"Abstraction in Python"

Abstraction is an important principle of object-oriented programming. It refers to a programming approach by which only the relevant data about an object is exposed, hiding all the other details. This approach helps in reducing the complexity & increasing the efficiency of application development.

"Types of Python Abstraction"

There are two types of abstraction.

1) Data abstraction :→ The original data entity is hidden via a data structure that can internally work through the hidden data entities

2) Process abstraction :→ It refers to hiding the underlying implementation details of a process.

A simple example of data abstraction can be a "Car". A car has an accelerator, clutch & brake, & we all know that pressing the accelerator will increase the speed of the car & applying the brakes can stop the car, but we do not know the internal mechanism of the car & how these functionalities can work. This detail hiding is known as data abstraction.

In python abstraction is achieved using abstract methods & abstract classes

(PTO)

Python Abstract class :⇒ In OOPs, a class is said to be an abstract class if it cannot be instantiated, that is you can not have an object of an abstract class. You can however use it as a base or parent class for constructing other classes.

Creating an abstract class :⇒

To create an abstract class in Python, it must inherit the ABC class defined in the abc module. This module is available in Python's standard library. Moreover, the class must have atleast one abstract method.

Note :⇒ An abstract method is the one which cannot be called but can be overridden. You need to decorate it with @abstractmethod decorator.

Eg :⇒
```
from abc import ABC, abstractmethod
class demo(ABC):
    @abstractmethod
    def method1(self)
      print("abstract method")
      return
    def method2(self):
      print("concrete method")
```

Note :⇒ The demo class inherits ABC class. There is a method1() which is an abstract method. Note that the class may have other non-abstract (concrete) methods.

Note :⇒ if you try to declare an object of demo class, Python raises TypeError.

The demo class here may be used as parent for another class. However, the child class must override the abstract method in parent class.

Eg:->

```
from abc import ABC, abstractmethod
class democlass(ABC):
    @abstractmethod
    def method1(self):
        print("abstract method")
        return
    def method2(self):
        print("Concrete method")
class concreteclass(democlass):
    def method1(self):
        super().method1()
        return

obj = concreteclass()
obj.method1()
obj.method2()
```

O/p
```
abstract method
Concrete method
```

Concrete method :-> It is a method defined in an abstract base class with their complete implementation. Concrete methods are required to avoid replication of code in subclasses. For eg: In an abstract base class there may be a method whose implementation is to be same in all its subclasses. So, we write the implementation of that method in abstract base class after which we do not need to write the implementation of the concrete method again & again in every subclass.

→ Implementation of Data Abstraction in Python :→

In the below code, we have implemented data abstraction using abstract class & method. Firstly, we import the required modules or classes from abc library then we create a base class 'Car' that inherited from 'ABC' class that we have imported.

Inside base class. we create init function, abstract function & non-abstract functions. To declare abstract function printDetails we use @abstractmethod. decorator. After that we create child class hatchback and suv. Since, these child classes inherited from abstract class so, we need to write the implementation of all abstract functions declared in the base class. We write the implementation of abstract method in both child classes. We create an instance of a child class & call the printDetails method. In this way we can achieve the data abstraction.

```
from abc import ABC, abstractmethod
class Car(ABC):
    def __init__(self, brand, model, year):
        self.brand = brand
        self.model = model
        self.year = year

    @abstractmethod
    def printDetails(self):
        pass

    def accelerate(self):
        print("Speed up ---")
```

[P.T.O]

```
    def break_applied(self):
        print("Car stopped")

class Hatchback(car):
    def printDetails(self):
        print("Brand:", self.brand)
        print('Model:", self.model)
        print("year:", self.year)

    def sunroof(self):
        print("Not having this feature")

class Suv(car):
    def printDetails(self):
        print("Brand:", self.brand)
        print("model:", self.model)
        print("Year:", self.year)

    def sunroof(self):
        print("Available")

car1 = Hatchback("Maruti", "Alto", "2022")
car1.printDetails()
car1.accelerate()
car1.sunroof()
```

O/p:→ Brand: Maruti
Model: Alto
year: 2022
Speed up ---
Not having this feature.