Introduction

Why Python?

- Open source
- Cross platform
- Huge community support
- Huge set of libraries and tools

what is Python?

- Python is an interpreted, general-purpose (or multi-purpose) high-level programming language with an easy-to-understand syntax and dynamic semantics.
- Monty Python's Flying Circus: This is the title of a famous British comedy series.
- Guido van Rossum: He is credited as the creator of the Python programming language.

Why do we need a Translator?

Translator: A translator (compiler/interpreter) converts high-level language into machine code.

- High-Level Languages: Programs are written in human-readable languages like Python, Java, or C++.
- Human Understanding: Only humans can naturally understand high-level languages, not computers.
- Machine Execution: Translators enable computers to run human-written programs by converting them into machine-readable code.

What is a Compiler?

- A compiler is a complex piece of software whose job is to convert source code machine understandable code (or binary code) in one go.
- c language

#include <stdio.h>

int main() {

```
    int sum, a = 10, b = 20;

    sum = a + b;

    printf("%d", sum);
```

}

- Compiler (Executable code) 10101010011111010001 0 1001001010010010001 0101010101001001010 1001100101001010010 01 0010101010101010010

- Friend's Machine only comipler code require

What is an Interpreter?

- An interpreter is a software program written to translate source code to machine code but it does that line by line.
- java script

```
var x, y, z;

x = 5;

y = 10;

z = x + y;

document.getElementById("para").innerHTML = "The value of z is " + z + ".";
```

- copy of souce code

```
var x, y, z;

x = 5;

y = 10;

z = x + y;

document.getElementById ("para").innerHTML = "The value of z is " + z + ".";
```

- My Machine The value of z is 15.
- Friend's Machine require copy of source code

Pros of Compiled Languages

- Private code.
- Faster execution.
- Fully optimized.

Cons of Compiled Languages

- No portability.
- Extra compilation step.

Pros of Interpreted Languages

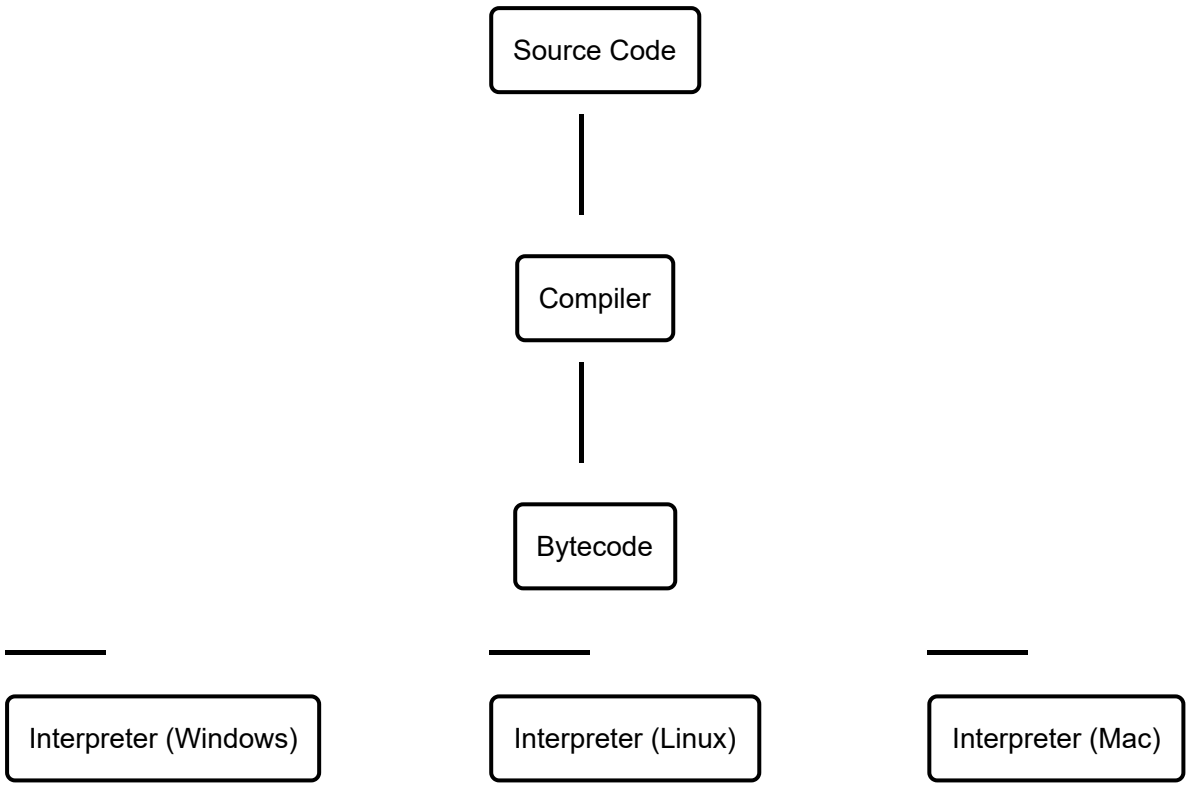
- Portable.
- Easy debugging.

