第七章：函数与函数式编程

Table of Contents

[1. 序言 2](#_Toc51162250)

[1.1 章节简介 2](#_Toc51162251)

[2. 定义函数 3](#_Toc51162252)

[2.1 pass语句 4](#_Toc51162253)

[2.2 参数检查 5](#_Toc51162254)

[2.3 函数返回多个值 5](#_Toc51162255)

[3. 调用函数 6](#_Toc51162256)

[3.1 为函数提供说明文档 7](#_Toc51162257)

[4. 函数的参数 8](#_Toc51162258)

[4.1 位置参数 8](#_Toc51162259)

[4.2 默认参数 9](#_Toc51162260)

[4.3 可变参数 10](#_Toc51162261)

[4.4 关键字参数 11](#_Toc51162262)

[4.5 命名关键字参数 12](#_Toc51162263)

[4.6 参数的顺序 13](#_Toc51162264)

[5. 递归函数 13](#_Toc51162265)

[6. 高阶函数 14](#_Toc51162266)

[6.1 高阶函数定义 14](#_Toc51162267)

[6.2 sorted函数举例 15](#_Toc51162268)

[7. 返回函数 16](#_Toc51162269)

[8. 匿名函数 17](#_Toc51162270)

[9. 函数装饰器 18](#_Toc51162271)

[10. 偏函数 19](#_Toc51162272)

# 序言

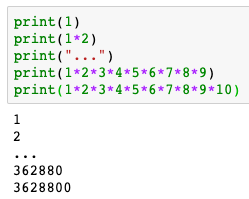
## 章节简介

所谓函数，就是把具有独立功能的代码块组织成为一个小模块，在需要的时候调用。在开发时，使用函数可以提高编写的效率以及代码的重用。Python 中函数的应用非常广泛，前面章节中我们已经接触过多个函数，比如 input() 、print()、range()、len() 函数等等，这些都是 Python 的内置函数，可以直接使用。除了可以直接使用的内置函数外，Python 还支持自定义函数，即将一段有规律的、可重复使用的代码定义成函数，从而达到一次编写、多次调用的目的。在本章里我们先详细定义函数的概念，然后还有一些关于函数式编程的高级内容，也会为大家介绍。具体内容为：

* 定义函数
* 调用函数
* 函数的参数
* 递归函数
* 高阶函数
* 返回函数
* 匿名函数
* 函数装饰器
* 偏函数

# 定义函数

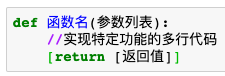
我们经常希望把完成相同功能的代码片段抽象出来，比如我们想使用Python计算整数的阶乘，如果我们计算10以内每个整数的阶乘，不使用函数，可能我们代码需要这样完成：



有此我们希望，能否有一个函数，可以抽象出整数阶乘的计算过程，当我们将参数输入给函数时，函数会自动进行计算并将结果返回给我们。借助抽象，我们才能不关心底层的具体计算过程，而直接在更高的层次上思考问题。写计算机程序也是一样，函数就是最基本的一种代码抽象的方式。

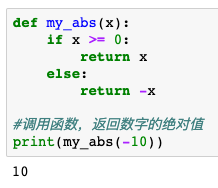
好，那如何定义一个函数，也就是创建一个实现某种功能的工具呢？定义函数需要用 def 关键字实现，具体的语法格式如下，其中，用 [] 括起来的为可选择部分，即可以使用，也可以省略。此格式中，各部分参数的含义如下：

* 函数名：其实就是一个符合 Python 语法的标识符，但不建议读者使用 a、b、c 这类简单的标识符作为函数名，函数名最好能够体现出该函数的功能（如上面的 my\_len，即表示我们自定义的 len() 函数）。
* 形参列表：设置该函数可以接收多少个参数，多个参数之间用逗号（ ，）分隔。
* [return [返回值] ]：整体作为函数的可选参参数，用于设置该函数的返回值。也就是说，一个函数，可以用返回值，也可以没有返回值，是否需要根据实际情况而定。



