Gowtham SB
www.linkedin.com/in/sbgowtham/ Instagram - @dataengineeringtamil

🔓 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/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

Gowtham SB
www.linkedin.com/in/sbgowtham/ Instagram - @dataengineeringtamil

💬 **Whats App** - https://lnkd.in/g5JrHw8q

📧 **Email** - atozknowledge.com@gmail.com

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