INTRODUCTION TO PROGRAMMING

[@kyoumah_07]

{PYTHON}
AND
[MCA - SEMESTER - 1]

[MODULE - 1]

⊛ {What is Python?:

- Python is a high level, interpreted and general purpose programming language used widely in a variety of application domains.
- Python is dynamically type, interpreted, cross platform, open source object-oriented programming language.
- Python works both as a scripting and programming language depending how it is used.

⊛ PYTHON AND HISTORY:

- Developed by Guido Van Rossum in the late 1980s.
- Influenced by "ABC" and "Modula-3" programming language.
- Started as a hobby project over christmas holidays, intending to create an easy-to-read powerful language.

Python 0.9.0 was released in market with features like:
- → classes and inheritance.
- → Exception handling.
- → function and modules.

Python 1.0 was released in January 1994 with an official release.
- functional tools such as map(), filter(), reduce(), lambda.
- Modules allowing the reuse of code.

Python 2.0 released on October 16, 2000 with some key features:

- List Comprehensions → way of creating lists
- Garbage collections → automatic memory management
- Unicode support → handle text in multiple languages

Became a very dominant version and used for a decade.

Python 3.0 released in December 3rd, 2008

Not backward compatible with python 2.0

- Print function
- Better Unicode support
- Improved integer division
- Removal of odd constructs

[Note: Python is named after British Comedy group Monty Python and their television series "Monty Python's flying Circus".

Python 2.0 (2010) remained in use even after the introduction of Python 3.0, largely due to massive codebase built on Python 2, particularly legacy systems and received long-term support till January 1, 2020 when it was officially discontinued.

- **PRESENT DAY:**

standard version, and the latest releases have many features

→ pattern matching.
→ Improved performance.
→ Type hinting and optimization.
→ Remains one of the most popular prog languages worldwide due to simplicity, versatility.

⊛[PYTHON PROGRAM EXECUTION :

Steps involved in a python programm Execution :

STEP 1 : Writing the python code (Source Code)

ex: → print("Hello, World!")

STEP 2: Python Interpreter loads the code and execute the code line by line. The interpreter processes the code in two keys steps:

STEP 3: COMPILATION TO BYTECODE :

- Before the python code is executed, the python interpreter compiles the source code into an intermediate form called bytecode.
- ByteCode : → is a lower-level, platform-independent representation of the code. Not a machine code but rather a set of instruction that python interpreter can understand.
- Python generates a ".pyc" (python compiled) file that contains bytecode and stored in the —pycache— directory.
- The compilation step is implict and happens automatically.

STEP 4: BYTE EXECUTION BY PYTHON VIRTUAL MACHINE (PVM)

- PVM that is a part of python interpreter executes the bytecode.
- PVM is a stack-based machine that executes the bytecode instructions line by line.
- PVM reads and interprets the bytecode to perform the actual operations specified in the original source code.

