url = <https://www.scaler.com/topics/python/data-abstraction-in-python/>

**Introduction**

We use television to watch shows, news or movies, etc. We use the TV remote to switch the TV ON or OFF, switch to different channels, and raise or lower the volume. The TV user only knows he/she may use the buttons on the remote to do it. What they don’t know is how all this is happening internally, for example how the TV sensor is capturing signals from the TV remote and then how it is processing the received signals to perform the required action of changing the channel, etc. All the internal functionality is hidden, as for the user it might not be necessary for them to know how that is happening.

The example we saw above is one of the examples of abstraction in real life. In object-oriented programming, we shall call it ‘Data Abstraction’. Let us define data abstraction:

**The process by which data and functions are defined in such a way that only essential details can be seen and unnecessary implementations are hidden is called Data Abstraction.**

The main focus of data abstraction is to separate the interface and the implementation of the program.

**Data Abstraction in OOP**

Abstraction is really powerful for making complex tasks and codes simpler when used in Object-Oriented Programming. It reduces the complexity for the user by making the relevant part accessible and usable leaving the unnecessary code hidden. Also, there are times when we do not want to give out sensitive parts of our code implementation and this is where data abstraction can also prove to be very functional.

From a programmer’s perspective, if we think about data abstraction, there is more to it than just hiding unnecessary information. One other way to think of abstraction is as synonymous with generalization. If, for instance, you wanted to create a program to multiply eight times seven, you wouldn't build an application to only multiply those two numbers.

Instead, you'd create a program capable of multiplying any two numbers. To put it another way, abstraction is a way of thinking about a function's specific use as separate from its more generalized purpose. Thinking this way lets you create flexible, scalable, and adaptable functions and programs. You’ll get a better understanding of data abstraction and it’s purposes by the end of this article.

**Data Abstraction in Python**

Data Abstraction in Python can be achieved through creating abstract classes and inheriting them later. Before discussing what abstract classes are, let us have a brief introduction of inheritance.

Inheritance in OOP is a way through which one class inherits the attributes and methods of another class. The class whose properties and methods are inherited is known as the Parent class. And the class that inherits the properties from the parent class is the Child class/subclass.

The basic syntax to implement inheritance in Python is:

class parent\_class:

body of parent class

class child\_class( parent\_class):

body of child class

Let us now talk about abstract classes in python:

**Abstract Classes in Python**

**Abstract Class:** The classes that cannot be instantiated. This means that we cannot create objects of an abstract class and these are only meant to be inherited. Then an object of the derived class is used to access the features of the base class. These are specifically defined to lay a foundation of other classes that exhibit common behavior or characteristics.

The abstract class is an interface. Interfaces in OOP enable a class to inherit data and functions from a base class by extending it. In Python, we use the NotImplementedError to restrict the instantiation of a class. Any class having this error inside method definitions cannot be instantiated.

We can understand that an abstract class just serves as a template for other classes by defining a list of methods that the classes must implement. To use such a class, we must derive them keeping in mind that we would either be using or overriding the features specified in that class.

Consider an example where we create an abstract class Fruit. We derive two classes Mango and Orange from the Fruit class that implement the methods defined in this class. Here the Fruit class is the parent abstract class and the classes Mango and Apple become the sub/child classes. We won’t be able to access the methods of the Fruit class by simply creating an object, we will have to create the objects of the derived classes: Mango and Apple to access the methods.

**Why use Abstract Base Class?**

Defining an Abstract Base Class lets us create a common Application Programming Interface (API) for multiple subclasses. It is useful while working in large teams and code-bases so that all of the classes need not be remembered and also be provided as library by third parties.

**Working of Abstract Class**

Unlike other high-level programming languages, Python does not provide the abstract class itself. To achieve that, we use the abc module of Python, which provides the base and necessary tools for defining Abstract Base Classes (ABC). ABCs give the feature of virtual subclasses, which are classes that don’t inherit from a class and can still be recognized by

isinstance()

and

issubclass()

We can create our own ABCs with this module.

from abc import ABC

class MyABC(ABC):

pass

The abc module provides the metaclass ABCMeta for defining ABCs and a helper class ABC to alternatively define ABCs through inheritance. The abc module also provides the @abstractmethod decorator for indicating abstract methods.

ABC is defined in a way that the abstract methods in the base class are created by decorating with the @abstractmethod keyword and the concrete methods are registered as implementations of the base class.

