廖雪峰的官方网站 □编程 □读书 👸 Java教程 🕝 Python教程 🔞 JavaScript教程 📦 SQL教程 👂 Git教程 💬 问答 ●登录 ♀ 函数的参数 目录 2 O x □ Python教程 阅读: 3454799 Python简介 定义函数的时候,我们把参数的名字和位置确定下来,函数的接口定义就完成了。对于函数的调用者来说,只需要知道如何传递正确的参数,以及函数将返回什么样的值就够了,函数内部的复杂逻辑被封装起来,调用者无 田 安装Python 需了解。 ⊞ 第一个Python程序 Python的函数定义非常简单,但灵活度却非常大。除了正常定义的必选参数外,还可以使用默认参数、可变参数和关键字参数,使得函数定义出来的接口,不但能处理复杂的参数,还可以简化调用者的代码。 ① Python基础 □ 函数 位置参数 调用函数 我们先写一个计算x2的函数: 定义函数 def power (x): 函数的参数 return x * x 递归函数 ⊞ 高级特性 对于 power(x) 函数,参数x就是一个位置参数。 ⊞ 函数式编程 当我们调用 power 函数时,必须传入有且仅有的一个参数 x: 田 模块 >>> power (5) ⊞ 面向对象编程 25 ⊞ 面向对象高级编程 >>> power (15) 田 错误、调试和测试 田 IO编程 现在,如果我们要计算 x^3 怎么办?可以再定义一个 power3 函数,但是如果要计算 x^4 、 x^5怎么办?我们不可能定义无限多个函数。 ⊞ 进程和线程 你也许想到了,可以把 power(x) 修改为 power(x, n),用来计算xⁿ,说干就干: 正则表达式 def power (x, n): 田 常用内建模块 s = 1田 常用第三方模块 while $n \ge 0$: n = n - 1virtualenv s = s * x 田 图形界面 return s 田 网络编程 对于这个修改后的 power(x, n) 函数,可以计算任意n次方: 田 电子邮件 田 访问数据库 >>> power (5, 2) 25 ⊞ Web开发 >>> power (5, 3) ⊞ 异步IO 田 实战 修改后的 power(x, n) 函数有两个参数: x 和 n , 这两个参数都是位置参数 , 调用函数时 , 传入的两个值按照位置顺序依次赋给参数 x 和 n 。 FAQ 默认参数 期末总结 新的 power(x, n) 函数定义没有问题,但是,旧的调用代码失败了,原因是我们增加了一个参数,导致旧的代码因为缺少一个参数而无法正常调用: 关于作者 >>> power (5) Traceback (most recent call last): File "<stdin>", line 1, in <module> TypeError: power() missing 1 required positional argument: 'n' Python的错误信息很明确:调用函数 power() 缺少了一个位置参数 n。 这个时候,默认参数就排上用场了。由于我们经常计算x2,所以,完全可以把第二个参数n的默认值设定为2: def power (x, n=2): s = 1 自己的Python课程 while $n \ge 0$: n = n - 1s = s * x return s Python商业爬虫全解码 这样, 当我们调用 power(5) 时, 相当于调用 power(5, 2): 让天下没有爬不到的数据! >>> power (5) >>> power (5, 2) 数据分析 Python爬虫 + 而对于 n > 2 的其他情况,就必须明确地传入n,比如 power(5, 3)。 深度学习 Python机器学习 从上面的例子可以看出,默认参数可以简化函数的调用。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。 使用默认参数有什么好处?最大的好处是能降低调用函数的难度。 举个例子,我们写个一年级小学生注册的函数,需要传入 name 和 gender 两个参数: def enroll (name, gender): print('name:', name) print('gender:', gender) 廖雪峰老师 这样,调用 enroll() 函数只需要传入两个参数: 自己的Java课程 >>> enroll('Sarah', 'F') name: Sarah gender: F Java高级架构师 如果要继续传入年龄、城市等信息怎么办?这样会使得调用函数的复杂度大大增加。 我们可以把年龄和城市设为默认参数: 源码分析专题 🕇 微服务架构专题 def enroll(name, gender, age=6, city='Beijing'): print('name:', name) print('gender:', gender) 高并发分布式专题 十 性能优化专题 print('age:', age) print('city:', city) 这样,大多数学生注册时不需要提供年龄和城市,只提供必须的两个参数: 找廖雪峰老师 >>> enroll('Sarah', 'F') name: Sarah gender: F age: 6 city: Beijing Google 已关闭此广告 只有与默认参数不符的学生才需要提供额外的信息: 停止显示此广告 enroll ('Bob', 'M', 7) enroll ('Adam', 'M', city='Tianjin') 为什么显示该广告? ▷ 可见,默认参数降低了函数调用的难度,而一旦需要更复杂的调用时,又可以传递更多的参数来实现。