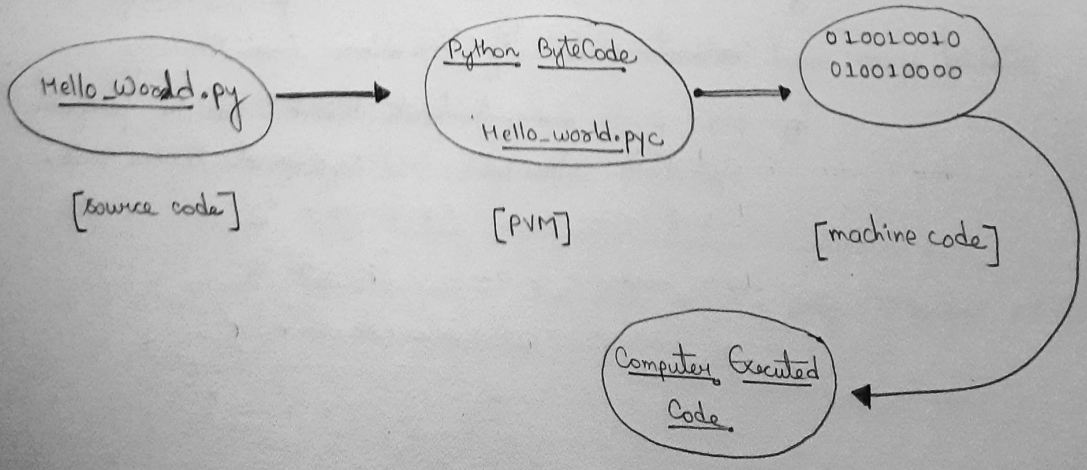
STEP 5: Execution flow and Runtime

- During execution python handles various tasks such as memory management, variable allocation, and object creation.
- Python supports dynamic typing, meaning that variable types are determined at runtime.
- The python runtime environment includes:
  - Garbage collection
  - Error handling

STEP 6: INTERACTION WITH OPERATING SYSTEM

- Python Interpreter interacts with the operating system (OS) through system calls for tasks like:
- Input/Output operations (reading & writing files)
  Allocating memory
- Managing processes and threads.

[Note: The interpreter inside the PVM translates the program line by line thereby consuming a lot of time. To overcomes this a JIT (Just-in time) compiler is added to PVM. Not used in all PVM's.



⭐: PYTHON'S KEY CHARACTERISTICS & FEATURES

- **SIMPLE & READABLE SYNTAX :→**
  - Clean and easy to understand, resembling plain English making it accessible for beginners and allows developers to write less code.
  - Identation is used to define code blocks instead of brackets.

- **INTERPRETED LANGUAGE: →**
  - It is an interpreted language, meaning the code is executed line by line by python interpreter.
  - makes python platform independent.

