Python系列分享会——第二讲 函数&类与对象&魔法方法

金融科技协会 2020年10月23日

目录

[一、函数 3](#_Toc55074843)

[1.0 什么是函数 callable() 3](#_Toc55074844)

[1.1 函数定义 3](#_Toc55074845)

[1.1.1 函数头：包括函数名和参数（形式参数） 4](#_Toc55074846)

[1.1.2 调用函数 4](#_Toc55074847)

[1.1.3 返回多个值 4](#_Toc55074848)

[1.1.4 定义空函数与pass的使用 5](#_Toc55074849)

[1.2 函数参数 5](#_Toc55074850)

[1.2.1 参数可变吗？（考虑字符串 str 和列表 list） 5](#_Toc55074851)

[1.2.2 位置参数 6](#_Toc55074852)

[1.2.3关键字参数 6](#_Toc55074853)

[1.2.4 含默认值的参数 7](#_Toc55074854)

[1.2.5 收集参数 7](#_Toc55074855)

[1.3 函数文档 8](#_Toc55074856)

[1.3.1 help() 8](#_Toc55074857)

[1.3.2 *doc* 9](#_Toc55074858)

[1.4 函数作用域 9](#_Toc55074859)

[1.4.1 什么是函数作用域 9](#_Toc55074860)

[1.4.2 全局作用域与局部作用域的区别 9](#_Toc55074861)

[1.4.3 在局部作用域中，全局变量遮盖的问题 10](#_Toc55074862)

[1.4.4 全局变量、外部非全局变量，在局部作用域中的重新关联 11](#_Toc55074863)

[1.5 递归调用 12](#_Toc55074864)

[1.5.1 递归是什么 12](#_Toc55074865)

[1.5.2 递归在二分查找中的应用 12](#_Toc55074866)

[二、类与对象 12](#_Toc55074867)

[2.1 基本概念 12](#_Toc55074868)

[2.1.1什么是类？什么是对象？ 12](#_Toc55074869)

[2.1.2什么是子类？什么是超类？ 12](#_Toc55074870)

[2.2 定义类 13](#_Toc55074871)

[2.2.1 定义长方形和长方体类 13](#_Toc55074872)

[2.2.2 类的相关介绍 14](#_Toc55074873)

[2.2.3 什么是封装 14](#_Toc55074874)

[2.2.4 什么是多态 14](#_Toc55074875)

[2.2.5 什么是继承 15](#_Toc55074876)

[2.3 类的命名空间 15](#_Toc55074877)

[2.3.1 类体内函数体外的变量 15](#_Toc55074878)

[2.3.2该类变量的应用 15](#_Toc55074879)

[2.4 如何修改属性（数据成员）的值 16](#_Toc55074880)

[2.4.1 直接修改 16](#_Toc55074881)

[2.4.2 通过成员函数修改 16](#_Toc55074882)

[2.5 多重继承 16](#_Toc55074883)

[2.6抽象基类 17](#_Toc55074884)

[2.7 hasattr() setattr() getattr() 17](#_Toc55074885)

[三、魔法方法 18](#_Toc55074886)

[3.1 概念介绍：“方法”VS“函数”VS“属性” 18](#_Toc55074887)

[3.2基本的序列（字符串、列表、元祖）和映射（字典）的协议（方法） 19](#_Toc55074888)

[3.3静态方法和类方法 21](#_Toc55074889)

[3.3.1 概念介绍 21](#_Toc55074890)

[3.3.2类方法: 22](#_Toc55074891)

[3.3.3静态方法： 22](#_Toc55074892)

[3.4类和对象的魔法方法 23](#_Toc55074893)

# 一、函数

## 1.0 什么是函数 callable()

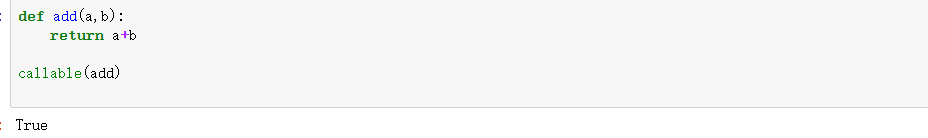
callable() 函数用于检查一个对象是否是可调用的。可调用返回 True，否则返回 False。

对于函数、lambda 函式、 类以及实现了 **call** 方法的类实例, 它都返回 True。（如图1）

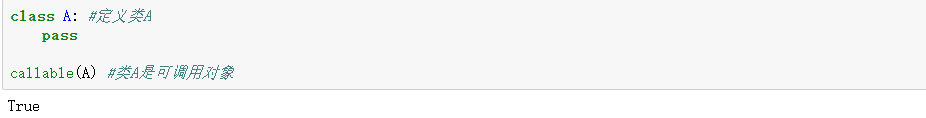
1603973481

图表 1 callable()方法用法

类对象都是可被调用对象，类的实例对象是否可调用对象，取决于类是否定义了**call**方法。（如图2-5）



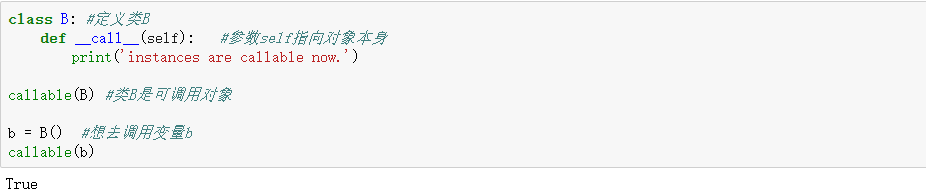
图表 2 add函数可以被成功调用



图表 3 类A是可调用对象



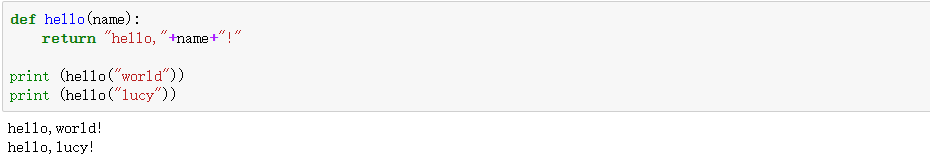
图表 4 没有实现call,变量不可被调用



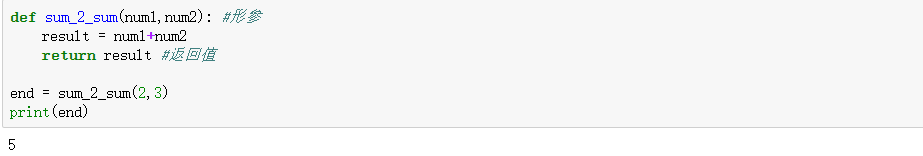
图表 5 实现call 可以调用变量b

## 1.1 函数定义

运行这些代码后，将有一个名为hello的新函数。它返回一个字符串，其中包含向唯一参数指定的人发出的问候语。你可像使用内置函数那样使用这个函数。（如图6-7）



图表 6 自定义函数hello



图表 7 自定义函数sum\_2\_sum

**即，定义函数包括以下要素：**

### 1.1.1 函数头：包括函数名和参数（形式参数）

1. 函数名

def是英文define的缩写 函数名称应该能够表达函数封装代码的功能，方便后续的调用注意，在创建函数时，即使函数不需要参数，也必须保留一对空的“()”，否则 Python 解释 器将提示“invaild syntax”错误。

