

INDEX

Python教程

Python简介

安装Python

第一个Python程序

Python基础

函数

高级特性

函数式编程

模块

面向对象编程

类和实例

访问限制

继承和多态

获取对象信息

实例属性和类属性

面向对象高级编程

错误、调试和测试

IO编程

进程和线程

正则表达式

常用内建模块

常用第三方模块

virtualenv

图形界面

网络编程

电子邮件

访问数据库

Web开发


异步IO

实战

FAQ

期末总结

关于作者



廖雪峰北京 朝阳区

+ 加关注

类和实例

Reads: 51704882

面向对象最重要的概念就是类（Class）和实例（Instance），必须牢记类是抽象的模板，比如Student类，而实例是根据类创建出来的一个个具体的“对象”，每个对象都拥有相同的方法，但各自的数据可能不同。

仍以Student类为例，在Python中，定义类是通过 `class` 关键字：

```
class Student(object):
    pass
```

`class` 后面紧接着是类名，即 `Student`，类名通常是大写开头的单词，紧接着是 `(object)`，表示该类是从哪个类继承下来的，继承的概念我们后面再讲，通常，如果没有合适的继承类，就使用 `object` 类，这是所有类最终都会继承的类。

定义好了 `Student` 类，就可以根据 `Student` 类创建出 `Student` 的实例，创建实例是通过类名+()实现的：

```
>>> bart = Student()
>>> bart
<__main__.Student object at 0x10a67a590>
>>> Student
<class '__main__.Student'>
```

可以看到，变量 `bart` 指向的就是一个 `Student` 的实例，后面的 `0x10a67a590` 是内存地址，每个object的地址都不一样，而 `Student` 本身则是一个类。

可以自由地给一个实例变量绑定属性，比如，给实例 `bart` 绑定一个 `name` 属性：

```
>>> bart.name = 'Bart Simpson'
>>> bart.name
'Bart Simpson'
```

由于类可以起到模板的作用，因此，可以在创建实例的时候，把一些我们认为必须绑定的属性强制填写进去。通过定义一个特殊的 `__init__` 方法，在创建实例的时候，就把 `name`，`score` 等属性绑上去：

```
class Student(object):

    def __init__(self, name, score):
        self.name = name
        self.score = score
```

⚠ 注意：特殊方法“`__init__`”前后分别有两个下划线！！

注意到 `__init__` 方法的第一个参数永远是 `self`，表示创建的实例本身，因此，在 `__init__` 方法内部，就可以把各种属性绑定到 `self`，因为 `self` 就指向创建的实例本身。

有了 `__init__` 方法，在创建实例的时候，就不能传入空的参数了，必须传入与 `__init__` 方法匹配的参数，但 `self` 不需要传，Python解释器自己会把实例变量传进去：

```
>>> bart = Student('Bart Simpson', 59)
>>> bart.name
'Bart Simpson'
>>> bart.score
59
```

和普通的函数相比，在类中定义的函数只有一点不同，就是第一个参数永远是实例变量 `self`，并且，调用时，不用传递该参数。除此之外，类的方法和普通函数没有什么区别，所以，你仍然可以用默认参数、可变参数、关键字参数和命名关键字参数。

数据封装

面向对象编程的一个重要特点就是数据封装。在上面的 `Student` 类中，每个实例就拥有各自的 `name` 和 `score` 这些数据。我们可以通过函数来访问这些数据，比如打印一个学生的成绩：

```
>>> def print_score(std):
...     print('%s: %s' % (std.name, std.score))
...
>>> print_score(bart)
Bart Simpson: 59
```

但是，既然 `Student` 实例本身就拥有这些数据，要访问这些数据，就没有必要从外面的函数去访问，可以直接在 `Student` 类的内部定义访问数据的函数，这样，就把“数据”给封装起来了。这些封装数据的函数是和 `Student` 类本身是关联起来的，我们称之为类的方法：

```
class Student(object):

    def __init__(self, name, score):
        self.name = name
        self.score = score

    def print_score(self):
        print('%s: %s' % (self.name, self.score))
```

要定义一个方法，除了第一个参数是 `self` 外，其他和普通函数一样。要调用一个方法，只需要在实例变量上直接调用，除了 `self` 不用传递，其他参数正常传入：

```
>>> bart.print_score()
Bart Simpson: 59
```

这样一来，我们从外部看 `Student` 类，就只需要知道，创建实例需要给出 `name` 和 `score`，而如何打印，都是在 `Student` 类的内部定义的，这些数据和逻辑被“封装”起来了，调用很容易，但却不用知道内部实现的细节。

封装的另一个好处是可以给 `Student` 类增加新的方法，比如 `get_grade`：

```
class Student(object):
    ...

    def get_grade(self):
        if self.score >= 90:
            return 'A'
        elif self.score >= 60:
            return 'B'
        else:
            return 'C'
```

同样的，`get_grade` 方法可以直接在实例变量上调用，不需要知道内部实现细节：

```
# -*- coding: utf-8 -*-

class Student(object):
    def __init__(self, name, score):
        self.name = name
        self.score = score

    def get_grade(self):
        if self.score >= 90:
            return 'A'
        elif self.score >= 60:
            return 'B'
        else:
            return 'C'

lisa = Student('Lisa', 99)
bart = Student('Bart', 59)
print(lisa.name, lisa.get_grade())
print(bart.name, bart.get_grade())
```

▶ Run

小结

类是创建实例的模板，而实例则是一个一个具体的对象，各个实例拥有的数据都互相独立，互不影响；

方法就是与实例绑定的函数，和普通函数不同，方法可以直接访问实例的数据；

通过在实例上调用方法，我们就直接操作了对象内部的数据，但无需知道方法内部的实现细节。

和静态语言不同，Python允许对实例变量绑定任何数据，也就是说，对于两个实例变量，虽然它们都是同一个类的不同实例，但拥有的变量名称都可能不同：

```
>>> bart = Student('Bart Simpson', 59)
>>> lisa = Student('Lisa Simpson', 87)
>>> bart.age = 8
>>> bart.age
8
>>> lisa.age
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: 'Student' object has no attribute 'age'
```

参考源码

[student.py](#)

读后有收获可以支付宝请作者喝咖啡，读后有疑问请加微信群讨论：



还可以分享给朋友：

分享到微博

◀ Previous Page

Next Page ▶

Comments

Make a comment

Sign in to make a comment



Feedback
License