- **DYNAMICALLY TYPED :→**
  - No need to declare the data type of variables, it automatically assigns or detects data type of the variable at runtime on the value assigned to it.
  - ```
    x = 10      # an integer
    x = "Hello" # a string
    ```

- **OBJECT-ORIENTED PROGRAMMING (OOP) SUPPORT :**
  - Supports object oriented Programming (OOP) allowing developers to create reusable and modular code using concepts like classes, inheritance, encapsulation and polymorphism.

- **EXTENSIVE STANDARD LIBRARY : →**
  - python comes with a rich and extensive standard library that includes modules and packages for handling various tasks such as file I/o, regular expressions etc.
    ```

① CROSS-PLATFORM COMPATIBILITY :

Python is a cross-platform and works on different operating systems like Windows, Linux, OSX etc.

② LARGE ECOSYSTEM & ACTIVE COMMUNITY :

- Python has a massive ecosystem of third-party libraries and frameworks.
- Available through PyPI → python package Index.
- Large & active community, extensive support, documentation.

③ HIGH LEVEL LANGUAGE :

- Abstracts away many details about the computers hardware such as memory management, processor-specific instruction.
- REPL → (Read-Eval-Print-Loop) allows users to execute code line by line, making it easy to test small code snippets.

④ SUPPORTS MULTIPLE PROGRAMMING PARADIGMS :→

- Supports OOP, procedural programming → functions and procedures.
  functional programming → treats computation as the evaluation of the mathematical functions.

⑤ OPEN SOURCE : →

- Python is open source and freely available to use, modification and distribution.

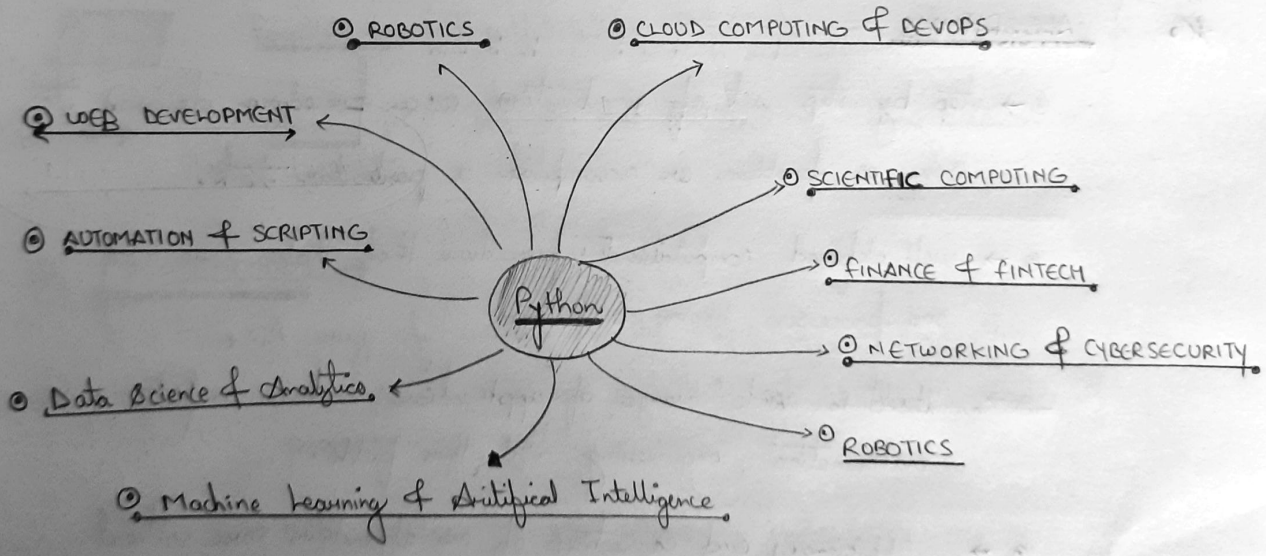
⑥ INTERATION COMPABILITIES :→

- Python can easily integrate with other programming languages like C, C++, Java and .NET through tools like SWIG and Cython.
- Allows developers can write python code that interacts with C/C++ libraries for performance-critical apps.

- **MEMORY MANAGEMENT & EXCEPTION HANDLING:**
