Mayank Singh

6, Dec, 2022

Abstraction

Abstraction in python is defined as a process of handling complexity by hiding unnecessary information from the user.

In Python, abstraction can be achieved by using abstract classes and interfaces. An abstract class can be considered as a blueprint for other classes. It allows you to create a set of methods that must be created within any child classes built from the **abstract class**. **A class which contains one or more abstract methods is called an abstract class**.

An abstract method is a method that has a declaration but does not have an implementation. While we are designing large functional units we use an abstract class. When we want to provide a common interface for different implementations of a component, we use an abstract class.

Why use Abstract Base Classes :

By defining an abstract base class, you can define a common Application Program Interface(API) for a set of subclasses. This capability is especially useful in situations where a third-party is going to provide implementations, such as with plugins, but can also help you when working in a large team or with a large code-base where keeping all classes in your mind is difficult or not possible.

How Abstract Base classes work :

By default, Python does not provide abstract classes. Python comes with a module that provides the base for defining Abstract Base classes(ABC) and that module name is ABC. ABC works by decorating methods of the base class as abstract and then registering concrete classes as implementations of the abstract base. A method becomes abstract when decorated with the keyword **@abstractmethod**.

In [1]:
```
from abc import ABC,abstractmethod
class BankApp(ABC):

    def database(self):
        print('connected to database')

    @abstractmethod
    def security(self):
        pass

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

In [2]:
```
class MobileApp(BankApp):

    def mobile_login(self):
        print('login into mobile')

    def security(self):
        print('mobile security')

    def display(self):
        print('display')
```

In [3]:
```
mob = MobileApp()
```

In [4]:
```
mob.security()
```

```
mobile security
```

In [5]:
```
## Abstract class does not have the capability to create object
obj = BankApp()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [5], in <cell line: 1>()
----> 1 obj = BankApp()

TypeError: Can't instantiate abstract class BankApp with abstract methods displ
ay, security
```

In [ ]:
```

```