

目录

第 1 章 入门准备

01 开篇词：你为什么要学 Python ？

02 我会怎样带你学 Python ？

03 让 Python 在你的电脑上安家落户

04 如何运行 Python 代码 ？

第 2 章 通用语言特性

05 数据的名字和种类—变量和类型

06 一串数据怎么存—列表和字符串

07 不只有一条路—分支和循环

08 将代码放进盒子—函数

09 知错能改—错误处理、异常机制

10 定制一个模子—类

11 更大的代码盒子—模块和包

12 练习—密码生成器

第 3 章 Python 进阶语言特性

13 这么多的数据结构（一）：列表、元祖、字符串

14 这么多的数据结构（二）：字典、集合

15 Python大法初体验：内置函数

16 深入理解下迭代器和生成器

17 生成器表达式和列表生成式

18 把盒子升级为豪宅：函数进阶 [最近阅读](#)

19 让你的模子更好用：类进阶

20 从小独栋升级为别墅区：函数式编程

18 把盒子升级为豪宅：函数进阶

更新时间：2019-09-26 09:49:00



“ 不经一翻彻骨寒，怎得梅花扑鼻香。 ”
——宋帆

位置参数

位置参数这个东西我们并不陌生，之前所编写的函数使用的就是位置参数。位置参数，顾名思义，传入函数时每个参数都是通过位置来作区分的。函数调用时，传入的值需按照位置与参数一一对应。

比如：

```
def overspeed_rate(current, max, min):
    if current > max:
        return (current - max) / max # 超过最大时速，结果为正
    elif current < min:
        return (current - min) / min # 超过最小时速，结果为负
    else:
        return 0 # 不超速，结果为 0
```

这个函数用来判断车辆在高速上行驶时超速的比例。它接受三个参数，current 表示当前时速，max 参数表示当前路段的允许的最大时速，min 表示所允许的最小时速。

位置参数需要按位置顺序来传递，否则结果不可预期。

```
>>> overspeed_rate(150, 120, 90)
0.25 # 超过最大时速 25%
>>> overspeed_rate(80, 100, 60)
0 # 不超速
```

<div><div>← 慕课专栏</div><div>☰ 你的第一本Python基础入门书 / 18 把盒子升级为豪宅：函数进阶</div></div>	
目录	
第 1 章 入门准备	
01 开篇词：你为什么要学 Python ？	
02 我会怎样带你学 Python ？	
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码 ？	
第 2 章 通用语言特性	
05 数据的名字和种类—变量和类型	
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	
08 将代码放进盒子—函数	
09 知错能改—错误处理、异常机制	
10 定制一个模子—类	
11 更大的代码盒子—模块和包	
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶 最近阅读	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	

参数默认值

前面的函数中，如果最大时速和最小时速比较固定，那么每次函数调用时都输入这个两个参数就显得有些繁琐，这时我们可以使用参数默认值。

参数默认值也就是给参数设置默认值，之后函数调用时便可以传入这个参数，Python 自动以默认值来填充参数。如果一个有默认值的参数依然被传入了值，那么默认值将会被覆盖。

函数定义时，以 `参数=值` 来指定参数默认值。如下：

```
def 函数(参数1, 参数2=默认值):  
    pass
```

例如上面的 `overspeed_rate` 函数，`max` 和 `min` 通常比较固定，我们可以使用一个常用值来作为默认值。

```
def overspeed_rate(current, max=120, min=90):  
    if current > max:  
        return (current - max) / max  
    elif current < min:  
        return (current - min) / min  
    else:  
        return 0
```

```
>>> overspeed_rate(192)  
0.6  
>>> overspeed_rate(45)  
-0.5
```