Cons of Interpreted Languages

- Requires interpreter.
- Slower.
- Public code.

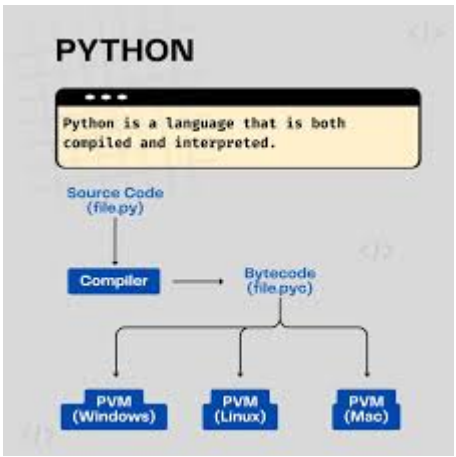
Hybrid approach

- Best of both the worlds - compiler and interpreter.
- Code privacy from compilation and portability from interpretation.



Examples of Compiled, Interpreted, and Hybrid Languages

- Compiled languages: C, C++, etc.
- Interpreted languages: Javascript, PHP, etc.
- Hybrid languages: Java, C#, Kotlin, etc.



```
Microsoft Windows [Version 10.0.22631.4169]
(c) Microsoft Corporation. All rights reserved.

C:\Users\VISHAL>cd C:\Users\VISHAL\OneDrive\Desktop\python demo

C:\Users\VISHAL\OneDrive\Desktop\python demo>python python_demo.py
10

C:\Users\VISHAL\OneDrive\Desktop\python demo>python -m py_compile python_demo.py

C:\Users\VISHAL\OneDrive\Desktop\python demo>python -m dis python_demo.py
  1           0 LOAD_CONST               0 (5)
              2 STORE_NAME               0 (a)

  2           4 LOAD_CONST               0 (5)
              6 STORE_NAME               1 (b)

  3           8 LOAD_NAME                2 (print)
             10 LOAD_NAME                0 (a)
             12 LOAD_NAME                1 (b)
             14 BINARY_ADD
             16 CALL_FUNCTION            1
             18 POP_TOP
             20 LOAD_CONST               1 (None)
             22 RETURN_VALUE

C:\Users\VISHAL\OneDrive\Desktop\python demo>
```

Indentation

Basics of Indentation

- Indentation is the leading whitespace before any statement in Python.
- Purposes:

**Improves readability.

- C program

char* name = "Sam";

if (name == "Sam")

{

printf("Hello Sam!");

}

- python program

name = 'Sam'

if name == 'Sam':

```
    print('Hello Sam')
```

Working of Indentation

**Helps in indicating a block of code.

Rules for Indentation

- Rule #1: Minimum one space is necessary to represent an indented statement.
- Rule #2: The first line of Python code cannot have indentation.
- Rule #3: Indentation is mandatory to define a block of code.
- Rule #4: The number of space must be uniform

Advantages and Disadvantages of Indentation

**Advantages:

- Better readability and identification of the block of code.
- Reduced lines of code.

**Disadvantages:

- Code must be carefully indented specially in the large programs.