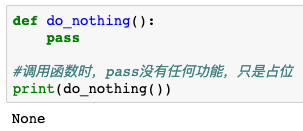
注意，在创建函数时，即使函数不需要参数，也必须保留一对空的“()”，否则 Python 解释器将提示“invaild syntax”错误。另外，如果想定义一个没有任何功能的空函数，可以使用 pass 语句作为占位符。

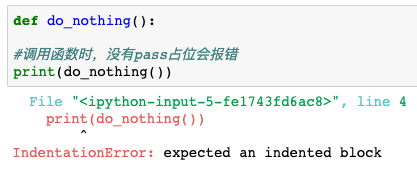
我们以自定义一个求绝对值的my\_abs函数为例：



## pass语句

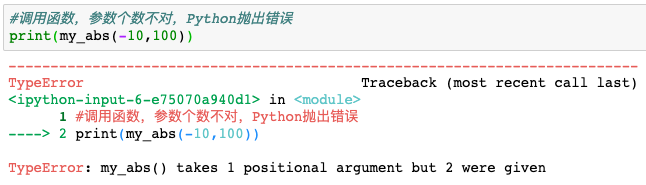
有时我们希望代码段或者函数什么也不做时会使用pass，pass有什么用？实际上pass可以用来作为占位符，比如现在还没想好怎么写函数的代码，就可以先放一个pass，让代码能运行起来。





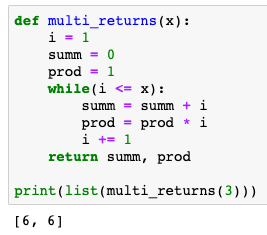
## 参数检查

当我们定义一个函数以后，如果调用时提供给函数的参数个数不对，Python会自动检测代码并抛出一个错误(TypeError):



## 函数返回多个值

我们定义的函数也可以返回多个值，下面的函数计算一个整数的和与阶乘，并一起返回，举例如下：

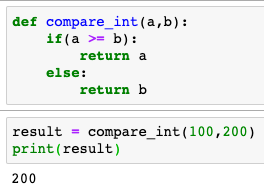


# 调用函数

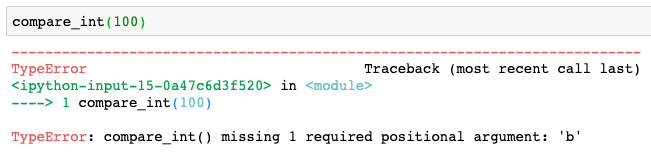
调用函数也就是执行函数。如果把创建的函数理解为一个具有某种用途的工具，那么调用函数就相当于使用该工具。函数调用的基本语法格式如下所示：*[返回值] = 函数名([形参值])。*

其中，函数名即指的是要调用的函数的名称；形参值指的是当初创建函数时要求传入的各个形参的值。如果该函数有返回值，我们可以通过一个变量来接收该值，当然也可以不接受。需要注意的是，创建函数有多少个形参，那么调用时就需要传入多少个值，且顺序必须和创建函数时一致。即便该函数没有参数，函数名后的小括号也不能省略。

对于下面示例中调用 compare\_int() 函数，由于当初定义该函数为其设置了 2 个参数，因此这里在调用该参数，就必须传入 2 个参数。同时，由于该函数内部还使用了 return 语句，因此我们可以使用 result变量来接收该函数的返回值。



调用函数的时候，如果传入的参数数量不对，会报TypeError的错误，并且Python会明确地告诉你：compare\_int ()有且仅有2个参数，但调用时只给出了1个：

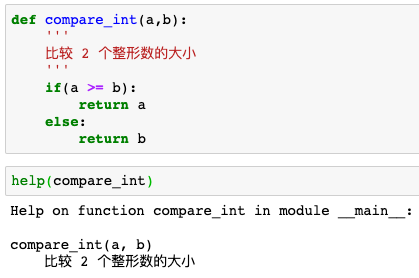


## 为函数提供说明文档

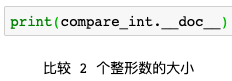
前面章节讲过，通过调用 Python 的 help() 内置函数或者 \_\_doc\_\_ 属性，我们可以查看某个函数的使用说明文档。事实上，无论是 Python 提供给我们的函数，还是自定义的函数，其说明文档都需要设计该函数的程序员自己编写。