无论是简单调用还是复杂调用,函数只需要定义一个。 有多个默认参数时,调用的时候,既可以按顺序提供默认参数,比如调用 enroll('Bob', 'M', 7),意思是,除了 name , gender 这两个参数外,最后1个参数应用在参数 age 上, city 参数由于没有提供,仍然使用默 认值。 也可以不按顺序提供部分默认参数。当不按顺序提供部分默认参数时,需要把参数名写上。比如调用 enroll('Adam', 'M', city='Tianjin') ,意思是, city 参数用传进去的值,其他默认参数继续使用默认值。 默认参数很有用,但使用不当,也会掉坑里。默认参数有个最大的坑,演示如下: 先定义一个函数,传入一个list,添加一个 END 再返回: def add_end(L=[]): L. append ('END') return L 当你正常调用时,结果似乎不错: >>> add_end([1, 2, 3]) [1, 2, 3, 'END'] >>> add_end(['x', 'y', 'z']) ['x', 'y', 'z', 'END'] 当你使用默认参数调用时,一开始结果也是对的: >>> add_end() Google 已关闭此广告 ['END'] 停止显示此广告 但是,再次调用 add_end()时,结果就不对了: 为什么显示该广告? ▷ >>> add_end() ['END', 'END'] >>> add_end() ['END', 'END', 'END'] 很多初学者很疑惑,默认参数是[],但是函数似乎每次都"记住了"上次添加了 'END' 后的list。 原因解释如下: Python函数在定义的时候,默认参数 L的值就被计算出来了,即 [] ,因为默认参数 L也是一个变量,它指向对象 [] ,每次调用该函数,如果改变了 L的内容,则下次调用时,默认参数的内容就变了,不再是函数定义 时的[]了。 ▲ 定义默认参数要牢记一点:默认参数必须指向不变对象! 要修改上面的例子,我们可以用 None 这个不变对象来实现: def add_end(L=None): if L is None: L = [] L. append ("END") return L 现在,无论调用多少次,都不会有问题: >>> add_end() ['END'] >>> add_end() ['END'] 为什么要设计str、None这样的不变对象呢?因为不变对象一旦创建,对象内部的数据就不能修改,这样就减少了由于修改数据导致的错误。此外,由于对象不变,多任务环境下同时读取对象不需要加锁,同时读一点问 题都没有。我们在编写程序时,如果可以设计一个不变对象,那就尽量设计成不变对象。 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。 我们以数学题为例子,给定一组数字a,b,c....,请计算 $a^2 + b^2 + c^2 +$ 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: def calc(numbers): sum = O for n in numbers: sum = sum + n * nreturn sum 但是调用的时候,需要先组装出一个list或tuple: >>> calc([1, 2, 3]) >>> calc((1, 3, 5, 7)) 如果利用可变参数,调用函数的方式可以简化成这样: >>> calc(1, 2, 3) >>> calc(1, 3, 5, 7) 所以,我们把函数的参数改为可变参数: def calc (*numbers): sum = O for n in numbers: sum = sum + n * nreturn sum 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: >>> calc(1, 2) >>> calc() 如果已经有一个list或者tuple,要调用一个可变参数怎么办?可以这样做: >>> nums = [1, 2, 3] >>> calc(nums[0], nums[1], nums[2]) 这种写法当然是可行的,问题是太繁琐,所以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: >>> nums = [1, 2, 3] >>> calc(*nums) 14 *nums 表示把 nums 这个list的所有元素作为可变参数传进去。这种写法相当有用,而且很常见。 关键字参数 可变参数允许你传入0个或任意个参数,这些可变参数在函数调用时自动组装为一个tuple。而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。请看示例: def person(name, age, **kw): print('name:', name, 'age:', age, 'other:', kw) 函数 person 除了必选参数 name 和 age 外,还接受关键字参数 kw 。在调用该函数时,可以只传入必选参数: >>> person ('Michael', 30) name: Michael age: 30 other: {} 也可以传入任意个数的关键字参数: >>> person('Bob', 35, city='Beijing') name: Bob age: 35 other: {'city': 'Beijing'} >>> person('Adam', 45, gender='M', job='Engineer') name: Adam age: 45 other: {'gender': 'M', 'job': 'Engineer'} 关键字参数有什么用?