2. 参数（形参和实参）

形参：定义函数时，小括号中的参数，是用来接收参数用的，在函数内部作为变量使用

实参：调用函数时，小括号中的参数，是用来把数据传递到函数内部用的

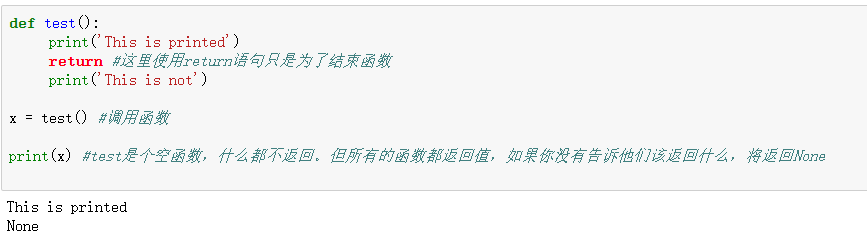
3. 函数的返回值

**return的作用：**

在函数中使用return可以从函数返回结果 数学意义上的函数总是返回根据参数计算得到的结果。

函数体中 return 语句有指定返回值时返回的就是其值。在Python中，有些函数什么都不返回。

函数体中没有 return 语句，或有return语句，但没有在后面指定值，函数运行结束会隐含返回一个 None 作为返回值。（如图8）

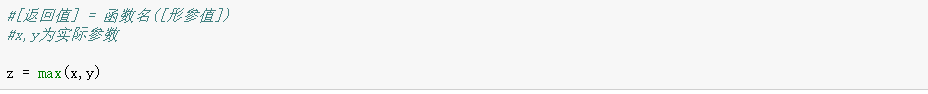


图表 8 为了结束函数的return用法

### 1.1.2 调用函数

调用函数也就是执行函数，如果把创建的函数理解为一个具有某种用途的工具，那么调用函数就相当于使用该工具。

函数调用的基本语法格式如下所示：（如图9）



图表 9 函数调用的基本语法格式

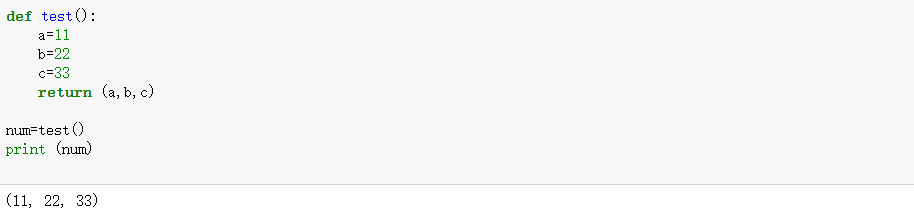
### 1.1.3 返回多个值

函数可以返回多个值吗？答案是肯定的。但其实这只是一种假象，Python函数返回的仍然是单一值，返回多值其实就是返回一个tuple，但写起来更方便。

无论定义的是返回什么类型，return 只能返回单值，但值可以存在多个元素；

return [1,3,5] 是指返回一个列表，是一个列表对象，1,3,5 分别是这个列表的元素；

return 1,3,5 看似返回多个值，隐式地被Python封装成了一个元祖返回。（如图10）



图表 10 函数返回多个值

### 1.1.4 定义空函数与pass的使用

Python pass 是空语句，是为了保持程序结构的完整性。

pass 不做任何事情，一般用做占位语句。 如果想定义一个什么事也不做的空函数，可以用pass语句。（如图11）

1603974132(1)

图表 11 空函数的使用

## 1.2 函数参数

### 1.2.1 参数可变吗？（考虑字符串 str 和列表 list）

**为何要修改参数**

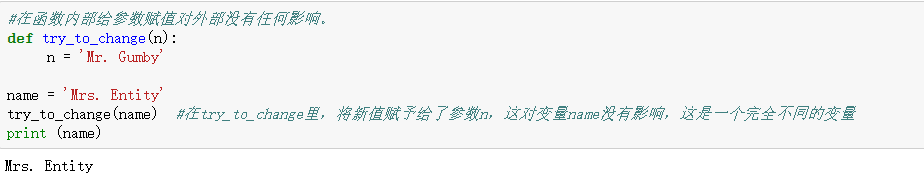
在提高程序的抽象程度方面，使用函数来修改数据结构（如列表或字典）是一种不错的方式。

可以将python中常见数据类型按照可变与不可变大致分为两类

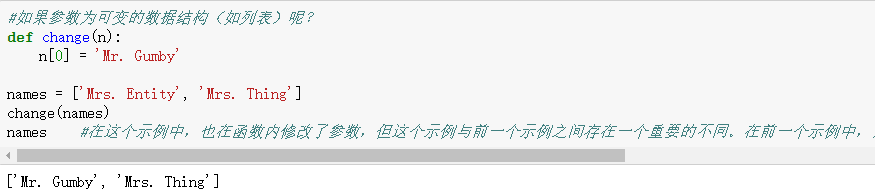
不可变数据（四个）：Number（数字）、String（字符串）、Tuple（元组）、Sets（集合）；

可变数据（两个）：List（列表）、Dictionary（字典）

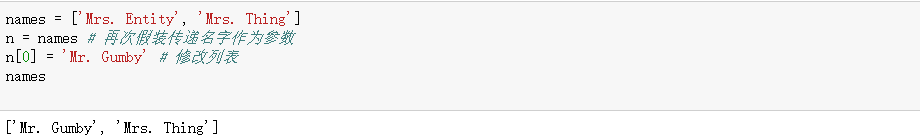
字符串（以及数和元组）是不可变的（immutable），这意味着你不能修改它们（即只能替换为新值），也就是说丢弃原来的存储空间，将变量名链接到新的空间中。而像list和dict是支持增删改的。（如图12-14）



图表 12 字符串替换为新值



图表 13 可变数据列表

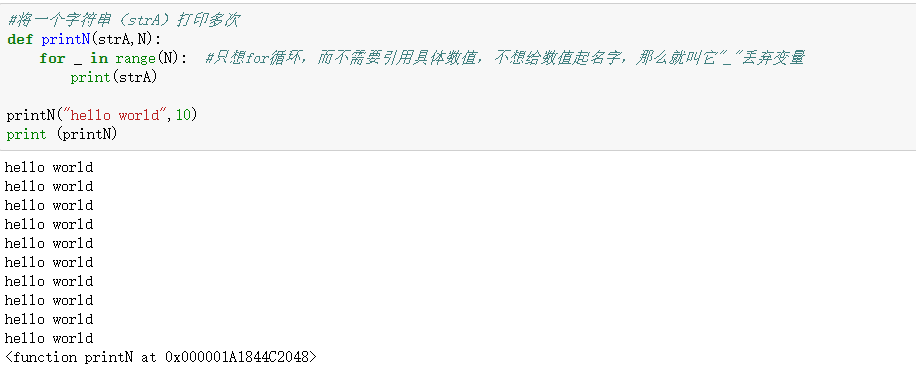


图表 14 可变数据列表

### 1.2.2 位置参数

位置参数要求参数的顺序和个数要和函数定义中一致，比如funcB(100, 99)和funcB(99, 100)的执行结果是不一样的。

在下图调用printN时，我们可以按照位置参数的规则传递实参，即实参必须和函数头中定义的形参在顺序、个数、类型上匹配。（如图15）



图表 15 位置参数实参必须和形参在顺序个数类型上匹配

### 1.2.3关键字参数

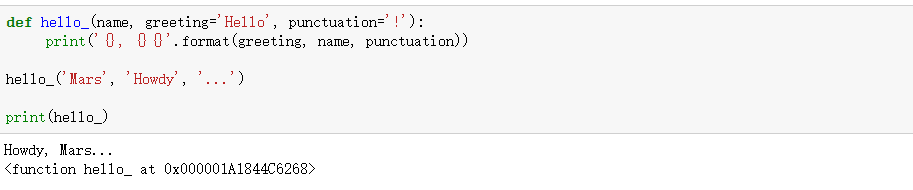
除了使用位置参数的规则传递参数外，也可以使用关键字参数的规则传递参数。也就是在调用函数的时候，明确指定参数值付给那个形参。

有时候，参数的排列顺序可能难以记住，尤其是参数很多时。为了简化调用工作，可指定参数的名称。（如图16）



图表 16 关键字参数的应用

另外，在函数调用中，可以混合使用基于位置匹配的参数和关键字参数，前提是先给出固定位置的参数，否则解释器将不知道它们是哪个参数（即不知道参数对应的位置）。比如图17-18。



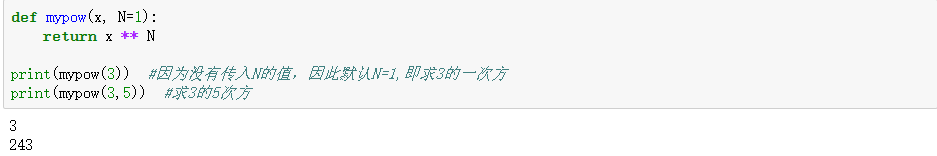
图表 17 混合使用位置参数和关键字参数



图表 18 混合使用位置参数和关键字参数

### 1.2.4 含默认值的参数

关键字参数最大的优点在于，可以指定默认值。 带有默认值的参数一定是右连续的，有默认值的参数的右边的参数不能没有默认值。（如图18）

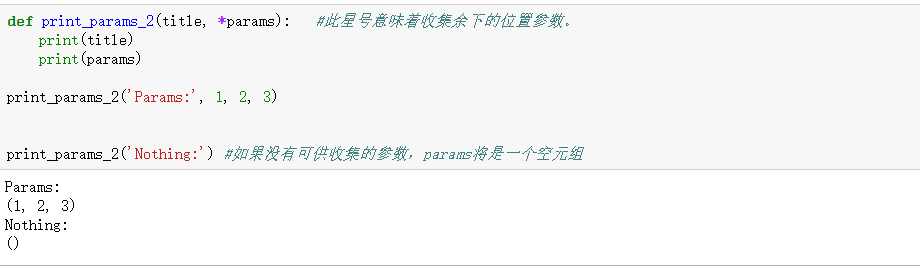


图表 19 含默认值的关键字参数

### 1.2.5 收集参数

**收集位置参数的\***

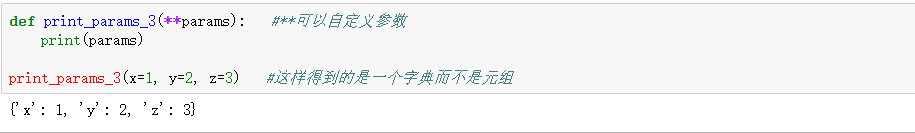
在参数前加了一个 \* 号，函数可以接收零个或多个值作为参数。返回结果是一个元组。传递零个参数时函数并不报错，而是返回一个空元组。但以上这种方法也有局限性，它不能收集关键字参数。（如图20）



