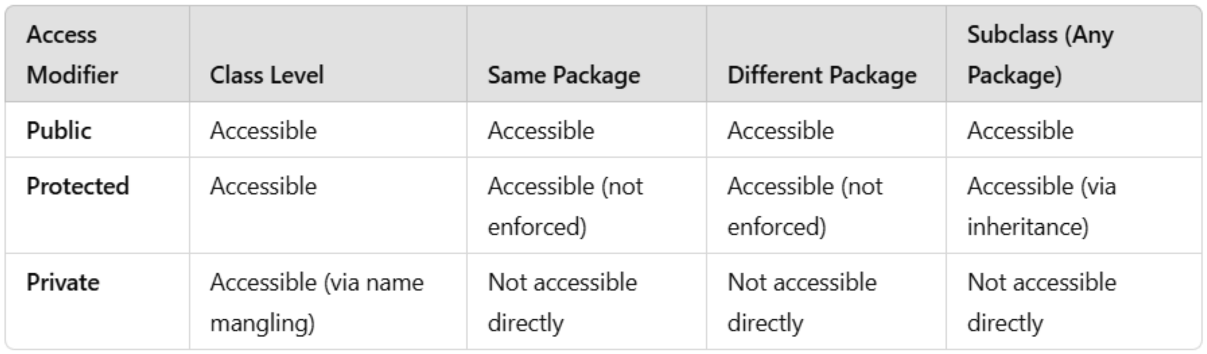
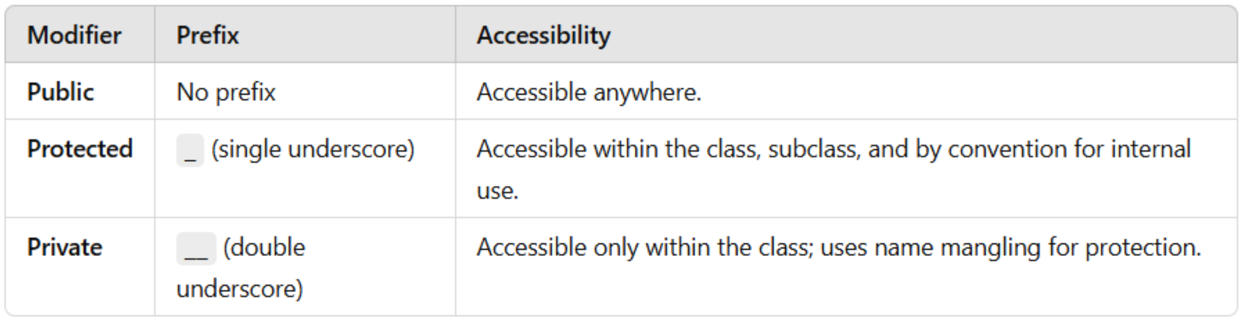
Access Modifiers

Access Level :



Access Modifier	Class Level	Same Package	Different Package	Subclass (Any Package)
Public	Accessible	Accessible	Accessible	Accessible
Protected	Accessible	Accessible (not enforced)	Accessible (not enforced)	Accessible (via inheritance)
Private	Accessible (via name mangling)	Not accessible directly	Not accessible directly	Not accessible directly

Representation :



Modifier	Prefix	Accessibility
Public	No prefix	Accessible anywhere.
Protected	`_` (single underscore)	Accessible within the class, subclass, and by convention for internal use.
Private	`__` (double underscore)	Accessible only within the class; uses name mangling for protection.

Access Modifiers in Python

In Python, access modifiers are not strictly enforced by the language but are implemented using naming conventions. Python has three types of access modifiers:

1. **Public (`public`): Accessible from anywhere.**
2. **Protected (`_protected`): Indicates internal use only; accessible within the class and its subclasses.**
3. **Private (`__private`): Restricted to the class where it is defined.**

Scenario 1: Public Access Modifier

Description: Public members can be accessed from anywhere in the program, including outside the class and across files.

File: `package1/class_public.py`

```
class PublicExample:

    def __init__(self, value):

        self.public_var = value  # Public attribute


    def show(self):

        return f"Public variable: {self.public_var}"
```

File: `main.py`

```
from package1.class_public import PublicExample

# Access public member

obj = PublicExample("Hello")

print(obj.public_var)  # Accessible

print(obj.show())  # Accessible
```

Scenario 2: Protected Access Modifier

Description: Protected members are intended for internal use and can be accessed by the class, its subclasses, and instances of the class (with care). Conventionally prefixed with a single underscore (_).

