面向对象编程

（下载幻灯片和 .py 文件，请跟随！）

6.0001 第 8 讲

对象

Python 支持多种不同类型的数据

1234　　3.14159　　“你好”　　[1, 5, 7, 11, 13]

{“CA”：“加利福尼亚”，“MA”：“马萨诸塞”}

每个都是一个对象，每个对象都有：

- 一种类型
- 内部数据表示（原始或复合）
- 一组与对象交互的程序

一个对象是一个类型的一个实例

- 1234 是 int 的一个实例
- “hello”是字符串的一个实例

面向对象编程（开放）

Python 中的一切都是一个对象（并且有一个类型）

可以创建某种类型的新对象

可以操纵物体

可以破坏物体

- 明确使用 del 或只是“忘记”它们
- python 系统将回收被破坏或无法访问的对象 称为“垃圾收集”

什么是对象？

对象是一种数据抽象
捕捉…

(1)内部表示 ·通过数据属性

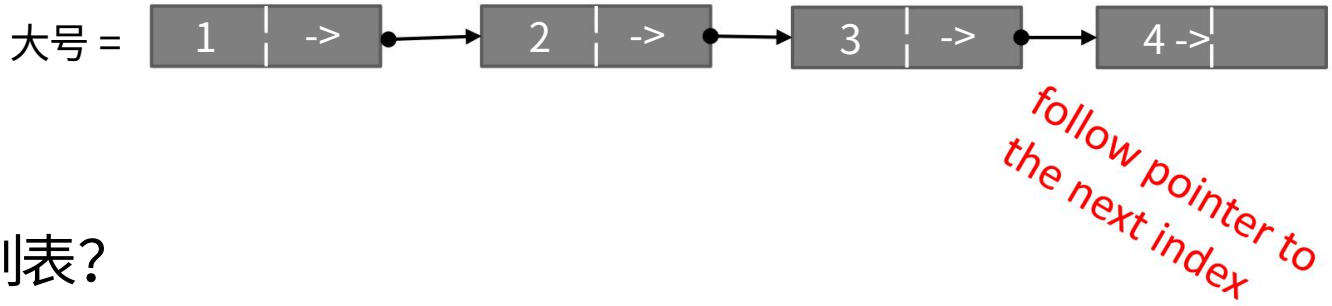
(2)与对象交互的接口 ·
通过方法（也称为过程/函数）

·定义行为但隐藏实施

示例：

[1,2,3,4] 具有类型列表

列表如何在内部表示?单元格链表



如何操作列表?

- L[i]、L[i:j]、+
- len()、min()、max()、del(L[i])
- L.append(),L.extend(),L.count(),L.index(),
  L.insert(),L.pop(),L.remove(),L.reverse(),L.sort()

内部代表应该是私人的

如果你直接操纵内部表示,正确的行为可能会受到影响

面向对象的优势

将数据捆绑到包中,并通过定义明确的接口对它们进行处理

分而治之的发展

- 分别实现和测试每个类的行为
- 增加的模块化降低了复杂性

类使重用代码变得容易

- 许多 Python 模块定义了新的类
- 每个类都有一个单独的环境（函数名没有冲突）
- 继承允许子类重新定义或扩展超类行为的选定子集

创建和使用您的拥有类的类型

区分创建类和使用类的实例

创建类涉及

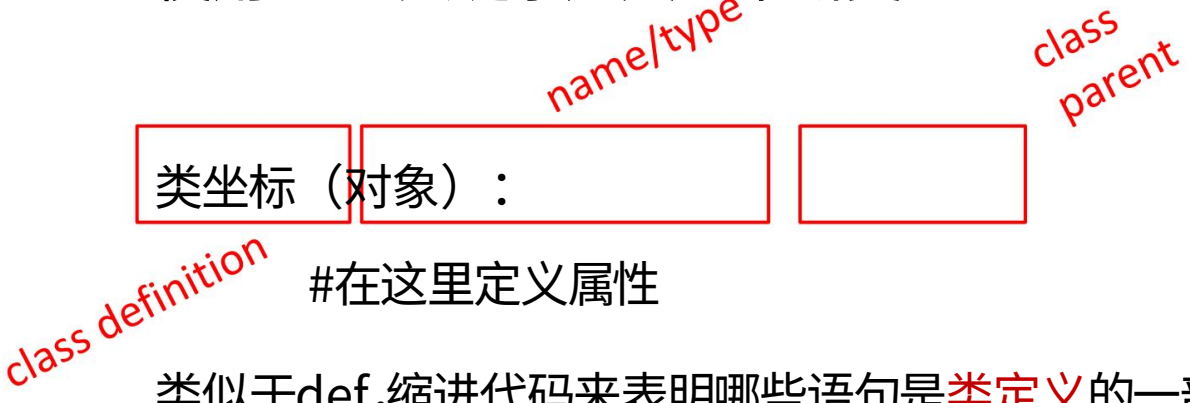
- 定义类名
- 定义类属性
- 例如,有人编写代码来实现一个列表类

使用类涉及

- 创建对象的新实例
- 对实例进行操作
- 例如,L=[1,2] 和 len(L)