图表 20 收集位置参数的\*

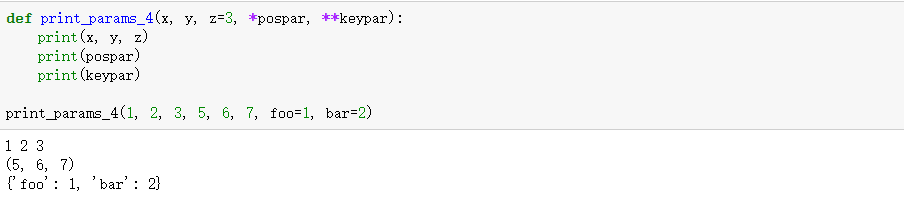
**收集关键字参数的\*\***

对关键字参数进行收集的另一种 收集参数 机制：使用两个星号 ( \*\* ) ，用法同上。最后返回一个以参数名为键、参数值为键值的字典。（如图21）



图表 21 收集关键字参数的\*\*

和 \* 是可以一起使用的，返回特定的结果。（如图22）



图表 22 混合使用收集位置参数\*和收集关键字参数\*\*

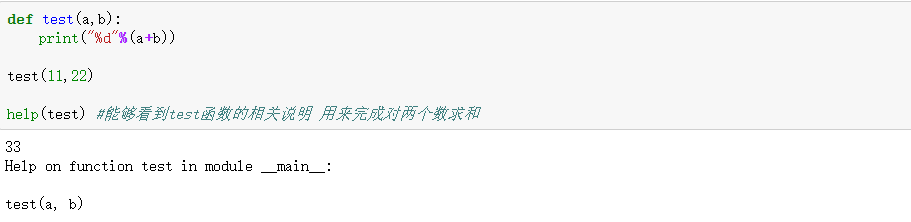
## 1.3 函数文档

### 1.3.1 help()

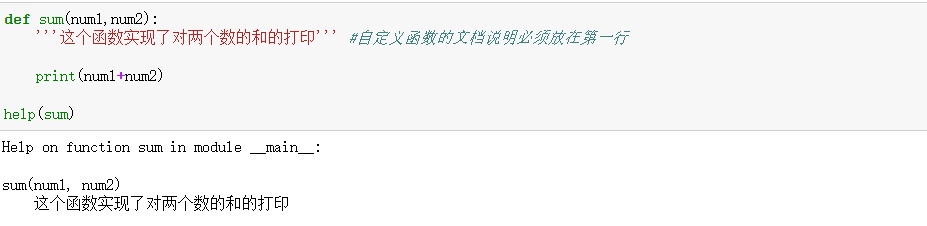
只要是函数，都可以用help（）来查看函数的文档，也就是函数的说明书

如果是自定义函数，如何查看它的文档？在def后面首行使用''' '''三个单引号对自定义函数进行说明，之后使用help() 就可以查看自定义文档的自定义说明了。

说明要包括的内容:1. 对函数的具体描述 2. 对函数的参数的说明（如图23-24）



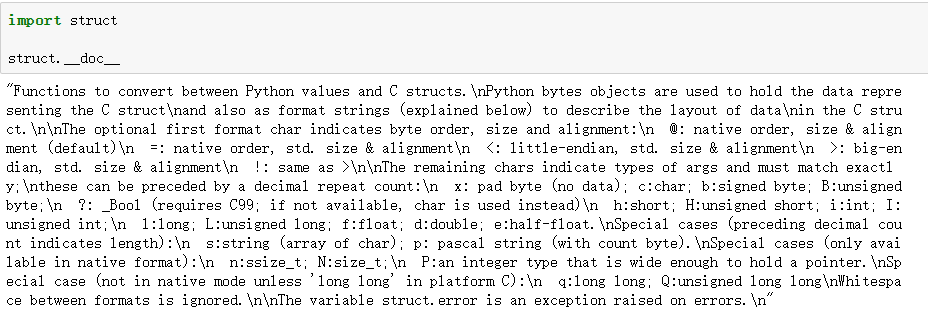
图表 23 help()查看函数文档



图表 24 自定义函数的文档说明必须放在定义后的第一行

### 1.3.2 doc

**doc**是每个对象都有的属性，其存放了对象文档。（如图25）



图表 25 doc查看对象文档

## 1.4 函数作用域

### 1.4.1 什么是函数作用域

变量到底是什么呢？可将其视为指向值的名称。因此，执行赋值语句x=1后，名称x指向值1。这几乎与使用字典一样(字典中的键指向值)，只是你使用的是"看不见"的字典。实际上，这种解释已经离真相不远。有一个名为vars的内置函数，它返回这个不可见的字典

Python 中，程序的变量并不是在哪个位置都可以访问的，访问权限决定于这个变量是在哪里赋值的。

变量的作用域决定了在哪一部分程序可以访问哪个特定的变量名称。

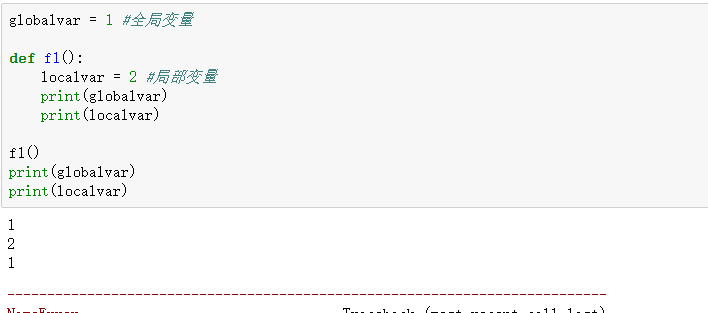
### 1.4.2 全局作用域与局部作用域的区别

**局部变量**

在函数内部被定义的变量，只能在函数内部访问。

**全局变量**

在所有函数之外创建的变量，可以被所有的函数访问。



图表 26 局部变量lovalvar不能在函数外访问

globalvar是全局变量，localvar是局部变量，globalvar可以在函数中访问，但是localvar却不能在函数外访问，会报错。（如图26）

### 1.4.3 在局部作用域中，全局变量遮盖的问题

如果有一个局部变量或参数与你要访问的全局变量同名，就无法直接访问全局变量，因为它被局部变量遮住了。

全局变量与局部变量同名：如果局部变量与全局变量同名，那么函数中将优先使用的是局部变量的值，而不是使用全局变量。（如图27）

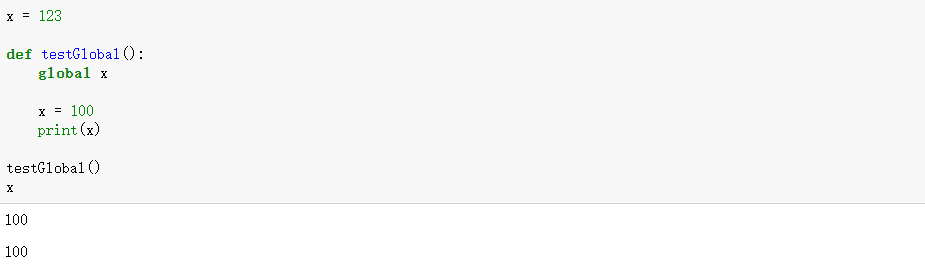


图表 27 局部变量与全局变量同名，优先访问局部变量的值

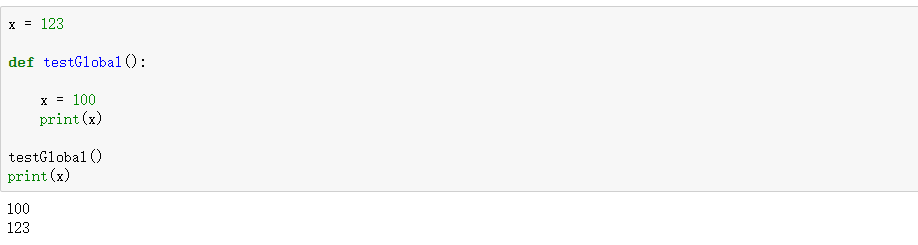
python访问变量的顺序： 当前作用域局部变量->外层作用域变量->当前模块中的全局变量->python内置变量.