**Concrete Methods in Abstract Base Class in Python**

We now know that we use abstract classes as a template for other similarly characterized classes. Using this, we can define a structure, but do not need to provide complete implementation for every method, such as:

from abc import ABC, abstractmethod

class MyClass(ABC):

@abstractmethod

def mymethod(self):

*#empty body*

pass

The methods where the implementation may vary for any other subclass are defined as abstract methods and need to be given an implementation in the subclass definition.

On the other hand, there are methods that have the same implementation for all subclasses as well. There are characteristics that exhibit the properties of the abstract class and so, must be implemented in the abstract class itself. Otherwise, it will lead to repetitive code in all the inherited classes. These methods are called concrete methods.

from abc import ABC, abstractmethod

class Parent(ABC):

*#common function*

def common\_fn(self):

print('In the common method of Parent')

@abstractmethod

def abs\_fn(self): *#is supposed to have different implementation in child classes*

pass

class Child1(Parent):

def abs\_fn(self):

print('In the abstract method of Child1')

class Child2(Parent):

def abs\_fn(self):

print('In the abstract method of Child2')

An abstract class can have both abstract methods and concrete methods.

We can now access the concrete method of the abstract class by instantiating an object of the child class. We can also access the abstract methods of the child classes with it. Interesting points to keep in mind are:

* We always need to provide an implementation of the abstract method in the child class even when implementation is given in the abstract class.
* A subclass must implement all abstract methods that are defined in the parent class otherwise it results in an error.

**Examples of Data Abstraction**

Let us take some examples to understand the working of abstract classes in Python. Consider the Animal parent class and other child classes derived from it.

from abc import ABC,abstractmethod

class Animal(ABC):

*#concrete method*

def sleep(self):

print("I am going to sleep in a while")

@abstractmethod

def sound(self):

print("This function is for defining the sound by any animal")

pass

class Snake(Animal):

def sound(self):

print("I can hiss")

class Dog(Animal):

def sound(self):

print("I can bark")

class Lion(Animal):

def sound(self):

print("I can roar")

class Cat(Animal):

def sound(self):

print("I can meow")

Our abstract base class has a concrete method sleep() that will be the same for all the child classes. So, we do not define it as an abstract method, thus saving us from code repetition. On the other hand, the sounds that animals make are all different. For that purpose, we defined the sound() method as an abstract method. We then implement it in all child classes.

Now, when we instantiate the child class object, we can access both the concrete and the abstract methods.

c = Cat()

c.sleep()

c.sound()

c = Snake()

c.sound()

This will give the output as:

I am going to sleep in a while

I can meow

I can hiss

Now, if we want to access the sound() function of the base class itself, we can use the object of the child class, but we will have to invoke it through super().

class Rabbit(Animal):

def sound(self):

super().sound()

print("I can squeak")

c = Rabbit()

c.sound()

This will produce the following output:

This function is for defining the sound by any animal

I can squeak

If we do not provide any implementation of an abstract method in the derived child class, an error is produced. Notice, even when we have given implementation of the sound() method in the base class, not providing an implementation in the child class will produce an error.

class Deer(Animal):

def sound(self):

pass

c = Deer()

c.sound()

c.sleep()

This will produce the following error:

Traceback (most recent call last):

File "F:/Python/Test/Parent.py", line 38, in <module>

c = Deer()

TypeError: Can't instantiate abstract class Deer with abstract methods sound

**NOTE:** Had there been more than one abstract method in the base class, all of them are required to be implemented in the child classes, otherwise, an error is produced.

**Why Data Abstraction is Important?**

Now that we know what Data Abstraction in Python is, we can also conclude how it is important.

Data Abstraction firstly saves a lot of our time as we do not have to repeat the code that may be the same for all the classes. Moreover, if there are any additional features, they can be easily added, thus improving flexibility. Not to mention, working in large teams becomes easier as one won’t have to remember every function and the basic structure can be inherited without any confusions.

**Conclusion**

Now that we have learned about Data Abstraction in Python, recall and tryto answer some questions for your better understanding:

* Try thinking of an example of ‘abstraction’ in your everyday life.
* Why do you think Data Abstraction can be useful?
* How can abstraction be achieved in Python?
* What are the few things to keep in mind about abstract and concrete methods while working with abstract classes?