Access Modifiers in Python

- Access modifiers are used to limit the access to the variables and functions of a class. Most of the languages use three types of access modifiers, they are - **private**, **public** and **protected**.
- Python makes the use of underscores (_) to specify the access modifier for a specific data member and method in a class.
- Access modifiers play an important role to protect the data from unauthorized access as well as protecting it from getting manipulated.
- When inheritance is implemented there is a huge risk for the data to get modified during the transfer of unwanted data from the parent class to the child class. Thus, it is required to give the right access modifiers for different data members and member functions depending upon the requirements.

Access Modifiers in Python

- In fact, when we didn't use any underscore with members names, they defaulted to be public in Python. Means, those can be accessed from anywhere.
- The members declared as **Public** are accessible from outside the Class through an object of the class.

```
class Account:
    def __init__(self,acc_no,acc_holder,bal):
        self.acc_no=acc_no
        self.acc_holder=acc_holder
        self.bal=bal
    def details(self):
        print('Account Number: ',self.acc_no)
        print('Account Holder: ',self.acc_holder)
        print('Account Balance: ',self.bal)

a = Account(12345,'Sunil Kumar',6500.35)
a.bal=6585.75
a.details()
```

Restricting the Access - Protected

- The members declared as **Protected** are accessible from outside the class but only in a class derived from it that is in the child or subclass.
- adding a prefix _(single underscore) to a variable name makes it protected. No additional keyword is required.

```
class Account:

    def __init__(self,acc_no,acc_holder,bal):
        self._acc_no=acc_no
        self._acc_holder=acc_holder
        self._bal=bal
    def details(self):
        print('Account Number: ',self._acc_no)
        print('Account Holder: ',self._acc_holder)
        print('Account Balance: ',self._bal)
```

```
Traceback (most recent call last):
  File "C:/User/Python/Python37/oop7.py", line 12, in <
    print(a.bal)
AttributeError: 'Account' object has no attribute 'bal'
```

- we have made the instance variables protected by prefixing an _(underscore), so now we can access them as follows:

```
a = Account(12345,'Sunil Kumar',6500.35)
print(a._bal)
a.details()
```

If you use print(a.bal)

Restricting the Access - Protected

- Similarly if there is a child class extending the class Account then it can also access the protected member variables of the class Account.

```
class Account:
    def __init__(self,acc_no,acc_holder,bal):
        self._acc_no=acc_no
        self._acc_holder=acc_holder
        self._bal=bal
    def details(self):
        print('Account Number: ',self._acc_no)
        print('Account Holder: ',self._acc_holder)
        print('Account Balance: ',self._bal)
```

Output:

```
Account Number:  12345
Account Holder:  Sunil Kumar
Account Balance:  6500.35
Rate of Interest:  3.5
```

Being a subclass, SavingAccount can access the Protected members of the base class Account.

```
from oop7 import Account
class SavingAccount(Account):
    def details(self,roi):
        self.roi=3.5
        print('Account Number: ',self._acc_no)
        print('Account Holder: ',self._acc_holder)
        print('Account Balance: ',self._bal)
        print('Rate of Interest: ',self.roi)
a = SavingAccount(12345,'Sunil Kumar',6500.35)
a.details(3.5)
```

Restricting the Access - Private

- The members declared as **Private** are only accessible from within the class. No outside Access is allowed.
- adding a prefix __(double underscore) to a variable name makes it private.

```
class Account:

    def __init__(self,acc_no,acc_holder,bal):
        self.__acc_no=acc_no
        self.__acc_holder=acc_holder
        self.__bal=bal
    def details(self):
        print('Account Number: ',self.__acc_no)
        print('Account Holder: ',self.__acc_holder)
        print('Account Balance: ',self.__bal)
a = Account(12345,'Sunil Kumar',6500.35)
a.details()
a.__bal=6585.75
```

- Last line will give an error because there is no outside access to any attributes in Account.