



《Python程序设计基础教程（微课版）》

<http://dblabb.xmu.edu.cn/post/python>



第6章 函数

林子雨 博士/副教授

厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn ▶▶▶

主页: <http://dblabb.xmu.edu.cn/post/linziyu>





主讲教师



2017年度厦门大学奖教金获得者

2020年度厦门大学奖教金获得者

主讲教师：厦门大学 林子雨 博士/副教授

中国高校首个“数字教师”提出者和建设者

2009年7月从事教师职业以来

累计**免费**网络发布超过**1500万**字高价值教学和科研资料

网络浏览量超过**1500万**次



提纲

- 6.1 普通函数
- 6.2 匿名函数
- 6.3 参数传递
- 6.4 参数类型

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社
《Python程序设计基础教程（微课版）》教材官方网站：
<http://dbl原因.xmu.edu.cn/post/python>

高等院校程序设计新形态精品系列

PYTHON

Python Programming Language

Python 程序设计基础教程

| 微课版 |

林子雨 赵江声 陶继平 编著

- 名师精品**
多年计算机教学实践的厚积薄发
- 深入浅出**
清晰呈现 Python 语言学习路径
- 实例丰富**
有效提升编程语言的学习趣味
- 资源全面**
构建全方位一站式在线服务体系

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS



6.1 普通函数

- 6.1.1 基本定义及调用
- 6.1.2 文档字符串
- 6.1.3 函数标注
- 6.1.4 return语句
- 6.1.5 变量作用域
- 6.1.6 函数的递归调用

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社
《Python程序设计基础教程（微课版）》教材官方网站：
<http://dblalab.xmu.edu.cn/post/python>

高等院校程序设计新形态精品系列

PYTHON

Python Programming Language

Python 程序设计基础教程

| 微课版 |

林子雨 赵江声 陶继平 编著



名师精品

多年计算机教学实践的厚积薄发



深入浅出

清晰呈现 Python 语言学习路径



实例丰富

有效提升编程语言的学习趣味



资源全面

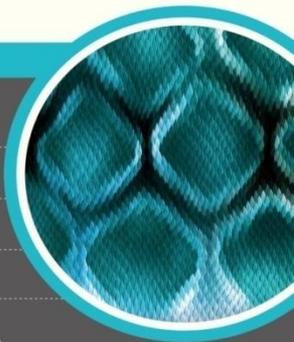
构建全方位一站式在线服务体系



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS





6.1.1 基本定义及调用

定义函数的语法

def 函数名(参数列表):
 函数体

定义函数需要遵循以下规则:

- 函数代码块从形式上包含函数名部分和函数体部分;
- 函数名部分以 **def** 关键词开头, 后接函数标识符名称和圆括号 (), 以冒号 “:” 结尾;
- 圆括号内可以定义参数列表 (可以为0个、1个或多个参数), 即使参数个数为0, 圆括号也必须有; 函数形参不需要声明其类型;
- 函数的第一行语句可以选择性地使用文档字符串用于存放函数说明;
- 函数体部分的内容需要缩进;
- 使用 “**return** [表达式] ” 结束函数, 选择性地返回一个值给调用方, 不带表达式的 **return** 语句相当于返回 **None**。



6.1.1 基本定义及调用

函数定义完成之后，就可以被调用了。函数可通过另一个函数调用执行，也可以直接从 Python 命令提示符执行。

在下面的例子中，我们先定义一个hello()函数，没有带参数，然后调用：

```
>>> def hello():  
    print("Hello Python")  
>>> hello()  
Hello Python
```



6.1.1 基本定义及调用

下面再给出一个带有一个参数的函数的实例，通过这个实例可以发现，对于已经定义的函数，可以多次调用，这样就提高了代码的复用性。

【例6-1】 定义一个带有参数的函数。

```
01 # i_like.py
02 # 定义带有参数的函数
03 def like(language):
04     """打印喜欢的编程语言！"""
05     print("我喜欢{}语言！".format(language))
06     return
07 # 调用函数
08 like("C")
09 like("C#")
10 like("Python")
```

代码的执行结果如下：
我喜欢C语言！
我喜欢C#语言！
我喜欢Python语言！



6.1.1 基本定义及调用

需要注意的是，函数的第一行语句使用文档字符串来进行函数说明，可以用内置函数`help()`查看函数的说明，具体如下：

```
>>> help(like)
```

```
Help on function like in module __main__:
```

```
like(language)
```

```
打印喜欢的编程语言！
```



6.1.1 基本定义及调用

下面再给出一个带有多个参数的函数的实例。

【例6-2】 求出从整数a1到整数a2中所有整数之和。

```
01 # sum_seq.py
02 # 定义函数
03 def sum_seq(a1,a2):
04     val = (a1 + a2) * (abs(a2 - a1)+1)/2
05     return val
06 #调用函数
07 print(sum_seq(1,9))
08 print(sum_seq(3,4))
09 print(sum_seq(2,11))
```

代码的执行结果如下：

45.0

7.0

65.0



6.1.2 文档字符串

在前面函数定义原则中曾提到，函数的第一行语句可以选择性地使用“文档字符串”（**documentation strings**）用于存放函数说明。文档字符串有如下一些约定：

- 第一行应为对象目的的简要描述；
- 如果有多行，则第二行应为空白。其目的是将摘要与其他描述有个视觉上的分隔。后面几行应该是一个或多个段落，描述对象的调用约定、副作用等。

文档字符串及其约定其实是可选而非必需；若没有增加文档字符串，并不会造成语法错误。当然，如果用规范的文档字符串为函数增加注释，则可以为程序阅读者提供友好的提示和使用说明，提高函数代码的可读性。

可以用内置函数**help()**或者函数名**.__doc__**来查看函数的注释。



6.1.2 文档字符串

【例6-3】在函数中使用文档字符串。

```
01 # docstr.py
02 # 函数定义
03 def docstr_demo(n):
04     """函数的简要描述
05
06     函数参数n: 传递函数的参数的描述"""
07     return
08
09 # 打印函数文档字符串的两种方式
10 help(docstr_demo)
11 print("-----")
12 print(docstr_demo.__doc__)
```



