Python基础_什么是类

Python是面向对象的高级编程语言，在Python里面“一切都是对象”：数字、字符串、元组、列表、字典、集合等内置数据类型，以及函数、方法、类、模块都是对象。

语言本身提供了上述的基本对象，但在实际编程中，我们要创造各种各样的对象，Python就为我们提供了创造我们自己的对象的方法：**类**。

类（Class），就是组合数据和功能的方法，它让我们创建一个新**类型**的对象，并可以创建该类型的新**实例**。类组合的数据，就是保存自己状态的**属性**，而它组合的功能（函数）就是改变自己状态的（定义在类中的）**方法**。类内部定义的函数，称为类的**方法**。

Python中的类和其它语言（比如C++）有很多相似的特征但也有些区别。如果你已了解其它语言的类的概念，可以在学习Python类时做一定的对比进行学习；如果你没有学过其它语言也不要紧，学过之后你会发现，类的概念是如此简单。

类的定义

类的定义是通过关键字class实现的，下面是最简单的类的定义的样子：

```
class ClassName:
    语句1
    ...
    语句n
```

是不是这个形式跟函数的定义（def 语句）很像。因为类是数据和功能的组合，所以语句1可能是内部变量（数据）的定义和赋值语句，也可能是内部方法（函数）的定义语句。类内部的函数定义通常具有一种特别形式的参数列表，这是方法调用的约定规范里面指明的。这个特别形式就是第一个参数必须是self，后面将详细介绍。

进入类定义时，就会创建一个新的命名空间，并把它用作局部作用域。因此，所有对局部变量的赋值都是在这个新命名空间内进行的。特别的，函数定义会绑定到这个局部作用域里的新函数名称。

正常离开（从结尾出）类定义时，就会创建一个**类对象**。它基本上是一个包围在类定义所创建的命名空间内容周围的包装器。元素的（在进入类定义之前起作用的）局部作用域将重新生效，类对象将在这里被绑定到类定义头给出的类名称（在上面的例子中就是ClassName）。

类对象

类对象（比如上面例子的ClassName）支持两种操作：**属性引用**和**实例化**。

属性引用的语法跟Python中所有属性引用的方法一样：obj.name。类对象被创建时存在于类命名空间内的所有名称都是有效的属性名称。下面是一个包含数据和方法的简单的类定义：

```
class KaiKeBa:
    """A demo for class"""
    name = '开课吧'
def say_hi(self):
    print('Hello World')
```

对这个类的有效的属性引用就是：
KaiKeBa.name
KaiKeBa.say_hi，
它们分别返回一个字符串和一个函数对象。
类属性也可以被赋值，因此可以通过赋值来更改KaiKeBa.name的值。
类的__doc__也是一个有效的属性，对他的引用会返回所属类的文档字符串：'A demo of class'。
类的**实例化**，是使用函数表示法，可以把类对象看做是会返回一个新的类实例的函数。比如上面类对象的实例化就是：

```
kkb = KaiKeBa()
```

这就创建了一个类的新实例并将词对象分配给局部变量kkb。
实例化操作可以看成是“调用”类对象。但我们在创建类实例时都想要做些初始化操作，为此类定义时可以定义一个名为__init__()的特殊方法。它是类实例化的初始化方法，跟C++语言中 的构造函数类似。

```
def __init__(self):
self.data = None
```

定义了__init__()方法后，类的实例化操作会自动调用该方法。
当然，__init__()方法也可以有额外（除self之外）的参数以实现更灵活的初始化操作。类对象实例化时（ “调用”类对象）传递的参数会被传递给__init__()方法。例如：

```
class Point:
        def __init__(self, x, y):
        self.x = x
         self.y = y

p = Point(7, 8)

print （p.x, p.y)
#(7, 8)
```

实例对象

类实例化后我们就得到了实例对象，对它的操作就是：属性引用。这里的有效属性名称是数据属性和方法。

数据属性，数据属性不需要声明，它像普通变量一样，在第一次赋值时产生。比如p是声明创建的Point的实例，则以下代码会打印数值8：

```
p.times = 1
while p.times < 5:
    p.times = p.times * 2
print(p.times)
del p.times
```