**global**

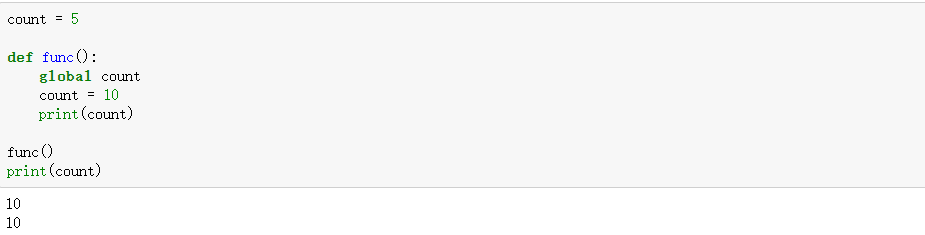
global 可以将局部变量的作用域改为全局。 如果需要，可使用函数global来访问全局变量。这个函数类似于vars，返回一个包含全局变量的字典。（locals返回一个包含局部变量的字典。）（如图28-30）



图表 28 global实现在函数内部访问全局变量



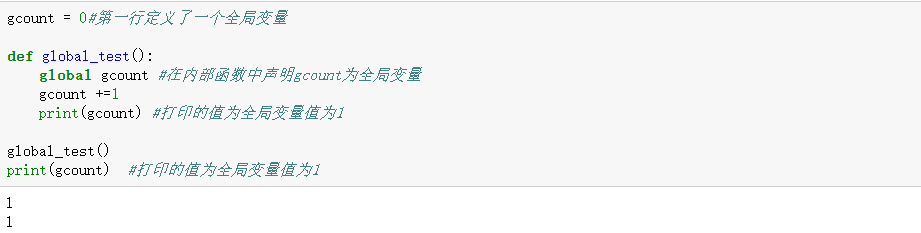
图表 29 没有实现global依然返回全局变量的值



图表 30 使用global在函数内部修改全局变量count的值

### 1.4.4 全局变量、外部非全局变量，在局部作用域中的重新关联

全局变量在局部作用域中的重新关联。（如图31）



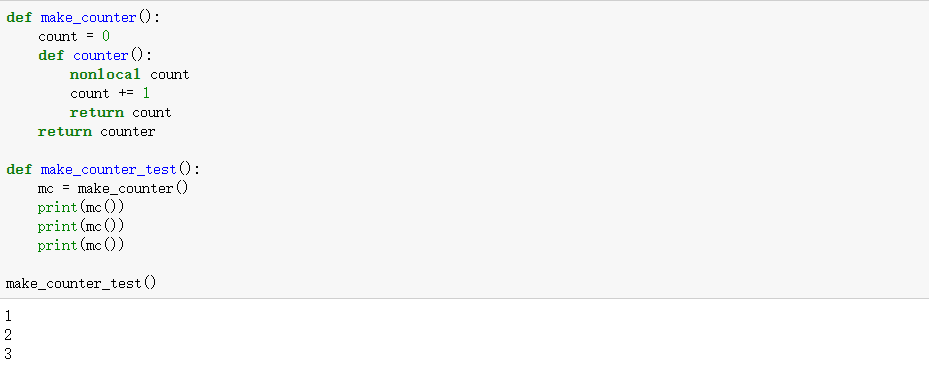
图表 31 全局变量在局部作用域中的重新关联

外部非全局变量在局部作用域中的重新关联

nonlocal关键字用来在函数或其他作用域中使用外层(非全局)变量

nonlocal 适用于在局部函数中调用另一个局部函数的局部变量， 把最内层的局部变量设置成外层局部或者其它局部可用，但不可在全局使用

简而言之，nonlocal的作用：在一个函数中调用另一个函数的私有化变量



图表 32 外部非全局变量在局部作用域中的重新关联

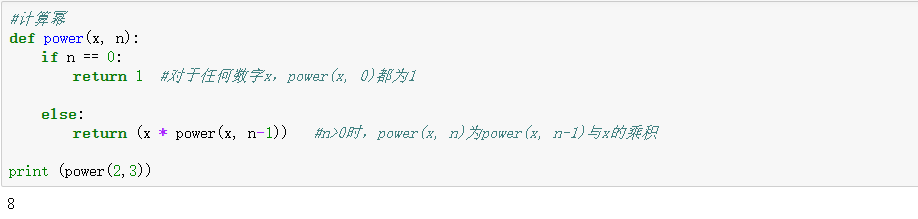
## 1.5 递归调用

### 1.5.1 递归是什么

递归意味着引用（这里是调用）自身。可使用递归完成的任何任务都可使用循环来完成，但有时使用递归函数的可读性更高。

**基线条件（针对最小的问题）**：满足这种条件时函数将直接返回一个值。

**递归条件：**包含一个或多个调用，这些调用旨在解决问题的一部分。（如图33）

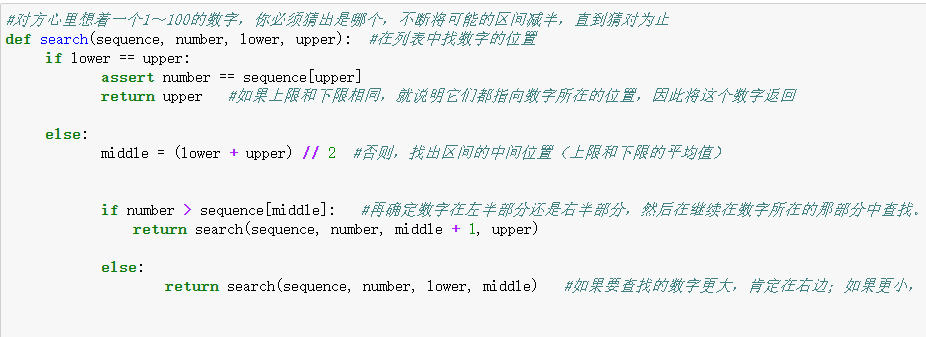


图表 33 计算幂的递归调用

### 1.5.2 递归在二分查找中的应用

如果上限和下限相同，就说明它们都指向数字所在的位置，因此将这个数字返回。

否则，找出区间的中间位置（上限和下限的平均值），再确定数字在左半部分还是右半部分。然后在继续在数字所在的那部分中查找。（如图34）



图表 34 递归在二分查找中的应用

# 二、类与对象

## 2.1 基本概念

### 2.1.1什么是类？什么是对象？

对象是对客观事物的抽象，类是对对象的抽象。两者的关系：对象是类的实例，类是对象的模板。

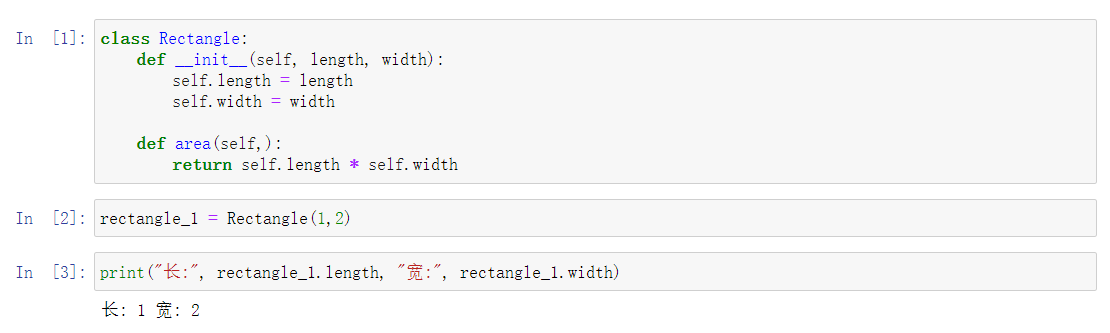
### 2.1.2什么是子类？什么是超类？

被继承的类一般称为“超类”或“父类”，继承的类称为“子类”。鸟类是一个非常通用（抽象）的类，它有多个子类：你看到的那只鸟可能属于子类“云雀”。你可将“鸟类”视为由所有鸟组成的集合，而“云雀”是其一个子集。一个类的对象为另一个类的对象的子集时，前者就是后者的子类。因此“云雀”为“鸟类”的子类，而“鸟类”为“云雀”的超类。

