OOPS - Abstraction

August 25, 2022

Data Abstraction In Python

What Is Meant By Abstraction ?

Abstraction aims to hide complexity from users and show them only relevant information..

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

For example, if you're driving a car, you don't need to know about its internal workings.

How abstraction can be achieved :

Python abstraction can be achieved through the use of abstract classes and methods in programs

Abstract Class in Python :

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

These are created to declare a set of methods that must be created in any child class built on top of abstract class.

These are specifically defined to lay a foundation of other classes that exhibit common behavior.

The abstract class is an interface which enable a sub class to inherit data and functions and extend it

Abstract class just serves as a template/blueprint for other classes

Abstract Method in Python :

Similarly, an abstract method is one that doesn't have any implementation.

Abstract methods force the child classes to give the implementation of these methods in them.

A class containing one or more abstract method is called an abstract class.

Benefits of Abstraction :

Hides the underlying complexity of data.

Helps avoid repetitive code.

Presents only the signature of internal functionality.

Gives flexibility to programmers to change the implementation of abstract behavior.

Abstract Base 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).

We can use the following syntax to create an abstract class in Python :

To define an abstract method we use the @abstractmethod decorator of the abc module.

We can use the following syntax to create an abstract method in Python :

[2]:
```
# let's take following example to demonstrate abstract classes:
from abc import ABC, abstractmethod
class DemoAbstractClass(ABC):
    @abstractmethod
    def abstract_method_name(self):
        Pass
```

Python Abstract Class Example :

[2]:
```
#Let's create a basic abstract class Shape which would contain the blueprint␣
 ↪for specific shapes
from abc import ABC, abstractmethod

class Shape(ABC):
    def __init__(self, shape_name):
        self.shape_name = shape_name

    @abstractmethod
    def draw(self):
        pass
```

[3]:
```
#Lets create a class Circle that will inherit the Shape class and implement the␣
 ↪draw() method.
class Circle(Shape):
    def __init__(self):
        super().__init__("circle")

    def draw(self):
        print("Drawing a Circle")

#create a circle object
circle = Circle()
circle.draw()
```

```
Drawing a Circle
```

[4]:
```
#Lets create a class Triangle that will inherit the Shape class and implement␣
 ↪the draw() method.
```

```
class Triangle(Shape):
    def __init__(self):
        super().__init__("triangle")
    def draw(self):
        print("Drawing a Triangle")

#create a triangle object
triangle = Triangle()
triangle.draw()
```

```
Drawing a Triangle
```

Importance of Abstract Classes :

The importance of using abstract class in Python is that if our subclasses don't follow that blueprint, Python will give an error.

Thus we can make sure that our classes follow the structure and implement all the abstract methods defined in our abstract class.

[6]:
```
#Let's see what happens if we do that
class Circle(Shape):
    def __init__(self):
        super().__init__("circle")

    def draw_circle(self):
        print("drawing circle")

#create a circle object
circle = Circle()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-6-f6cfba957473> in <module>
      8
      9 #create a circle object
---> 10 circle = Circle()

TypeError: Can't instantiate abstract class Circle with abstract methods draw
```

Abstract Properties

Abstract class in Python also provides the functionality of abstract properties in addition to abstract methods

The abc module has a @abstractproperty decorator to use abstract properties

[12]:
```
from abc import ABC, abstractmethod, abstractproperty

class Shape(ABC):
    def __init__(self, shape_name):
        self.shape_name = shape_name

    @abstractproperty
    def name(self):
        pass

    @abstractmethod
    def draw(self):
        pass
```

[14]:
```
class Circle(Shape):
    def __init__(self):
        super().__init__("circle")

    def draw_circle(self):
        print("drawing circle")

circle = Circle()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-14-4e8f9859ccea> in <module>
      6         print("drawing circle")
      7
----> 8 circle = Circle()

TypeError: Can't instantiate abstract class Circle with abstract methods draw,␣
 ↪name
```

[18]:
```
# Let's rectify this by defining the name property in the Circle class:
class Circle(Shape):
    def __init__(self):
        super().__init__("circle")
    @property
    def name(self):
        return self.shape_name
    def draw(self):
        print("Drawing a Circle")

circle = Circle()
print(f"The shape name is: {circle.name}")
```

```
The shape name is: circle
```

Abstract Class Instantiation

Abstract classes are not complete as they may have some methods that are not defined

So we cannot create an instance or object of an abstract class in Python.

[8]:
```
#Let's verify this by trying to instantiate our Shape class
try :
    shape = Shape()
except Exception as error:
    print("===========ERROR OCCURRED==============")
    print(error)
```

```
===========ERROR OCCURRED==============
Can't instantiate abstract class Shape with abstract methods draw
```

Invoke Methods from Abstract Classes

Unlike some other programming languages, Python abstract methods don't need to be completely abstract

They can have some basic level implementation that can be used by all the concrete classes.

A Concrete class is a class that has a definition for all its methods and has no abstract method.

[21]:
```
from abc import ABC, abstractmethod, abstractproperty
#Abstract Class
class Shape(ABC):
    def __init__(self, shape_name):
        self.shape_name = shape_name

    @abstractmethod
    def draw(self):
        print("Preparing the Canvas")

#Concret Class
class Circle(Shape):
    def __init__(self):
        super().__init__("circle")

    def draw(self):
        super().draw()
        print("Drawing a Circle")

#create a circle object
circle = Circle()
circle.draw()
```

```
Preparing the Canvas
Drawing a Circle
```

Concrete Methods in Abstract Base Classes

The abstract classes may also contain concrete methods that have the implementation of the method and can be used by all the concrete classes.

To define a concrete method in an abstract class, we simply define a method with implementation and don't decorate it with the @abstractmethod decorator.

If needed, we may also override this concrete method in the concrete class to provide any additional functionality as per user needs

These methods will help to reduce the repeated code

[22]:
```
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")
```

[24]:
```
c = Cat()
c.sleep()
c.sound()

c = Snake()
c.sleep()
c.sound()
```

```
I am going to sleep in a while
I can meow
I am going to sleep in a while
```

```
I can hiss
```

[25]:
```
#we can access the sound() function of the base class itself - Invoke Methods␣
 ↪from Abstract Classes
class Rabbit(Animal):
    def sound(self):
        super().sound()
        print("I can squeak")

c = Rabbit()
c.sound()
```

```
This function is for defining the sound by any animal
I can squeak
```

Abstraction Vs Encapsulation :

Encapsulation means-hiding data like using getter and setter etc.

Abstraction means- hiding implementation using abstract class and interfaces etc.

Abstraction solves an issue at the design level.

Encapsulation solves an issue at implementation level.

Summary :

Data Abstraction firstly saves a lot of our time as we do not have to repeat the code that is same for all classes.

Moreover, if there are any additional features, they can be easily added, thus improving flexibility.

We can make abstract classes in Python using abc module of Python.

We can implement abstract classes by inheriting the ABC class.

We use @abstractmethod decorators to define abstract methods in Python.

Abstract classes help to provide the blueprint for another class.

Abstract classes may contain concrete methods.

Abstract classes cannot be instantiated.

Data Abstraction FAQ's :

What is data abstraction ?

How to achieve data abstraction ?

What is an abstract class ?

Can you create an instance of an abstract class ?

What is an interface ?

Differentiate between data abstraction and encapsulation.