基础语法文档（四）

Section1.序言

面向对象编程（OOP）是一种用于组织程序的方法，该方法将本章介绍的许多想法融合在一起。像数据抽象中的功能一样，类在数据的使用和实现之间创建抽象障碍。像调度字典一样，对象响应行为请求。像可变数据结构一样，对象具有局部状态，无法从全局环境直接访问。 Python对象系统提供了方便的语法，以促进使用这些技术来组织程序。许多其他面向对象的编程语言都共享这种语法。

对象系统不仅提供便利。它为设计程序提供了一个新的隐喻，其中几个独立的代理在计算机内进行交互。每个对象以抽象两者的复杂性的方式将本地状态和行为捆绑在一起。对象彼此通信，并且通过交互作用来计算有用的结果。对象不仅传递消息，而且还在其他相同类型的对象之间共享行为，并从相关类型继承特征。

面向对象编程的范式具有自己的词汇表。我们已经看到，对象是具有方法和属性的数据值，可以通过点表示法对其进行访问。每个对象还具有一个称为其类的类型。为了创建新的数据类型，我们实现了新的类。

Section2. 类的意义

一个类用作类型为该类的对象的模板，每一个对象都是一个类的实例。字符串如此，函数如此，整数如此。到目前为止，我们使用的对象都有内置的类，但是也可以创建新的用户定义的类。类定义指定了该类对象共享的属性和方法。我们将使用银行账户的实例来介绍第一点：类的使用

2.1 类的使用

类的使用实际上创建该类对象，修改该类对象的属性和调用该类对象的方法。

```
a = Account("limzh")
a.holder # >>> 'limzh'
a.balance # 0
a.deposit(15) # 15
a.withdraw(10) # 10
a.balance  # 5
a.withdraw(10) # 'Insuficient funds'
```

上图是一个实例，假设我们已经拥有了一个类的模板`Account`,它定义了一个银行账户的对象应当拥有怎么样的属性和方法，这里我们省略该类的定义。首先我们创建了一个该类的实例，通过`.`来访问该实例的属性和方法。在上图，holder, balance是属性，deposit, withdraw是方法。可以看到，我们可以像使用函数一样使用方法。

Section2.2 类的定义

目前为止我们使用的类都是内建类型的，现在我们要开始自定义一个类。

```
class <name>:
    <suite>
```

一个类定义的完整语法如上，指的是类的名字，指的是类的内容。只要填充好这两块内容，我们就说这个类已经定义好了。最简单的，语法有效的类可以这样实现：

```
class Account:
    pass
```

这样的类虽然语法完整，但是不具有实际意义。当类语句被执行，一个新的类被创建，在全局作用域中新分配一块独立的作用空间，并且将这块作用域绑定到你所定义的上，就像简单的变量赋值一样。之后在这块新分配的作用域中执行的全部内容。这就是class实际做的事情，按照这样的理解，不难发现，上面这个看上去特别简单的类在语法上是完整的。

然而，只满足于语法完整还远远不够，我们当然还希望它能做一些事情。

```
class Account:
    def __init__(self,holder):
        self.holder = holder
        self.balance = 0
```

如上图，我们用`def __init__(self)`替换了`pass`。这样我们定义了一个初始函数。初始函数的意思就是，当Account的一个实例被创建以后，会自动地调用该函数来对你的属性进行初始化。它的参数里面有self, holder两个形式参数。第一个，即self，绑定到新创建的Account对象。第二个参数holder绑定到在调用要实例化时传递给类的参数。

构造函数将实例属性名称余额绑定为0。还将属性名称所有者绑定到名称account_holder的值。形式参数account_holder是**init**方法中的本地名称。另一方面，通过最终赋值语句绑定的名称持有者将保留，因为它使用点表示法存储为self的属性。

定义好初始化函数之后，我们就可以实例化它.

```
a = Account('limzh')
```

虽然初始化函数有两个参数，但我们只需要传入除了self 以外的其他参数就可以了。这是因为Python会自动地将创建的对象的名字传入self.

Section3. 实例的属性

对类定义最重要的两样东西是属性和方法，在这里我们先说属性。我们默认，属性指的是类的实例的成员属性。

```
class Account:
    def __init__(self,holder):
        self.holder = holder
        self.balance = 0
```

在上面的例子里，属性已经在初始化函数里添加了。这意味着每一个Account的实例都具有holder 和balance两个成员属性。但我们要问的问题是，成员属性必须要在初始化函数里声明吗？我们略作尝试。

```
class Account:
    def __init__(self,holder):
        self.holder = holder
        self.balance = 0
    def someMethod(self):
        self.exchange = 10
```

运行如下程序

```
a = Account('limzh')
a.holder
a.balance
a.exchange # error
a.someMethod()
a.exchange # 10
```

我们发现，在执行someMethod方法之前，exchange这个属性并没有添加到类的实例里面中去，但是当执行完该方法之后就成功添加了。难道说实例的属性可以被动态添加和删除的嘛?

```
a.yyr = 1
a.yyr # 1
```

原来确实如此，Python中成员属性是可以动态添加和更改的。即使是在实例的外部，也可以像访问创建一个新变量一样对其成员属性进行添加。这样做被允许的原因是，本质上`a.xxx`做了的事情只是通过变量的名字访问到其所独立拥有的作用域仅此而已，在实例作用域内动态的创建、删除一个变量当然是被允许的。

Section4. 实例的方法

所谓实例的方法仅仅是函数而已。函数的一切性质就是实例方法的一切性质，而实例方法仅仅是调用方式和传入的`self`有所不同而已。只要记住这一点就具备了对类方法的基本掌握。而而一些高级特性以及其内部的机制，之后再慢慢引入。但我要说明的是，和属性不一样，实例的方法是静态添加的，不可增加和删改。

Section5. 总结和练习

截至本文档，Python的类和函数的基本语法就介绍完了。

下面是一点练习，参考答案我已经放到day5中，请参考。

练习:

1. 请为中国银行定义一个类`Account`，用于生成一个用户的账户实例。要求可以记录用户的姓名，密码，账户余额。在其中定义转账支出和收入的方法，使得调用支出时可以减少相应的余额，调用收入时可以增加相应的余额。
2. 请定义一个函数，输入是两个账户A、B和一个金额（A是第一个参数，B是第二个参数，金额是第三个参数），使得A向B转入输入的金额数。如果转入成功（A中有足够的金额）则返回1， 转入失败则返回0.