# Access Modifiers in Python : Public, Private and Protected

**Prerequisites:**[Underscore (\_) in Python](https://www.geeksforgeeks.org/underscore-_-python/), [Private Variables in Python](https://www.geeksforgeeks.org/private-variables-python/)

Encapsulation is one of the four principles used in Object Oriented Paradigm. It is used to bind and hide data to the class. Data hiding is also referred as Scoping and the accessibility of a method or a field of a class can be changed by the developer. The implementation of scoping is different for different programming language. For example, statically typed, compiled language has direct support to scoping with the help of keywords which are mentioned when the method or field is declared. However Python does not have such keywords since it is a scripting language, and it is interpreted instead of being compiled. Mainly, Access Modifiers can be categorized as **Public**, **Protected** and **Private** in a class.

Python uses the ‘\_’ symbol to determine the access control for a specific data member or a member function of a class. Access specifiers in Python have an important role to play in securing data from unauthorized access and in preventing it from being exploited. But it is not like other languages like Java and C++ since Python uses the concept of Name Mangling for achieving data hiding.

A Class in Python has three types of access modifiers:

* **Public Access Modifier:**Theoretically, public methods and fields can be accessed directly by any class.
* **Protected Access Modifier:**Theoretically, protected methods and fields can be accessed within the same class it is declared and its subclass.
* **Private Access Modifier:**Theoretically, private methods and fields can be only accessed within the same class it is declared.

We are mentioning “Theoretically” because python doesn’t follow the textbook definition of such specifications. Instead, it depends on the programmer/organization as well as a unique feature of python called as name mangling using which we can mimic the actual security provided by access modifiers.

**Public Access Modifier:**

The members of a class that are declared public are easily accessible from any part of the program. All data members and member functions of a class are public by default.

Python

*# program to illustrate public access modifier in a class*

**class** **Student**:

*# constructor*

**def** \_\_init\_\_(self, name, age):

*# public data members*

self.sName = name

self.sAge = age

*# public member function*

**def** displayAge(self):

*# accessing public data member*

print("Age: ", self.sAge)

*# creating object of the class*

obj = Student("Ron", 20)

*# finding all the fields and methods which are present inside obj*

print("List of fields and methods inside obj:", dir(obj))

*# accessing public data member*

print("Name:", obj.sName)

*# calling public member function of the class*

obj.displayAge()

**Output**

List of fields and methods inside obj: ['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'displayAge', 'sAge', 'sName']  
Name: Ron  
Age: 20

We are using dir() function to list down all the member variables and functions of the Students object which can be accessed. We can clearly see sName, sAge, displayAge and other inbuilt methods such as \_\_str\_\_, \_\_sizeof\_\_, etc. In the above program, sName and sAge are supposed to be public data members and displayAge() method is a public member function of the class Student. These data members of the class Student can be accessed from anywhere in the program since they are present in the list returned by dir() as it is.

**Protected Access Modifier:**

The members of a class that are declared protected are only accessible within the class where it is declared and its subclass. To implement protected field or method, the developer follows a specific convention mostly by adding prefix to the variable or function name. Popularly, a single underscore “\_” is used to describe a protected data member or method of the class. Note that the python interpreter does not treat it as protected data like other languages, it is only denoted for the programmers since they would be trying to access it using plain name instead of calling it using the respective prefix. For example,

Python

*# program to illustrate protected access modifier in a class*

*# super class*

**class** **Student**:

*# protected data members*

\_name = **None**

\_roll = **None**

\_branch = **None**

*# constructor*

**def** \_\_init\_\_(self, name, roll, branch):

self.\_name = name

self.\_roll = roll

self.\_branch = branch

*# protected member function*

**def** \_displayRollAndBranch(self):

*# accessing protected data members*

print("Roll:", self.\_roll)

print("Branch:", self.\_branch)

*# derived class*

**class** **Geek**(Student):

*# constructor*

**def** \_\_init\_\_(self, name, roll, branch):

Student.\_\_init\_\_(self, name, roll, branch)

