ABC in Python (Abstract Base Class)

nijanthan · Follow
Published in Analytics Vidhya
2 min read · Mar 12, 2021

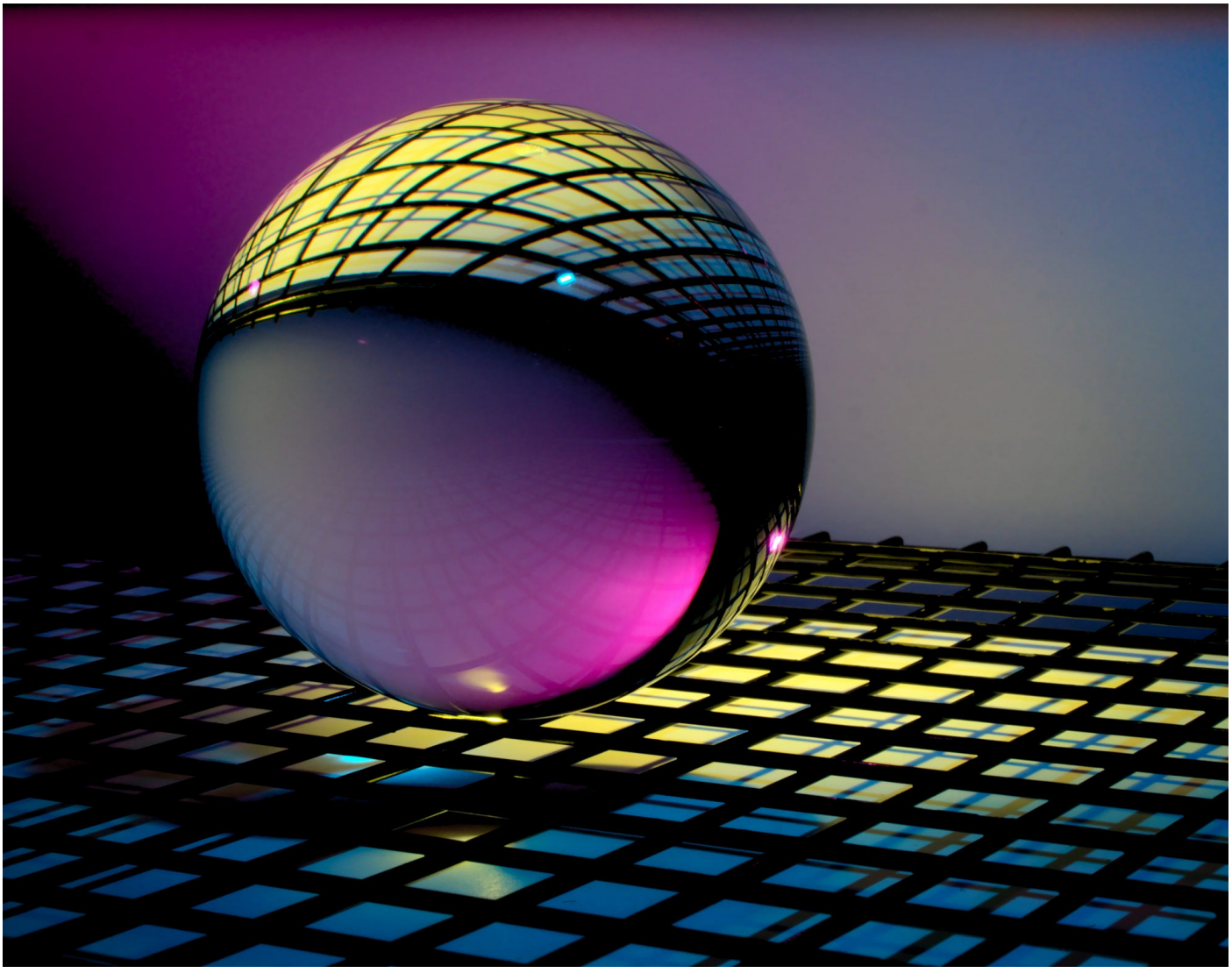

Photo by Michael Dziedzic on Unsplash

Introduction

An Abstract class is one of important concept in object oriented programming(oops). It is like blueprint for other classes.

