# **🔓 Python Access Specifiers - Complete Guide**

## **✅ What are Access Specifiers?**

Access specifiers (also called access modifiers) control **where and how variables and methods can be accessed** in Python.

Python has 3 main access levels:

* public – accessible everywhere
* \_protected – accessible within the class and subclasses
* \_\_private – accessible only within the class (uses name mangling)

## **🧪 Example 1: Variable Access (Same Class, Subclass, Outside Class)**

class Parent:

def \_\_init\_\_(self):

self.public\_var = "Public"

self.\_protected\_var = "Protected"

self.\_\_private\_var = "Private"

def access\_from\_same\_class(self):

print("Inside Parent class:")

print("Public:", self.public\_var)

print("Protected:", self.\_protected\_var)

print("Private:", self.\_\_private\_var)

class Child(Parent):

def access\_from\_subclass(self):

print("Inside Child class (Subclass):")

print("Public:", self.public\_var)

print("Protected:", self.\_protected\_var)

try:

print("Private:", self.\_\_private\_var)

except AttributeError:

print("Private: ❌ Cannot access (AttributeError)")

class Stranger:

def access\_from\_other\_class(self, obj):

print("Inside Stranger class (Unrelated):")

print("Public:", obj.public\_var)

print("Protected:", obj.\_protected\_var) # ⚠️ Not recommended

try:

print("Private:", obj.\_\_private\_var)

except AttributeError:

print("Private: ❌ Cannot access (AttributeError)")

## **🧪 Example 2: Method Access (Same Class, Subclass, Outside Class)**

class Parent:

def public\_method(self):

print("Public method")

def \_protected\_method(self):

print("Protected method")

def \_\_private\_method(self):

print("Private method")

def access\_from\_same\_class(self):

print("Inside Parent class:")

self.public\_method()

self.\_protected\_method()

self.\_\_private\_method()

class Child(Parent):

def access\_from\_subclass(self):

print("Inside Child class:")

self.public\_method()

self.\_protected\_method()

try:

self.\_\_private\_method()

except AttributeError:

print("Private method: ❌ Cannot access")

class Stranger:

def access\_from\_other\_class(self, obj):

print("Inside Stranger class:")

obj.public\_method()

obj.\_protected\_method() # ⚠️ Not recommended

try:

obj.\_\_private\_method()

except AttributeError:

print("Private method: ❌ Cannot access")

## **✅ Summary Table – Variable Access**

| **Access Location** | **public\_var** | **\_protected\_var** | **\_\_private\_var** |
| --- | --- | --- | --- |
| Same class | ✅ Yes | ✅ Yes | ✅ Yes |
| Subclass | ✅ Yes | ✅ Yes | ❌ No |
| Outside class | ✅ Yes | ⚠️ Yes (not advised) | ❌ No |

## **✅ Summary Table – Method Access**

| **Access Location** | **public\_method()** | **\_protected\_method()** | **\_\_private\_method()** |
| --- | --- | --- | --- |
| Same class | ✅ Yes | ✅ Yes | ✅ Yes |
| Subclass | ✅ Yes | ✅ Yes | ❌ No |
| Outside class | ✅ Yes | ⚠️ Yes (not advised) | ❌ No |

## **🔐 Name Mangling in Python**

### **✅ What is it?**

When you define a private variable using \_\_var, Python **internally renames it** to \_ClassName\_\_var to avoid accidental access or overriding.

### **🔧 Example:**

class BankAccount:

def \_\_init\_\_(self):

self.\_\_balance = 1000

acc = BankAccount()

# print(acc.\_\_balance) # ❌ Error

print(acc.\_BankAccount\_\_balance) # ✅ Works (name mangling)

⚠️ Name mangling is **not for security** — it's just to **avoid accidental misuse**. You **can still access** it using the mangled name, but **you shouldn't**.

## **🧠 Summary Points**

* Use public for open access
* Use \_protected for internal use or subclassing
* Use \_\_private for internal-only logic
* Name mangling protects variable/method from accidental overrides in inheritance

### **About the Author**

**Gowtham SB** is a **Data Engineering expert, educator,** **and content creator** with a passion for **big data technologies, as well as cloud and Gen AI** . With years of experience in the field, he has worked extensively with **cloud platforms, distributed systems, and data pipelines**, helping professionals and aspiring engineers master the art of data engineering.

Beyond his technical expertise, Gowtham is a **renowned mentor and speaker**, sharing his insights through engaging content on **YouTube and LinkedIn**. He has built one of the **largest Tamil Data Engineering communities**, guiding thousands of learners to excel in their careers.

Through his deep industry knowledge and hands-on approach, Gowtham continues to **bridge the gap between learning and real-world implementation**, empowering individuals to build **scalable, high-performance data solutions**.

𝐒𝐨𝐜𝐢𝐚𝐥𝐬

🎥𝐘𝐨𝐮𝐓𝐮𝐛𝐞 - https://www.youtube.com/@dataengineeringvideos

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - <https://instagram.com/dataengineeringtamil>

📸𝐈𝐧𝐬𝐭𝐚𝐠𝐫𝐚𝐦 - [https://instagram.com/](https://instagram.com/dataengineeringtamil)thedatatech.in

🤝𝐂𝐨𝐧𝐧𝐞𝐜𝐭 𝐟𝐨𝐫 𝟏:𝟏 - https://topmate.io/dataengineering/

💼𝐋𝐢𝐧𝐤𝐞𝐝𝐈𝐧 - https://www.linkedin.com/in/sbgowtham/

🌐𝐖𝐞𝐛𝐬𝐢𝐭𝐞 - https://codewithgowtham.blogspot.com

💻𝐆𝐢𝐭𝐇𝐮𝐛 - http://github.com/Gowthamdataengineer

💬𝐖𝐡𝐚𝐭𝐬 𝐀𝐩𝐩 - https://lnkd.in/g5JrHw8q

📧𝐄𝐦𝐚𝐢𝐥 - atozknowledge.com@gmail.com

📱𝐀𝐥𝐥 𝐌𝐲 𝐒𝐨𝐜𝐢𝐚𝐥𝐬 - <https://lnkd.in/gf8k3aCH>