  - Python has built-in automatic memory management and a garbage collector that handles the allocation and deallocation of memory.
  - Python provides robust exception handling mechanism to catch and manage errors during runtime, improving code reliability.

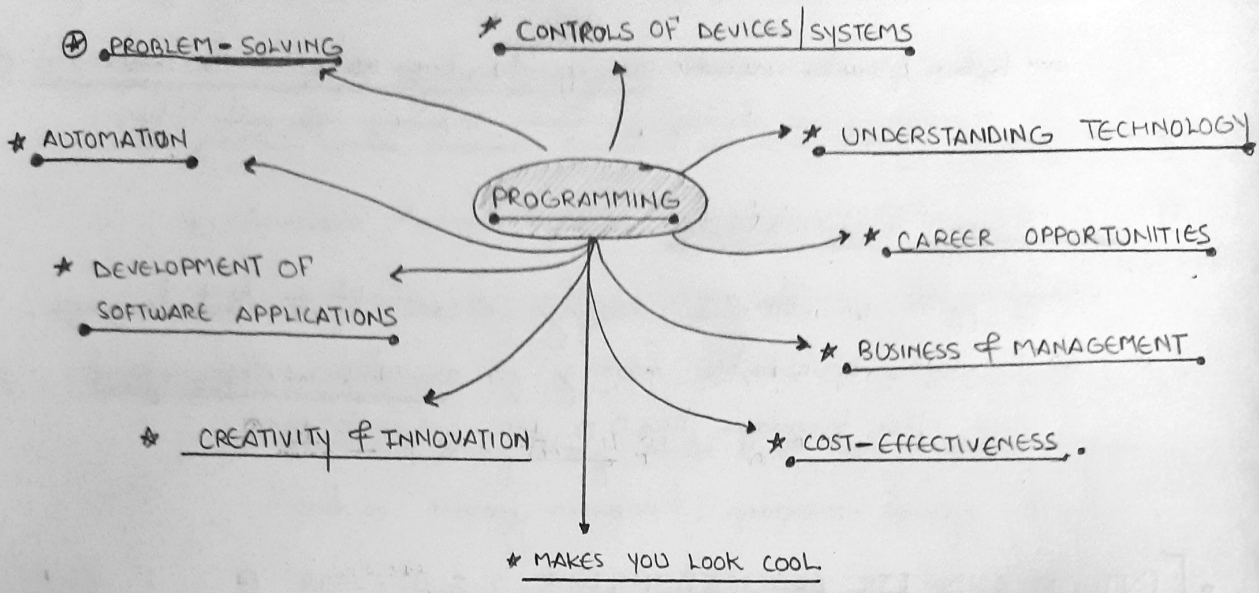
- **EXTENSIBLE & EMBEDDABLE :→**
  - Python is extensible, meaning you can integrate other languages.
  - Also embeddable, meaning you can include python code in other languages like C or Java for scripting.

- **[PYTHON AND ITS APPLICATIONS :→**



NEED OF PROGRAMMING: ?

[valuable skill in todays world.]



ALGORITHMS

- → step by step set of instructions or a procedure designed to solve a problem or accomplish a particular task.
- → well defined computational procedure that takes some input and processes some output.
- → Used in wide range of applications such as computer science, mathematics, engineering and other fields.
- → efficiency and correctness of an algorithm are critical factors that determines its usefulness.
- → can be represented using flowcharts, pseudocode or natural language.

⭐ FLOWCHARTS :

→ graphical representations of a process or algorithms, showing the flow of steps through various shapes connected by arrows.

→ Each shape represents a different type of action or decision.

→ Flowcharts are useful for illustrating how a system or process works at a high level.

FLOW CHART SYMBOL: →

• **Oval** (Terminator): Represents the start or end of a process.

• **Rectangle** (Process): Represents a task or action to be performed.

Diamond (Decision): Represents a decision point, where the process can branch into different paths.

Parallelogram (Input/Output): Represents an input or output operations.

→ **Arrows**: Indicate the direction of flow between steps.

⭐ PSEUDOCODE: →

→ A way of writing out the steps of an algorithm in plain language, resembling code but not written in any specific programming syntax.

→ Helps bridge the gap between a high-level idea and actual code by focusing on the logic rather than the syntax of a programming language.

Example: →

