Python的面向对象

面向对象简述

我不想讨论面向对象和面向过程到底有什么区别，也不想讨论python到底是不是一门面向对象语言...这些讨论没什么意义；python提供了类，继承，重写...等一系列功能，让我们能够抽象出类，实例化对象，

类和对象

类和对象是什么?

在正式开始之前必须阐明：Python遵循万物皆为对象的理念；python中其实没有真正意义上的类，python中的“类”本身只是一个特殊的对象罢了，这点在后续学习“self”的时候会详细的解释

“类”是对“一类东西”的抽象；

“对象”是“类”的一个实例；

举个例子：

1. 我们日常生活中，所说的“学生”就是一个类，“学生”是我们对所有上学的人抽象出来的一个概念，当我们提及“学生”这个类时，我们都知道“学生”应该有学号，有班级...等；学号，班级...这些东西我们统称为“类的属性”
2. 如果我们提及**具体**的某位同学，那么这位同学就是“对象”；“对象”是“类”的实例

如何创建一个类

```
class staff_class:
    '这是一个员工类'
    num_of_staff = 0 # 共有属性
    _num_of_resigned_employees = 0 # 保护属性
    __salary_budget = 0  #私有属性

    def __init__(self,name,salary): # 初始化函数
        self.name = name # 对象属性
        self.salary = salary
        staff_class.num_of_staff += 1

    def show_staff_info(self):# 对象方法
        print(self.name)
        print(self.salary)

    @classmethod
    def show_num_of_staff(cls): # 类方法
        print(staff_class.num_of_staff)


staff_obj = staff_class("li",1000) # 使用类的名字可以调用初始化函数，无需传入self这个参数
```

```
staff_obj.show_staff_info()

staff_class.show_num_of_staff() # 类方法既可以通过“类.方法”的方式去调用
staff_obj.show_num_of_staff() # 也可以通过"对象.方法"的方式去调用


# 输出如下
# PS C:\Users\lzz_l\tmp> python.exe .\test.py
# li
# 1000
# 1
```

在使用类时，self是什么?

self是理解python中的“类”的关键：

python中实际上是没有“类”的，我们用的“class”实际上本身就是一个特殊的“对象”；

在上面的例子中，我们可以看到，有一些方法中有一个叫做“self”的参数，self指向的是当前的“对象”；当调用初始化函数生成了一个对象之后，self就指向这个对象，随后的调用中，凡是“self.属性”定义的属性，对象之间是不共享的；

权限控制

python也有“共有属性（方法）”，“保护属性（方法）”，“私有属性（方法）”这种设计；

python的权限控制是用下划线表示的:

```
__a__ # 头尾双下划线，一般是系统定义的名字，比如__init__();

a # 没有下划线修饰，表示这是一个共有属性（方法），任何地方都可以调用

_a # 头部单下划线，表示这是一个保护属性（方法），只可以在本类和本类的继承类里使用

__a # 头部双下划线，表示这是一个私有属性（方法），只可以在本类中使用
```

析构函数

```
class my_class:
    def __init__(self):
        self.test = 0

obj = my_class()
del obj # 这行代码使用了析构函数，将obj这个对象删除掉
```

析构函数的底层逻辑是“引用计数器”，对象被构建时计数器会赋值为0，随着该对象被引用，计数器会+1；不过一般而言，我们在初始化对象的时候都会以：

```
obj = class()
```

这种形式初始化，一开始其引用计数器就是1

如果一个对象的引用计数器为0，那么解释器会自动回收内存，当然也可以手动的del，即使引用计数器不是0，del也会生效，请注意不要随便del以防引起错误，python解释器会自动帮你做这些事...

```
class staff_class:
    def __init__(self,name,salary):
        self.name = name
        self.salary = salary
        staff_class.num_of_staff += 1
    def __del__(self):
        print("销毁成功")



staff_obj = staff_class("li",1000)

staff_obj_2 = staff_obj

del staff_obj

# 执行这个代码
# PS C:\Users\lzz_l\tmp> python.exe .\test.py
# 销毁成功
```

可见即使staff_obj被staff_obj_2引用，但是还能被删除

当对象作为参数被传递

python中不存在值传递，所有的传递都是指针传递；即：当对象被作为参数传递时，对象在内存中只有一份，即：

```
#
class my_class():
    def __init__(self):
        self.value = 0
obj = my_class()

def my_func_a(my_class):
    my_class.value = 1

def my_func_b(my_class_obj):
    my_class_obj.value = 2
    print(my_class_obj.value)
    my_func_a(my_class_obj) # 这里将对象当作参数传入函数a，在函数a内，对象的值被修改
    print(my_class_obj.value) # 虽然对象在函数a内被修改，但函数b内的对象的值也变了，说明这是一个对象

my_func_b(obj)

# 执行结果
# PS C:\Users\lzz_l\tmp> python.exe .\test.py
# 2
```

```
21 # 1
```

继承、重写、重载

略...这部分内容一般是用不到的...等用到的时候再学习....