# Python实验11 类

## 一、实验介绍

在Python中，所有数据类型都可以视为对象，当然也可以自定义对象。自定义的对象数据类型就是面向对象中的类（Class）的概念。

知识点

类的定义

对象初始化

## 二、实验步骤

本部分实验中将通过定义一些简单的 Python 类，来学习 Python 面向对象编程的基本概念。

实验安排如下：

定义简单的类

\_\_init\_\_ 方法

Python 中的继承

多继承

删除对象

属性读取方法

@property 装饰器

2.1 定义类

在写你的第一个类之前，你应该知道它的语法。我们以下面这种方式定义类：

class nameoftheclass(parent\_class):

statement1

statement2

statement3

在类的声明中你可以写任何 Python 语句，包括定义函数（在类中我们称为方法）。

>>> class MyClass(object):

... """A simple example class"""

... i = 12345

... def f(self):

... return 'hello world'

2.2 \_\_init\_\_ 方法

类的实例化使用函数符号。只要将类对象看作是一个返回新的类实例的无参数函数即可。例如（假设沿用前面的类）:

x = MyClass()

以上创建了一个新的类实例并将该对象赋给局部变量 x。

这个实例化操作创建一个空的对象。很多类都倾向于将对象创建为有初始状态的。因此类可能会定义一个名为 \_\_init\_\_() 的特殊方法，像下面这样:

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

self.data = []

类定义了 \_\_init\_\_() 方法的话，类的实例化操作会自动为新创建的类实例调用 \_\_init\_\_() 方法。所以在下例中，可以这样创建一个新的实例:

x = MyClass()

当然，出于弹性的需要，\_\_init\_\_() 方法可以有参数。事实上，参数通过\_\_init\_\_() 传递到类的实例化操作上。例如：

>>> class Complex:

... def \_\_init\_\_(self, realpart, imagpart):

... self.r = realpart

... self.i = imagpart

...

>>> x = Complex(3.0, -4.5)

>>> x.r, x.i

(3.0, -4.5)

2.3 继承

当一个类继承另一个类时，它将继承父类的所有功能（如变量和方法）。这有助于重用代码。

在下一个例子中我们首先创建一个叫做 Person 的类，然后创建两个派生类 Student 和 Teacher。当两个类都从 Person 类继承时，它们的类除了会有 Person 类的所有方法还会有自身用途的新方法和新变量。

2.3.1 student\_teacher.py

代码写入文件 /home/shiyanlou/student\_teacher.py：

#!/usr/bin/env python3

class Person(object):

"""

返回具有给定名称的 Person 对象

"""

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

self.name = name

def get\_details(self):

"""

返回包含人名的字符串

"""

return self.name

class Student(Person):

"""

返回 Student 对象，采用 name, branch, year 3 个参数

"""

def \_\_init\_\_(self, name, branch, year):

Person.\_\_init\_\_(self, name)

self.branch = branch

self.year = year

def get\_details(self):

"""

返回包含学生具体信息的字符串

"""

return "{} studies {} and is in {} year.".format(self.name, self.branch, self.year)

class Teacher(Person):

"""

返回 Teacher 对象，采用字符串列表作为参数

"""

def \_\_init\_\_(self, name, papers):

Person.\_\_init\_\_(self, name)

self.papers = papers

def get\_details(self):

return "{} teaches {}".format(self.name, ','.join(self.papers))

person1 = Person('Sachin')

student1 = Student('Kushal', 'CSE', 2005)

teacher1 = Teacher('Prashad', ['C', 'C++'])

print(person1.get\_details())

print(student1.get\_details())

print(teacher1.get\_details())

运行程序

在这个例子中你能看到我们是怎样在 Student 类和 Teacher 类中调用 Person 类的 \_\_init\_\_ 方法。

我们也在 Student 类和 Teacher 类中重写了 Person 类的 get\_details() 方法。

因此，当我们调用 student1 和 teacher1 的 get\_details() 方法时，使用的是各自类（Student 和 Teacher）中定义的方法。

2.4 多继承

一个类可以继承自多个类，具有父类的所有变量和方法，语法如下：

class MyClass(Parentclass1, Parentclass2,...):

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

Parentclass1.\_\_init\_\_(self)

Parentclass2.\_\_init\_\_(self)

...

...

这里没有提供额外的实例，将会在其他实验楼的项目实战中用到。

2.5 删除对象

现在我们已经知道怎样创建对象，现在我们来看看怎样删除一个对象。我们使用关键字 del 来做到这个。

>>> s = "I love you"

>>> del s

>>> s

Traceback (most recent call last):

File "<stdin>", line 1, in <module>

NameError: name 's' is not defined

del 实际上使对象的引用计数减少一，当对象的引用计数变成零的时候，垃圾回收器会删除这个对象。

2.6 属性（attributes）读取方法

在 Python 里请不要使用属性（attributes）读取方法（getters 和 setters）。如果你之前学过其它语言（比如 Java），你可能会想要在你的类里面定义属性读取方法。请不要这样做，直接使用属性就可以了，就像下面这样：

>>> class Student(object):

... def \_\_init\_\_(self, name):

... self.name = name

...

>>> std = Student("Kushal Das")

>>> print(std.name)

Kushal Das

>>> std.name = "Python"

>>> print(std.name)

Python

2.7 装饰器

你可能想要更精确的调整控制属性访问权限，你可以使用 @property 装饰器，@property 装饰器就是负责把一个方法变成属性调用的。

下面有个银行账号的例子，我们要确保没人能设置金额为负，并且有个只读属性 cny 返回换算人名币后的金额。

代码写入文件 /home/shiyanlou/property.py

#!/usr/bin/env python3

class Account(object):

"""账号类,

amount 是美元金额.

"""

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

self.\_\_amt = 0

self.rate = rate

@property

def amount(self):

"""账号余额（美元）"""

return self.\_\_amt

@property

def cny(self):

"""账号余额（人名币）"""

return self.\_\_amt \* self.rate

@amount.setter

def amount(self, value):

if value < 0:

print("Sorry, no negative amount in the account.")

return

self.\_\_amt = value

if \_\_name\_\_ == '\_\_main\_\_':

acc = Account(rate=6.6) # 基于课程编写时的汇率

acc.amount = 20

print("Dollar amount:", acc.amount)

print("In CNY:", acc.cny)

acc.amount = -100

print("Dollar amount:", acc.amount)

运行程序：

## 三、总结

实验知识点回顾：

类的定义

对象初始化

本实验我们了解了类的定义，类的继承以及多继承，并且最后我们还接触了装饰器这个概念，本质上，装饰器也是一种高阶函数。