6.1.2 文档字符串

上面代码的执行结果如下：

```
Help on function docstr_demo in module __main__:
```

```
docstr_demo(n)
```

函数的简要描述

函数参数n： 传递函数的参数的描述

函数的简要描述

函数参数n： 传递函数的参数的描述



6.1.3 函数标注

- 函数及函数的形参都可以不用指定类型，但是这往往会导致在阅读程序或函数调用时无法知道参数的类型。
- Python提供“函数标注”（Function Annotations）的手段为形参标注类型。函数标注是关于用户自定义函数中使用的类型的元数据信息，它以字典的形式存放在函数的“`__annotations__`”属性中，并且不会影响函数的任何其他部分。
- 在下面示例中，位置参数、默认参数以及函数返回值都被标注了类型，形参标注的方式是在形参后加冒号和数据类型，函数返回值类型的标注方式是在形参列表和`def`语句结尾的冒号之间加上复合符号“`->`”和数据类型。
- 值得注意的是，函数标注仅仅是标注了参数或返回值的类型，但并不会限定参数或返回值的类型，在函数定义和调用时，参数和返回值的类型是可以改变的。



6.1.3 函数标注

【例6-4】函数参数和返回值类型的标注。

```
01 # anno_demo.py
02 # 函数定义
03 def anno_demo(p1:str,p2:str = "is my favorite!")->str :
04     s = p1 + " " + p2
05     print("函数标注: ",anno_demo.__annotations__)
06     print("传递的参数: ",p1,p2)
07     return s
08
09 # 函数调用
10 print(anno_demo("Python"))
```

代码的执行结果如下：

函数标注： {'p1': <class 'str'>, 'p2': <class 'str'>, 'return': <class 'str'>}

传递的参数： Python is my favorite!

Python is my favorite!



6.1.4 return语句

“**return** [表达式]”语句用于退出函数，选择性地向调用方返回一个表达式。特别地，不带表达式的**return**返回**None**。

【例6-5】使用**return**语句根据条件判断有选择性地返回。

```
01 # quotient.py
02 # 求商
03 def quotient(dividend,divisor):
04     if (divisor == 0):
05         return
06     else:
07         return dividend/divisor
08
09 # 函数调用
10 # 除数不为0
11 a = 99
12 b = 3
```

```
13 print( a,"/" ,b," = ",quotient(a,b))
14
15 # 除数为0
16 a = 99
17 b = 0
18 print( a,"/" ,b," = ",quotient(a,b))
```

代码的执行结果如下：

99 / 3 = 33.0

99 / 0 = None

特别地，如果函数没有**return**语句或者没有执行到**return**语句就退出函数，则该函数是以返回**None**结束。



6.1.5 变量作用域

- 在函数内部或者外部，会经常用到变量。函数内部定义的变量一般为局部变量，函数外部定义的变量为全局变量。变量起作用的代码范围称为“变量的作用域”。
- 变量（包括局部变量和全局变量）的作用域都是从定义的位置开始，在定义之前访问则会报错。
- 比如，在解释器环境中，直接使用没有定义的变量，就会报错：

```
>>> print(x,y)
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    print(x,y)
NameError: name 'x' is not defined
```



6.1.5 变量作用域

在独立代码文件中，直接使用没有定义的变量，也会报错。

【例6-6】 在独立代码文件中，直接使用没有定义的变量。

```
01 # func_var1.py
02 def func():
03     x = 1
04     print(x,y)
05     y = 100
06
07 func()
```

代码执行以后会出现类似如下的错误信息：

Traceback (most recent call last):

File "C:/Python38/mycode/chapter05/func_var1.py",
line 7, in <module>

func()

File "C:/Python38/mycode/chapter05/func_var1.py",
line 4, in func

print(x,y)

UnboundLocalError: local variable 'y' referenced before
assignment



6.1.5 变量作用域

函数内定义的局部变量，其作用域仅在函数内；一旦函数运行结束，则局部变量都被删除而不可访问。

【例6-7】全局变量与局部变量的作用域示例。

```
01 # func_var2.py
02 x,y = 2,200      #全局变量
03
04 def func():
05     x,y = 1,100   #局部变量作用域仅在函数内部
06     print("函数内部: x=%d,y=%d" % (x,y))
07
08 print("函数外部: x=%d,y=%d" % (x,y))
09 func()
10 print("函数外部: x=%d,y=%d" % (x,y))
```

代码的执行结果如下：
函数外部: x=2,y=200
函数内部: x=1,y=100
函数外部: x=2,y=200



6.1.5 变量作用域

- 从本例可以看出，虽然全局变量与局部变量名称相同，但由于作用域不同，其所包含的值也不相同。
- 在函数内部可以通过`global`定义的方式来定义全局变量，该全局变量在函数运行结束后依然存在并可访问。
- 下面对上例做简单的修改，在函数内部使用`global`定义全局变量`x`，其同名全局变量在函数外已经定义，该变量在函数内外是同一个变量，所以在函数内部该变量所有的运算结果也反映到函数外；如果在函数内部用`global`定义的全局变量，在函数外部没有同名的，则调用该函数后，创建新的全局变量。



6.1.5 变量作用域

【例6-8】在函数内部用global定义全局变量。

```
01 # func_var3.py
02 x,y = 2,200      #全局变量
03
04 def func():
05     global x
06     x,y = 1,100   #局部变量作用域仅在函数内部
07     print("函数内部: x=%d,y=%d" % (x,y))
08
09 print("函数外部: x=%d,y=%d" % (x,y))  #函数调用前
10 func()
11 print("函数外部: x=%d,y=%d" % (x,y))  #函数调用后
```

代码的执行结果如下：
函数外部: x=2,y=200
函数内部: x=1,y=100
函数外部: x=1,y=200



6.1.5 变量作用域

通过这个实例可以发现，对于变量`x`而言，在函数`func()`调用前，`x`的值为2；在函数`func()`调用时，`x`的值由2变为1，所以打印为1；在函数`func()`调用后，由于函数内部`x`是全局变量，所以其值也反映到函数外部。