## 2.2 定义类

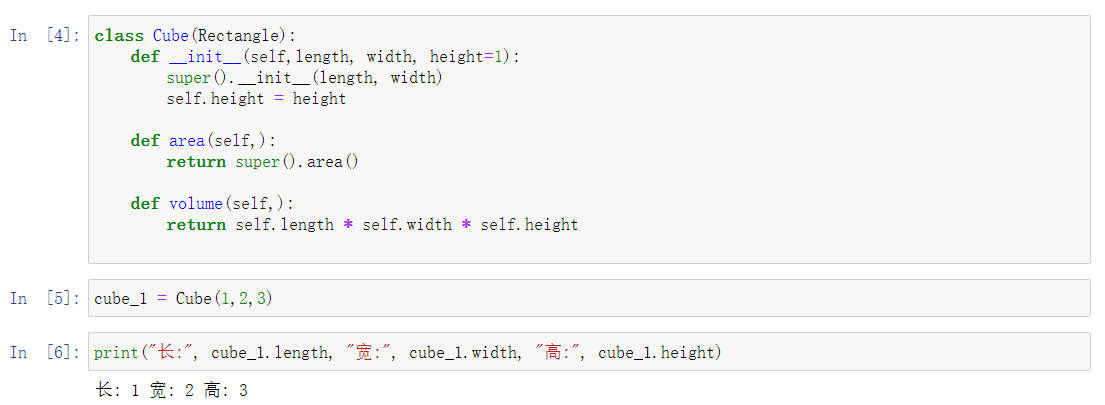
### 2.2.1 定义长方形和长方体类

长方形类，数据成员为长、宽。（如图35）



图表 35定义长方形

长方形木板类（继承长方形），数据成员为长、宽、厚度，设定厚度的默认值为 1。在定义长方形木板时，由于其本身含有长方形的一些属性，所以我们选择继承长方形类，并在此基础上添加只属于长方形木板类的特有属性和成员函数。（如图37）



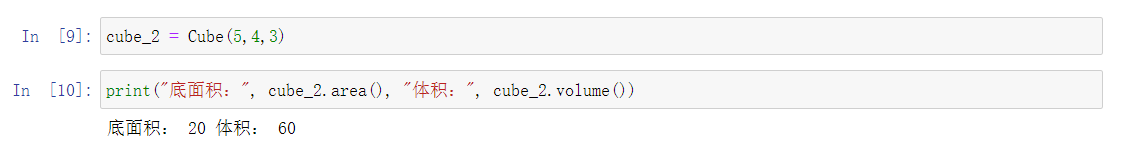
图表 36定义长方形模板类

长方形类的成员函数为计算面积。（如图38）



图表 37长方形成员函数

长方形木板类的成员函数为计算底面积和计算体积。（如图39）



图表 38长方形模木板类成员函数

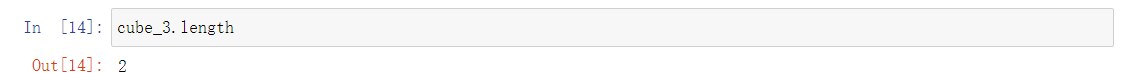
### 2.2.2 类的相关介绍

对象的生成：实例化对象 = 类名(参数)。如图（40-42）



图表 39对象的生成

属性: 数据成员。（如图6）



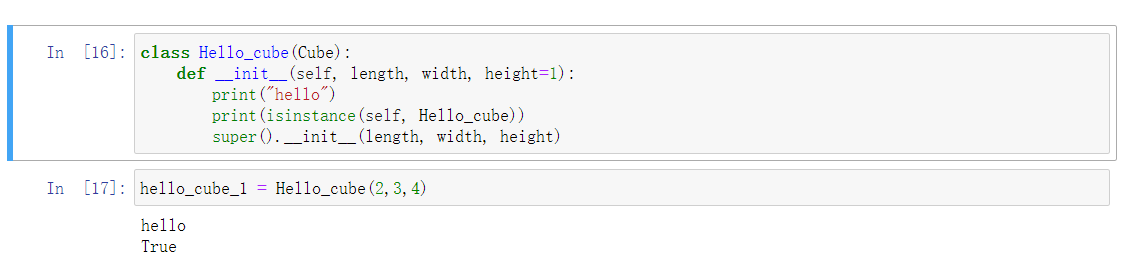
图表 40属性：数据成员

普通成员函数是实现类或者对象中的功能的函数，只有在调用时才会作用。（如图8）



图表 41普通成员函数

构造函数每次实例化类时都会执行一次。super().\_\_ init \_\_: 继承父类的init方法。self: 指向对象本身。（如图43）



图表 42构造函数

对象的方法(成员函数的调用)：对象名.成员函数。（如图44）



图表 43函数成员的调用

### 2.2.3 什么是封装

指的是向外部隐藏不必要的细节。

### 2.2.4 什么是多态

即便不知道变量指向的是哪种对象，也能够对其执行操作，且操作的行为将随对象所属的类型（类）而异。（如图45）



图表 44多态

### 2.2.5 什么是继承

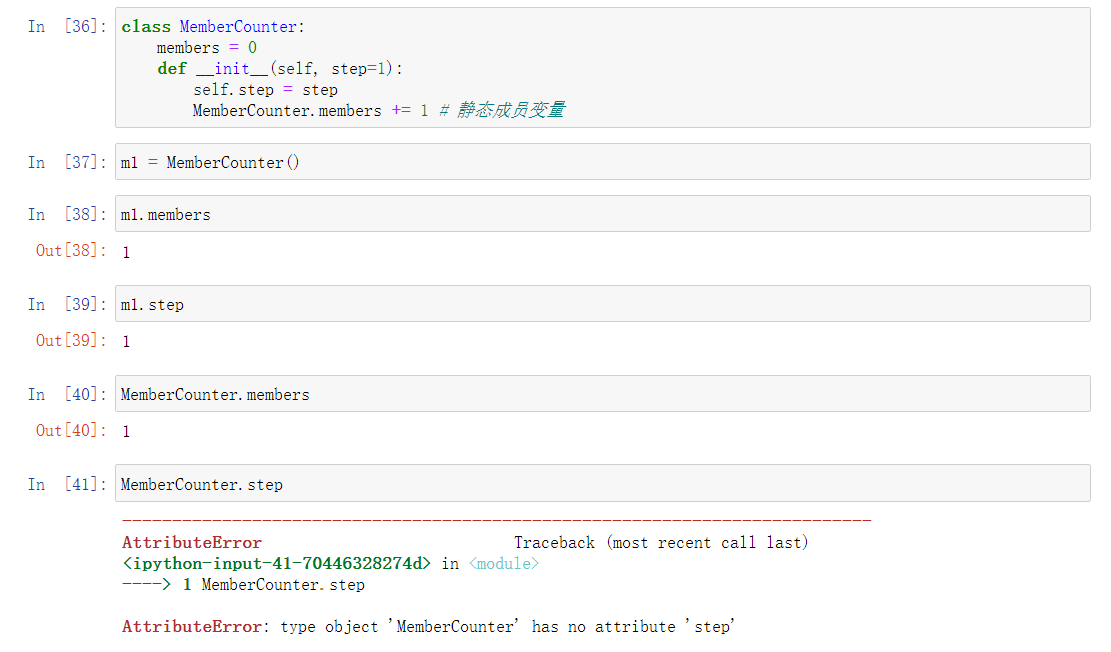
它可以使用现有类的所有功能，并在无需重新编写原来的类的情况下对这些功能进行扩展。