其实，函数的说明文档，本质就是一段字符串，只不过作为说明文档，字符串的放置位置是有讲究的，函数的说明文档通常位于函数内部、所有代码的最前面。

以上面程序中的 compare\_int () 函数为例，下面演示了如何为其设置说明文档：



上面程序中，还可以使用 \_\_doc\_\_ 属性来获取 compare\_int () 函数的说明文档，即使用最后一行的输出语句，其输出结果为：



最后一点提醒大家，对于Python内置的很多函数，我们直接可以调用，关于函数的说明文档，可以在Python的官方网站查询：<https://docs.python.org/3.6/library/functions.html>

实际工作中我们需要经常自己查询Python内置的函数或者其他人开发的函数的说明文档以了解如何调用某个函数，大家要养成 自己查询开发文档的习惯。

# 函数的参数

函数的参数一共有5种形式，这里分别介绍。

## 位置参数

位置参数，有时也称必备参数，指的是必须按照正确的顺序将实际参数传到函数中，换句话说，调用函数时传入实际参数的数量和位置都必须和定义函数时保持一致。实参和形参数量必须一致。在调用函数，指定的实际参数的数量，必须和形式参数的数量一致（传多传少都不行），否则 Python 解释器会抛出 TypeError 异常，并提示缺少必要的位置参数。例如下面的例子，抛出的异常类型为 TypeError，具体是指 girth() 函数少一个必要的height参数：



在调用函数时，传入实际参数的位置必须和形式参数位置一一对应，否则会产生错误，例如下面的例子，当实际参数类型和形式参数类型不一致，并且在函数中这两种类型之间不能正常转换，此时就会抛出 TypeError 异常。



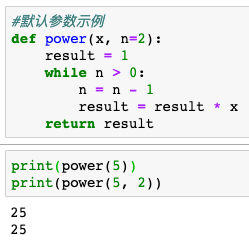
## 默认参数

我们知道，在调用函数时如果不指定某个参数，Python 解释器会抛出异常。为了解决这个问题，Python 允许为参数设置默认值，即在定义函数时，直接给形式参数指定一个默认值。这样的话，即便调用函数时没有给拥有默认值的形参传递参数，该参数可以直接使用定义函数时设置的默认值。Python 定义带有默认值参数的函数，其语法格式如下：



注意，在使用此格式定义函数时，指定有默认值的形式参数必须在所有没默认值参数的最后，否则会产生语法错误。

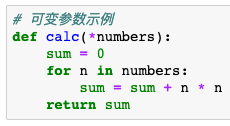
下面程序演示了如何定义和调用有默认参数的函数，从下面的例子可以看出，默认参数可以简化函数的调用，调用时如果我们不给默认参数赋值，被调用函数在运行时就使用默认参数的默认值。当我们调用时给默认参数赋值时，被调用函数在运行时就会用我们传递的值覆盖默认参数的默认值，在函数中运行中使用：



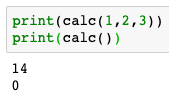
设置默认参数时，有几点要注意：一是必选参数在前，默认参数在后，否则Python的解释器会报错（思考一下为什么默认参数不能放在必选参数前面）；二是如何设置默认参数。当函数有多个参数时，把变化大的参数放前面，变化小的参数放后面。变化小的参数就可以作为默认参数。使用默认参数最大的好处是能降低调用函数的难度。

## 可变参数

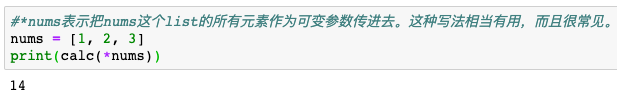
在Python函数中，还可以定义可变参数。顾名思义，可变参数就是传入的参数个数是可变的，可以是1个、2个到任意个，还可以是0个。要定义出这个函数，我们必须确定输入的参数。由于参数个数不确定，我们首先想到可以把a，b，c……作为一个list或tuple传进来，这样，函数可以定义如下：