定义你自己的类型

使用class关键字定义一个新类型



类似于def,缩进代码来表明哪些语句是类定义的一部分

对象一词的意思是Coordinate是一个Python对象并且继承了它的所有属性（继承下一讲） · Coordinate是对象的子类

- object 是 Coordinate 的超类

什么是属性？

“属于”类的数据和过程

数据属性

- 将数据视为构成类的其他对象
- 例如，坐标由两个数字组成

方法（过程属性）

- 将方法视为仅适用于此类的函数
- 如何与对象交互
- 例如，您可以定义两个坐标对象之间的距离，但两个列表对象之间的距离没有意义

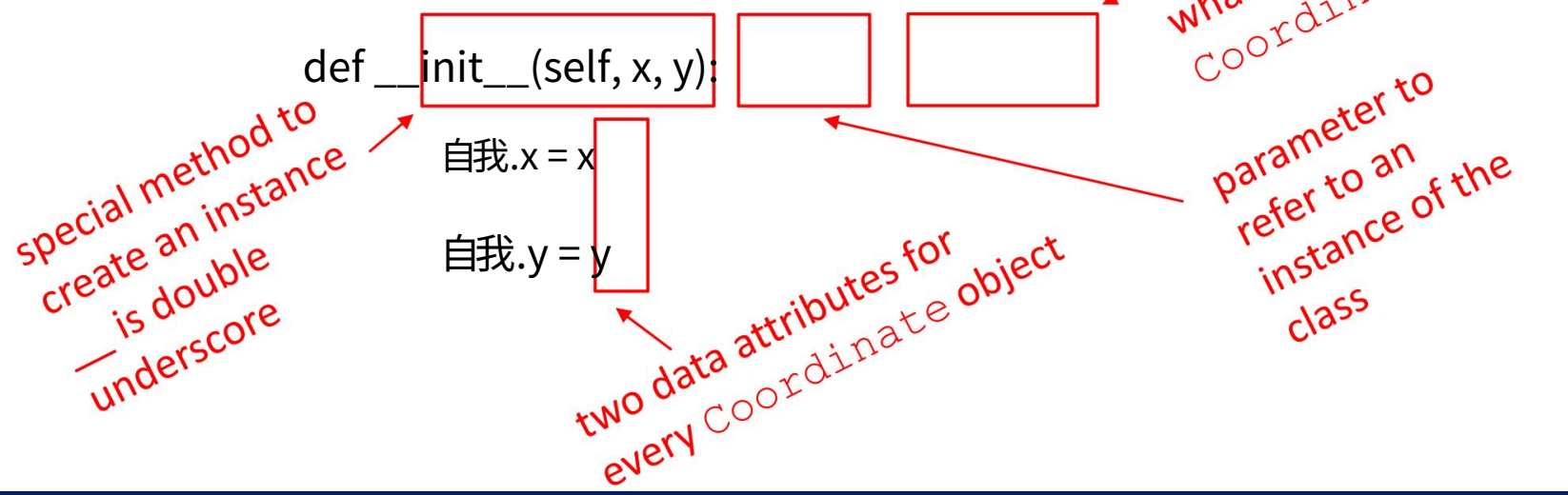
定义如何创建类的实例

首先必须定义如何创建对象的实例

使用一种叫做__init__的特殊方法来初始化一些数据属性

类坐标（对象）：

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



实际创建一个类的实例

```
c = 坐标（3,4）
原点 = 坐标（0,0）
打印（CX）
打印（原点.x）
```

create a new object of type Coordinate and pass in 3 and 4 to the __init__

use the dot to access an attribute of instance c

实例的数据属性称为实例变量

不要为 self 提供参数，Python 会自动执行此操作

什么是方法?

