4.创建型模式-原型模式

概述：
    当需要在原有对象的基础上创建一个该对象的副本时，我们就可以使用原型模式．在Python里可以很简单的通过copy.copy或copy.deepcopy函数来实现原型模式．
    简而言之原型模式就是
    第一种情况：你有一个 ApplePen，想要一个一模一样的ApplePen，可以直接制作一个ApplePen的副本．

    第二种情况：你有一个 ApplePen，想要一个PineapplePen 可以通过制作一个ApplePen的副本，然后修改这个副本为PineapplePen的方式来达成目的．

生活中的例子：
    大家如果用过类似于Photoshop的平面设计软件，一定都知道图层的概念．图层概念的提出，使得设计、图形修改等操作更加便利．设计师既可以修改和绘制当前图像对象，又可以保留其它图像对象，逻辑清晰且可以及时得到反馈．

示例代码如下：

    首先设计一个图层对象，在图层上花一个"狗"的图案，并且用[0,0,0,1]作为颜色填充背景色．

In [3]:
```
class simpleLayer:

    @property
    def picture(self):
        return self.__picture

    @picture.setter
    def picture(self, value):
        self.__picture = value


    @property
    def background(self):
        return self.__background

    @background.setter
    def background(self, value):
        if not value:
            self.__background = [0,0,0,0]
        self.__background = value


if __name__ == '__main__':
    x = simpleLayer()
    x.picture = "dog"
    x.background = [0, 0, 0, 1]

    print("Drow a picture %s" %x.picture)
    print("Background color %s" %x.background)
```

```
Drow a picture dog
Background color [0, 0, 0, 1]
```

    接下来，如果需要再生成一个同样的图层，再填充同样的颜色，再画一只同样狗，该如何做呢？还是按照新建图层、填充背景、画的顺序么？ 或许你已经发现了，这里可以用复制的方法来实现，而复制(clone)这个动作，就是原型模式的精髓了．

    按照此思路，在图层类中新加入一个新方法：deep_clone．

示例代码如下：

In [4]:
```
from copy import deepcopy

class simpleLayer:

    @property
    def picture(self):
        return self.__picture

    @picture.setter
    def picture(self, value):
        self.__picture = value

    @property
    def background(self):
        return self.__background

    @background.setter
    def background(self, value):
        if not value:
            self.__background = [0,0,0,0]
        self.__background = value

    def deep_clone(self):
        return deepcopy(self)

if __name__ == '__main__':
    origin = simpleLayer()
    origin.picture = "dog"
    origin.background = [0, 0, 0, 1]
    print("origin Drow a picture %s" %origin.picture)
    print("origin Background color %s" %origin.background)
    print('#' * 20)

    second_one = origin.deep_clone()
    second_one.background = [0, 1, 1, 1]
    print("second_one Drow a picture %s" % second_one.picture)
    print("second_one Background color %s" % second_one.background)
    print("origin Drow a picture %s" % origin.picture)
    print("origin Background color %s" % origin.background)
    print('#' * 20)
```

```
origin Drow a picture dog
origin Background color [0, 0, 0, 1]
####################
second_one Drow a picture dog
second_one Background color [0, 1, 1, 1]
origin Drow a picture dog
origin Background color [0, 0, 0, 1]
####################
```

注意：
    进行clone操作后，新对象的构造函数没有被二次执行，新对象的内容是从内存里直接拷贝的．因为在second_one中，可以直接打印出dog，并没有调用second_one.picture="dog".

优点：
    1.性能极佳，直接拷贝比在内存里直接新建实例节省不少的资源．
    2.简化对象创建，同时避免了构造函数的约束，不受构造函数的限制直接复制对象，是优点 也有隐患，这一点还是需要多留意一些．

使用场景：

1.对象在修改过后，需要复制多份的场景，如本例和其它一些涉及到复制、粘贴的场景.
2.需要优化资源的情况，如需要在内存中创建非常多的实例，可以通过原型模式来减少资源消耗；此时原型模式与工厂模式配合起来，不管在逻辑上还是结构上，都会达到不错的效果.

In [ ]: