在上一课的内容中，我们一起开发了一个名为MyCoin的小项目，设计了自己的硬币。而在这一课的内容中，我们将学习关于类的更多知识，同时扩展MyCoin项目，让它可以生成更多种类的硬币。

打开Jupyter Notebook,打开上一课的MyCoin项目。

首先我们要创建一个通用的硬币类，使用以下代码即可：

class Coin:

def \_\_init\_\_(self,rare = False,clean = True, heads = True):

self.is\_rare = rare

self.is\_clean = clean

self.heads = heads

if self.is\_rare:

self.value = self.original\_value \* 1.25

else:

self.value = self.original\_value

if self.is\_clean:

self.color = self.clean\_color

else:

self.color = self.rusty\_color

在上面的代码中，我们定义了一个新的Coin类。在其构造方法中，我们设置了一些特定的属性值。

接着在Coin的类定义中添加几个类方法：

def rust(self):

self.color = self.rusty\_color

def clean(self):

self.color = self.clean\_color

def \_\_del\_\_(self):

print("Coin spent!")

def flip(self):

heads\_options = [True,False]

choice = random.choice(heads\_options)

self.heads = choice

在以上的代码中，我们采用了上一课所用到的知识，添加了几个通用的类方法。不过和MyCoin类不同的是，这里面的states值并没有特别指定。

好了，接下来我们可以删除MyCoin类定义中的原有代码，并从Coin这个通用类继承来生成：

class MyCoin(Coin):

def \_\_init\_\_(self):

data = {

"original\_value":1.00,

"clean\_color":"gold",

"rusty\_color":"greenish"

"num\_edges":1,

"diameter":22.5,

"thickness":3.15,

"mass":9.5

}

super().\_\_init\_\_(\*\*data)

在以上代码中，我们让MyCoin类继承自Coin类。在初始化方法中定义了一系列的数据，然后调用父类（super)的初始化方法，把data的信息都传递过去。

这一点是Python中很有意思的用法。

此时我们还需要完成一个事情，更改Coin类的init初始化方法，添加一个新的参数：

class Coin:

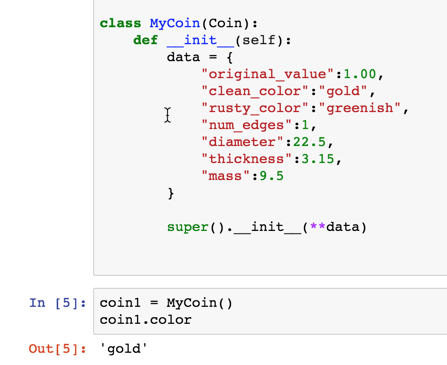
def \_\_init\_\_(self,rare = False,clean = True, heads = True,\*\*kwargs):

for key,value in kwargs.items():

setattr(self,key,value)

通过这样的方式，我们就实现了上一课中设置states默认值的效果。

试着运行一下，看看效果：



好了，这一课的内容就先到这里了，可能有些概念还不是很清楚。特别是涉及到数据unpack的部分，建议大家复习下函数部分的相关内容。

我们下一课再见。