jyakgohvr

May 12, 2025

[3]:
```
# Encapsulation in python
# Access modifier
# public private protected
# obj=Ajit("aman",10,"sonipat")
# obj1=Ajit("anushka",19,"delhi")


# obj=Ajit("aman",10,"sonipat")
# get_data(value)
# obj.get_data(value)
# obj.set_data("akash")
# set_data("19")
# # data hiding
# name="anushka"#public-----it can be used inside or outside the class
# _salary=salary#protected----it can be used inside the class or member of the␣
 ↪class we use _ to define protected method
# __pan_card=pan_card#private---private can be used only within the class and␣
 ↪with the help of class name we use __ to define private
```

[9]:
```
# Public access modifier
class Example:
    def __init__(self,Name):
        self.Name=Name#public
    def Info(self):
        print(f"my name is {self.Name}")
obj=Example("mansi")
print(obj.Name)
```

```
mansi
```

[11]:
```
#Protected access modifier
class Example:
    def __init__(self,Name):
        self._Name=Name#Protected
    def Info(self):
        print(f"my name is {self.Name}")
obj=Example("mansi")
```

```
print(obj._Name)
```

```
mansi
```

[5]:
```
class Car:
    def __init__(self,make,model,year,price):
        self.make=make#public
        self._model=model#protected
        self.year=year
        self.__price=price#price is private
#now define getter method to get the values
    def get_make(self):
        return self.make
    def _get_model(self):
        return self._model
    def get_year(self):
        return self.year
    def __get_price(self):
        return self.__price
# Now define setter mathed tp set the values
    def set_model(self,make):
        self.make=make
    def set_model(self,model):
        self._model=model
    def set_year(self,year):
        self.year=year
    def set_price(self,price):
        self.__price=price
obj=Car("maruti","Brezza",2019,2300000)
print(obj.get_make())
print(obj._get_model())
print(obj._Car__get_price())
obj.set_model("Tyota")
obj.set_year(2008)
print(obj._get_model())
print(obj.get_year())
```

```
maruti
Brezza
2300000
Tyota
2008
```

[35]:
```
#Protected access modifier
class Example:
    def __init__(self,Name):
        self.__Name=Name#Protected
    def __Info(self):
        print(f"my name is {self.__Name}")
obj=Example("mansi")
print(obj._Example__Name)
obj._Example__Info()
obj._Example__Info()
```

```
mansi
my name is mansi
my name is mansi
```

[15]:
```
class Details:
    def __init__(self,name,dept,salary):
        self.name=name#public
        self._dept=dept#protected
        self.__salary=salary#private
# lets get some value with the help of getter
    def get_name(self):
        return self.name
    def _get_dept(self):
        return self._dept
    def __get_salary(self):
        return self.__salary
#lets set some new value with the help of setter
    def set_name(self,name):
        self.name=name
    def set_dept(self,dept):
        self._dept=dept
    def set_salary(self,salary):
        self.__salary=salary
obj=Details("ajit","AI",1000)
print(obj.get_name())
print(obj._get_dept())
print(obj._Details__get_salary())
# now set new values
obj.set_name("akansha ")
obj.set_dept("Front end")
obj.set_salary(2900)
print(obj.get_name())
print(obj._get_dept())
print(obj._Details__get_salary())
```

```
ajit
AI
1000
akansha
Front end
2900
```

[17]:
```
from abc import ABC , abstractmethod
```

[19]:
```
class Animal(ABC):
    @abstractmethod
    def make_sound(self):
        pass
class Dog(Animal):
    def make_sound(self):
        return "hello how are you"
class Cat(Animal):
    def make_sound(self):
        return "Hii how are you "
obj=Cat()
print(obj.make_sound())
obj1=Dog()
print(obj1.make_sound())
```

```
Hii how are you
hello how are you
```

[23]:
```
class BankAccount(ABC):
    @abstractmethod
    def Deposite(self,amount):
        pass
    def Withdraw(self,amount):
        pass
class Saving(BankAccount):
    def __init__(self,balance=0):
        self.balance=balance
    def Deposite(self,amount):
        self.balance+=amount
        print(f"amount deposited {amount}, new amount {self.balance}")
    def Withdraw(self,amount):
        self.balance-=amount
        print(f"amount withdraw {amount}, new amount {self.balance}")
account=Saving()
account.Deposite(100000)
account.Withdraw(999)
```

```
amount deposited 100000, new amount 100000
amount withdraw 999, new amount 99001
```

[ ]:
```

```