而对于变量`y`而言，这个实例中的全局变量`y`和局部变量`y`是两个不同的变量。局部变量`y`在函数`func()`调用过程中不会改变全局变量`y`的值。这也说明，如果局部变量和全局变量名字相同，则局部变量在函数内部会“屏蔽”同名的全局变量。



6.1.6 函数的递归调用

递归（**recursion**）是一种特殊的函数调用形式。函数在定义时直接或间接调用自身的一种方法，目的是将大型复杂的问题转化为一个相似的但规模较小的问题。构成递归需要具备以下条件：

- 子问题须与原来的问题为同样的问题，但规模较小或更为简单；
- 调用本身须有出口，不能无限制调用，即有边界条件。

例如，求非负整数的阶乘，公式为 $n! = 1 \times 2 \times 3 \times \dots \times n$ 。可以用循环的方式来实现，即按照公式从1乘到n来获得结果。但仔细分析后可以发现，n的阶乘其实是n-1的阶乘与n的乘积，即 $n! = (n-1)! \times n$ ，这符合递归所需的条件。

下面分别用循环和递归的方式来实现，可以比较这两种实现方式。一般而言，递归会大大地减少了程序的代码量，让程序更加简洁。



6.1.6 函数的递归调用

【例6-9】用循环的方式实现非负整数的阶乘。

```
01 # factorial_loop.py
02 def factorial_loop(n):
03     """用循环的方式求非负整数n的阶乘"""
04     val = 1
05     if n==0:
06         return val
07     else:
08         i = 1
09         while i<=n:
10             val = val * i
11             i += 1
12         return val
13
14 # 调用函数
15 print(factorial_loop(5))
```

调用函数factorial_loop(5)
val=1
n不等于0，执行else语句
i=1，然后执行5次循环，计
算出1*2*3*4*5，然后退出
循环，最后返回结果120。



6.1.6 函数的递归调用

【例6-10】用递归的方式实现非负整数的阶乘。

```
01 # factorial_recursion.py
02 def factorial_recursion(n):
03     "用递归的方法求非负整数n的阶乘"
04     if n==0:
05         return 1
06     else:
07         return n*factorial_recursion(n-1)
08
09 # 调用函数
10 print(factorial_recursion(5))
```

- 调用函数factorial_recursion(5)
n不等于0, 执行else分支, 返回5*
factorial_recursion(4)
- 调用函数factorial_recursion(4)
n不等于0, 执行else分支, 返回4*
factorial_recursion(3)
- 调用函数factorial_recursion(3)
n不等于0, 执行else分支, 返回3*
factorial_recursion(2)
- 调用函数factorial_recursion(2)
n不等于0, 执行else分支, 返回2*
factorial_recursion(1)
- 调用函数factorial_recursion(1)
n不等于0, 执行else分支, 返回1*
factorial_recursion(0)
- 调用函数factorial_recursion(0)
n等于0, 执行if分支, 返回1
最终返回值就是 $5*4*3*2*1=120$



6.1.6 函数的递归调用

求斐波那契数列是另一个典型的递归案例。

斐波那契数列是这样一个数列：0、1、1、2、3、5、8、13、21、34、.....

在数学上，斐波那契数列以如下递归的方法定义：

$$F(0)=0$$

$$F(1)=1$$

$$F(n)=F(n - 1)+F(n - 2) \quad (n \geq 2, n \in \mathbb{N}^*)$$



6.1.6 函数的递归调用

【例6-11】使用递归方法求斐波那契数列中第n个元素。

```
01 # fibonacci.py
02 def fibonacci(n):
03     "求斐波那契数列中第n个元素"
04     fn = 0
05     if n == 1:
06         fn = 0
07     elif n == 2:
08         fn = 1
09     else:
10         fn = fibonacci(n-2) + fibonacci(n-1)
11     return fn
12
13 # 调用函数
14 for i in range(1,10):
15     print (fibonacci(i))
```



6.2 匿名函数

前面曾经提到，Python有一种特殊的函数叫“匿名函数”，它其实是没有使用def语句定义函数的标准方式，而是用lambda的方式来简略定义的函数。匿名函数没有函数名，其lambda表达式只可以包含一个表达式，常用于不想定义函数但又需要函数的代码复用功能的场合。

匿名函数具有以下特点：

- lambda表达式中只包含一个表达式，函数体比 def 简单很多，所以匿名函数更加简洁；
- lambda表达式的主体是一个表达式，而不是一个代码块。仅仅能在lambda表达式中封装有限的逻辑进去；
- lambda函数拥有自己的命名空间，且不能访问自己参数列表之外或全局命名空间里的参数。



6.2 匿名函数

匿名函数定义如下：

匿名函数名 = lambda [arg1 [,arg2,.....argn]]:expression

其中，arg*为参数列表，expression为表达式，表示函数要进行的操作。



6.2 匿名函数

【例6-12】 分别用普通函数和匿名函数的方式求两个数的平方差。

```
01 # variance.py
02 # 计算两个数的平方差
03
04 # 以普通函数方式定义
05 def variance1(a,b):
06     return b**2 - a**2
07
08 # 以匿名函数方式定义
09 variance2 = lambda a,b: b**2 - a**2
10
11 x,y = 4,5
12
13 # 普通函数调用
14 print("以普通函数方式定义的函数计算: ")
15 print("{}*{} - {}*{} = {}".format(y,y ,x ,x,variance1(x,y)))
16
17 # 匿名函数调用
18 print("=====")
19 print("以匿名函数方式定义的函数计算: ")
20 print("{}*{} - {}*{} = {}".format(y,y ,x ,x,variance2(x,y)))
```

代码的执行结果如下:

