language. Programs are executed line by line by an interpreter (e.g., Python, JavaScript).

Compiler: A compiler is a software that converts the source code to the object code. In other words, it converts the high-level language to machine/binary language.

Interpreter: An interpreter is a computer program that executes instructions in a programming language without converting them to machine code. It translates source code into object code one instruction at a time, and then immediately executes that object code

Compile Time: compile time is the time it takes to convert a program's source code into machine code. This process is done by a compiler.

How it works

1. A compiler translates the source code into machine code
2. The compiler checks for errors in the code
3. If there are errors, the compiler won't execute the code until the errors are fixed
4. The compiled code is loaded into memory and executed by a processor.

Run Time: Runtime is the period during which a program is executed by the computer. This is when the program's instructions are carried out by the processor.

How it Works

1. The processor executes the compiled machine code (or interpreted code) line by line or instruction by instruction.
2. During runtime, the program checks for and handles errors that occur while the program is running (e.g., division by zero, null pointer exceptions, or file not found errors).
3. If errors occur during runtime, the program may crash, display an error message, or handle the error gracefully, depending on how it was programmed.
4. The program dynamically allocates and deallocates memory, interacts with hardware, and manages system resources (e.g., opening files, network connections) while it is running.

Key Differences:

Compile Time: Focuses on translating source code into machine code and checking for syntax or semantic errors before execution.
Runtime: Focuses on executing the compiled or interpreted code, handling dynamic errors, and managing system resources during program execution.

Syntax Error: A mistake in the structure of the code that violates the rules of the programming language.
Example: Missing semicolons, unmatched parentheses, or typos in keywords.

Run Time Error: An error that occurs **while the program is running**. The code compiles successfully but fails during execution.
Example: Division by zero, accessing an invalid index in an array, or null pointer dereference.

Compile Time Error: An error detected **during compilation** (before the program runs). Includes syntax errors and other issues like:

- Type mismatches (e.g., assigning a string to an integer variable).
- Undeclared variables or functions.
- Missing imports or incorrect method signatures

Case-sensitive: Case sensitivity describes a programming language's ability to distinguish between upper and lower case versions of a letter. Examples of case sensitive programming languages include python, C# and java.

Syntax: In programing, syntax refers to the set of rules that defines the structure and order of symbols, keywords, and punctuation used to write valid code in specific programing language.

Example:

- In Python, to print "Hello, World!" the syntax is: `print("Hello, World!")`
- This means that the "print" keyword needs to be used followed by parentheses containing the string to be printed.
-

Character Set: A character set is a collection of characters used to represent text in a computer system. It includes letters, numbers, symbols, and other characters that a computer can display or process.

Reserve words: Reserved words in Python are words that are reserved by the Python language for specific purposes. These words **cannot** be used as identifiers (like variable names, function names, etc.) because they are meant to have a special meaning or are kept for future use in the language.

Examples of reserved words in Python: False, await, async, None, True, yield, def, if, else, return, import, from, etc.

Keywords: Keywords are a **subset** of reserved words that are **actively used** in the language to perform specific operations or define language structures. Keywords are essential for the syntax and control flow of the program.

Examples of keywords in Python: def, if, else, while, for, break, continue, return, import, class, try, except.

These words are part of the Python syntax and perform specific tasks in the language. For instance:

- `def` is used to define functions.
- `if`, `else` are used for conditional statements.
- `for`, `while` are used for loops.
- `return` is used to exit a function and return a value

Special Words: are words used in certain contexts that aren't keywords but have a special meaning or behavior in those contexts (like `self` in Python classes).

Memory: In programming languages, "memory" refers to a dedicated space within a computer where data (like numbers, text, or instructions) is stored and accessed by the program during execution, essentially acting like a collection of "buckets" with unique addresses, allowing the program to retrieve and manipulate information as

needed; it's like a temporary workspace where the program keeps track of its current information while running.

Key points about memory in programming:

- **Data storage:**
  The primary function of memory is to hold the data a program needs to operate, including variables, function arguments, and intermediate results.
- **Addressing:**
  Each piece of data in memory has a unique address, which allows the program to access it specifically.
- **Logical vs. Physical Memory:**
  While a programmer interacts with "logical memory" (an abstraction of memory addresses), the actual physical hardware (RAM) stores the data in a different way, managed by the operating system.
- **Memory management:**
  Programmers need to manage memory by allocating space for data when needed and releasing it when no longer required to avoid memory leaks.

Data-type: In programming languages, data types are classifications that define the kind of value a variable can hold, including common types like integers, floating-point numbers (floats), characters, strings, and booleans, which are essential for performing operations on data accurately and efficiently.

- **Numeric data types:**
  - **int (integer):** Whole numbers like 1, 2, 3, -5
  - **float (floating-point):** Numbers with decimal places like 3.14, -2.5
- **Text data type:**
  - **str (string):** Sequences of characters enclosed in quotes like "Hello World"
- **Boolean data type:**
  - **bool (boolean):** Represents logical values True or False

Variables: Variables are the logical names of memory locations where we can store our data or variable has ability to change its values.