它可以扩展函数的功能。比如,在 person 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 = {'city': 'Beijing', 'job': 'Engineer'} >>> person('Jack', 24, city=extra['city'], job=extra['job']) name: Jack age: 24 other: {'city': 'Beijing', 'job': 'Engineer'} 当然,上面复杂的调用可以用简化的写法: >>> extra = {'city': 'Beijing', 'job': 'Engineer'} >>> person (' Jack', 24, **extra) name: Jack age: 24 other: {'city': 'Beijing', 'job': 'Engineer'} **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。 仍以 person() 函数为例,我们希望检查是否有 city 和 job 参数: def person(name, age, **kw): if 'city' in kw: # 有city参数 pass if 'job' in kw: # 有job参数 print ('name:', name, 'age:', age, 'other:', kw) 但是调用者仍可以传入不受限制的关键字参数: >>> person('Jack', 24, city='Beijing', addr='Chaoyang', zipcode=123456) 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: def person(name, age, *, city, job): print(name, age, city, job) 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。 调用方式如下: >>> person('Jack', 24, city='Beijing', job='Engineer') Jack 24 Beijing Engineer 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: def person(name, age, *args, city, job): print(name, age, args, city, job) 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: >>> person('Jack', 24, 'Beijing', 'Engineer') Traceback (most recent call last): File "<stdin>", line 1, in <module> TypeError: person() takes 2 positional arguments but 4 were given 由于调用时缺少参数名 city 和 job , Python解释器把这4个参数均视为位置参数 , 但 person() 函数仅接受2个位置参数。 命名关键字参数可以有缺省值,从而简化调用: def person(name, age, *, city='Beijing', job): print(name, age, city, job) 由于命名关键字参数 city 具有默认值,调用时,可不传入 city 参数: >>> person('Jack', 24, job='Engineer') Jack 24 Beijing Engineer 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: def person(name, age, city, job): # 缺少 *, city和job被视为位置参数 参数组合 在Python中定义函数,可以用必选参数、默认参数、可变参数、关键字参数和命名关键字参数,这5种参数都可以组合使用。但是请注意,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。 比如定义一个函数,包含上述若干种参数: def f1 (a, b, c=0, *args, **kw): print('a =', a, 'b =', b, 'c =', c, 'args =', args, 'kw =', kw) def f2 (a, b, c=0, *, d, **kw): print('a =', a, 'b =', b, 'c =', c, 'd =', d, 'kw =', kw) 在函数调用的时候, Python解释器自动按照参数位置和参数名把对应的参数传进去。 >>> f1(1, 2) $a = 1 b = 2 c = 0 args = () kw = {}$ >>> f1 (1, 2, c=3) $a = 1 b = 2 c = 3 args = () kw = {}$ >>> f1 (1, 2, 3, 'a', 'b') $a = 1 b = 2 c = 3 args = ('a', 'b') kw = {}$ >>> f1 (1, 2, 3, 'a', 'b', x=99) $a = 1 b = 2 c = 3 args = ('a', 'b') kw = {'x': 99}$ >>> f2(1, 2, d=99, ext=None) $a = 1 b = 2 c = 0 d = 99 kw = {'ext' : None}$ 最神奇的是通过一个tuple和dict,你也可以调用上述函数: >>> args = (1, 2, 3, 4) >>> kw = {' d' : 99, 'x' : '#'} >>> f1 (*args, **kw) $a = 1 b = 2 c = 3 args = (4,) kw = {'d': 99, 'x': '#'}$ >>> args = (1, 2, 3) >>> kw = {' d' : 88, 'x' : '#'} >>> f2(*args, **kw) $a = 1 b = 2 c = 3 d = 88 kw = {'x': '#'}$ 所以,对于任意函数,都可以通过类似 func(*args, **kw) 的形式调用它,无论它的参数是如何定义的。 ● 虽然可以组合多达5种参数,但不要同时使用太多的组合,否则函数接口的可理解性很差。 练习 以下函数允许计算两个数的乘积,请稍加改造,变成可接收一个或多个数并计算乘积: # -*- coding: utf-8 -*def product(x, y): return x * y # 测试 print('product(5) =', product(5)) print('product(5, 6) =', product(5, 6)) print('product(5, 6, 7) =', product(5, 6, 7)) print('product(5, 6, 7, 9) =', product(5, 6, 7, 9)) if product(5) != 5: print('测试失败!') elif product(5, 6) != 30: print('测试失败!') elif product(5, 6, 7) != 210: print('测试失败!') elif product(5, 6, 7, 9) != 1890: print('测试失败!') else: try: product() print('测试失败!') except TypeError: print('测试成功!') ▶ Run 小结 Python的函数具有非常灵活的参数形态,既可以实现简单的调用,又可以传入非常复杂的参数。 默认参数一定要用不可变对象,如果是可变对象,程序运行时会有逻辑错误! 要注意定义可变参数和关键字参数的语法: *args 是可变参数, args接收的是一个tuple; **kw 是关键字参数,kw接收的是一个dict。 以及调用函数时如何传入可变参数和关键字参数的语法: 可变参数既可以直接传入: func(1, 2, 3),又可以先组装list或tuple,再通过*args 传入: func(*(1, 2, 3)); 关键字参数既可以直接传入: func(a=1, b=2), 又可以先组装dict, 再通过**kw 传入: func(**{'a': 1, 'b': 2})。 使用 *args 和 **kw 是Python的习惯写法,当然也可以用其他参数名,但最好使用习惯用法。 命名的关键字参数是为了限制调用者可以传入的参数名,同时可以提供默认值。 定义命名的关键字参数在没有可变参数的情况下不要忘了写分隔符*,否则定义的将是位置参数。 参考源码 var_args.py kw_args.py 读后有收获可以请作者喝咖啡,读后有疑问请加群讨论: 还可以分享给朋友: 6 分字到微博 下一页》 く上一页 () 阿里云 ACM金牌得主 廖雪峰推荐 高性能云服务器首台5折 **Python** JAVA进阶教程 企业上云事半功倍,最大20Gbps内网带宽,450万PPS 全球顶尖名企一线数据科学家倾力指导 商业爬虫全解码 人工智能与自然语言/计算机视觉课程培训 原价1599元 Artificial Intelligence For NLP/CV Courses 找廖雪峰老师 0元领取 立即选购 无offer退全款 广告× .Python内部教材+ .Python内部教材+ 100G全套学习视频 100G全套学习视频 每晚8:30免费直播公开课 每晚8:30免费直播公开课 python大型免费高级进阶公开课 python大型免费高级进阶公开课 授课模式:在线直播+课后视频,从 授课模式:在线直播+课后视频,从 零基础到中高级开发工程师 零基础到中高级开发工程师 评论 微尘初心 created at 21 hours ago, Last updated at 21 hours ago def product(x=0, y=1, *args): result = x * y if args: for number in args: rint (测试失败2') if product (5, 6, 7) != 210: print (测试失败3') if product (5, 6, 7, 8) != 1680: print (测试失败4') try: product () print (ok') except Exception as e: with (log. txt', 'w') as f: 至部讨论 每 回复

作业:加改造,变成可接收一个或多个数并计算乘积。