以普通函数方式定义的函数计算:

$$5*5 - 4*4 = 9$$

=====

以匿名函数方式定义的函数计算:

$$5*5 - 4*4 = 9$$



6.2 匿名函数

- 从这个实例可以看出，对于不复杂的仅用一个表达式即可实现的函数，匿名函数与普通函数差别不大。
- 与普通函数比较而言，匿名函数的调用形式相同，定义会更简洁一些。
- 所以，对于只需要包含一个表达式的函数，可以选择匿名函数；而对于复杂的、不能用一个表达式来完成任务，匿名函数则力不从心，此时使用普通函数是正确的选择。



6.3 参数传递

- 参数是函数的重要组成部分。
- 函数定义语法中，函数名后括号内参数列表是用逗号分隔开的形参列表（**parameters**），可以包含0个或多个参数。
- 在调用函数时，向函数传递实参（**arguments**），根据不同的实参参数类型，将实参的值或引用传递给形参。



6.3 参数传递

- 通过前面章节的学习我们已经知道，在Python中，各种数据类型都是对象，其中字符串、数字、元组等是不可变（immutable）对象，列表、字典等是可变（mutable）对象。

- 而变量赋值某个数据类型的对象时，不需要事先声明变量名及其类型，直接赋值即可，此时，不仅变量的“值”发生变化，变量的“类型”也随之发生变化。这里之所以打引号，是因为变量本身并无类型，也不直接存储值，而是存储了值的内存地址或者引用（指针）。

Python函数在传递不可变对象和可变对象这两种参数时，变量的值和存储的地址是否发生变化？只有明晰这个问题，才能更好地编写函数。



6.3 参数传递

6.3.1 给函数传递不可变对象

6.3.2 给函数传递可变对象

6.3.3 关于参数传递的总结

本PPT是如下教材的配套讲义：

《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社

《Python程序设计基础教程（微课版）》教材官方网站：

<http://dbl原因.xmu.edu.cn/post/python>

高等院校程序设计新形态精品系列

PYTHON

Python Programming Language

Python

程序设计基础教程

| 微课版 |

林子雨 赵江声 陶继平 编著



名师精品

多年计算机教学实践的厚积薄发



深入浅出

清晰呈现 Python 语言学习路径



实例丰富

有效提升编程语言的学习趣味



资源全面

构建全方位一站式在线服务体系



中国工信出版集团



人民邮电出版社
POSTS & TELECOM PRESS



6.3.1 给函数传递不可变对象

下面的示例函数的参数传递的是数字类型，是不可变对象，这里要重点观察其定义及调用阶段变量的变化情况。



6.3.1 给函数传递不可变对象

【例6-13】给函数传递不可变对象。

```
01 # transfer_immutable.py
02 # 函数定义
03 def transfer_immutable(var):
04     print("-----函数内部-----")
05     print("函数内部赋值前，变量值：",var," --- 变量地址：",id(var))
06     var += 77
07     print("函数内部赋值后，变量值：",var," --- 变量地址：",id(var))
08     print("-----函数内部-----")
09     return(var)
10
11 # 函数调用
12 var_a = 11
13 print("函数外部调用前，变量值：",var_a," --- 变量地址：",id(var_a))
14 transfer_immutable(var_a)
15 print("函数外部调用后，变量值：",var_a," --- 变量地址：",id(var_a))
```



6.3.1 给函数传递不可变对象

上面代码的执行结果如下：

函数外部调用前，变量值： 11 --- 变量地址： 140705103677424

-----函数内部-----

函数内部赋值前，变量值： 11 --- 变量地址： 140705103677424

函数内部赋值后，变量值： 88 --- 变量地址： 140705103679888

-----函数内部-----

函数外部调用后，变量值： 11 --- 变量地址： 140705103677424

从这个实例可以看出，不可变对象在传递给函数时，实参和形参是同一个对象（值和地址都相同），但在函数内部，不可变对象类型的变量重新赋值后，形参变成一个新的对象（地址发生改变），函数外部的实参在函数调用前后并未发生改变。可见，对于不可变对象的实参，传递给函数的仅仅是值，函数不会影响其引用（存放地址）。



6.3.2 给函数传递可变对象

将上面的实例做简单的修改，函数内部几乎相同，不同的是实参的类型是可变对象的列表。

【例6-14】给函数传递可变对象。

```
01 # transfer_mutable.py
02 # 函数定义
03 def transfer_mutable(varlist):
04     print("-----函数内部-----")
05     print("函数内部赋值前，变量值：",varlist," --- 变量地址：",id(varlist))
06     varlist += [4,5,6,7]
07     print("函数内部赋值后，变量值：",varlist," --- 变量地址：",id(varlist))
08     print("-----函数内部-----")
09     return(varlist)
10
11 # 函数调用
12 var_a = [1,2,3]
13 print("函数外部调用前，变量值：",var_a," --- 变量地址：",id(var_a))
14 transfer_mutable(var_a)
15 print("函数外部调用后，变量值：",var_a," --- 变量地址：",id(var_a))
```



6.3.2 给函数传递可变对象

上面代码的执行结果如下：

函数外部调用前，变量值： [1, 2, 3] --- 变量地址： 2353139853056

-----函数内部-----

函数内部赋值前，变量值： [1, 2, 3] --- 变量地址： 2353139853056

函数内部赋值后，变量值： [1, 2, 3, 4, 5, 6, 7] --- 变量地址：

2353139853056

-----函数内部-----

函数外部调用后，变量值： [1, 2, 3, 4, 5, 6, 7] --- 变量地址：

2353139853056

通过这个实例可以发现，传递可变对象的实参，事实上是把值和引用都传递给形参。函数内部对于形参的改变，也同时改变了实参。



6.3.3 关于参数传递的总结

关于Python函数对于不可变对象和可变对象这两种参数传递形式，这里做如下总结：

- 不可变对象：对于函数`fun(a)`，调用时传递不可变对象类型的值给`a`，传递的只是`a`的值，没有影响`a`对象本身。如果在`fun(a)`内部修改`a`的值，则是新生成一个`a`。
- 可变对象：对于`fun(la)`，调用时传递可变对象类型的值给`la`，则是将`la`真正地传过去，在函数`fun(la)`内部修改`la`后，`fun(la)`外部的`la`也会受影响。



