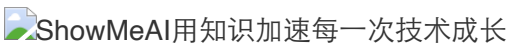


图解 Python 编程(18) | 函数

🕒 2021-11-09 👁 1589 💬 0 📁 工具教程 python 编程语言



作者：韩信子@ShowMeAI
教程地址：<https://www.showmeai.tech/tutorials/56>
本文地址：<https://www.showmeai.tech/article-detail/81>
声明：版权所有，转载请联系平台与作者并注明出处
收藏ShowMeAI更多精彩内容

Python函数

函数是组织好的，可重复使用的，用来实现单一， 或相关联功能的代码段。

函数能提高应用的模块性，和代码的重复利用率。你已经知道Python提供了许多内建函数，比如print()。你也可以自己创建函数，这被叫做用户自定义函数。

定义一个函数

你可以定义一个由自己想要功能的函数，以下是简单的规则：

- 函数代码块以 **def** 关键词开头，后接函数标识符名称和圆括号 **()**。
- 任何传入参数和自变量必须放在圆括号中间，圆括号之间可以用于定义参数。
- 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。
- 函数内容以冒号 **:** 起始，并且缩进。
- **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** 相当于返回 **None**。

一个函数实例

函数可以输入参数，并设置返回值

```
def关键字定义函数
def my_max(a,b):
    if a > b:
        return a
    else:
        return b
return关键字设置返回值
```

a,b为函数参数

函数体

a,b为函数的输入参数

搜索 | 微信 ShowMeAI 研究中心

<http://www.showmeai.tech/>

语法

Python 定义函数使用 **def** 关键字，一般格式如下：

1. **def** 函数名（参数列表）：
2. 函数体

默认情况下，参数值和参数名称是按函数声明中定义的顺序匹配起来的。如下为示例代码（代码可以在在线python3环境中运行）：

让我们使用函数来输出”Hello World! ”：

1. **def** hello() :
2. **print**(”Hello World!”)
- 3.
4. hello()

更复杂点的应用，函数中带上参数变量：

1. **def** my_max(a, b):
2. **if** a > b:
3. **return** a
4. **else:**
5. **return** b
- 6.
7. a = 4
8. b = 5
9. **print**(my_max(a, b))

以上实例输出结果：

1. 5

计算面积函数，如下为示例代码（代码可以在在线python3环境中运行）：

1. **# 计算面积函数**
2. **def** cal_area(width, height):
3. **return** width * height
- 4.
5. **def** my_welcome(name):
6. **print**(”Welcome”, name)
- 7.
8. my_welcome(”ShowMeAI”)
9. w = 4
10. h = 5

```
11. print("width =", w, " height =", h, " area =", cal_area(w, h))
```


以上实例输出结果：

```
1. Welcome ShowMeAI
2. width = 4 height = 5 area = 20
```

函数调用

定义一个函数：给了函数一个名称，指定了函数里包含的参数，和代码块结构。

这个函数的基本结构完成以后，你可以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。



定义函数 & 调用函数

a接收了11，b接收了22

```
#定义函数
def add_num(a,b): ← a,b为形参
    c = a + b
    print(c)

#调用参数
add_num(11,22) ← 11,22为实参
```

a

b

11

22

↑ ↑

搜索 | 微信 ShowMeAI 研究中心

<http://www.showmeai.tech/>

如下代码调用了 `print_myself()` 函数：

```
1. # 定义函数
2. def print_myself( str ):
3.     # 打印任何传入的字符串
4.     print (str)
5.     return
6.
7. # 调用函数
8. print_myself("调用用户自定义函数!")
9. print_myself("再次调用同一函数")
```

以上实例输出结果：

```
1. 调用用户自定义函数!
2. 再次调用同一函数
```

参数传递

在 python 中，类型属于对象，变量是没有类型的：

```
1. a=[1,2,3]
2.
3. a="ShowMeAI"
```

以上代码中，`[1,2,3]` 是 List 类型，“`ShowMeAI`” 是 String 类型，而变量 `a` 是没有类型，它仅仅是一个对象的引用（一个指针），可以是指向 List 类型对象，也可以是指向 String 类型对象。

