ABSTRACT BASE CLASS

Abstract base classes

While duck typing is useful, it is not always easy to tell in advance if a class is going to fulfill the protocol you require. Therefore, Python introduced the idea of abstract base classes. Abstract base classes, or ABCs, define a set of methods and properties that a class must implement in order to be considered a duck-type instance of that class. The class can extend the abstract base class itself in order to be used as an instance of that class, but it must supply all the appropriate methods. In practice, it's rarely necessary to create new abstract base classes, but we may find occasions to implement instances of existing ABCs. We'll cover implementing ABCs first, and then briefly see how to create your own if you should ever need to.

Abstract Classes in Python

An abstract class can be considered as a blueprint for other classes, allows you to create a set of methods that must be created within any child classes built from your abstract class. A class which contains one or abstract methods is called an abstract class. An abstract method is a method that has declaration but not has any implementation. Abstract classes are not able to instantiated and it needs sub-classes to provide implementations for those abstract methods which are defined in abstract classes. While we are designing large functional units we use an abstract class. When we want to provide a common implemented functionality for all implementations of a component, we use an abstract class. Abstract classes allow partially to implement classes when it completely implements all methods in a class, then it is called interface.

Why use Abstract Base Classes :

Abstract classes allow you to provide default functionality for the sub-classes. Compared to interfaces abstract classes can have an implementation. By defining an abstract base class, you can define a common Application Program Interface(API) for a set of sub-classes. This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins in an application, but can also help you when working on a large team or with a large code-base where keeping all classes in your head at the same time is difficult or not possible.

How Abstract Base classes work:

In python by default, it is not able to provide abstract classes, but python comes up with a module which provides the base for defining Abstract Base classes(ABC) and that module name is `ABC`. by marking methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes an abstract by decorated it with a keyword `@abstractmethod`. For Example –

Code 1:

```
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod

class Polygon(ABC):

    # abstract method
    def noofsides(self):
        pass

class Triangle(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 3 sides")

class Pentagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 5 sides")

class Hexagon(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 6 sides")

class Quadrilateral(Polygon):

    # overriding abstract method
    def noofsides(self):
        print("I have 4 sides")

# Driver code
R = Triangle()
R.noofsides()
```

```
K = Quadrilateral()
K.noofsides()

R = Pentagon()
R.noofsides()

K = Hexagon()
K.noofsides()
```

Output:

```
I have 3 sides
I have 4 sides
I have 5 sides
I have 6 sides
```

Code 2:

```
# Python program showing
# abstract base class work

from abc import ABC, abstractmethod
class Animal(ABC):

    def move(self):
        pass

class Human(Animal):

    def move(self):
        print("I can walk and run")

class Snake(Animal):

    def move(self):
        print("I can crawl")

class Dog(Animal):

    def move(self):
        print("I can bark")

class Lion(Animal):

    def move(self):
        print("I can roar")


# Driver code

R = Human()
```

```
R.move()

K = Snake()
K.move()

R = Dog()
R.move()

K = Lion()
K.move()
```

Output:

```
I can walk and run

I can crawl

I can bark

I can roar
```

Implementation Through Sub-classing :

By sub-classing directly from the base, we can avoid the need to register the class explicitly. In this case, the Python class management is used to recognize `PluginImplementation` as implementing the abstract `PluginBase`.

```
# Python program showing
# implementation of abstract
# class through subclassing

import abc
class parent
    def method(self):
        pass

class child(parent):
    def method(self):
        print("child class")

# Driver code

print( issubclass(child, parent))
print( isinstance(child(), parent))
```

Output:

```
True

True
```

A side-effect of using direct sub-classing is it is possible to find all of the implementations of your plugin by asking the base class for the list of known classes derived from it.

Concrete Methods in Abstract Base Classes :

Concrete classes contain only concrete (normal) methods whereas abstract class contains both concrete methods as well as abstract methods. Concrete class provide an implementation of abstract methods, the abstract base class can also provide an implementation by invoking the methods via `super()`.
Lets look over the example to invoke the method using super():

```
# Python program invoking a
# method using super()

import abc

from abc import ABC, abstractmethod

class R(ABC):
    def rk(self):
        print("Abstract Base Class")

class K(R):
    def rk(self):
        super().rk()
        print("subclass ")

# Driver code

r = K()
r.rk()
```

Output:

```
Abstract Base Class
subclass
```

In the above program, we can invoke the methods in abstract classes by using `super()`.