6.4 参数类型

- 6.4.1 位置参数
- 6.4.2 关键字参数
- 6.4.3 默认参数
- 6.4.4 不定长参数
- 6.4.5 特殊形式
- 6.4.6 参数传递的序列解包

本PPT是如下教材的配套讲义：
《Python程序设计基础教程（微课版）》

厦门大学 林子雨,赵江声,陶继平 编著，人民邮电出版社
《Python程序设计基础教程（微课版）》教材官方网站：
<http://dblalab.xmu.edu.cn/post/python>

高等院校程序设计新形态精品系列

PYTHON

Python Programming Language

Python

程序设计基础教程

| 微课版 |

林子雨 赵江声 陶继平 编著

-  **名师精品**
多年计算机教学实践的厚积薄发
-  **深入浅出**
清晰呈现 Python 语言学习路径
-  **实例丰富**
有效提升编程语言的学习趣味
-  **资源全面**
构建全方位一站式在线服务体系

中国工信出版集团 人民邮电出版社
POSTS & TELECOM PRESS



6.4.1 位置参数

- 位置参数，顾名思义，对于定义的此类参数，在函数调用时是必须有的，而且顺序和数量都要保持一致。
- 下面实例中定义了一个有两个位置参数的函数，第一次调用时没有参数，运行时报错显示缺少两个必需的参数；第二次调用时仅有一个参数，运行时报错信息不同，显示缺少第二个参数；第三次调用时，提供了两个参数，函数正确运行。



6.4.1 位置参数

```
>>> #定义一个函数，其有两个位置参数
```

```
>>> def required_param(p1,p2):  
    print(p1,p2)  
    return
```

```
>>> #调用函数，缺少2个参数
```

```
>>> required_param()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#9>", line 1, in <module>  
    required_param()
```

```
TypeError: required_param() missing 2 required positional arguments: 'p1' and 'p2'
```

```
>>> #调用函数，缺少1个参数
```

```
>>> required_param("a string")
```

```
Traceback (most recent call last):
```

```
File "<pyshell#11>", line 1, in <module>  
    required_param("a string")
```

```
TypeError: required_param() missing 1 required positional argument: 'p2'
```

```
>>> #调用函数，提供2个参数
```

```
>>> required_param("Hello","World")
```

```
Hello World
```



6.4.2 关键字参数

在函数调用时，参数的传入使用了参数的名称，则该类参数称为“关键字参数”。使用关键字参数允许函数调用时参数的顺序与声明时不一致，因为Python解释器能够用参数名匹配参数值。所以，在函数调用中，多个关键字参数的顺序可以随意，但是关键字参数必须跟随在位置参数的后面。实例如下：

```
>>> # 定义一个函数，有1个位置参数、2个关键字参数
>>> def key_param(p1,kp1,kp2):
    print(p1,kp1,kp2)
    return
>>> # 调用函数
>>> key_param("p1",kp2="kp2",kp1="kp1")
p1 kp1 kp2
```



6.4.2 关键字参数

在这个例子中，在`key_param("p1",kp2="kp2",kp1="kp1")`这种调用下，函数定义中的`kp1`、`kp2`是关键字参数，而`p1`是位置参数，这是由函数的调用而决定的。此时，函数调用时，`p1`必须有，而且位置跟声明时一致，即放在第一个位置；而`kp1`、`kp2`则顺序可随意。

如果函数调用改为：`key_param(p1="p1",kp2="kp2",kp1="kp1")`，那么此时`p1`是何种参数呢？这时`p1`也变为关键字参数了，可见关键字参数与函数调用紧密相关。实际上，函数调用也可以改为`key_param(kp2="kp2",p1="p1",kp1="kp1")`，也就是说，当三个参数都是关键字参数的时候，三个参数传递的顺序可以随意。



6.4.3 默认参数

- 如果在函数定义时，某个参数使用了默认值，则该参数是默认参数；如果函数调用时没有传递该参数，则使用默认值。
- 下面以某学生社团纳新的会员管理为例，假定社团的新成员一般年级为“大一”，所以`grade`参数使用默认参数。



6.4.3 默认参数

【例6-15】在函数定义中使用默认参数。

```
01 # param1.py
02 # 定义一个函数
03 # 某社团纳新，会员一般为大一学生
04 def new_member(name,student_id,grade="大一"):
05     print("姓名",name)
06     print("学号",student_id)
07     print("年级",grade)
08     print("-----")
09     return
10
11 # 调用函数
12 new_member("张三","0001")
13 new_member("李四","0002","大二")
```

代码的执行结果如下：

```
姓名 张三
学号 0001
年级 大一
```

```
-----
姓名 李四
学号 0002
年级 大二
-----
```



6.4.3 默认参数

值得注意的是，默认参数是在函数定义的时候就计算的。下面的示例中，函数调用`func()`会打印什么值呢？答案是在函数定义时就确定的值，即“大一”，而非在函数定义之后才被赋值的“大二”。

```
>>> gradeone = '大一'
>>> def func(grade=gradeone):
    print(grade)
>>> gradeone = '大二'
>>> #函数调用
>>> func()
大一
```



6.4.4 不定长参数

如果我们希望函数参数的个数不确定时，往往需要用到不定长参数。不定长函数的定义方式主要有两种方式：`*parameter`和`**parameter`，前者是接收多个实参并将其放在一个元组中，后者则是接收键值对并将其放在字典中。这两种定义方式的基本语法分别如下：

方式一：

```
def functionname([formal_args,] *var_args_tuple ):
    function_suite
    return [expression]
```

方式二：

```
def functionname([formal_args,] **var_args_dict ):
    function_suite
    return [expression]
```



6.4.4 不定长参数

下面给出第一种方式的示例函数，它使用了不定长参数。

【例6-16】对所有数字参数求和。

```
01 # param2.py
02 # 定义函数
03 def cal_sum(*a):
04     sum = 0
05     for ele in a:
06         sum += ele
07     return sum
08
09 # 调用函数
10 print(cal_sum(1,2))
11 print(cal_sum(1,2,3,4))
```

