**Private and Public Members in Python Programming**

In Python, class members (attributes and methods) can have different levels of access control. Python follows a convention rather than strict access control like other languages (e.g., Java, C++). It mainly relies on naming conventions to indicate access restrictions.

**1. Public Members**

* Public members are accessible from anywhere—both inside and outside the class.
* There are no restrictions on their accessibility.
* Public members are defined without any special naming convention.

**Example:**

class Car:

def \_\_init\_\_(self, brand, model):

self.brand = brand # Public attribute

self.model = model # Public attribute

def display\_info(self): # Public method

return f"Car: {self.brand} {self.model}"

# Creating an object

car1 = Car("Toyota", "Corolla")

# Accessing public members

print(car1.brand) # Toyota

print(car1.model) # Corolla

print(car1.display\_info()) # Car: Toyota Corolla

💡 **Since brand, model, and display\_info are public, they can be accessed and modified directly.**

**2. Private Members**

* Private members are intended to be **hidden** from outside access.
* They are declared using **double underscores (\_\_)** as a prefix.
* Python does not enforce true private variables, but it uses **name mangling** to make direct access difficult.
* Private members can still be accessed using name mangling (\_ClassName\_\_memberName), but this is not recommended.

**Example:**

class BankAccount:

def \_\_init\_\_(self, account\_number, balance):

self.\_\_account\_number = account\_number # Private attribute

self.\_\_balance = balance # Private attribute

def deposit(self, amount):

self.\_\_balance += amount

return f"Deposited {amount}. New Balance: {self.\_\_balance}"

def withdraw(self, amount):

if amount > self.\_\_balance:

return "Insufficient funds!"

self.\_\_balance -= amount

return f"Withdrew {amount}. Remaining Balance: {self.\_\_balance}"

def get\_balance(self):

return self.\_\_balance # Accessing private attribute inside the class

# Creating an object

account = BankAccount("123456", 1000)

# Trying to access private attributes (will raise an AttributeError)

# print(account.\_\_balance) # ❌ AttributeError

# Accessing private attributes using name mangling (not recommended)

print(account.\_BankAccount\_\_balance) # ✅ Works but not recommended

# Using public methods to interact with private data

print(account.deposit(500)) # Deposited 500. New Balance: 1500

print(account.withdraw(300)) # Withdrew 300. Remaining Balance: 1200

print(account.get\_balance()) # 1200

💡 **Since \_\_balance and \_\_account\_number are private, they can't be accessed directly from outside the class. However, we can still access them using name mangling (\_BankAccount\_\_balance).**

**Key Differences:**

| **Feature** | **Public Members** | **Private Members** |
| --- | --- | --- |
| **Access Modifier** | No prefix (self.attribute) | Prefix with double underscores (\_\_self.attribute) |
| **Access** | Accessible from anywhere | Accessible only within the class (name-mangled outside) |
| **Encapsulation** | Not enforced | Helps in hiding data from external access |
| **Use Case** | Use for general attributes/methods | Use for sensitive or internal attributes/methods |

**Conclusion**

* Python does not have true access modifiers like private, protected, or public, but it follows naming conventions.
* **Public members** are accessible everywhere.
* **Private members** are indicated by a double underscore (\_\_), making them harder to access outside the class.
* Use private members when you want to **encapsulate** data and restrict direct modification.