## 2.3 类的命名空间

在class语句中定义的代码都是在一个特殊的命名空间（类的命名空间）内执行的，而类的所有成员都可访问这个命名空间。

### 2.3.1 类体内函数体外的变量

类体内函数外的变量，对于每个该类初始化的对象来说都是公共的。（如图46）



图表 45类体内函数体外的变量

### 2.3.2该类变量的应用

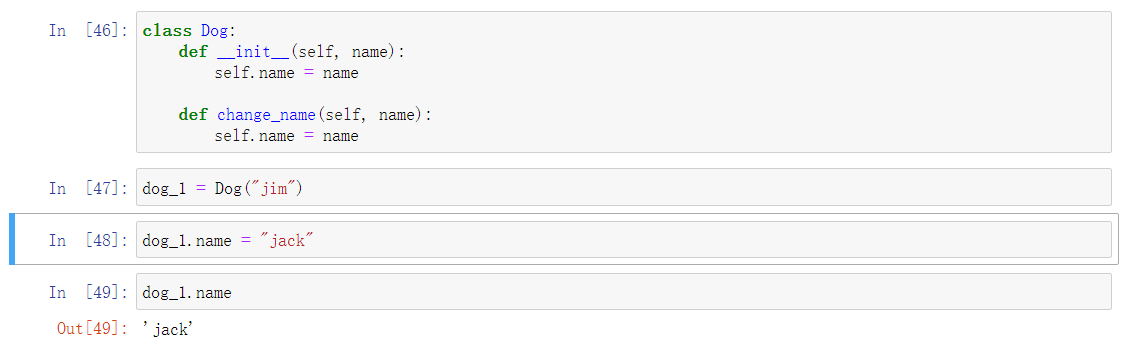
该类变量的主要应用适用于对象计数。（如图47）



图表 46该类变量的应用

## 2.4 如何修改属性（数据成员）的值

### 2.4.1 直接修改



图表 47直接修改

### 2.4.2 通过成员函数修改



图表 48通过成员函数修改

一旦我们将对象的属性设为私有的，那么我们之恩那个通过成员函数来进行修改。这样的好处在于防止误操作。（如图49）

## 2.5 多重继承

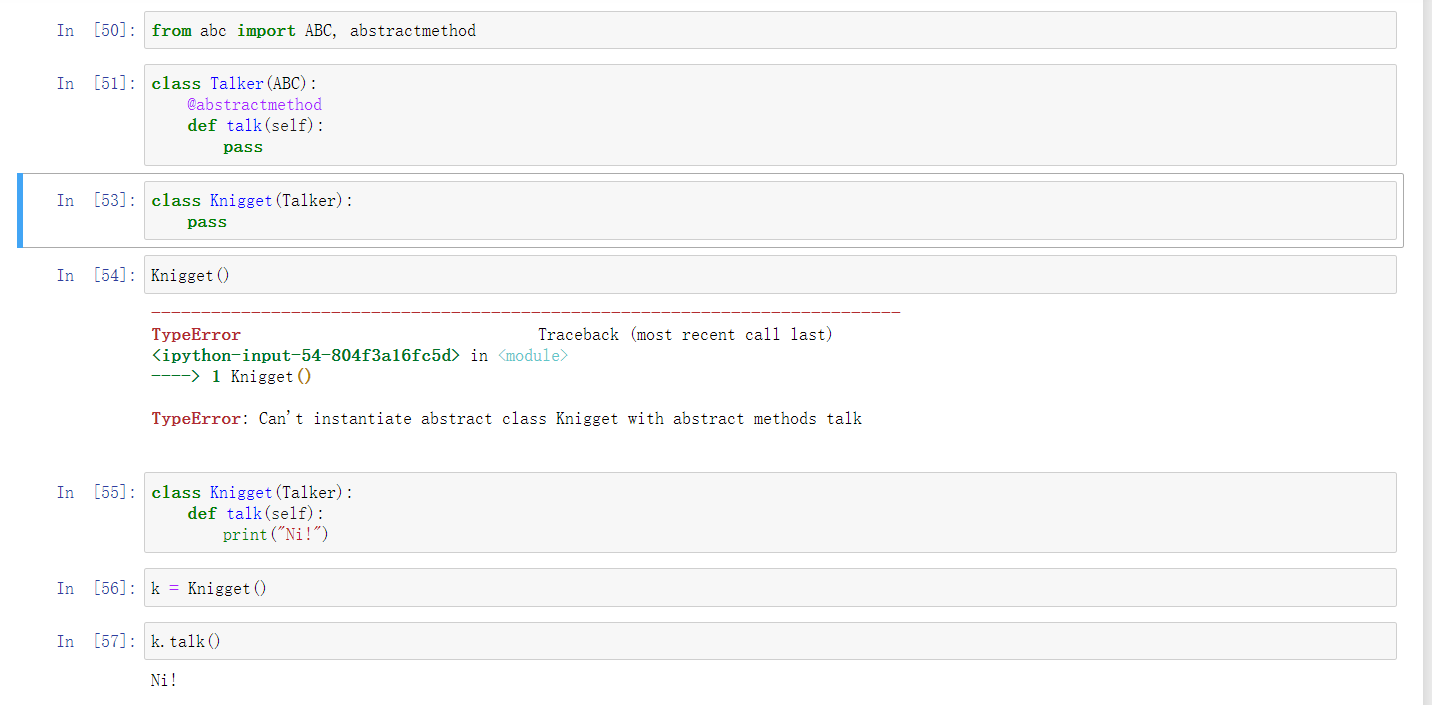
哺乳类：能跑的哺乳类，能飞的哺乳类；鸟类：能跑的鸟类，能飞的鸟类。（如图50）



图表 49多重继承

## 2.6抽象基类

"抽象基类"这个词可能听着比较"深奥",其实"基类"就是"父类","抽象"就是"假"的意思, "抽象基类"就是"假父类."



图表 50抽象基类

抽象基类的好处在与我们定义一个类的时候，并不要清楚是如何实现的。在具体操作的时候可以进行具体化，从而实现多态性。（如图51）

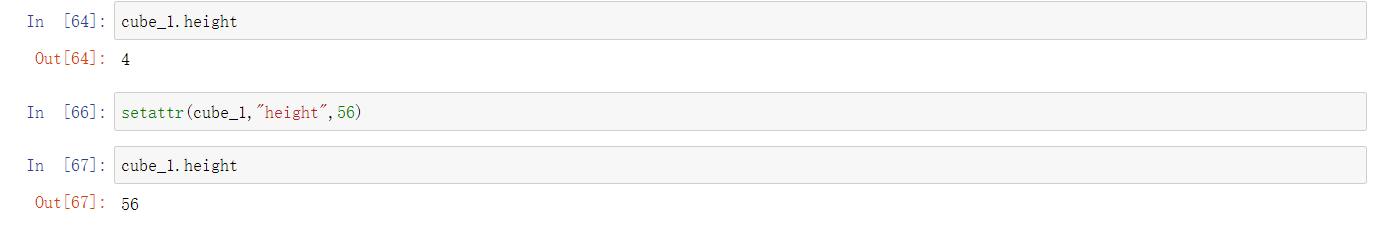
## 2.7 hasattr() setattr() getattr()

hasattr() 函数用于判断对象是否包含对应的属性。（如图52）



图表 51 hasattr() 函数

setattr() 函数的功能相对比较复杂，它最基础的功能是修改类实例对象中的属性值。（如图53）



图表 52 setattr() 函数

getattr() 函数获取某个类实例对象中指定属性的值。（如图54）



图表 53 getattr() 函数

# 三、魔法方法

## 3.1 概念介绍：“方法”VS“函数”VS“属性”

1.函数（FunctionType）：函数是封装了一些独立的功能，可以直接调用，能将一些数据（参数）传递进去进行处理，然后返回一些数据（返回值），也可以没有返回值。可以直接在模块中进行定义使用。 所有传递给函数的数据都是显式传递的。

2.方法（MethodType）：方法和函数类似，同样封装了独立的功能，但是方法是只能依靠类或者对象来调用的，表示针对性的操作。方法中的数据self和cls是隐式传递的，即方法的调用者；方法可以操作类内部的数据。

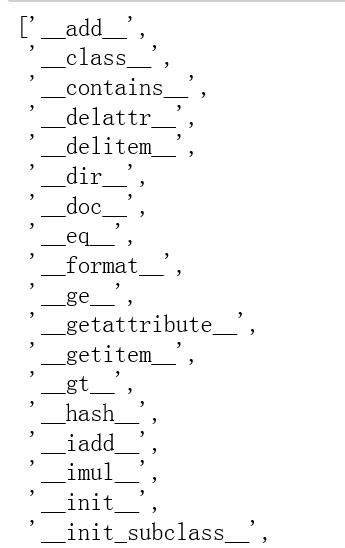
简单的说，函数在python中独立存在，可直接使用的，而方法是必须被别人调用才能实现的。