代码的执行结果是：

```
3
10
```



6.4.4 不定长参数

下面给出第二种方式的示例函数，它使用了不定长参数，实参传递进函数后被转变为字典类型。

【例6-17】 使用不定长参数，实参传递进函数后被转变为字典类型。

```
01 # param3.py
02 # 函数定义
03 def userinfo(**p):
04     print(p)
05     for k,v in p.items():
06         print(k,":",v)
07
08 # 函数调用
09 userinfo(name = 'zhangsan' , id = '0001' , sex= 'male')
10 print("=====")
11 userinfo(name = 'lisi' , id = '0002' , sex= 'female')
12 print("=====")
13 userinfo(name = 'wangwu' , id = '0003' , sex= 'female')
```



6.4.4 不定长参数

上面代码的执行结果如下：

```
{'name': 'zhangsan', 'id': '0001', 'sex': 'male'}
```

```
name : zhangsan
```

```
id : 0001
```

```
sex : male
```

```
=====
```

```
{'name': 'lisi', 'id': '0002', 'sex': 'female'}
```

```
name : lisi
```

```
id : 0002
```

```
sex : female
```

```
=====
```

```
{'name': 'wangwu', 'id': '0003', 'sex': 'female'}
```

```
name : wangwu
```

```
id : 0003
```

```
sex : female
```



6.4.5 特殊形式

Python中为了确保可读性和运行效率，可以对参数传递形式进行限制。通过在函数定义的参数列表中增加“/”或“*”（可选），来确定参数项是仅按位置、按位置也按关键字，还是仅按关键字传递。以下为函数定义参数列表中增加“/”或“*”的相关语法和规定：

```
def f(pos1, pos2, / , pos_or_kwd, * , kwd1, kwd2):
```

①

②

③

①仅允许位置参数（Positional only）

②可位置参数或关键字参数（Positional or keyword）

③仅允许关键字参数（Keyword only）



6.4.5 特殊形式

- 如果函数定义中未使用“/”和“*”，则参数传递的类型可以是位置参数或关键字参数。
- 如果函数定义中使用“/”，则“/”前面的形参为仅限位置参数，这意味着函数调用时，实参应与形参一一对应，也不能按照关键字参数的方式来传递；如果函数定义中使用“*”，则其后的形参为仅限关键字参数；而在“/”和“*”之间的形参可以为位置参数或者关键字参数。



6.4.6 参数传递的序列解包

在函数定义的参数列表中，如果包含有多个位置参数的形参，则可以用列表、元组、集合、字典或其他可迭代的对象作为实参来进行参数传递，这需要在实参名称前加一个星号（*），此时Python解释器会将实参进行所谓的解包操作，将序列中的值分别传递给多个单变量的形参。



6.4.6 参数传递的序列解包

下面是序列解包的实例：

```
>>> def func(p1,p2,p3):  
    # 形参列表中有多个位置参数  
    print(p1,p2,p3)  
    return  
  
>>> list1 = ["a1","a2","a3"]  
>>> func(*list1) #对列表进行解包  
a1 a2 a3  
  
>>> tup1 = ("a1","a2","a3")  
>>> func(*tup1) #对元组进行解包  
a1 a2 a3
```



6.4.6 参数传递的序列解包

```
>>> dict={'a':1,'b':2,'c':3}
>>> func(*dict) #对字典的键进行解包
a b c
>>> func(*dict.values()) #对字典的值进行解包
1 2 3
>>> set1 = {'a','b','c'}
>>> func(*set1) #对集合进行解包
a b c
>>> r1 = range(1,4)
>>> func(*r1) #对其他序列类型进行解包
1 2 3
```



6.4.6 参数传递的序列解包

这是单个星号（*）对参数传递的序列解包的情形，而两个星号（**）则是针对字典的值进行解包的。需要注意的是，字典的键须要与形参的名称保持一致，否则会报错。实例如下：

```
>>> def func(p1,p2,p3):
    # 形参列表中有多个位置参数
    print(p1,p2,p3)
    return
>>> p = {'p1':1,'p2':2,'p3':3}
>>> func(**p)    #对字典进行解包，注意键与形参一致
1 2 3
>>> func(*p.values())    #对字典的值进行解包
1 2 3
```



6.4.6 参数传递的序列解包

```
>>> p = {'a1':1,'a2':2,'a3':3} #字典的键与形参名称不一致
>>> func(**p) #解包时出错
Traceback (most recent call last):
  File "<pyshell#58>", line 1, in <module>
    func(**p) #解包时出错
TypeError: func() got an unexpected keyword argument 'a1'
```



