📝 Comprehensive Python Study Notes

I. Introduction and Core Execution

Features of Python 🐍

Python is a **dynamic, high-level, free and open-source, and interpreted** programming language.

- **Easy to Code & Read:** Simple, English-like syntax that enforces structure using **indentation**.
- **Object-Oriented:** Supports OOP principles (classes, objects) easily.
- **Portable (Cross-platform):** Code runs on different operating systems (Windows, Linux, macOS) without modification.
- **Interpreted:** Code is executed line by line, which simplifies debugging.
- **Dynamically Typed:** Variable types are determined at **runtime** (e.g., `x = 10` is automatically an integer).
- **Extensive Standard Library:** A huge collection of modules for various tasks, from web development to data science.

Execution of Python Program

The process converts source code into machine-readable instructions:

1. **Compilation Phase:** The Python source code (`.py` file) is converted line by line into an intermediate format called **Python Bytecode** (`.pyc` file).
2. **Execution Phase:** The **Python Virtual Machine (PVM)** reads the bytecode and executes it by translating it into the **native machine code** for the specific operating system and processor.

- **Viewing the Bytecode:** The `.pyc` files are typically saved in a **`__pycache__`** directory.

Python Virtual Machine (PVM)

- **Role:** The PVM is Python's **runtime engine**.
- **Function:** It is the software layer that executes the bytecode.
- **Benefit:** The PVM is the reason for Python's **platform independence**; the same bytecode runs wherever a PVM is installed.

Frozen Binaries

- **Concept:** A standalone executable program that includes the application's **bytecode** and a minimal, bundled copy of the **PVM** (interpreter).

- **Purpose:** Allows users to run a Python application on a machine **without needing to install Python** or its libraries separately.

II. Memory Management and Comparison

Memory Management of Python

Python uses **automatic** memory management.

- **Mechanism:** All Python objects are stored in a **private heap space**.
- **Garbage Collection (GC):** Memory deallocation is handled automatically using two main techniques:
    1. **Reference Counting:** Tracks the number of references pointing to an object. When the count drops to zero, the memory is immediately freed.
    2. **Generational Garbage Collection:** A secondary mechanism used to detect and clean up **reference cycles** (objects that reference each other but are unreachable from the main program).

Comparison: C vs Python

Aspect	C	Python
Type	**Compiled** Language	**Interpreted** Language
Typing	**Statically Typed** (Type must be declared before use)	**Dynamically Typed** (Type checked at runtime)
Speed	**Faster** (Compiles directly to machine code)	**Slower** (Interpreted through PVM)
Memory	**Manual** memory management (`malloc`, `free`)	**Automatic** memory management (GC)
Use Case	System programming, OS, embedded systems	Data science, scripting, rapid prototyping

Comparison: Java vs Python

Aspect	Java	Python
Type	Compiled to **Bytecode**, then executed by **JVM**	Interpreted to **Bytecode**, then executed by **PVM**
Typing	**Statically Typed** (More robust compile-time error checking)	**Dynamically Typed** (More flexible, errors can occur at runtime)
Speed	**Generally Faster** (due to Just-In-Time compilation by JVM)	**Generally Slower** (purely interpreted)
Syntax	More **verbose** and structured (uses braces)	More **concise** and readable (uses indentation)

III. Python Fundamentals (Syntax)

1. Python Character Set

The allowed characters Python recognizes, including:

- **Letters:** A-Z and a-z
- **Digits:** 0-9
- **Special Symbols:** All standard symbols (`!`, `@`, `+`, `=`, etc.)
- **Whitespace:** Space, tab, newline.

2. Tokens (Lexical Units)

The smallest recognizable elements in a Python program:

Token Type	Description	Examples
Keywords	Reserved words with special meaning.	`False`, `if`, `def`, `while`, `import`

Identifiers	Names for variables, functions, and classes. **Case-sensitive.**	`my_name`, `calculate_sum`, `_value`
Literals	Fixed values or constants.	`100` (Numeric), `"hello"` (String), `True` (Boolean)
Operators	Symbols that perform operations.	`+`, `-`, `*`, `=`, `==`, `and`
Punctuations	Symbols that structure statements.	`( )`, `[ ]`, `{ }`, `:`, `,`

3. Comments in Python

Lines ignored by the interpreter, used for explanation.

- **Single-line Comment:** Use the **hash symbol (#)**.
- **Multi-line Comment:** The convention is to use a **triple-quoted string** (`"""..."""` or `'''...'''`) not assigned to a variable.

```
Python

# This is a single-line comment.

"""
This text block serves as a
multi-line comment (or a docstring
if placed after a definition).
"""
```

4. Variables and Assignments

A variable is a name referencing a value in memory.

- **Creating a Variable:** A variable is created the moment you **assign a value** using the **assignment operator (=)**.
- Python

```
age = 25    # Variable created and assigned integer value
city = "London"
```

-
-

- **Multiple Assignments:**
    - **Same Value:** `x = y = z = 50`
    - **Different Values (Unpacking):** `name, score, valid = "Tom", 95, True`