3.属性(Attribute)：类和对象的属性，储存某个值（int,str,bool...），代表对象的某种性质或特征。

然而，对于本模块需要讲解的“魔法方法”，我们可以理解为与普通方法的调用和使用有所不同的方法，具有特殊的固定名称，便于人们辨别。

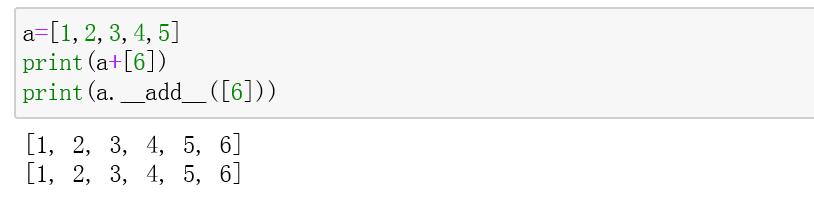
我们可以使用函数dir()来查看一划线“\_\_xxx\_\_”的为隐函数（图X），我们今天所介绍的魔法方法就包含其中。（如图55）





图表 54查看类的方法和属性

列表类中的add函数，作为一种“魔法方法”，我们可以用“+”对其调用（如图56）。



图表 55魔法方法——加法

## 3.2基本的序列（字符串、列表、元祖）和映射（字典）的协议（方法）

序列和映射基本上是元素（item）的集合，要实现它们的基本行为（协议），不可变对象需要实现2个方法，而可变对象需要实现4个。

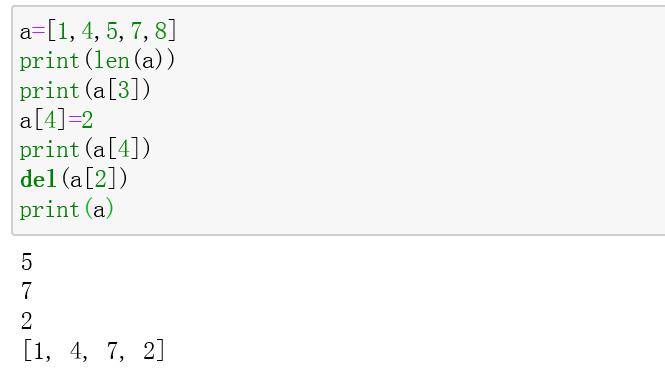
1.\_\_len\_\_(self)：这个方法应返回集合包含的项数，对序列来说为元素个数，对映射来说为键值对数。如果\_\_len\_\_返回零（且没有实现覆盖这种行为的\_\_nonzero\_\_），对象在布尔上下文中将被视为假（就像空的列表、元组、字符串和字典一样）。

2.\_\_getitem\_\_(self, key)：这个方法应返回与指定键相关联的值。对序列来说，键应该是0~n-1的整数，其中n为序列的长度。对映射来说，键可以是任何类型。

3.\_\_setitem\_\_(self, key, value)：这个方法应以与键相关联的方式存储值，以便以后能够使用\_\_getitem\_\_来获取。当然，仅当对象可变时才需要实现这个方法。(

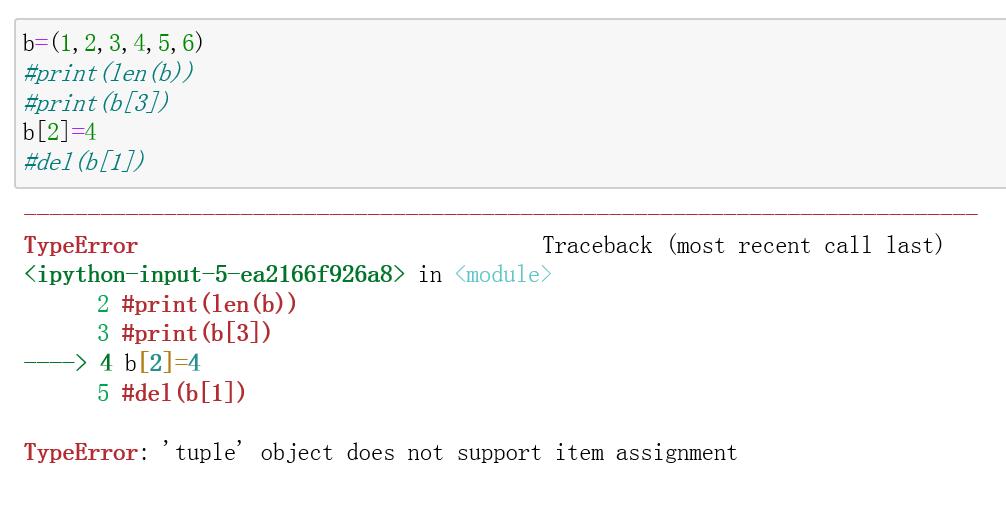
4.\_\_delitem\_\_(self, key)：这个方法在对对象的组成部分使用\_\_del\_\_语句时被调用，应删除与key相关联的值。同样，仅当对象可变（且允许其项被删除）时，才需要实现这个方法。

例如：列表作为可变对象，可以使用以上四种方法（如图57）



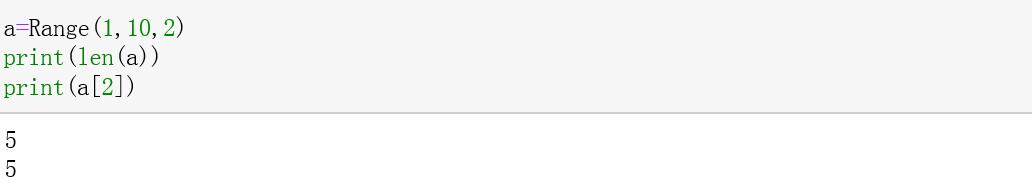
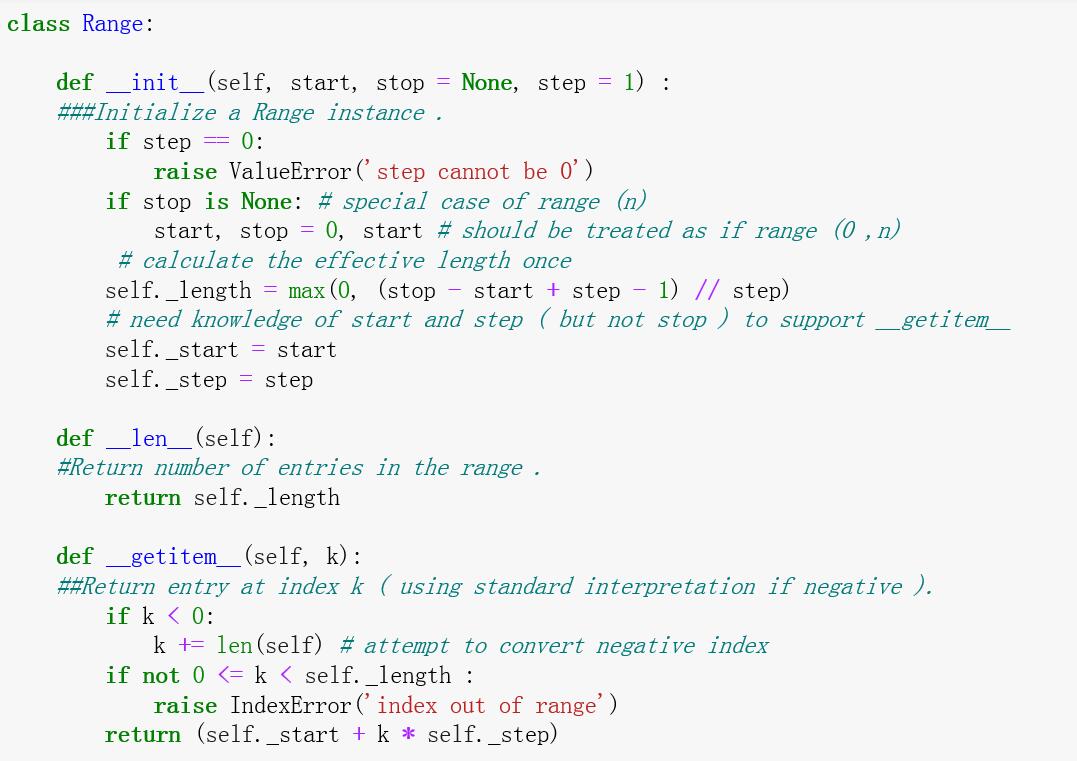
图表 56可变对象适用四种方法

但是，元组作为不可变对象，只能使用前两种方法，使用”\_\_setitem\_\_”和“\_\_delitem\_\_”时会报错（如图58）



图表 57不可变对象只适用两种方法

我们知道range函数既可以生成一系列连续的整数，也可以生成指定步长的序列。下面我们将改写” \_\_len\_\_”和“\_\_getitem\_\_”，实现这一功能。（如图59）