定义可变参数和定义一个list或tuple参数相比，仅仅在参数前面加了一个\*号。在函数内部，参数numbers接收到的是一个tuple，因此，函数代码完全不变。但是，调用该函数时，可以传入任意个参数，包括0个参数：

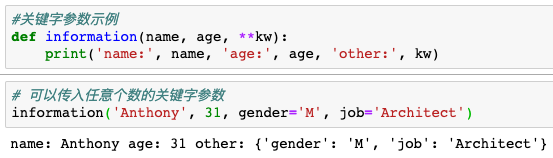


如果已经有一个list或者tuple，可以这样调用可变参数的函数。Python允许你在list或tuple前面加一个\*号，把list或tuple的元素变成可变参数传进去：



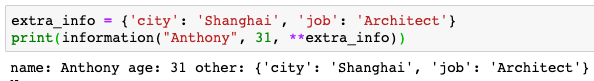
## 关键字参数

本节将介绍的关键字参数，可以避免牢记参数位置的麻烦，令函数的调用和参数传递更加灵活方便。关键字参数是指使用形式参数的名字来确定输入的参数值。通过此方式指定函数实参时，不再需要与形参的位置完全一致，只要将参数名写正确即可。关键字参数允许你传入0个或任意个含参数名的参数，这些关键字参数在函数内部自动组装为一个dict。



关键字参数有什么用？它可以扩展函数的功能。比如，在information函数里，我们保证能接收到name和age这两个参数，但是，如果调用者愿意提供更多的参数，我们也能收到。试想你正在做一个用户注册的功能，除了用户名和年龄是必填项外，其他都是可选项，利用关键字参数来定义这个函数就能满足注册的需求。

和可变参数类似，也可以先组装出一个dict，然后，把该dict转换为关键字参数传进去。\*\*extra表示把extra这个dict的所有key-value用关键字参数传入到函数的\*\*kw参数，kw将获得一个dict，注意kw获得的dict是extra的一份拷贝，对kw的改动不会影响到函数外的extra。

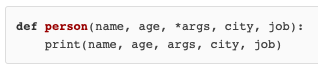


## 命名关键字参数

对于关键字参数，函数的调用者可以传入任意不受限制的关键字参数。至于到底传入了哪些，就需要在函数内部通过kw检查。调用者仍可以传入不受限制的关键字参数。如果要限制关键字参数的名字，就可以用命名关键字参数，例如，只接收city和job作为关键字参数。这种方式定义的函数如下.和关键字参数\*\*kw不同，命名关键字参数需要一个特殊分隔符\*，\*后面的参数被视为命名关键字参数。



如果函数定义中已经有了一个可变参数，后面跟着的命名关键字参数就不再需要一个特殊分隔符\*了。使用命名关键字参数时，要特别注意，如果没有可变参数，就必须加一个\*作为特殊分隔符。如果缺少\*，Python解释器将无法识别位置参数和命名关键字参数：



命名关键字参数必须传入参数名，这和位置参数不同。如果没有传入参数名，调用将报错：



命名关键字参数也可以有缺省值，这样可以简化我们调用：



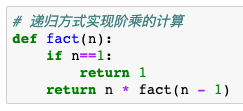
## 参数的顺序

我们使用Python定义函数时，5中参数类型必选参数、默认参数、可变参数、关键字参数和命名关键字参数，这5种参数都可以混合使用。需要小心的是，参数定义的顺序必须是：必选参数、默认参数、可变参数、命名关键字参数和关键字参数。

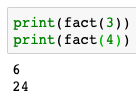
虽然可以使用5中类型的参数进行组合，实际工作时不要同时使用太多的组合，否则函数的可理解性和可读性很差，给函数调用者带来疑惑。

