Inheritance

Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object's variable can only be changed by an object's method. Those types of variables are known as private variables. A class is an example of encapsulation as it encapsulates all the data that is member functions, variables, etc. Consider a real-life example of encapsulation, in a company, there are different sections like the accounts section, finance section, sales section etc. The finance section handles all the financial transactions and keeps records of all the data related to finance. Similarly, the sales section handles all the sales-related activities and keeps records of all the sales. Now there may arise a situation when for some reason an official from the finance section needs all the data about sales in a particular month. In this case, he is not allowed to directly access the data of the sales section. He will first have to contact some other officer in the sales section and then request him to give the particular data. This is what encapsulation is. Here the data of the sales section and the employees that can manipulate them are wrapped under a single name "sales section". Using encapsulation also hides the data. In this example, the data of the sections like sales, finance, or accounts are hidden from any other section.	Kapsülleme, nesne yönelimli programlamanın (OOP) temel kavramlarından biridir. Verileri sarma fikrini ve tek bir birim içindeki veriler üzerinde çalışan yöntemleri açıklar. Bu, değişkenlere ve yöntemlere doğrudan erişime kısıtlamalar getirir ve verilerin yanlışlıkla değiştirilmesini önleyebilir. Kazara değişikliği önlemek için, bir nesnenin değişkeni yalnızca bir nesnenin yöntemiyle değiştirilebilir. Bu tür değişkenler özel değişkenler olarak bilinir. Bir sınıf, üye işlevler, değişkenler vb. olan tüm verileri kapsüllediği için bir kapsülleme örneğidir. Gerçek hayattan bir kapsülleme örneğini ele alalım, bir şirkette hesaplar bölümü, finans bölümü, satış bölümü vb. Aynı şekilde satış bölümü de satışla ilgili tüm faaliyetleri yürütür ve tüm satışların kaydını tutar. Şimdi, herhangi bir nedenle finans bölümünden bir yetkilinin belirli bir aydaki satışlarla ilgili tüm verilere ihtiyaç duyduğu bir durum ortaya çıkabilir. Bu durumda, satış bölümünün verilerine doğrudan erişmesine izin verilmez. Önce satış bölümünde başka bir memurla iletişime geçmesi ve ardından ondan belirli verileri vermesini istemesi gerekecektir. Kapsülleme budur. Burada satış bölümü verileri ve bunları manipüle edebilen çalışanlar tek bir isim altında "satış bölümü" altında toplanmıştır. Kapsülleme kullanmak da verileri gizler. Bu örnekte satış, finans veya hesaplar gibi bölümlerin verileri diğer bölümlerden gizlenmiştir.

Protected members

Protected members (in C++ and JAVA) are those members of the class that cannot be accessed outside the class but can be accessed from within the class and its subclasses. To accomplish this in Python, just follow the convention by prefixing the name of the member by a single underscore "_". Although the protected variable can be accessed out of the class as well as in the derived class(modified too in derived class), it is customary(convention not a rule) to not access the protected out the class body. Note: The __init__ method is a constructor and runs as soon as an object of a class is instantiated.	Korumalı üyeler (C++ ve JAVA'da), sınıfın dışından erişilemeyen ancak sınıfın ve alt sınıflarının içinden erişilebilen sınıfın üyeleridir. Python'da bunu başarmak için , üyenin adının önüne tek bir alt çizgi "_" koyarak kuralı takip etmeniz yeterlidir . Korumalı değişkene, türetilmiş sınıfta olduğu gibi sınıfın dışında da erişilebilmesine rağmen (türetilmiş sınıfta da değiştirilebilir), korunan sınıf gövdesine erişmemek alışılmış bir durumdur (gelenek değil kural). Not: __init__ yöntemi bir yapıcıdır ve bir sınıfın nesnesi başlatılır başlatılmaz çalışır.

```python
# Python program to
# demonstrate protected members

# Creating a base class
class Base:
    def __init__(self):

        # Protected member
        self._a = 2

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling protected member of base class: ",
              self._a)

        # Modify the protected variable:
        self._a = 3
        print("Calling modified protected member outside class: ",
              self._a)


obj1 = Derived()

obj2 = Base()

# Calling protected member
# Can be accessed but should not be done due to convention
print("Accessing protected member of obj1: ", obj1._a)

# Accessing the protected variable outside
print("Accessing protected member of obj2: ", obj2._a)
```

```
Calling protected member of base class:  2
Calling modified protected member outside class:  3
Accessing protected member of obj1:  3
Accessing protected member of obj2:  2
```

Private members

Private members are similar to protected members, the difference is that the class members declared private should neither be accessed outside the class nor by any base class. In Python, there is no existence of Private instance variables that cannot be accessed except inside a class. However, to define a private member prefix the member name with double underscore “__”. Note: Python’s private and protected members can be accessed outside the class through python name mangling.	Özel üyeler, korumalı üyelere benzer, fark, özel olarak bildirilen sınıf üyelerine, sınıfın dışından veya herhangi bir temel sınıf tarafından erişilmemesidir. Python'da, bir sınıf dışında erişilemeyen Özel örnek değişkenleri yoktur. Ancak, özel bir üye tanımlamak için üye adının ön ekini çift alt çizgi “__” ile yapın. Not: Python'un özel ve korumalı üyelerine sınıf dışından, python name mangling aracılığıyla erişilebilir .

```python
# Python program to
# demonstrate private members

# Creating a Base class


class Base:
    def __init__(self):
        self.a = "GeeksforGeeks"
        self.__c = "GeeksforGeeks"

# Creating a derived class
class Derived(Base):
    def __init__(self):

        # Calling constructor of
        # Base class
        Base.__init__(self)
        print("Calling private member of base class: ")
        print(self.__c)


# Driver code
obj1 = Base()
print(obj1.a)

# Uncommenting print(obj1.c) will
# raise an AttributeError

# Uncommenting obj2 = Derived() will
# also raise an AtrributeError as
# private member of base class
# is called inside derived class
```

```
Traceback (most recent call last):
  File "/home/f4905b43bfcf29567e360c709d3c52bd.py", line 25, in <module>
    print(obj1.c)
AttributeError: 'Base' object has no attribute 'c'

Traceback (most recent call last):
  File "/home/4d97a4efe3ea68e55f48f1e7c7ed39cf.py", line 27, in <module>
    obj2 = Derived()
  File "/home/4d97a4efe3ea68e55f48f1e7c7ed39cf.py", line 20, in __init__
    print(self.__c)
AttributeError: 'Derived' object has no attribute '_Derived__c'
```