可更改(mutable)与不可更改(immutable)对象

在 python 中，strings, tuples, 和 numbers 是不可更改的对象，而 list,dict 等则是可以修改的对象。

- 不可变类型：变量赋值 `a=10` 后再赋值 `a=5`，这里实际是新生成一个 int 值对象 5，再让 `a` 指向它，而 10 被丢弃，不是改变 `a` 的值，相当于新生成了 `a`。
- 可变类型：变量赋值 `l=[1,2,3,4]` 后再赋值 `l[2]=5` 则是将 list `l` 的第三个元素值更改，本身 `l` 没有动，只是其内部的一部分值被修改了。

python 函数的参数传递：

- 不可变类型：类似 C++ 的值传递，如整数、字符串、元组。如 `func(a)`，传递的只是 `a` 的值，没有影响 `a` 对象本身。如果在 `func(a)` 内部修改 `a` 的值，则是新生成一个 `a` 的对象。
- 可变类型：类似 C++ 的引用传递，如 列表，字典。如 `func(l)`，则是将 `l` 真正的传过去，修改后 `func` 外部的 `l` 也会受影响

python 中一切都是对象，严格意义我们不能说值传递还是引用传递，我们应该说传不可变对象和传可变对象。

python传不可变对象实例

通过 `id()` 函数来查看内存地址变化：

```
1. def my_change(a):
2.     print(id(a)) # 指向的是同一个对象
3.     a=10
4.     print(id(a)) # 一个新对象
5.
6. a=5
7. print(id(a))
8. my_change(a)
```

以上实例输出结果为：

```
1. 9788736
2. 9788736
3. 9788896
```

01 17:00:19

可以看见在调用函数前后，形参和实参指向的是同一个对象（对象 id 相同），在函数内部修改形参后，形参指向的是不同的 id。

传可变对象实例

可变对象在函数里修改了参数，那么在调用这个函数的函数里，原始的参数也被改变了。如下为示例代码（代码可以在在线python3环境中运行）：

```
1. def change_list( mylist ):
2.     "修改传入的列表"
3.     mylist.append([1,2,3,4])
4.     print ("函数内取值: ", mylist)
5.     return
6.
7. # 调用changeme函数
8. mylist = [10,20,30]
9. change_list( mylist )
10. print ("函数外取值: ", mylist)
```

传入函数的和在末尾添加新内容的对象用的是同一个引用。故输出结果如下：

```
1. 函数内取值: [10, 20, 30, [1, 2, 3, 4]]
2. 函数外取值: [10, 20, 30, [1, 2, 3, 4]]
```

参数

以下是调用函数时可使用的正式参数类型：

- 必需参数
- 关键字参数
- 默认参数
- 不定长参数



函数调用

定义	def fun(name , age , *args , gender = '男')
调用1	fun('ShowMeAI' , 3 , 1,2,3, '男'] <small>没有关键词参数，这些全部传入 *args</small>
定义	def fun(name , age , *args , gender = '男')
调用2	fun('ShowMeAI' , 3 , 1,2,3, gender = '男'] <small>普通参数全部完全匹配 下一个参数开始之后的普通实参传入*args 直到遇到一个关键实参结束</small>



 <http://www.showmeai.tech/>  搜索 | 微信 **ShowMeAI** 研究中心

必需参数

必需参数须以正确的顺序传入函数。调用时的数量必须和声明时的一样。

调用 print_myself() 函数，你必须传入一个参数，不然会出现语法错误：

```
1. def print_myself( str ):
2.     "打印任何传入的字符串"
3.     print(str)
4.     return
5.
6. # 调用 print_myself 函数，不加参数会报错
7. print_myself()
```

以上实例输出结果：

```
1. Traceback (most recent call last):
2.   File "test.py", line 10, in <module>
3.     print_myself()
4. TypeError: print_myself() missing 1 required positional argument: 'str'
```

关键字参数

关键字参数和函数调用关系紧密，函数调用使用关键字参数来确定传入的参数值。

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。

以下实例在函数 print_myself() 调用时使用参数名：

```
1. def print_myself( str ):
2.     "打印任何传入的字符串"
3.     print(str)
4.     return
5.
6. #调用print_myself函数
7. print_myself( str = "ShowMeAI知识社区")
```

