Data Abstraction:

==========

> It is the process of hiding the data
> To improve security for the data.

In Python Data abstraction is possible by using the following modifiers

1.private

1.The members are which are declared by using __ then those members are considered by PVM as private members.

2.private members can be accessed only with in the same class.

3.private members are not inherited

4.private members can't be accessed outside of the class.

Example 1:

```
class SuperA:
    def __init__(self):
        self.__x=10 #private member

    def method1(self):
        print("mtd-1 of SuperA ")
        print("private x : ",self.__x)

class SubB(SuperA):
    def method2(self):
        print("mtd-2 of SubB")
        print("private x : ",self.__x)
```

```
#calling
s=SubB()
s.method1()
s.method2()

Example :2
class Super:
   def __init__(self):
      self.__x=10 #private members

   def method1(self): #instance mtd
      print("Mtd-1 of Super ")
      print("private members x val is : ",self.__x)

#calling
s=Super()
s.method1()
'''we can access the Instance fields outside of the class
by using an object reference '''

print("From outside of the class ")
print("private x val is : ",s.__x)

==========================================
```

2.protected

> The attribute which are declared by using single _
called protected members
Eg: _x=10 #protected

> The protected members are inherited
> The protected members are can't be accessed from
outside of the class.

Example 1:

```
class SuperA:
   def __init__(self):
      self._x=10 #protected

   def method1(self):
      print("Mtd-1 of SuperA")
      print("protected x val is : ",self._x)

#calling
s=SuperA( )
s.method1()
```

Example 2:

```
class SuperA:
   def __init__(self):
      self._x=10

   def method1(self):
      print("Ins mtd-1 of SuperA")
      print("protected x val is : ",self._x)
```

```
class SubB(SuperA):
   def method2(self):
      print("Ins mtd-2 of SubB ")
      print("protected x val is : ",self._x)

#calling
s=SubB()
s.method1()
s.method2()
```

==========

3.public

> If any members are declared without any underscorll then they considered by PVM as "public" members

> by default all the members are public only.

> public members can be used with in the same class.

> public members are inherited

> public members can be accessed outside of the class.

Example_1:

```
class Super:
   def __init__(self):
      self.x=10  #public

   def method1(self):
      print("Ins method-1 of Super ")
      print("public x val is : ",self.x)

#calling
```

```
s=Super()
s.method1()

print("From outside of the class ")
print("public x val is : ",s.x)
```

Example 2:

```
class SuperA:
   def __init__(self):
      self.x=10

   def method1(self):
      print("Mtd-1 of SuperA")
      print("public x val is : ",self.x)

class SubB(SuperA):
   def method2(self):
      print("Mtd-2 of SubB")
      print("public x val is : ",self.x)

#calling
sb=SubB()
sb.method1()
sb.method2()
```

Note:

1.public members can be accessed outside of the class.
2.protected members are also accessed
  from outside of the class.
3.private members are also accessed from outside
of the class.

```
class Sample:
   def __init__(self):
      self.__x=10 #private
      self._y=20   # protected
      self.z=30    #public

   def method1(self):
      print("Mtd-1 of Sample ")
      print("private x : ",self.__x)
      print("protected y : ",self._y)
      print("public z val is : ",self.z)


#calling
s=Sample()
s.method1()

print("============================")
print(" From outside of the class")
print(" public z val is  : ",s.z)
print(" protected y val is : ",s._y)
print(" private x val is : ",s._Sample__x)
```

We can access private members outside of the class by using the following Syntax.

#Syn: objectreference._classname__privatemember