Example

File: package1/class_protected.py

```
class ProtectedExample:

    def __init__(self, value):

        self._protected_var = value  # Protected attribute


    def _show(self):  # Protected method

        return f"Protected variable: {self._protected_var}"


class SubClass(ProtectedExample):

    def access_protected(self):

        return f"Accessed in subclass: {self._protected_var}"
```

File: main.py

```
from package1.class_protected import ProtectedExample, SubClass


# Access protected member

obj = ProtectedExample("Protected")

print(obj._protected_var)  # Technically accessible but
discouraged

# print(obj._show())  # Technically accessible but discouraged


# Access in subclass

sub_obj = SubClass("SubProtected")

print(sub_obj.access_protected())
```

Scenario 3: Private Access Modifier

Description: Private members are prefixed with double underscores (__) to restrict access to the defining class. Python performs name mangling to make these attributes inaccessible outside the class.

Example

File: package1/class_private.py

python

Copy code

```
class PrivateExample:

    def __init__(self, value):
        self.__private_var = value  # Private attribute

    def __private_method(self):  # Private method
        return f"Private variable: {self.__private_var}"

    def access_private(self):
        return self.__private_method()  # Access private method
internally
```

File: `main.py`

python

Copy code

```
from package1.class_private import PrivateExample

# Access private member
obj = PrivateExample("Private")

# Direct access will fail
# print(obj.__private_var)  # AttributeError
# print(obj.__private_method())  # AttributeError

# Indirect access via public method
```

```
print(obj.access_private())
```

Output:

vbnet

Copy code

```
Private variable: Private
```

Scenario 4: Access Across Packages

Description

Let's demonstrate how access modifiers behave when classes are in different files or packages.

Structure:

```
project/
│
├── package1/
│   ├── class_public.py
│   ├── class_protected.py
│   ├── class_private.py
│
├── package2/
│   ├── main.py
```

File: **package1/class_demo.py**

python

Copy code

```
class Demo:
    def __init__(self):
        self.public_var = "I am Public"
        self._protected_var = "I am Protected"
        self.__private_var = "I am Private"

    def get_private(self):
        return self.__private_var
```

File: **package2/main.py**

python

Copy code

```
from package1.class_demo import Demo

obj = Demo()

# Public Access
```

```
print(obj.public_var)  # Accessible

# Protected Access
print(obj._protected_var)  # Accessible but not recommended

# Private Access
# print(obj.__private_var)  # AttributeError
print(obj.get_private())  # Access private attribute via method
```

Output:

mathematica

Copy code

```
I am Public
I am Protected
I am Private
```

Scenario: Using a Protected Attribute in a Subclass from Another Package

We'll create two packages: `parent_package` and `child_package`.

- `parent_package` contains a base class with a protected attribute.

- `child_package` contains a subclass that accesses the protected attribute.

Directory Structure

```
project/

│

├── parent_package/

│   ├── __init__.py

│   ├── parent.py

│

├── child_package/

│   ├── __init__.py

│   ├── child.py

│

└── main.py
```

Code

File: `parent_package/parent.py`

```
class Parent:
```

```python
    def __init__(self):

        self._protected_var = "This is a protected attribute."


    def display_protected(self):

        return f"Parent Class: {self._protected_var}"
```

File: **`child_package/child.py`**

python

Copy code

```python
from parent_package.parent import Parent


class Child(Parent):

    def access_protected(self):

        return f"Child Class: {self._protected_var}"
```

File: **`main.py`**

python

Copy code

```python
from child_package.child import Child

from parent_package.parent import Parent


# Create an instance of the Parent class
```

```
parent_instance = Parent()

print(parent_instance.display_protected())


# Attempting to access the protected variable directly (Not
recommended, but possible)

try:

    print(parent_instance._protected_var)

except AttributeError as e:

    print(f"Error: {e}")


# Create an instance of the Child class

child_instance = Child()

print(child_instance.access_protected())
```

Output

plaintext

Copy code

```
Parent Class: This is a protected attribute.

This is a protected attribute.

Child Class: This is a protected attribute.
```