以上实例输出结果：

```
1. ShowMeAI知识社区
```

以下示例代码（在线python3环境）中演示了函数参数的使用不需要使用指定顺序：

```
1. def print_info( name, age ):
```

```
1. def print_info( name, age ):
2.     "打印任何传入的字符串"
3.     print ("名字: ", name)
4.     print ("年龄: ", age)
5.     return
6.
7. #调用print_info函数
8. print_info( age=30, name="ShowMeAI" )
```

以上实例输出结果：

```
1. 名字: ShowMeAI
2. 年龄: 30
```

默认参数

调用函数时，如果没有传递参数，则会使用默认参数。以下代码（[在线python3环境](#)）中如果没有传入 age 参数，则使用默认值：

```
1. def print_info( name, age = 35 ):
2.     "打印任何传入的字符串"
3.     print ("名字: ", name)
4.     print ("年龄: ", age)
5.     return
6.
7. #调用print_info函数
8. print_info( age=30, name="ShowMeAI" )
9. print("-----")
10. print_info( name="ShowMeAI" )
```

以上实例输出结果：

```
1. 名字: ShowMeAI
2. 年龄: 30
3. -----
4. 名字: ShowMeAI
5. 年龄: 35
```

不定长参数

你可能需要一个函数能处理比当初声明时更多的参数。这些参数叫做不定长参数，和上述 2 种参数不同，声明时不会命名。基本语法如下：

```
1. def function_name([formal_args,] *var_args_tuple ):
2.     "函数_文档字符串"
3.     function_suite
4.     return [expression]
```

加了星号 * 的参数会以元组(tuple)的形式导入，存放所有未命名的变量参数。

```
1. def print_info( arg1, *vartuple ):
2.     "打印任何传入的参数"
3.     print("输出: ")
4.     print(arg1)
5.     print(vartuple)
6.
7. # 调用print_info 函数
8. print_info( 100, 90, 80 )
```

以上实例输出结果：

```
1. 输出:
2. 100
3. (90, 80)
```

如果在函数调用时没有指定参数，它就是一个空元组。我们也可以不向函数传递未命名的变量。如下代码（[在线python3环境](#)）：

```
1. def print_info( arg1, *vartuple ):
2.     "打印任何传入的参数"
3.     print("输出: ")
4.     print(arg1)
5.     for var in vartuple:
6.         print (var)
7.     return
8.
9. # 调用printinfo 函数
10. print_info( 100 )
11. print_info( 90, 80, 70 )
```

以上实例输出结果：

```
1. 输出:
2. 100
3. 输出:
4. 90
5. 80
6. 70
```

还有一种就是参数带两个星号 **基本语法如下：


```
1. def function_name([formal_args,] **var_args_dict ):
2.     "函数_文档字符串"
3.     function_suite
4.     return [expression]
```

加了两个星号 ** 的参数会以字典的形式导入。

```
1. def print_info( arg1, **vardict ):
2.     "打印任何传入的参数"
3.     print("输出: ")
4.     print(arg1)
5.     print(vardict)
6.
7. # 调用print_info 函数
8. print_info(1, a=2,b=3)
```

以上实例输出结果：

```
1. 输出:
2. 1
3. {'a': 2, 'b': 3}
```

声明函数时，参数中星号 * 可以单独出现，例如：

```
1. def f(a,b,*,c):
2.     return a+b+c
```

如果单独出现星号 * 后的参数必须用关键字传入。


```
1. def f(a,b,*,c):
2.     return a+b+c
3.
4. f(1,2,c=3) # 正常
5. f(1,2,3)  # 报错
```

匿名函数

python 使用 lambda 来创建匿名函数。

所谓匿名，意即不再使用 def 语句这样标准的形式定义一个函数。

- lambda 只是一个表达式，函数体比 def 简单很多。
- lambda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。
- 虽然lambda函数看起来只能写一行，却不等同于C或C++的内联函数，后者的目的是调用小函数时不占用栈内存从而增加运行效率。