# 递归函数

在函数内部，可以调用其他函数。如果一个函数在内部调用自身本身，这个函数就是递归函数。举个例子，我们来计算阶乘n! = 1 x 2 x 3 x ... x n。用函数fact(n)表示，可以看出：fact(n)=fact(n−1)×n。于是我们可以定义一个递归函数用于计算阶乘：



我们可以像正常函数一样调用递归函数，如下示例：

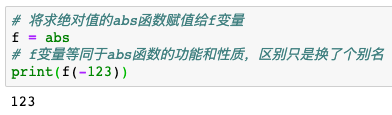


递归函数的优点是定义简单，逻辑清晰。理论上，所有的逻辑都可以使用循环或递归方式完成，但循环的逻辑不如递归的代码逻辑清晰。

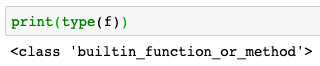
# 高阶函数

## 高阶函数定义

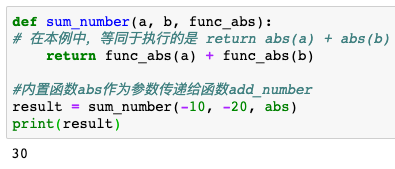
高阶函数是在Python中一个非常有用的功能函数，所谓高阶函数就是一个函数可以用来接收另一个函数作为参数，这样的函数叫做高阶函数。为了便于理解，我们从实际例子来看看函数当做参数被传递到另个函数是什么样的。我们把abs()函数赋值给了f变量，然后我们就可以像调用abs()函数本身那样使用f变量了，区别只是换了个名字而已。



通过查看下这个f变量的类型，显示为内置函数，因为它的实际就是指向内置abs()函数，是该函数的一个”别名”。



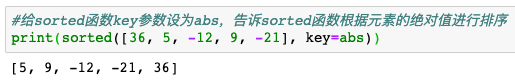
可以看到在Python语言中，变量可以指向函数，既然变量可以指向函数，而函数的参数是用来接收变量。那么自然地一个函数可以接收另一个函数作为它的参数，如下示例所示，我们定义了一个sum\_number函数，然后定义了3个函数参数变量分别为a，b，func\_abs。通过函数内部语句我们可以发现func\_abs其实是另一个函数体，所以func\_abs变量由于被调用的时候传值为abs函数，它的实际真身就是abs函数了：



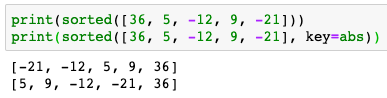
## sorted函数举例

给大家一个高阶函数的举例，集合排序也是在程序中经常要处理的问题。sorted()也是一个高阶函数。用sorted()排序的关键在于实现一个映射函数。排序算法里我们见过的像冒泡排序快速排序或者二分法排序法，其排序的核心处理两个元素的大小比较。如果是数字，我们可以直接比较，但如果是字符串或者两个dict呢？这些类型的对象直接比较数学上的大小是没有意义的，因此，比较的过程必须通过函数抽象出来。需要我们自己定义一个函数比较两个元素的大小，并将这个函数作为排序的依据传递给排序函数对集合进行排序。

Pyhton内置的sorted()函数也是一个高阶函数，它还可以接收一个key函数来实现自定义的排序，我们可以通过key这个参数指定对象的比较规则。例如按绝对值大小排序：



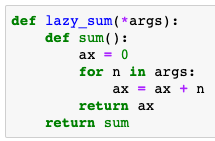
我们可以比较下两种不同比较方式排序后的结果：



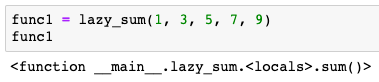
# 返回函数

在Python里，一切都是对象，包括函数，既然对象或者变量可以作为函数的返回值，同样的我们也可以把一个函数作为结果值返回。

比如我们定义一个函数，它的返回值是一个函数，这个返回的函数的作用是计算求和。



当我们调用lazy\_sum()时，返回的并不是求和结果，而是求和函数：



调用函数func1时，才真正计算求和的结果。在这个例子中，我们在函数lazy\_sum中又定义了函数sum，并且，内部函数sum可以引用外部函数lazy\_sum的参数和局部变量，当lazy\_sum返回函数sum时，相关参数和变量都保存在返回的函数中，这种称为“闭包（Closure）”的程序结构拥有极大的威力：