图表 58改写"\_\_len\_\_”和“\_\_getitem\_\_”

## 3.3静态方法和类方法

### 3.3.1 概念介绍

1.实例方法 ：第一个参数必须是实例对象，该参数名一般约定为“self”，通过它来传递实例的属性和方法（也可以传类的属性和方法）；只能由实例对象调用。

2.类方法：使用装饰器@classmethod。第一个参数必须是当前类对象，该参数名一般约定为“cls”，通过它来传递类的属性和方法（不能传实例的属性和方法）；实例对象和类对象都可以调用。

3.静态方法：使用装饰器@staticmethod。参数随意，没有“self”和“cls”参数，但是方法体中不能使用类或实例的任何属性和方法；实例对象和类对象都可以调用。

参考于<https://www.cnblogs.com/geogre123/p/10142510.html>

### 3.3.2类方法:

使用装饰器@classmethod。原则上，类方法是将类本身作为对象进行操作的方法。假设有个方法，且这个方法在逻辑上采用类本身作为对象来调用更合理，那么这个方法就可以定义为类方法。另外，如果需要继承，也可以定义为类方法。

例题：假设我有一个学生类和一个班级类，想要实现的功能为：

1.执行班级人数增加的操作、获得班级的总人数；

2.学生类继承自班级类，每实例化一个学生，班级人数都能增加；

3.最后，我想定义一些学生，获得班级中的总人数。（图X）

那么这个问题用类方法做比较合适，为什么？

因为我实例化的是学生，但是如果我从学生这一个实例中获得班级总人数，在逻辑上显然是不合理的。同时，如果想要获得班级总人数，如果生成一个班级的实例也是没有必要的。 （如图60）

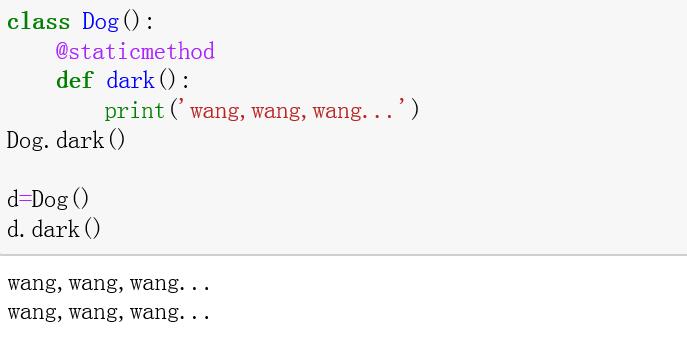


图表 59类方法示例

### 3.3.3静态方法：

使用装饰器@staticmethod。静态方法是类中的函数，不需要实例。静态方法主要是用来存放逻辑性的代码，逻辑上属于类，但是和类本身没有关系，也就是说在静态方法中，不会涉及到类中的属性和方法的操作。可以理解为，静态方法是个独立的、单纯的函数，它仅仅托管于某个类的名称空间中，便于使用和维护。

譬如，无论什么狗都是汪汪叫的.（如图61）



图表 60静态方法示例

如上，使用了静态方法（函数），然而方法体中并没使用（也不能使用）类或实例的属性（或方法）。若要获得当前时间的字符串时，并不一定需要实例化对象，此时对于静态方法而言，所在类更像是一种名称空间。

其实，我们也可以在类外面写一个同样的函数来做这些事，但是这样做就打乱了逻辑关系，也会导致以后代码维护困难。

## 3.4类和对象的魔法方法

既对类和对象的属性的系列操作：

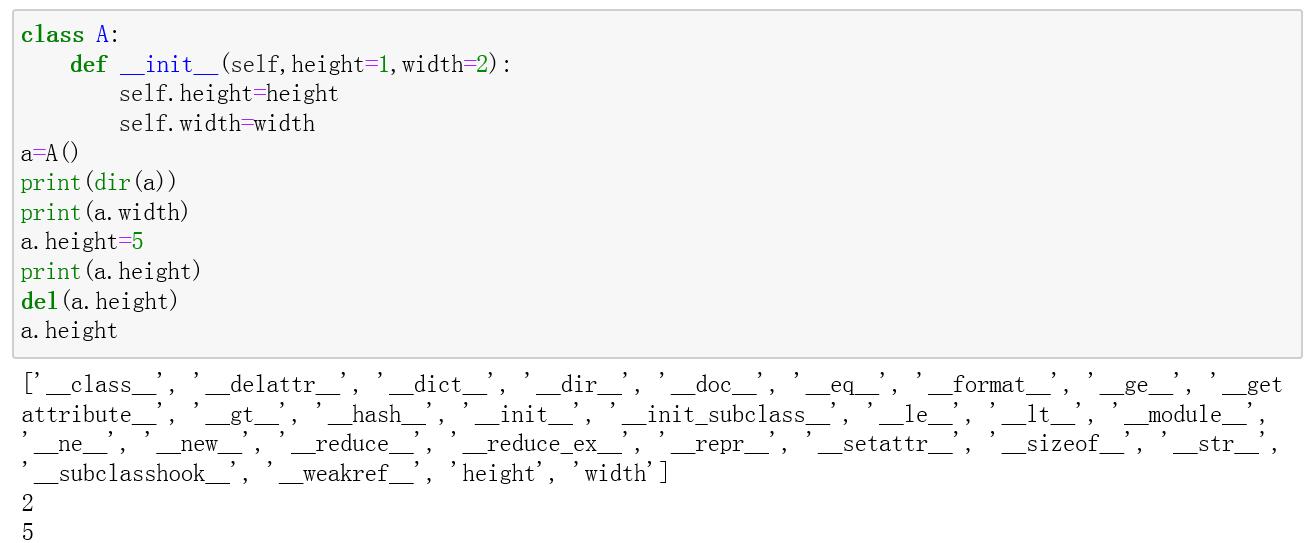
这些魔法方法是拦截对对象属性的所有访问企图，在属性被访问时执行的一段代码。下面的四个魔法方法提供了你需要的所有功能（在旧式类中，只需使用后面三个）。（Python3默认都是新式类）

1.\_\_getattribute\_\_(self, name)：在属性被访问时自动调用（只适用于新式类）。

2.\_\_getattr\_\_(self, name)：在属性被访问而对象没有这样的属性时自动调用。

3.\_\_setattr\_\_(self, name, value)：试图给属性赋值时自动调用。

4.\_\_delattr\_\_(self, name)：试图删除属性时自动调用。

我们随意构建一个类A，使用“dir()”函数可以看到类A具有以上四种方法。（”\_\_getattr\_\_”方法没有出现，由于其功能被” \_\_getattribute\_\_”所包含了。）（如图62）

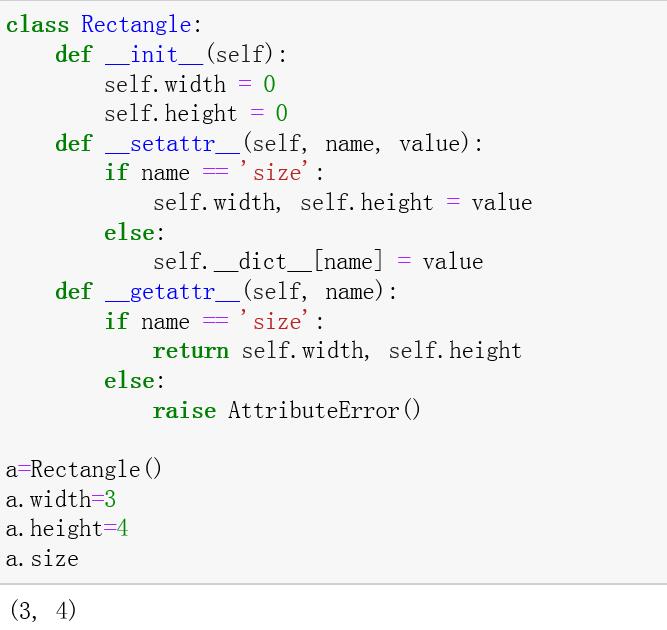
图表 61查看类和对象的魔法方法

以下是实验这四种方法的结果（如图63）：



图表 62这四种魔法方法的使用

我们通过构建“Recrangle”为例，通过改写“\_\_setattr\_\_”和“\_\_getattr\_\_”，实现设置和获取长方形长和宽的功能（如图64）：



图表 63改写"\_\_setattr\_\_”和“\_\_getattr\_\_”