```
start
  input num
    if num % 2 == 0:
    print("Even")
    else
      print("odd")

  End.
```

★ PROGRAM :

- → set of instructions or code written in a programming language. that tells what a computer what to do.
- → typically written by a programmer using a programming language such as python, C, Java etc.
- → Contains (input → processing → output) components

input → data that program uses to perform its task.

processing → involves manipulation of input according to program.

output → result of processing of programm.

★ STEPS TO SOLVING PROBLEMS :

★ Understanding the Problem :

identify input data, output that are expected, constraints and requirements to solve the program.

★ Plan & Design :

includes determining the programming language, algorithms, data structure, user interface, etc.

★ Code the program :

write the code for the program based on your design. This involves translating your plan into code using the chosen programming language.

★ : Test the program :

Test the program to ensure that it functions correctly and produces the correct output.

★ DEBUG THE PROGRAM :

identify and fix errors and bugs in your code.

④ **optimization** :→

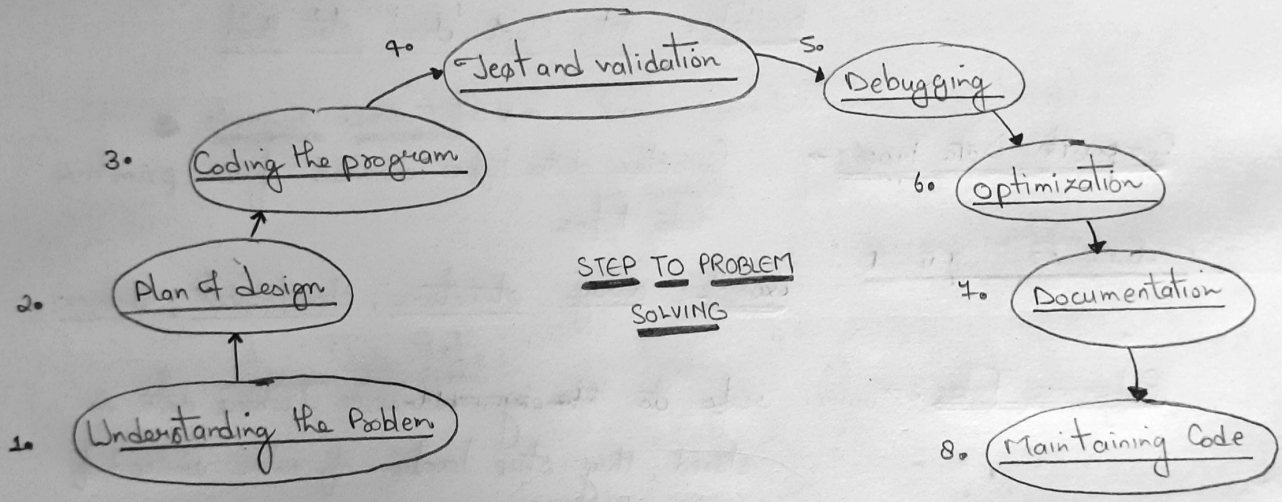
optimize the code to improve its performance and efficiency.

⑤ **Documentation** :→

necessary for readability and reusability, including the code and any instructions or user manuals.

⑥ **Maintain the program**:

maintain the program over time by updating it to fix bugs improve performance.



DATA TYPES :

Data types are classifications that specify the type of value or data a variable can hold in a programming language. Each data types defines values a variable can take, the operations that can be performed on it and how much memory it occupies.

There are various data types in general (not programming language specific)

- **Primitive Data type** → basic data types provided by a language
  example: int, float, char, str, bool

Composite Data Types :→ Complex data types derived from primitive data types.
example: array, structure, class/object, Union

Reference Type :→ refer to memory addresses where data is stored, they store location of value instead of value itself.
example:→ pointers, reference-types

Enumerations :→ user-defined data type consisting of a set of named values. Collection of constants.
example: enum Days { Sunday, Monday, ... }

Derived Data Types: derived from primitive but offer more functionality.
example list, Dictionary, sets.

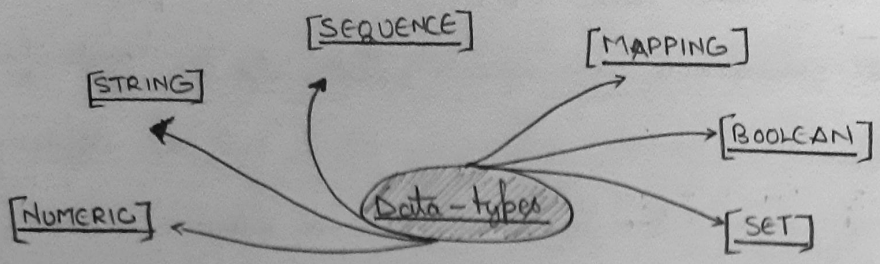
Abstract Data Type : defines behaviour and operation without specifying exact implementation.
example: stack, Queue, Tree, graphs.

How are these data-types implemented and used depends on the specific programming languages and their way to handle these data types.

Data Types in Python:

Python in an object-oriented programming, and the data types are actually classes and variables are instances (objects) of these classes.

- **Numeric Data type :→** holds numeric values (all kind).
  - it includes integers (int), float (decimal point) and complex (imaginary numbers)
- **string data type :** holds "strings" [series of characters] belongs to "str" class.
- **Sequence :→** list, tuple, range belongs to this data type → holds collection of items.
- **Mapping :→** dictionary belongs to this data-type (dict) holds data in key-value pair form.
- **Boolean :→** (bool), holds either [True] or [False] value.
- **Set :→** set, frozenset belongs to this data type holds collection of unique items.



[Note: → The "type()" function can be used to check which class or data-type does a variable belongs to.

example