Abstract Properties :

Abstract classes includes attributes in addition to methods, you can require the attributes in concrete classes by defining them with @abstractproperty.

```
# Python program showing
# abstract properties

import abc
from abc import ABC, abstractmethod

class parent(ABC):

    @abc.abstractproperty

    def method(self):
       return "parent class"

class child(parent):

    @property

    def method(self):
        return "child class"

try:
    r =parent()
    print( r.method)

except Exception as err:
    print (err)

r = child()
print (r.method)
```

Output:

```
Can't instantiate abstract class parent with abstract methods method

child class
```

In the above example, the Base class cannot be instantiated because it has only an abstract version of the property getter method.

Abstract Class Instantiation :

Abstract classes are incomplete because they have methods which have no body. If python allows creating an object for abstract classes then using that object if anyone calls the abstract method, but there is no actual implementation to invoke. So we use an abstract class as a template and according to the need we extend it and build on it before we can use it. Due to the fact, an abstract class is not a concrete class, it cannot be instantiated. When we create an object for the abstract class it raises an *error*.

```
# Python program showing
# abstract class cannot
# be an instantiation
from abc import ABC,abstractmethod

class Animal(ABC):
    @abstractmethod
    def move(self):
        pass
class Human(Animal):
    def move(self):
        print("I can walk and run")

class Snake(Animal):
    def move(self):
        print("I can crawl")

class Dog(Animal):
    def move(self):
        print("I can bark")

class Lion(Animal):
    def move(self):
        print("I can roar")

c=Animal()
```

Output:

```
Traceback (most recent call last):
  File "/home/ffe4267d930f204512b7f501bb1bc489.py", line 19, in
    c=Animal()
TypeError: Can't instantiate abstract class Animal with abstract methods
move
```

Problems

1. Create an abstract class 'Parent' with a method 'message'. It has two sub-classes each having a method with the same name 'message' that prints "This is first subclass" and "This is second subclass" respectively. Call the methods 'message' by creating an object for each subclass.

2. Create an abstract class 'Bank' with an abstract method 'getBalance'. $100, $150 and $200 are deposited in banks A, B and C respectively. 'BankA', 'BankB' and 'BankC' are subclasses of class 'Bank', each having a method named 'getBalance'. Call this method by creating an object of each of the three classes.

3. We have to calculate the percentage of marks obtained in three subjects (each out of 100) by student A and in four subjects (each out of 100) by student B. Create an abstract class 'Marks' with an abstract method 'getPercentage'. It is inherited by two other classes 'A' and 'B' each having a method with the same name which returns the percentage of the students. The constructor of student A takes the marks in three subjects as its parameters and the marks in four subjects as its parameters for student B. Create an object for each of the two classes and print the percentage of marks for both the students.

4. An abstract class has a constructor which prints "This is constructor of abstract class", an abstract method named 'a_method' and a non-abstract method which prints "This is a normal method of abstract class". A class 'SubClass' inherits the abstract class and has a method named 'a_method' which prints "This is abstract method". Now create an object of 'SubClass' and call the abstract method and the non-abstract method. (Analyze the result)

5. Create an abstract class 'Animals' with two abstract methods 'cats' and 'dogs'. Now create a class 'Cats' with a method 'cats' which prints "Cats meow" and a class 'Dogs' with a method 'dogs' which prints "Dogs bark", both inheriting the class 'Animals'. Now create an object for each of the subclasses and call their respective methods.

6. We have to calculate the area of a rectangle, a square and a circle. Create an abstract class 'Shape' with three abstract methods namely 'RectangleArea' taking two parameters, 'SquareArea' and 'CircleArea' taking one parameter each. The parameters of 'RectangleArea' are its length and breadth, that of 'SquareArea' is its side and that of 'CircleArea' is its radius. Now create another class 'Area' containing all the three methods 'RectangleArea', 'SquareArea' and 'CircleArea' for printing the area of rectangle, square and circle respectively. Create an object of class 'Area' and call all the three methods.

7. Repeat the above question for 4 rectangles, 4 squares and 5 circles. Hint- Use list of objects.