过程属性,就像一个只适用于这个类的函数

Python 总是将对象作为第一个参数传递

- 约定是使用self作为所有方法的第一个参数的名称

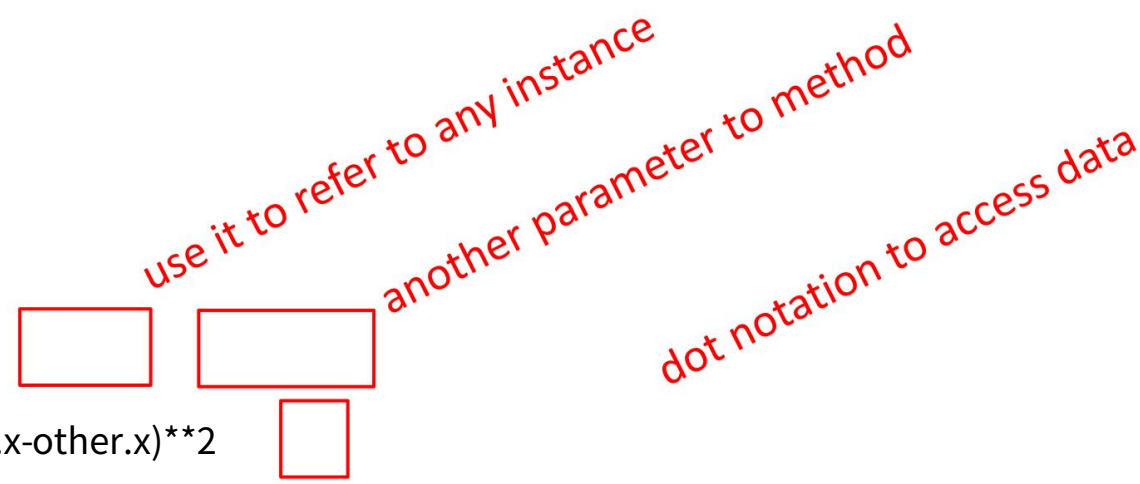
“。”运算符用于访问任何属性

- 对象的数据属性
- 对象的方法

定义一个方法

坐标类

```
类坐标（对象）：
    def __init__(self, x, y):
        自我.x = x
        自我.y = y
    def 距离（自己,其他）：
        x_diff_sq = (self.x-other.x)**2
        y_diff_sq = (self.y-other.y)**2
        返回 (x_diff_sq + y_diff_sq)**0.5
```



除了 self 和 dot notation 之外,方法的行为只是类似函数（获取参数,执行操作,返回）

如何使用方法

```
def distance(self, other): # 代码在这里
```

method def

使用类： 常规方式

```
c = 坐标（3,4）
零 = 坐标 (0,0)
打印（c.距离（零））
```

object to call method on

name of method

parameters not including `self` (`self` is implied to be `c`)

相当于

```
c = 坐标（3,4）
零 = 坐标 (0,0)
打印（坐标。距离（c,零））
```

name of class

name of method

parameters, including an object to call the method on, representing `self`

打印代表一个东西

```
>>> c = 坐标（3,4） >>> 打印（c）

<__main__.坐标对象在0x7fa918510488>
```

默认情况下无信息的打印表示

为一个类定义一个__str__ 方法

Python 在类对象上与 print 一起使用时调用 __str__ 方法

你选择它做什么!说当我们打印一个 Coordinate 对象时,想要显示

```
>>> 打印（c）
<3,4>
```

定义你自己的印刷品方法

类坐标（对象）：

```
def __init__(self, x, y):
        自我.x = x
        自我.y = y
def 距离（自己,其他）：
        x_diff_sq = (self.x-other.x)**2 y_diff_sq = (self.y-other.y)**2
        return (x_diff_sq + y_diff_sq)**0.5 def __str__(self): return
          <   +str( self.x)+    ,    +str(self.y)+    >
```

name of special method

must return a string

包扎你的头

围绕类型和类别

可以询问对象实例的类型>>> c = Coordinate(3,4)
>>> print(c) <3,4> >>> print(type(c)) <class __main__.Coordinate> 这是有道理的,因为

return of the __str__ method

the type of object c is a class Coordinate

>>> print(Coordinate) <class __main__.Coordinate> >>> print(type(Coordinate))
<type type >

a Coordinate is a class

a Coordinate class is a type of object

使用 isinstance() 检查对象是否为坐标

>>> 打印（isinstance（c,坐标））
真的

特殊操作员

+、 -、==、<、>、len()、print 等

https://docs.python.org/3/reference/datamodel.html#basic-customization

像打印一样,可以覆盖这些以与您的班级一起工作

在之前/之后用双下划线定义它们

__add__(self, other) self + other

__sub__(self, other) self - other

__eq__(self, other) self == other

__lt__ （自我,其他)自我<其他

__len__(self) 打 镜头（自我）

__str__(self) ... 和其即自我

他

示例:分数

创建一个新类型来将数字表示为分数

内部表示是两个整数

- 分子
- 分母

接口又名方法又名如何与分数对象交互 ·加法,减法 ·打印表示,转换为浮点数 ·反转分数

代码在讲义中,检查一下!

面向对象的力量

将共享的对象捆绑在一起
- 共同的属性
  和
- 对这些属性进行操作的过程

使用抽象来区分如何实现对象与如何使用对象

构建对象抽象层，继承其他类对象的行为

在 Python 的基本类之上创建我们自己的对象类

6.0001 计算机科学和 Python 编程简介

2016 年秋季