面向对象编程-抽象

在面向对象的概念中，所有的对象都是通过类来描绘的，但并不是所有的类都是用来描述对象的；如果一个类没有足够的信息来描绘一个具体的对象这样的类就是抽象类，抽象类除了不能实例化对象以外，类的其他功能仍然存在，成员方法和构造方法的访问方式和普通类一样．由于抽象类不能实例化对象，所以抽象类必须被继承才能被使用．

什么是抽象类?

1.抽象类不能被实例化，如果实例化抽象类就会报错，只有抽象类的非抽象子类才能实例化对象．

2.抽象类中的抽象方法只是声明不包含方法体，也就是不给出方法的具体实现．

3.子类必须重写抽象父类中的抽象方法．

什么是抽象方法?

在抽象类中有这样一个方法，这个方法的具体实现由它的子类确定，这样的方法为抽象方法．可以通过@abstractmethod装饰器来声明抽象方法，抽象方法只有方法名，没有方法体．

In [16]:
```
#代码示例1: Python2中的抽象类代码

from abc import ABCMeta, abstractmethod

class Person:
    __metaclass__ =  ABCMeta  ##用于在python中创建抽象基类
    def __init__(self, name):
        self.name = name
        self.age = 19
    def getName(self):
        return self.name, self.age
    @abstractmethod
    def getTalk(self):
        pass
```

In [17]:
```
#代码示例2: Python3中的抽象类代码

from abc import ABCMeta, abstractmethod
class Person(metaclass=ABCMeta):
    def __init__(self, name):
        self.name = name
        self.age = 19
    def getName(self):
        return self.name, self.age
    @abstractproperty
    def getTalk(self):
        pass
```

In [7]:
```
p = Person()
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-7-f16ed64f4024> in <module>
----> 1 p = Person()

TypeError: Can't instantiate abstract class Person with abstract methods getTalk
```

通过Person类实例化对象p报错，因为抽象类不可以被实例化．

In [11]:
```
#示例代码3: 抽象类继承
class s1(Person):
    def __init__(self, name, job):
        Person.__init__(self, name)
        self.job = job

x = s1("lily", 'hr')
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-11-4627b9372af8> in <module>
      5         self.job = job
      6
----> 7 x = s1("lily", 'hr')

TypeError: Can't instantiate abstract class s1 with abstract methods getTalk
```

定义了一个子类1继承了Person类，继承抽象方法的子类必须重写该方法，因为必须有子类实现该抽象方法，否则无论是最初的父类还是子类都不能实例化对象．

In [14]:
```
#示例代码4: 抽象类继承并且子类重写了抽象方法

class s1(Person):
    def __init__(self, name, job):
        Person.__init__(self, name)
        self.job = job
    def getTalk(self):
        return 'i am a student %s' %self.job

s = s1("tom", "tech")
print(s.getTalk())
```

```
i am a student tech
```