Abstraction in Python

Abstraction is used to hide the internal functionality of the function from the users. The users only interact with the basic implementation of the function, but inner working is hidden. User is familiar with that **"what function does"** but they don't know **"how it does."**

In simple words, we all use the smartphone and very much familiar with its functions such as camera, voice-recorder, call-dialing, etc., but we don't know how these operations are happening in the background. Let's take another example - When we use the TV remote to increase the volume. We don't know how pressing a key increases the volume of the TV. We only know to press the "+" button to increase the volume.

Why Abstraction is Important?

In Python, an abstraction is used to hide the irrelevant data/class in order to reduce the complexity. It also enhances the application efficiency.

Abstraction classes in Python

In Python, abstraction can be achieved by using abstract classes and interfaces.

A class that consists of one or more abstract method is called the abstract class. Abstract methods do not contain their implementation. Abstract class can be inherited by the subclass and abstract method gets its definition in the subclass. Abstraction classes are meant to be the blueprint of the other class. An abstract class can be useful when we are designing large functions. An abstract class is also helpful to provide the standard interface for different implementations of components. Python provides the **abc** module to use the abstraction in the Python program. Let's see the following syntax.

Syntax

```
1. from abc import ABC
2. class ClassName(ABC):
```

Working of the Abstract Classes

Unlike the other high-level language, Python doesn't provide the abstract class itself. We need to import the abc module, which provides the base for defining Abstract Base classes (ABC). The ABC works by decorating methods of the base class as abstract. It registers concrete classes as the implementation of the abstract base. We use the ***@abstractmethod*** decorator to define an abstract method or if we don't provide the definition to the method, it automatically becomes the abstract method. Let's understand the following example.

Example -

```
# Python program demonstrate  abstract base class work
from abc import ABC, abstractmethod
class Car(ABC):
def mileage(self):
pass
class Tesla(Car):
def mileage(self):
print("The mileage is 30kmph")
class Suzuki(Car):
def mileage(self):
print("The mileage is 25kmph ")
class Duster(Car):
def mileage(self):
print("The mileage is 24kmph ")
class Renault(Car):
def mileage(self):
print("The mileage is 27kmph ")
t= Tesla ()
t.mileage()
r = Renault()
r.mileage()
s = Suzuki()
s.mileage()
d = Duster()
d.mileage()
```

```
# Python program to define
# abstract class

from abc import ABC

class Polygon(ABC):

  # abstract method
  def sides(self):
    pass

class Triangle(Polygon):


  def sides(self):
    print("Triangle has 3 sides")

class Pentagon(Polygon):


  def sides(self):
    print("Pentagon has 5 sides")

class Hexagon(Polygon):

  def sides(self):
    print("Hexagon has 6 sides")

class square(Polygon):

  def sides(self):
    print("I have 4 sides")
```

```
t = Triangle()
t.sides()

s = square()
s.sides()

p = Pentagon()
p.sides()

k = Hexagon()
K.sides()
```

Points to Remember

Below are the points which we should remember about the abstract base class in Python.

- An Abstract class can contain the both method normal and abstract method.
- An Abstract cannot be instantiated; we cannot create objects for the abstract class.

Abstraction is essential to hide the core functionality from the users. We have covered the all the basic concepts of Abstraction in Python.