```
a = 5
print (type(a))
#output= <class 'int'>
```

```
a = "Hello"
print (type(a))
#output= <class 'str'>
```

ⓐ VARIABLES IN PYTHON :

- Variables are named storage locations in a computer's memory that holds a value. The value of a variable can change or vary as a program executes, which is why its called a "variables"
- In programming language, variables are used to store data so that it can be referenced or manipulated later.

characteristics of variables :

- **Data Type :** → each variable has a data-type defining what value it holds.
- **Name :** → variables have names (identifiers).
- **Value :** → holds a value, can be changed during prog execution
- **Scope :** → region of the program where a variable is accessible.
- **Lifetime :** → The duration for which a variable exists in memory.

example

name1 = "Hello_world"

age = 12

★ PYTHON NAMING CONVENTIONS :

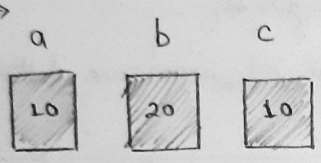
- Use lowercase letters.
- Use underscore (_) to seperate words (snake-case).
- names should be descriptive but not very long.
- Cannot start with a digit or special character (excluding '_')
- Variable names can contain digits.
- Constants names should be in all uppercase letters seperated by _ underscores.
- function names should be written in lowercase with words seperated by underscores _ .
- function name should be descriptive and indicate an action.
- classes should follow PascalCase (UpperCamelCase), each word starts with an uppercase letter, no underscore are used.
- classes names should be nouns because they usually represent objects.
- Module name (file) should be short and in snake-case.
- Package name (folder) should be in lowercase, avoid underscore.
- private variable and methods should start with underscore.
- Avoid using Reserved keywords for naming variables.

- In python, variable acts as a reference, not as containers.
- whenever we create a variable, we do not create a box that holds its value, but rather a label or reference pointing to specific object in memory.

Example:

```
int a = 10;
int b = 20;
int c = 10;
```

[In C]



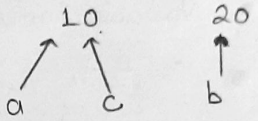
- 3 different storage for 3 variable
- physically stored

say d = b

→ where a copy is created of b and stored in d.

Python:

```
a = 10
b = 20
c = 10
```



- a, c reference to 10 in memory.
  b reference to 20 in memory.
- variables are just names that reference objects stored in memory.

→ say d = b

- now python doesn't create a new copy of b, it just references to value of b somewhere in memory. → d is made to point to the same object that b is pointing to.

→ Assignment doesn't copy objects in python instead reference it to the same object in the memory.

→ Mutability matters as changes to mutable object affects all references to the object. Immutable objects, however will create new objects when modified.

→ Garbage collection :→ when no variable references an object, python will eventually free the memory used by the object.

IDENTIFIERS

→ an identifier is a name that is used to identify a variable, function, class or any other object.

→ Can consist of letters, digits and underscores.

→ Python identifiers are case-sensitive.

KEYWORDS

→ reserved words that have specific meanings and functionalities in the language.

→ keywords cannot be used as identifiers.

{List of keywords}

→ False	• class	• finally	• is	• return
→ None	• continue	• for	• lambda	• try
→ True	• def	• from	• nonlocal	• while
• and	• del	• global	• not	• with
• as	• elif	• if	• or	• yield
• assert	• else	• import	• pass	
• break	• except	• in	• raise	