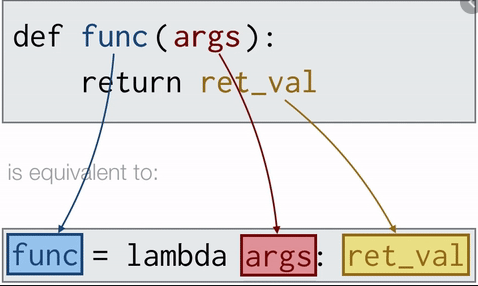


# 匿名函数

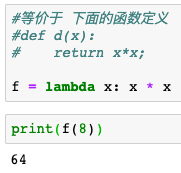
匿名函数，顾名思义，就是不需要显示的定义函数名，直接定义函数体。在Python中，我们用lambda表达式的方式来定义一个匿名函数，具体格式为：



换句话说，我们可以lambda用来表示一个匿名的函数，它可以有多个参数，但是只能是一个表达式。我们用lambda生成的函数实际上和一般方式定义的函数时完全一样的。



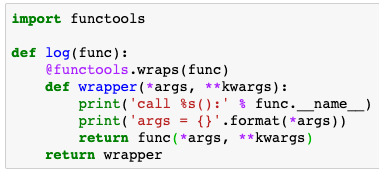
用匿名函数的好处是，函数没有名字所以不必担心函数名冲突。而且匿名函数也是一个函数对象，也可以把匿名函数赋值给一个变量，再使用变量来调用该函数：



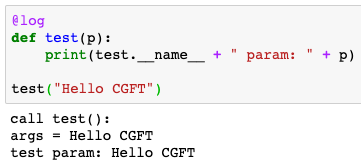
# 函数装饰器

Python装饰器看起来类似Java中的注解，然而和注解并不相同，同样也实现了类似面向切面编程。想要理解Python中的装饰器，我们先来回顾下闭包（closure）这一概念。我们在第7小节返回函数了解到，闭包就是引用了自有变量的函数，这个函数保存了执行的上下文，可以脱离原本的作用域独立存在。

下面来看看Python中的装饰器。一个普通的装饰器一般是这样, import functools是导入functools模块。模块的概念稍候讲解。现在，只需记住在定义wrapper()的前面加上@functools.wraps(func)即可：



这样就定义了一个打印出方法名及其参数的装饰器。我们用下面这种方式调用它：



观察上面的log，因为它是一个decorator，所以接受一个函数作为参数，并返回一个函数。我们要借助Python的@语法，把decorator置于函数的定义处。wrapper()函数的参数定义是(\*args, \*\*kw)，因此，wrapper()函数可以接受任意参数的调用。在wrapper()函数内，首先打印日志，再紧接着调用原始函数。

值得注意的是@functools.wraps(func)，这是python提供的装饰器。它能把原函数的元信息拷贝到装饰器里面的 func 函数中。函数的元信息包括docstring、name、参数列表等等。可以尝试去除@functools.wraps(func)，你会发现test.\_\_name\_\_的输出变成了wrapper。

# 偏函数

Python的functools模块提供了很多有用的功能，其中一个就是偏函数（Partial function）。要注意，这里的偏函数和数学意义上的偏函数不一样。使用functools.partial可以创建一个新的函数，这个新函数可以固定住原函数的部分参数，从而在调用时更简单。

偏函数是将所要承载的函数作为 partial() 函数的第一个参数，原函数的各个参数依次作为 partial() 函数后续的参数，除非使用关键字参数。示例如下，可以看到functools.partial的作用就是，把一个函数的某些参数给固定住（也就是设置默认值），返回一个新的函数，调用这个新函数会更简单。

