Private Variables

And python extended libraries

- In many programming languages, private variables are used to restrict access to certain attributes or methods of a class from outside that class.
- These are useful for encapsulation, which is a core principle of object-oriented programming (OOP). However, Python does not have true private variables in the same way as languages like Java or C++.
- Instead, Python relies on naming conventions and a mechanism called **name mangling** to provide a form of limited privacy.

Naming Conventions: The Underscore Prefix

- In Python, a variable or method name that starts with a single underscore (e.g., `_variable`) is meant to be treated as "non-public."
- This is a convention that signals to other programmers that this part of the code should not be accessed directly and is intended for internal use only.

Name Mangling: A More Robust Approach

- Python provides a mechanism called **name mangling** to create "private" variables that are harder to access from outside the class.
- This is done by prefixing a variable or method name **with two underscores (__)**, but not followed by another underscore.
- When you do this, *Python automatically changes the name of the variable in a way that includes the class name*.
- This makes it difficult to accidentally override or access these variables from subclasses or outside the class.

Why Name mangling?

Avoiding Name Collisions: When you have a base class and a subclass that both define private variables or methods with the same name, name mangling ensures that they don’t interfere with each other.

Limitations of Name Mangling

- **Not Truly Private**: Even with name mangling, these variables can still be accessed from outside the class if you know the mangled name. This is why Python's "private" variables are sometimes referred to as "**weakly private.**"

Understanding the Operating System

Interface in Python

Os in Python

- Python provides powerful tools for interacting with the operating system through modules like **`os and shutil`**.
- These modules allow you to perform tasks such as navigating the file system, running shell commands, and managing files and directories.
- Let’s break down how these modules work and how you can use them in your Python programs.

The os Module: Interacting with the Operating System

- The **os** module in Python offers a wide range of functions that let you interact with the operating system in a cross-platform way.
- This means you can write code that works on different operating systems like Windows, macOS, and Linux.
- **Example Usage:**

```
import os

current_directory = os.getcwd()

print(current_directory)
```

The `os.getcwd()` function returns the current working directory, which is the folder your Python script is running in.

- **os.chdir('/server/accesslogs')**
- The `os.chdir()` function changes the current working directory to the path you specify. This is like using the `cd` command in a terminal.

- **os.system('mkdir today')**
- The `os.system()` function allows you to run a system command from within your Python script. In this example, it runs the `mkdir today` command, which creates a new directory named `today`

Exploration in python

- **`dir(os)`**: This function lists all the functions and attributes available in the `os` module. It's a quick way to see what you can do with the module.

- **`help(os)`**: This function provides detailed documentation on the `os` module, including explanations of what each function does.

Understanding File Wildcards with the `glob` Module in Python

- When working with files in Python, you may often need to list all files in a directory that match a certain pattern.
- For example, you might want to find all `.txt` files in a folder or all files that start with a certain prefix. The **`glob`** module in Python provides a simple way to do this using **wildcards**.
- **What Are Wildcards?**
- Wildcards are special characters that represent unknown parts of a file name or extension. The most common wildcard is the asterisk (*), which matches any sequence of characters. For example:
- `*.py` matches all files with the `.py` extension.
- `data_*.csv` matches all `.csv` files that start with `data_` (e.g., `data_01.csv`, `data_summary.csv`).