在这一课的内容中，我们将继续完善MyCoin项目，并学习多态的概念。

首先在Jupyter Notebook中打开MyCoin项目,在MyCoin类定义的代码之后（或之前）添加一个新的类定义：

class One\_Pence(Coin):

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

data = {

"original\_value":0.01,

"clean\_color":"bronze",

"rusty\_color":"brownish",

"num\_edges":1,

"diameter":20.3,

"thickness":1.52,

"mass":3.56

}

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

可以看到，One\_Pence类继承自Coin类，只是将初始的数据信息做了更改。

接着添加另外一个类定义。

class Two\_Pence(Coin):

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

data = {

"original\_value":0.02,

"clean\_color":"bronze",

"rusty\_color":"brownish",

"num\_edges":1,

"diameter":25.9,

"thickness":1.85,

"mass":7.12

}

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

可以看到，One\_Pence类同样继承自Coin类，只是将初始的数据信息做了更改。

再添加一个新的类定义。

class Five\_Pence(Coin):

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

data = {

"original\_value":0.05,

"clean\_color":"silver",

"rusty\_color":None,

"num\_edges":1,

"diameter":18.0,

"thickness":1.77,

"mass":3.25

}

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

def rust(self):

self.color = self.clean\_color

def clean(self):

self.color = self.clean\_color

跟之前略有区别的是，Five\_Pence的类定义中对两个类方法重新进行了定义，这就是所谓的多态。

在Python中，所谓的多态就是子类可以重写父类的方法。

接下来再添加几个新的类定义：

class Ten\_Pence(Coin):

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

data = {

"original\_value":0.1,

"clean\_color":"silver",

"rusty\_color":None,

"num\_edges":1,

"diameter":24.5,

"thickness":1.95,

"mass":6.50

}

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

def rust(self):

self.color = self.clean\_color

def clean(self):

self.color = self.clean\_color

class Twenty\_Pence(Coin):

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

data = {

"original\_value":0.20,

"clean\_color":"silver",

"rusty\_color":None,

"num\_edges":1,

"diameter":21.4,

"thickness":1.7,

"mass":5

}

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

def rust(self):

self.color = self.clean\_color

def clean(self):

self.color = self.clean\_color

class Fifty\_Pence(Coin):

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

data = {

"original\_value":0.50,

"clean\_color":"silver",

"rusty\_color":None,

"num\_edges":7,

"diameter":27.3,

"thickness":1.78,

"mass":8.00

}

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

def rust(self):

self.color = self.clean\_color

def clean(self):

self.color = self.clean\_color

class Two\_Pound(Coin):

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

data = {

"original\_value":2.00,

"clean\_color":"gold",

“rusty\_color":"greenish",

"num\_edges":1,

"diameter":28.4,

"thickness":2.50,

"mass":12.00

}

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

然后把之前的MyCoin类更名为One\_Pound

接下来创建一个硬币列表：

coins = [One\_Pence(),Two\_Pence(),Fice\_Pence(),Ten\_Pence(),Twenty\_Pence(),Fifty\_Pence(),MyCoin(),Two\_Pound()]

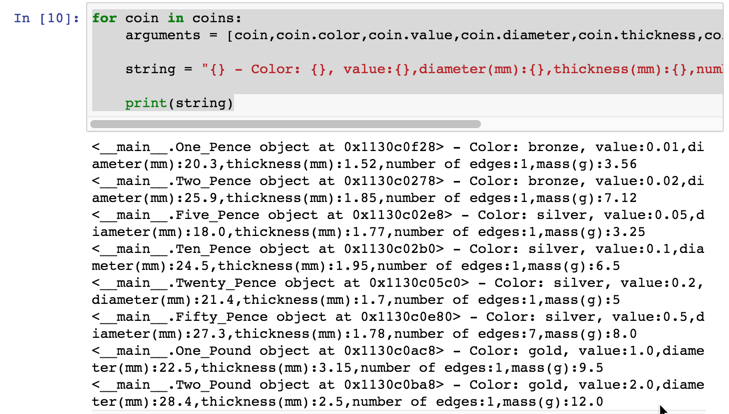
然后遍历每一种硬币，并输出硬币的相关信息：

for coin in coins:

arguments = [coin,coin.color,coin.value,coin.diameter,coin.thickness,coin.num\_edges,coin.mass]

string = "{} - Color: {}, value:{},diameter(mm):{},thickness(mm):{},number of edges:{},mass(g):{}".format(\*arguments)

print(string)

在Jupyter Notebook中试着跑一下，会看到类似下面的输出信息：

这样的输出方式看起来不是很清爽，让我们在Coin类定义中添加一个新的方法：

def \_\_str\_\_(self):

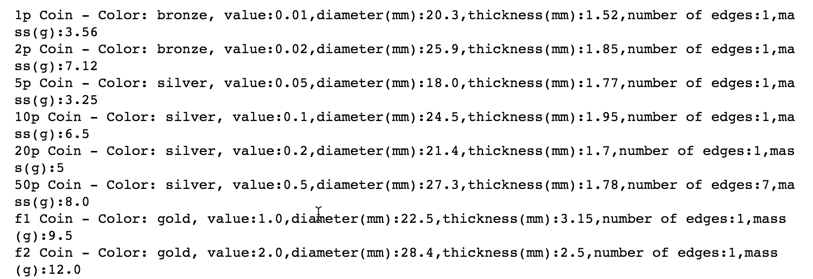
if self.original\_value >= 1:

return "f{} Coin".format(int(self.original\_value))

else:

return "{}p Coin".format(int(self.original\_value \* 100))

这样一来，当再次运行的时候，输出的就不再是对象的直接信息，而是以特定格式显示的信息：



好了，到这里我们已经创建了一整个硬币家族。

在该系列教程的最后一课中，我们将一起来学习开办一个自己的小银行~

我们下一课再见。