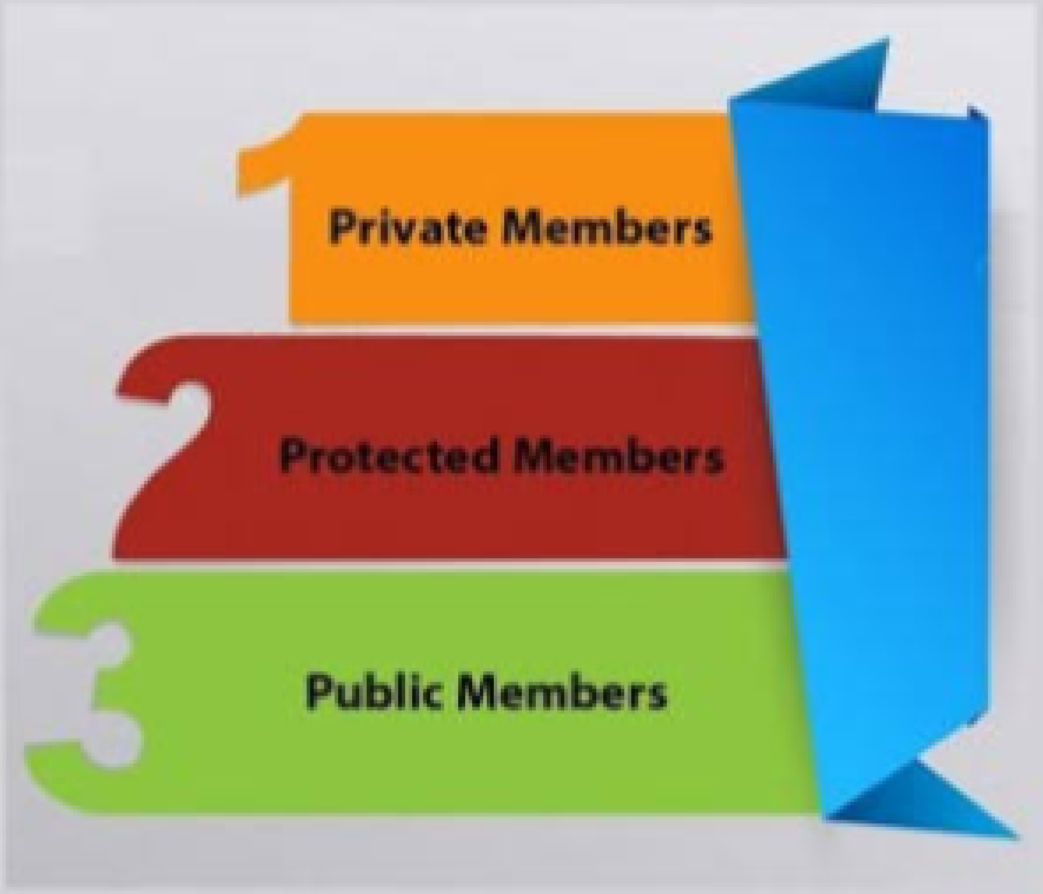
Python

Python Programming

Access Modifiers



Access Modifiers in Python : Public, Private and Protected

Prerequisites: Underscore (_) in Python, Private Variables in Python

Various object-oriented languages like C++, Java, Python control access modifications which are used to restrict access to the variables and methods of the class. Most programming languages has three forms of access modifiers, which are **Public**, **Protected** and **Private** in a class.
Python uses '_' 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.
A Class in Python has three types of access modifiers:

- **Public Access Modifier**
- **Protected Access Modifier**
- **Private Access Modifier**

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

Python3

```
# program to illustrate public access modifier in a class
class Geek:

    # constructor
    def __init__(self, name, age):

        # public data members
        self.geekName = name
        self.geekAge = age
    # public member function
    def displayAge(self):

        # accessing public data member
        print("Age: ", self.geekAge)
# creating object of the class
obj = Geek("R2J", 20)
# accessing public data member
print("Name: ", obj.geekName)
# calling public member function of the class
obj.displayAge()
```

Output:

```
Name:  R2J
Age:  20
```

In the above program, geekName and geekAge are public data members and displayAge() method is a public member function of the class Geek. These data members of the class Geek can be accessed from anywhere in the program.

Protected Access Modifier:

The members of a class that are declared protected are only accessible to a class derived from it. Data members of a class are declared protected by adding a single underscore '_' symbol before the data member of that class.

Python3

```
# 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()

# creating objects of the derived class
obj = Geek("R2J", 1706256, "Information Technology")

# calling public member functions of the class
obj.displayDetails()
```

Output:

```
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.

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.

Python3

```
# 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")

# calling public member function of the class
obj.accessPrivateFunction()
```

Output:

```
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.

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

Python3

```
# 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()

# Object can access protected member
print("Object is accessing protected member:", obj._var2)

# 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 !
```

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.