*# public member function*

**def** displayDetails(self):

*# accessing protected data members of super class*

print("Name:", self.\_name)

*# accessing protected member functions of super class*

self.\_displayRollAndBranch()

stu = Student("Alpha", 1234567, "Computer Science")

print(dir(stu))

*# protected members and methods can be still accessed*

print(stu.\_name)

stu.\_displayRollAndBranch()

*# Throws error*

*# print(stu.name)*

*# stu.displayRollAndBranch()*

*# creating objects of the derived class*

obj = Geek("R2J", 1706256, "Information Technology")

print("")

print(dir(obj))

*# calling public member functions of the class*

obj.displayDetails()

**Output**

['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_  
\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_branch', '\_displayRollAndBranch', '\_name', '\_roll']  
Alpha  
Roll: 1234567  
Branch: Computer Science  
  
['\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', '\_branch', '\_displayRollAndBranch', '\_name', '\_roll', 'displayDetails']  
Name: R2J  
Roll: 1706256  
Branch: Information Technology

In the above program, \_name, \_roll, and \_branch are protected data members and \_displayRollAndBranch() method is a protected method of the super class Student. The displayDetails() method is a public member function of the class Geek which is derived from the Student class, the displayDetails() method in Geek class accesses the protected data members of the Student class.

However, we can still access protected members of Student class directly by specifying the correct name of field and method i.e. adding underscore before them since it was declared by that name. We can also see that these declared fields and methods can be called since they are present in the list returned by the dir() function. If we try to access the using plain names such as stu.name and stu.displayRollAndBranch(), we get error since they are not saved by that name. Underscores are mainly used since other characters like “$”, “-“, “&”, etc. cannot be present in variable or function name.

**Private Access Modifier:**

The members of a class that are declared private are accessible within the class only, private access modifier is the most secure access modifier. Data members of a class are declared private by adding a double underscore ‘\_\_’ symbol before the data member of that class.

Python

*# program to illustrate private access modifier in a class*

**class** **Geek**:

*# private members*

\_\_name = **None**

\_\_roll = **None**

\_\_branch = **None**

*# constructor*

**def** \_\_init\_\_(self, name, roll, branch):

self.\_\_name = name

self.\_\_roll = roll

self.\_\_branch = branch

*# private member function*

**def** \_\_displayDetails(self):

*# accessing private data members*

print("Name:", self.\_\_name)

print("Roll:", self.\_\_roll)

print("Branch:", self.\_\_branch)

*# public member function*

**def** accessPrivateFunction(self):

*# accessing private member function*

self.\_\_displayDetails()

*# creating object*

obj = Geek("R2J", 1706256, "Information Technology")

print(dir(obj))

print("")

*# Throws error*

*# obj.\_\_name*

*# obj.\_\_roll*

*# obj.\_\_branch*

*# obj.\_\_displayDetails()*

*# To access private members of a class*

print(obj.\_Geek\_\_name)

print(obj.\_Geek\_\_roll)

print(obj.\_Geek\_\_branch)

obj.\_Geek\_\_displayDetails()

print("")

*# calling public member function of the class*

obj.accessPrivateFunction()

**Output**

'\_Geek\_\_branch', '\_Geek\_\_displayDetails', '\_Geek\_\_name', '\_Geek\_\_roll', '\_\_class\_\_', '\_\_delattr\_\_', '\_\_dict\_\_', '\_\_dir\_\_', '\_\_doc\_\_', '\_\_eq\_\_', '\_\_format\_\_', '\_\_ge\_\_', '\_\_getattribute\_\_', '\_\_getstate\_\_', '\_\_gt\_\_', '\_\_hash\_\_', '\_\_init\_\_', '\_\_init\_subclass\_\_', '\_\_le\_\_', '\_\_lt\_\_', '\_\_module\_\_', '\_\_ne\_\_', '\_\_new\_\_', '\_\_reduce\_\_', '\_\_reduce\_ex\_\_', '\_\_repr\_\_', '\_\_setattr\_\_', '\_\_sizeof\_\_', '\_\_str\_\_', '\_\_subclasshook\_\_', '\_\_weakref\_\_', 'accessPrivateFunction']  
  