Syllabus

Introduction to Python and Jupyter Notebooks

1. Basic Data Types and Operations

1.1 Basic Data Types

- Integers
- Booleans
- Floats
- Strings
- `type()` function
- Typecasting

1.2 Declaring and Using Variables

- Declaring variables
- Comments
- Reading input from users
- Printing output

1.3 Operators

- Arithmetic Operators: `+`, `-`, `*`, `/`, `**`, `//`, `%`
- Logical Operators: `and`, `or`, `not`
- Membership Operators: `in`, `not in`
- Assignment Operators: `=`, `+=`, `-=`, `*=`, `/=`, `**=`, `%=`
- Comparison Operators: `==`, `!=`, `>`, `<`, `>=`, `<=`
- Ternary Operators
- Operator Precedence
- Classification of Operators:
  - Arithmetic
  - Relational

- Logical
- Assignment
- Increment/Decrement
- Bitwise
- Special
- Unary
- Binary
- Ternary

1.4 Using Interactive Shell

- Editing, saving, and running a script

1.5 Jupyter Notebooks

- Creating, Opening, Saving, and Downloading Notebooks
- Running Cells

---

2. Conditional Execution and Iterations

2.1 Control Statements

- Simple `if`
- `if-else`
- `if-elif-else`
- Nested `if`

2.2 Loops

- `for` loop using `range()` with `else`
- `while` loop with `else`

2.3 Break, Continue, and Pass Statements

---

3. Functions, Scoping, and Abstraction

3.1 Declaring, Defining, and Invoking Functions

3.2 Function Arguments

- Keyword
- Default
- Positional
- Variable-length

3.3 Local vs. Global Variables

---

4. Immutable Data Structures

4.1 Strings

- Immutability
- Declaring and accessing through `for` loop
- Slicing
- Concatenation
- String Methods: `len()`, `capitalize()`, `enumerate()`, `isalnum()`, `isalpha()`, `islower()`, `isupper()`, `lower()`, `upper()`, `isnumeric()`, `find()`, `index()`, `split()`, `strip()`, `translate()`, `count()`

4.2 Tuples

- Immutability
- Create, Assign, Access, Delete
- Slicing, Concatenation
- Comparing tuples using `>`, `<`, `==`
- Tuple Methods: `count()`

4.3 Built-in Functions for Tuples

- `sorted()`
- `reversed()`
- `min()`
- `max()`

---

5. Mutable Data Structures

5.1 Lists

- Mutability
- Declaring, Accessing through `for` loop
- List Methods: `append()`, `count()`, `extend()`, `index()`, `insert()`, `pop()`, `remove()`, `reverse()`, `sort()`

5.2 Dictionaries

- Create and access using `for` loop
- Dictionary Methods: `clear()`, `copy()`, `fromkeys()`, `get()`, `has_key()`, `items()`, `keys()`, `update()`, `values()`

5.3 Sets

- Declaring, Accessing
- `set()`, `frozenset()`
- Set Operations: `issubset()`, `issuperset()`, `union()`, `intersection()`, `difference()`, `symmetric_difference()`, `copy()`

5.4 Lambda Functions

- `map()`
- `reduce()`
- `filter()`

---

6. Working with Files

6.1 Working with Text Files

- Opening files in `w`, `r`, and `a` modes with `open()` function

6.2 Reading Data from Files

- `read()`, `readline()`, `readlines()`
- `seek()` and `tell()` functions

6.3 Writing Data to Files

- `write()`, `writelines()`
- Closing files

---

7. Modules and Directories

7.1 Modules and Packages

- Difference between `import` and `from ... import`

7.2 Python OS Module

- `getcwd()`, `chdir()`, `mkdir()`, `listdir()`, `remove()`, `rmdir()`

---

8. Introduction to Object Oriented Programming and Exception Handling

8.1 Abstraction and Encapsulation

- Defining Classes, Attributes, Methods
- Constructors and Destructors
- Objects, Generators

8.2 Exception Handling

- `try`, `except`, `finally`
- Custom Exceptions

---

9. Advanced OOP Concepts and Introduction to NumPy

9.1 Polymorphism

- Overloading and Overriding

9.2 Inheritance

- Single, Multi-level
- Method Resolution Order (MRO)
- Abstract Class

9.3 Arrays in NumPy

- Creating a NumPy `ndarray` Object
- 1D, 2D, and 3D Arrays

9.4 Array Operations

- Indexing, Slicing, Shape, Reshaping, Iteration

9.5 Built-in Functions for Arrays

- `concatenate()`
- `array_split()`
- `where()`
- `sort()`

---

10. Introduction to Matplotlib

10.1 Basic Plotting Methods

- `plot()` with different markers, colors, linestyles
- `xlabel()`, `ylabel()`, `title()` with different fonts, colors, positions
- Grids and Subplots

```
In [ ]:  1
```