匿名函数（lambda函数）

匿名函数（也称**lambda函数**），指没有名字的函数。

```
lambda x , y : x + y
```

```
def func( x , y ):
    return x + y
```

lambda 参数1，参数2: 表达式

只能写一个表达式（不能直接return），
表达式的结果就是发返回值。
只能用于一些简单的操作处理。

搜索 | 微信 ShowMeAI 研究中心

http://www.showmeai.tech/

语法

lambda 函数的语法只包含一个语句，如下：

```
1. lambda [arg1 [,arg2,.....argn]):expression
```

如下实例：

```
1. my_sum = lambda arg1, arg2: arg1 + arg2
2.
3. # 调用my_sum函数
4. print ("相加后的值为 :", my_sum( 10, 20 ))
5. print ("相加后的值为 :", my_sum( 20, 20 ))
```

以上实例输出结果：

```
1. 相加后的值为 ： 30
2. 相加后的值为 ： 40
```

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。不带参数值的return语句返回None。之前的例子都没有示范如何返回数值，以下实例演示了 return 语句的用法：

```
1. def my_sum( arg1, arg2 ):
```

https://www.showmeai.tech/article-detail/81

5/7

```
2.  # 返回2个参数的和."
3.  total = arg1 + arg2
4.  print("函数内 :", total)
5.  return total
6.
7. # 调用sum函数
8. total = my_sum( 10, 20 )
9. print("函数外 :", total)
```

以上实例输出结果：

```
1. 函数内 : 30
2. 函数外 : 30
```

强制位置参数

Python3.8+ 新增了一个函数形参语法 / 用来指明函数形参必须使用指定位置参数，不能使用关键字参数的形式。

在以下的例子中，形参 a 和 b 必须使用指定位置参数，c 或 d 可以是位置形参或关键字形参，而 e 和 f 要求为关键字形参：

```
1. def f(a, b, /, c, d, *, e, f):
2.     print(a, b, c, d, e, f)
```

以下使用方法是正确的：

```
1. f(10, 20, 30, d=40, e=50, f=60)
```

以下使用方法会发生错误：

```
1. f(10, b=20, c=30, d=40, e=50, f=60)  # b 不能使用关键字参数的形式
2. f(10, 20, 30, 40, 50, f=60)         # e 必须使用关键字参数的形式
```

视频教程

也可以点击 [这里](#) 到B站查看有【中英字幕】的版本

【双语字幕+资料下载】Python 3全系列...

去bilibili观看

分享

扫一扫 手机看

00:00 / 10:10

360P

进入bilibili,一起发弹幕吐槽!

去吐槽

【双语字幕+资料下载】Python 3全系列...

去bilibili观看

分享

扫一扫 手机看

00:00 / 07:38

360P

进入bilibili,一起发弹幕吐槽!

去吐槽

一键运行所有代码

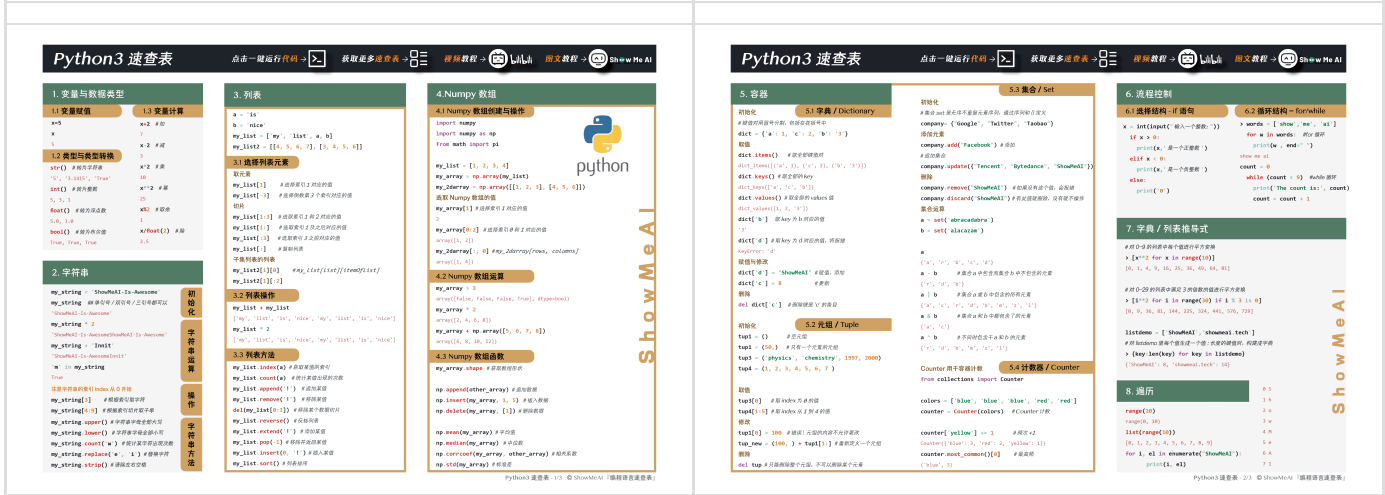
图解Python编程系列 配套的所有代码，可前往ShowMeAI 官方 [GitHub](#)，下载后即可在本地 Python 环境中运行。能访问 Google 的宝宝也可以直接借助 Google Colab一键运行与交互学习！