虽然p.times并没有在类定义时声明（数据属性不需要声明），但在任何时候，我们可以给实例赋值一个新的数据属性（这里是p.times），并可以随时删除实例的数据属性（del p.times）。

方法，是“从属于”对象的函数。方法这个术语并不是类实例独有的，其它对象也可以有方法。比如，列表对象有append(), insert(), sort()等方法。

实例对象的有效方法名称依赖于其所属的类。根据定义，一个类中所有是函数对象的属性都是定义了其实例的相应方法。因此在我们的示例中，kkb.say_hi是有效的方法引用，因为KaiKeBa.say_hi是一个函数；而kkb.name不是方法，因为KaiKeBa.name不是一个函数。这里要注意，kkb.say_hi与KaiKeBa.say_hi并不是一回事，它是一个方法对象，不是函数对象，通俗讲，前者是实例的方法，后者是类的函数。

方法对象

通常，调用方法的方法是：

```
kkb.say_hi()
```

在KaiKeBa示例中，这将打印字符串Hello World。但是，调用一个方法也可以换另外一种形式，把它赋值给一个变量后再调用。例如：

```
kkb_say = kkb.say_hi
kkb_say()
```

当我们调用kkb.say_hi()时并没有带参数，但say_hi()函数定义时指定了一个参数。实际上，方法的特殊之处就是实例对象会作为函数的第一个参数(self)被传入。调用kkb.say_hi()其实就相当于KaiKeBa.say_hi(kkn)。

类变量和实例变量

一般来说，类变量用于类的所有实例共享的属性和方法，而实例变量用于每个实例的唯一数据

如果类变量是列表或字典这种可变对象会导致令人惊讶的结果，我们要明白这种结果，从而可以在需要的时候利用这个特性，也可以在不需要的时候避免这个特性。是取是舍，全在于我们在编程时要定义的类的作用。下面，我们看看这个“令人惊讶”的结果是什么：

```
class Tiger:
        places = []
        def __init__(self, name):
            self.name = name
        def go_place(self, place):
            self.places.append(place)
a = Tiger('Kiro')

b = Tiger('Zim')

a.go_place('北京')

b.go_place('上海')

print(a.places)
#['北京', '上海']
```

这里，把places定义为类变量，它就记录了所有老虎（Kiro和Zim，以及后面实例化出来个各个老虎）去过的地方，所以，尽管a只去过北京，但是当我们通过a查看places，也看到了上海。这就是可变对象作为类变量时的特性。如果我们就是想记录所有老虎实例对象去过的地方，这样的用法就是恰到好处。

如果，我们只想记录每一只老虎去过的地方，那么就应该把places定义为实例变量，也就是在__init__()中进行初始化赋值。

需要注意的地方

（1）数据属性会覆盖掉具有相同名称的方法属性，因此写程序时为了避免名称冲突，可以使用某些规范来减少冲突，比如：

- 方法名称使用大写字母；
- 属性名称加上特殊的前缀（或加一个下划线）；
- 用动词来命名方法，用名词来命名数据属性。

这些规范称为“代码规范”，一个好的代码规范让代码可读性更高，也更利用团队合作。有名的Python代码规范有“PEP 8”，也有Google的Python风格规范。

（2）数据属性可以被方法以及对象之外的任何用户（使用该类的人）所引用。也就是说，Python不行C++类那样有私有成员，Python没有任何对象能强制隐藏数据。

（3）类对象的用户应该谨慎使用数据属性，直接操作数据属性可能破坏其中的固定变量。但我们可以想一个实例对象添加我们自己的数据属性，但要避免与原有的名称冲突。

（4）方法的第一个参数常常被命名为self，代表实例对象。但这只是大家普通认同的一个约定。你可以用任何单词来替代它，但是你的代码让气体程序员读起来就很费劲，也会对VS Code这样的编辑器造成困惑。

总结

类是组合数据和功能的方法，其中：

- 数据，是类的属性，从属于类的各种数据对象；
- 功能，是类的方法，定义在类内部的各种函数。

定义好一个类我们就创建了一个类对象，类对象支持两种操作：属性引用和实例化。
类对象实例化后就得到了实例对象，它有两种属性名称：数据属性和方法。
类变量是所有类的实例共享的属性和方法，实例变量是每个实例独有的数据。