Where we use abstract?

For larger projects, it is impossible to remember the class details, and also the reusability of code can increase the bug. Therefore, it plays a crucial role in our projects.

By default, Python does not provide abstract classes. The '**abc**' module in the Python library provides the infrastructure for defining custom abstract base classes.

Abstract **class** cannot be instantiated in python. An Abstract **method** can be call by its subclasses.

The above module provides us the following,

1. ABCMeta
2. ABC (helper class)

these are keywords used to create an abstract class.

ABCMeta:
It is a metaclass for defining the abstract class.

```python
from abc import ABCMeta, abstractmethod


class Calculation(metaclass=ABCMeta):

    @abstractmethod
    def add(self):
        pass

    @abstractmethod
    def subtract(self):
        pass

    def multiply(self):
        pass

    def division(self):
        pass


class Calculator(Calculation):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        print(self.a + self.b)

    def subtract(self):
        print(self.a - self.b)


take = Calculator(10, 5)
take.add()
take.subtract()
```

ABC:

It is a helper class. we can use it in the area where we want to avoid the confusion of metaclass usage.

```python
from abc import ABC, abstractmethod


class Calculation(ABC):

    @abstractmethod
    def add(self):
        pass

    @abstractmethod
    def subtract(self):
        pass

    def multiply(self):
        pass

    def division(self):
        pass


class Calculator(Calculation):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        print(self.a + self.b)

    def subtract(self):
        print(self.a - self.b)


take = Calculator(10, 5)
take.add()
take.subtract()
```

Both the steps are similar.

Note: The type of ABC (type(ABC)) is still ABCMeta. therefore inheriting from ABC requires the usual precautions regarding metaclass usage, as multiple inheritance may lead to metaclass conflicts.

register():

We can make a concrete subclass or built-in subclass as a virtual subclass using the register. But it is not visible in the Method Resolution Order(MRO). Its method is not even callable using super().

```
from abc import ABC, abstractmethod


class Calculation(ABC):

    @abstractmethod
    def add(self):
        pass

    @abstractmethod
    def subtract(self):
        pass

@Calculation.register
class Calculator:

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        print(self.a + self.b)

    # def subtract(self):
    #     print(self.a - self.b)


take = Calculator(10, 5)
take.add()
# subtract method is an abstract method but it is not throwing error due to virtual
# take.subtract()
print(issubclass(Calculator, Calculation)) # True
```

@abstractmethod:

It is used to make the method of the abstract class as an abstract method.

The abstract methods can be called using any of the normal 'super' call mechanisms. abstractmethod() may be used to declare abstract methods for properties and descriptors.

Dynamically adding abstract methods to a class, or attempting to modify the abstraction status of a method or class once it is created, are not supported. The abstractmethod() only affects subclasses derived using regular inheritance; "virtual subclasses" registered with the ABC's register() method are not affected.

Python Python Programming Object Oriented

```python
from abc import ABC, abstractmethod


class Calculation(ABC):

    @abstractmethod
    def add(self):
        pass

    @abstractmethod
    def subtract(self):
        pass

    def multiply(self):
        print('Method of an Abstract Class can be called without error',
              'because abstract method decorator is not used')

    @abstractmethod
    def division(self):
        pass

class Calculator(Calculation):

    def __init__(self, a, b):
        self.a = a
        self.b = b

    def add(self):
        print(self.a + self.b)

    def subtract(self):
        print(self.a - self.b)


take = Calculator(10, 5)
take.add() # 15
take.subtract() # 5
take.multiply() # Method of an Abstract Class can be called without error because ab
print(issubclass(Calculator, Calculation)) # True
# throws an error because abstract class method is an abstract method
take.division() # TypeError: Can't instantiate abstract class Calculator with abstra
```

There are other decorators (@abstractclassmethod, @abstractstaticmethod) and also some magic methods available. We can discuss about that further.