附录A：主讲教师林子雨简介



主讲教师：林子雨

单位：厦门大学计算机科学与技术系

E-mail: ziyulin@xmu.edu.cn

个人网页: <http://dmlab.xmu.edu.cn/post/linziyu>

数据库实验室网站: <http://dmlab.xmu.edu.cn>

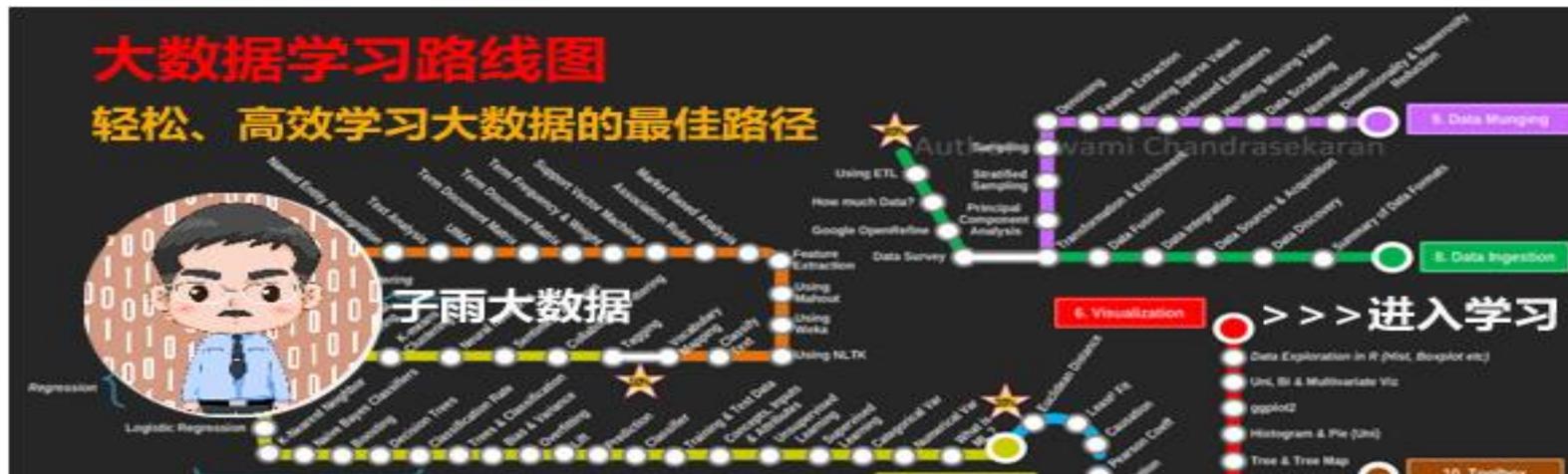


扫一扫访问个人主页

林子雨，男，1978年出生，博士（毕业于北京大学），全国高校知名大数据教师，现为厦门大学计算机科学系副教授，厦门大学信息学院实验教学中心主任，曾任厦门大学信息科学与技术学院院长助理、晋江市发展和改革局副局长。中国计算机学会数据库专业委员会委员，中国计算机学会信息系统专业委员会委员。国内高校首个“数字教师”提出者和建设者，厦门大学数据库实验室负责人，厦门大学云计算与大数据研究中心主要建设者和骨干成员，2013年度、2017年度和2020年度厦门大学教学类奖教金获得者，荣获2019年福建省精品在线开放课程、2018年厦门大学高等教育成果特等奖、2018年福建省高等教育教学成果二等奖、2018年国家精品在线开放课程。主要研究方向为数据库、数据仓库、数据挖掘、大数据、云计算和物联网，并以第一作者身份在《软件学报》《计算机学报》和《计算机研究与发展》等国家重点期刊以及国际学术会议上发表多篇学术论文。作为项目负责人主持的科研项目包括1项国家自然科学基金青年基金项目(No.61303004)、1项福建省自然科学基金青年基金项目(No.2013J05099)和1项中央高校基本科研业务费项目(No.2011121049)，主持的教改课题包括1项2016年福建省教改课题和1项2016年教育部产学协作育人项目，同时，作为课题负责人完成了国家发改委城市信息化重大课题、国家物联网重大应用示范工程区域试点泉州市工作方案、2015泉州市互联网经济调研等课题。中国高校首个“数字教师”提出者和建设者，2009年至今，“数字教师”大平台累计向网络免费发布超过1000万字高价值的研究和教学资料，累计网络访问量超过1000万次。打造了中国高校大数据教学知名品牌，编著出版了中国高校第一本系统介绍大数据知识的专业教材《大数据技术原理与应用》，并成为京东、当当网等网店畅销书籍；建设了国内高校首个大数据课程公共服务平台，为教师教学和学生学习大数据课程提供全方位、一站式服务，年访问量超过400万次，累计访问量超过1500万次。



附录B：大数据学习路线图



大数据学习路线图访问地址：<http://dblab.xmu.edu.cn/post/10164/>



附录C：林子雨大数据系列教材



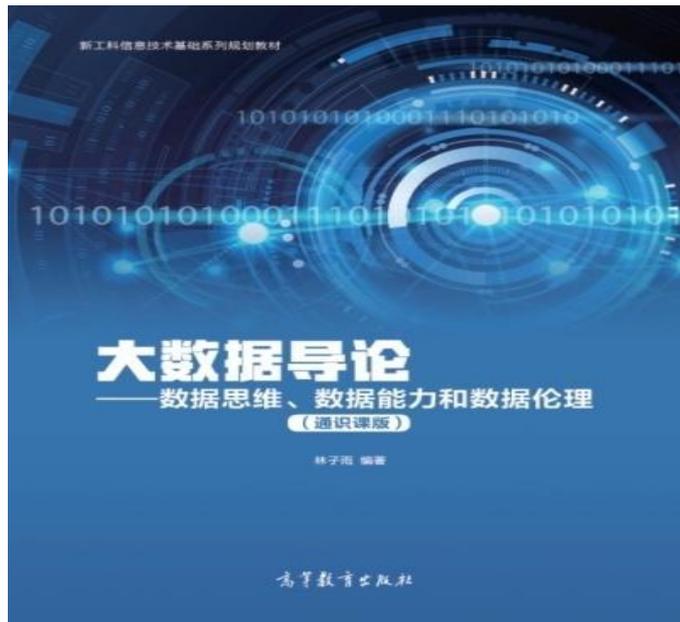
林子雨大数据系列教材
用于导论课、专业课、实训课、公共课

了解全部教材信息：<http://dbllab.xmu.edu.cn/post/bigdatabook/>



附录D：《大数据导论（通识课版）》教材

开设全校公共选修课的优质教材



本课程旨在实现以下几个培养目标：

- 引导学生步入大数据时代，积极投身大数据的变革浪潮之中
- 了解大数据概念，培养大数据思维，养成数据安全意识
- 认识大数据伦理，努力使自己的行为符合大数据伦理规范要求
- 熟悉大数据应用，探寻大数据与自己专业的应用结合点
- 激发学生基于大数据的创新创业热情

高等教育出版社 ISBN:978-7-04-053577-8 定价：32元 版次：2020年2月第1版
教材官网：<http://dbl原因.xmu.edu.cn/post/bigdataintroduction/>