下载Python要点速查表

Awesome cheatsheets | [ShowMeAI速查表大全](#) 系列包含『编程语言』『AI技能知识』『数据科学工具库』『AI垂直领域工具库』四个板块，追平到工具库当前最新版本，并跑通了所有代码。点击 [官网](#) 或 [GitHub](#) 获取~

ShowMeAI速查表大全

Python 速查表（部分）



拓展参考资料

- Python教程 - Python3文档
- Python教程 - 廖雪峰的官方网站

ShowMeAI图解Python编程系列推荐（要点速查版）

- ShowMeAI 图解 Python 编程(1) | 介绍
- ShowMeAI 图解 Python 编程(2) | 安装与环境配置
- ShowMeAI 图解 Python 编程(3) | 基础语法
- ShowMeAI 图解 Python 编程(4) | 基础数据类型
- ShowMeAI 图解 Python 编程(5) | 运算符
- ShowMeAI 图解 Python 编程(6) | 条件控制与if语句
- ShowMeAI 图解 Python 编程(7) | 循环语句
- ShowMeAI 图解 Python 编程(8) | while循环
- ShowMeAI 图解 Python 编程(9) | for循环
- ShowMeAI 图解 Python 编程(10) | break语句
- ShowMeAI 图解 Python 编程(11) | continue语句
- ShowMeAI 图解 Python 编程(12) | pass语句
- ShowMeAI 图解 Python 编程(13) | 字符串及操作
- ShowMeAI 图解 Python 编程(14) | 列表
- ShowMeAI 图解 Python 编程(15) | 元组
- ShowMeAI 图解 Python 编程(16) | 字典
- ShowMeAI 图解 Python 编程(17) | 集合
- ShowMeAI 图解 Python 编程(18) | 函数
- ShowMeAI 图解 Python 编程(19) | 迭代器与生成器
- ShowMeAI 图解 Python 编程(20) | 数据结构
- ShowMeAI 图解 Python 编程(21) | 模块
- ShowMeAI 图解 Python 编程(22) | 文件读写
- ShowMeAI 图解 Python 编程(23) | 文件与目录操作
- ShowMeAI 图解 Python 编程(24) | 错误与异常处理
- ShowMeAI 图解 Python 编程(25) | 面向对象编程
- ShowMeAI 图解 Python 编程(26) | 命名空间与作用域
- ShowMeAI 图解 Python 编程(27) | 时间和日期

ShowMeAI系列教程精选推荐

- 大厂技术实现：推荐与广告计算解决方案
- 大厂技术实现：计算机视觉解决方案
- 大厂技术实现：自然语言处理行业解决方案
- 图解Python编程：从入门到精通系列教程
- 图解数据分析：从入门到精通系列教程
- 图解AI数学基础：从入门到精通系列教程
- 图解大数据技术：从入门到精通系列教程
- 图解机器学习算法：从入门到精通系列教程
- 机器学习实战：手把手教你玩转机器学习系列
- 深度学习教程：吴恩达专项课程·全套笔记解读
- 自然语言处理教程：斯坦福CS224n课程·课程带学与全套笔记解读
- 深度学习与计算机视觉教程：斯坦福CS231n·全套笔记解读

图解 Python 编程(18) | 函数

◀ 上一篇

图解 Python 编程(17) | 集合

下一篇 ▶

图解 Python 编程(19) | 迭代器与生成器