关键字参数

对于 `overspeed_rate` 函数，我们还可以在函数调用时，以 `参数名=值` 的形式来向指定的参数传入值。

如：

```
overspeed_rate(100, min=80)
```

或者

```
overspeed_rate(current=100, min=80)
```

或者

```
overspeed_rate(current=100, max=100, min=80)
```

<div>← 慕课专栏</div>	<div>☰ 你的第一本Python基础入门书 / 18 把盒子升级为豪宅：函数进阶</div>
<div>目录</div>	<div>使用关键字时甚至可以打乱参数传递次序：</div>
<div>第 1 章 入门准备</div>	<div><pre>overspeed_rate(min=80, max=100, current=100)</pre></div>
<div>01 开篇词：你为什么要学 Python ？</div>	<div></div>
<div>02 我会怎样带你学 Python ？</div>	<div><pre>>>> overspeed_rate(min=80, max=100, current=100) 0</pre></div>
<div>03 让 Python 在你的电脑上安家落户</div>	<div></div>
<div>04 如何运行 Python 代码？</div>	<div>但要注意，关键字参数需要出现在位置参数之后，否则将抛出 <code>SyntaxError</code> 异常：</div>
<div>第 2 章 通用语言特性</div>	<div></div>
<div>05 数据的名字和种类—变量和类型</div>	<div><pre>>>> overspeed_rate(100, max=100, 80) File " ", line 1 SyntaxError: positional argument follows keyword argument</pre></div>
<div>06 一串数据怎么存—列表和字符串</div>	<div></div>
<div>07 不只有一条路—分支和循环</div>	<div>关键字参数的用法还不止如此。</div>
<div>08 将代码放进盒子—函数</div>	<div>当我们在定义函数时，如果参数列表中某个参数使用 <code>**参数名</code> 形式，那么这个参数可以接受一切关键字参数。如下：</div>
<div>09 知错能改—错误处理、异常机制</div>	<div><pre>def echo(string, **keywords): print(string) for kw in keywords: print(kw, ":", keywords[kw])</pre></div>
<div>10 定制一个模子—类</div>	<div></div>
<div>11 更大的代码盒子—模块和包</div>	<div><pre>>>> echo('hello' , today= '2019-09-04' , content= 'function' , section=3.6) hello today : 2019-09-04 content : function section : 3.6</pre></div>
<div>12 练习—密码生成器</div>	<div></div>
<div>第 3 章 Python 进阶语言特性</div>	<div></div>
<div>13 这么多的数据结构（一）：列表、元祖、字符串</div>	<div></div>
<div>14 这么多的数据结构（二）：字典、集合</div>	<div>显然，我们并没有在函数定义时定义 <code>today</code> 、 <code>content</code> 、 <code>section</code> 参数，但是我们却能接收到它们，这正是 <code>**keywords</code> 发挥了作用。函数会将所有接收到的关键字参数组装成一个字典，并绑定到 <code>keywords</code> 上。验证一下：</div>
<div>15 Python大法初体验：内置函数</div>	<div><pre>>>> def foo(**keywords): ... print(keywords) ... >>> foo(a=1, b=2, c=3) { 'a' : 1, 'b' : 2, 'c' : 3}</pre></div>
<div>16 深入理解下迭代器和生成器</div>	<div></div>
<div>17 生成器表达式和列表生成式</div>	<div></div>
<div>18 把盒子升级为豪宅：函数进阶 最近阅读</div>	<div></div>
<div>19 让你的模子更好用：类进阶</div>	<div></div>
<div>20 从小独栋升级为别墅区：函数式编程</div>	<div></div>

目录	如，计算任意个数的乘积：
第 1 章 入门准备	
01 开篇词：你为什么要学 Python ？	
02 我会怎样带你学 Python ？	
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码 ？	
第 2 章 通用语言特性	
05 数据的名字和种类—变量和类型	
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	
08 将代码放进盒子—函数	
09 知错能改—错误处理、异常机制	
10 定制一个模子—类	
11 更大的代码盒子—模块和包	
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶 最近阅读	
19 让你的模子更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	

```
def multiply(*nums):
    result = 1
    for n in nums:
        result *= n
    return result
```

```
>>> multiply(1,3,5,7)
105
```

这个函数能接收任意个参数，这正是 `*nums` 所发挥的作用。函数所有接收到的非关键字参数组装成一个元组，并绑定到 `nums` 上。来试验一下：

```
>>> def multiply(*nums):
...     print(nums)
...
>>> multiply(1, 2, 3, 4, 5)
(1, 2, 3, 4, 5)
```

多返回值

典型情况下，函数只有一个返回值，但是 Python 也支持函数返回多个返回值。

要返回多个返回值，只需在 `return` 关键字后跟多个值（依次用逗号分隔）。

例如：

```
def date():
    import datetime
    d = datetime.date.today()
    return d.year, d.month, d.day
```

`date()` 返回了今天的日期的年、月、日。

接收函数返回值时，用对应返回值数量的变量来分别接收它们。

```
>>> year, month, day = date()
>>> year
2019
>>> month
9
>>> day
4
```

<div><div>← 慕课专栏</div><div>≡ 你的第一本Python基础入门书 / 18 把盒子升级为豪宅：函数进阶</div></div>	
目录	后将元组返回。来验证下：
第 1 章 入门准备	<pre>>>> date() (2019, 9, 4)</pre>
01 开篇词：你为什么要学 Python ？	
02 我会怎样带你学 Python ？	接收返回值时， <code>year, month, day = date()</code> ，这样赋值写法，会将元组解包，分别将元素赋予单独的变量中。即：
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码 ？	<pre>>>> year, month, day = (2019, 9, 4) >>> year 2019 >>> month 9 >>> day 4</pre>
第 2 章 通用语言特性	
05 数据的名字和种类—变量和类型	<div>← 17 生成器表达式和列表生成式</div> <div>19 让你的模式更好用：类进阶 →</div>
06 一串数据怎么存—列表和字符串	
07 不只有一条路—分支和循环	精选留言 0
08 将代码放进盒子—函数	
09 知错能改—错误处理、异常机制	欢迎在这里发表留言，作者筛选后可公开显示
10 定制一个模式—类	
11 更大的代码盒子—模块和包	<div>!</div> <div>目前暂无任何讨论</div>
12 练习—密码生成器	
第 3 章 Python 进阶语言特性	千学不如一看，千看不如一练
13 这么多的数据结构（一）：列表、元祖、字符串	
14 这么多的数据结构（二）：字典、集合	
15 Python大法初体验：内置函数	
16 深入理解下迭代器和生成器	
17 生成器表达式和列表生成式	
18 把盒子升级为豪宅：函数进阶 最近阅读	
19 让你的模式更好用：类进阶	
20 从小独栋升级为别墅区：函数式编程	