附录E：《大数据导论》教材

- 林子雨 编著《大数据导论》
- 人民邮电出版社，2020年9月第1版
- ISBN:978-7-115-54446-9 定价：49.80元

教材官网：<http://dblab.xmu.edu.cn/post/bigdata-introduction/>



开设大数据专业导论课的优质教材



扫一扫访问教材官网



附录F：《大数据技术原理与应用（第3版）》教材

《大数据技术原理与应用——概念、存储、处理、分析与应用（第3版）》，由厦门大学计算机科学系林子雨博士编著，是国内高校第一本系统介绍大数据知识的专业教材。人民邮电出版社 ISBN:978-7-115-54405-6 定价：59.80元

全书共有17章，系统地论述了大数据的基本概念、大数据处理架构Hadoop、分布式文件系统HDFS、分布式数据库HBase、NoSQL数据库、云数据库、分布式并行编程模型MapReduce、Spark、流计算、Flink、图计算、数据可视化以及大数据在互联网、生物医学和物流等各个领域的应用。在Hadoop、HDFS、HBase、MapReduce、Spark和Flink等重要章节，安排了入门级的实践操作，让读者更好地学习和掌握大数据关键技术。

本书可以作为高等院校计算机专业、信息管理等相关专业的大数据课程教材，也可供相关技术人员参考、学习、培训之用。

欢迎访问《大数据技术原理与应用——概念、存储、处理、分析与应用》教材官方网站：

<http://dbllab.xmu.edu.cn/post/bigdata3>



扫一扫访问教材官网





附录G：《大数据基础编程、实验和案例教程（第2版）》

本书是与《大数据技术原理与应用（第3版）》教材配套的唯一指定实验指导书

大数据教材



1+1黄金组合
厦门大学林子雨编著

配套实验指导书



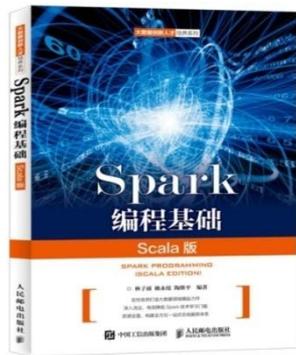
- 步步引导，循序渐进，详尽的安装指南为顺利搭建大数据实验环境铺平道路
- 深入浅出，去粗取精，丰富的代码实例帮助快速掌握大数据基础编程方法
- 精心设计，巧妙融合，八套大数据实验题目促进理论与编程知识的消化和吸收
- 结合理论，联系实际，大数据课程综合实验案例精彩呈现大数据分析全流程

林子雨编著《大数据基础编程、实验和案例教程（第2版）》

清华大学出版社 ISBN:978-7-302-55977-1 定价：69元 2020年10月第2版



附录H：《Spark编程基础（Scala版）》



《Spark编程基础（Scala版）》

厦门大学 林子雨，赖永炫，陶继平 编著

披荆斩棘，在大数据丛林中开辟学习捷径
填沟削坎，为快速学习Spark技术铺平道路
深入浅出，有效降低Spark技术学习门槛
资源全面，构建全方位一站式在线服务体系

人民邮电出版社出版发行，ISBN:978-7-115-48816-9
教材官网：<http://dblab.xmu.edu.cn/post/spark/>

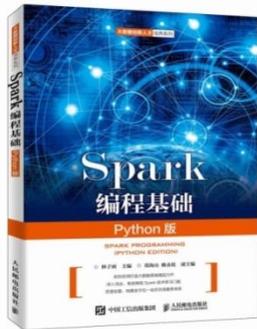


本书以Scala作为开发Spark应用程序的编程语言，系统介绍了Spark编程的基础知识。全书共8章，内容包括大数据技术概述、Scala语言基础、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作，以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源，包括讲义PPT、习题、源代码、软件、数据集、授课视频、上机实验指南等。



附录I: 《Spark编程基础 (Python版)》

《Spark编程基础 (Python版)》



厦门大学 林子雨, 郑海山, 赖永炫 编著

披荆斩棘, 在大数据丛林中开辟学习捷径
填沟削坎, 为快速学习Spark技术铺平道路
深入浅出, 有效降低Spark技术学习门槛
资源全面, 构建全方位一站式在线服务体系



人民邮电出版社出版发行, ISBN:978-7-115-52439-3

教材官网: <http://dbllab.xmu.edu.cn/post/spark-python/>

本书以Python作为开发Spark应用程序的编程语言, 系统介绍了Spark编程的基础知识。全书共8章, 内容包括大数据技术概述、Spark的设计与运行原理、Spark环境搭建和使用方法、RDD编程、Spark SQL、Spark Streaming、Structured Streaming、Spark MLlib等。本书每个章节都安排了入门级的编程实践操作, 以便读者更好地学习和掌握Spark编程方法。本书官网免费提供了全套的在线教学资源, 包括讲义PPT、习题、源代码、软件、数据集、上机实验指南等。



附录J：高校大数据课程公共服务平台



高校大数据课程

公 共 服 务 平 台

<http://dblab.xmu.edu.cn/post/bigdata-teaching-platform/>



扫一扫访问平台主页 扫一扫观看3分钟FLASH动画宣传片



附录K：高校大数据实训课程系列案例教材

为了更好地满足高校开设大数据实训课程的教材需求，厦门大学数据库实验室林子雨老师团队联合企业共同开发了《高校大数据实训课程系列案例》，目前已经完成开发的系列案例包括：

- 《电影推荐系统》（已经于2019年5月出版）
- 《电信用户行为分析》（已经于2019年5月出版）
- 《实时日志流处理分析》
- 《微博用户情感分析》
- 《互联网广告预测分析》
- 《网站日志处理分析》



系列案例教材将于2019年陆续出版发行，教材相关信息，敬请关注网页后续更新！

<http://dblab.xmu.edu.cn/post/shixunkecheng/>



扫一扫访问大数据实训课程系列案例教材主页

The background is a solid blue color with faint, light-blue silhouettes of people. At the top, there are two groups of people holding hands. On the right side, there is a silhouette of a person standing with their hand on their head. On the left side, there are silhouettes of people sitting or standing in a group.

Thank You!

Department of Computer Science, Xiamen University, 2022