R2J  
1706256  
Information Technology  
Name: R2J  
Roll: 1706256  
Branch: Information Technology  
  
Name: R2J  
Roll: 1706256  
Branch: Information Technology

In the above program, \_\_name, \_\_roll and \_\_branch are private members, \_\_displayDetails() method is a private member function (these can only be accessed within the class) and accessPrivateFunction() method is a public member function of the class Geek which can be accessed from anywhere within the program. The accessPrivateFunction() method accesses the private members of the class Geek.

However, we can still access private members of a class outside the class. We cannot directly call obj.\_\_name, obj.\_\_age, obj.\_\_branch, and obj.\_\_displayDetails() because they throw errors. We can notice that in the list of callable fields and methods, \_\_name is saved as \_Geek\_\_name, \_\_age is saved as \_Geek\_\_age, \_\_branch is saved as \_Geek\_\_branch and \_\_displayDetails() is saved as \_Geek\_\_displayDetails(). This conversion is called as name mangling, where the python interpreter automatically converts any member preceded with two underscores to \_<class name>\_\_<member name>. Hence, we can still call all the supposedly private data members of a class using the above convention.

**Below is a program to illustrate the use of all the above three access modifiers (public, protected**,**and private) of a class in Python:**

Python

*# program to illustrate access modifiers of a class*

*# super class*

**class** **Super**:

*# public data member*

var1 = **None**

*# protected data member*

\_var2 = **None**

*# private data member*

\_\_var3 = **None**

*# constructor*

**def** \_\_init\_\_(self, var1, var2, var3):

self.var1 = var1

self.\_var2 = var2

self.\_\_var3 = var3

*# public member function*

**def** displayPublicMembers(self):

*# accessing public data members*

print("Public Data Member:", self.var1)

*# protected member function*

**def** \_displayProtectedMembers(self):

*# accessing protected data members*

print("Protected Data Member:", self.\_var2)

*# private member function*

**def** \_\_displayPrivateMembers(self):

*# accessing private data members*

print("Private Data Member:", self.\_\_var3)

*# public member function*

**def** accessPrivateMembers(self):

*# accessing private member function*

self.\_\_displayPrivateMembers()

*# derived class*

**class** **Sub**(Super):

*# constructor*

**def** \_\_init\_\_(self, var1, var2, var3):

Super.\_\_init\_\_(self, var1, var2, var3)

*# public member function*

**def** accessProtectedMembers(self):

*# accessing protected member functions of super class*

self.\_displayProtectedMembers()

*# creating objects of the derived class*

obj = Sub("Geeks", 4, "Geeks!")

*# calling public member functions of the class*

obj.displayPublicMembers()

obj.accessProtectedMembers()

obj.accessPrivateMembers()

print()

*# Can also be accessed using*

obj.\_displayProtectedMembers()

obj.\_Super\_\_displayPrivateMembers()

print()

*# Object can access protected member*

print("Object is accessing protected member:", obj.\_var2)

print("Object is accessing private member:", obj.\_Super\_\_var3)

*# object can not access private member, so it will generate Attribute error*

*# print(obj.\_\_var3)*

**Output**

Public Data Member: Geeks  
Protected Data Member: 4  
Private Data Member: Geeks!  
  
Protected Data Member: 4  
Private Data Member: Geeks!  
  
Object is accessing protected member: 4  
Object is accessing private member: Geeks!

In the above program, the accessProtectedMembers() method is a public member function of the class *Sub* accesses the \_displayProtectedMembers() method which is protected member function of the class Super and the accessPrivateMembers() method is a public member function of the class Super which accesses the \_\_displayPrivateMembers() method which is a private member function of the class Super. Also note that all these access modifiers are not strict like other languages such as C++, Java, C#, etc. since they can still be accessed if they are called by their original or mangled names