

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 最近阅读

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

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

# 08 将代码放进盒子—函数

更新时间：2019-08-30 11:14:33



“

一个不注意小事情的人，永远不会成功大事业。

——戴尔·卡耐基

”

我们之前介绍过一些函数，如 `print()`、`int()`、`input()` 等。直接使用它们就可以获得一些功能，如向命令行输出内容、转换数字、获取命令行输入，那么它们到底是什么呢？

## 函数的初步理解

大家应该都非常熟悉数学上的函数，简单来说数学上的函数就是一个映射关系，给定一个  $x$ ，经映射后将得到  $y$  值，至于这其中的映射关系我们可以直接把它抽象为  $y=f(x)$ 。

程序中的函数与数学上的函数有一丝类似，我们也可以把它抽象地看作一个映射关系，给定输入参数后，经函数映射，返回输出结果。如之前我们使用过的 `int()` 和 `len()`：

数字 = `int`(字符串)

长度 = `len`(列表)

给定输入值，经函数处理，返回输出值，这是函数最单纯的模式。

## 函数如何定义

Python 中函数的定义方式如下：

```
def 函数名(参数1, 参数2, ...):  
    代码块
```

目录	函数内部的代码块就是函数的实现。所有的函数功能都实现于此。												
第 1 章 入门准备	函数的输出结果叫函数的返回值。函数可以没有返回值，也可以有一个或多个返回值。返回值通过 <code>return</code> 语句传递到函数外部，如：												
01 开篇词：你为什么要学 Python？	<pre>def add(x, y):     return x + y</pre>												
02 我会怎样带你学 Python？	函数定义示例												
03 让 Python 在你的电脑上安家落户	我们来编写一个函数试试，这个函数的需要的参数是年龄，返回值是年龄对应的人生阶段。												
04 如何运行 Python 代码？	<table><tr><th>年龄</th><th>人生阶段</th></tr><tr><td>0-6 岁</td><td>童年</td></tr><tr><td>7-17 岁</td><td>少年</td></tr><tr><td>18-40 岁</td><td>青年</td></tr><tr><td>41-65 岁</td><td>中年</td></tr><tr><td>65 岁之后</td><td>老年</td></tr></table>	年龄	人生阶段	0-6 岁	童年	7-17 岁	少年	18-40 岁	青年	41-65 岁	中年	65 岁之后	老年
年龄	人生阶段												
0-6 岁	童年												
7-17 岁	少年												
18-40 岁	青年												
41-65 岁	中年												
65 岁之后	老年												
第 2 章 通用语言特性	这个功能好像有点眼熟，没错，上一章节中我们完成过这个功能，现在把这个功能改写成函数。												
05 数据的名字和种类—变量和类型	可以这样来定义这个函数：												
06 一串数据怎么存—列表和字符串	<pre>def stage_of_life(age):     if age &lt;= 6:         return '童年'     elif 7 &lt;= age &lt;=17:         return '少年'     elif 18 &lt;= age &lt;= 40:         return '青年'     elif 41 &lt;= age &lt;= 65:         return '中年'     else:         return '老年'</pre>												
07 不只有一条路—分支和循环	我们给函数起名为 <code>stage_of_life</code> ，需要一个参数 <code>age</code> ，最终通过 <code>return</code> 语句返回对应的人生阶段，这个人生阶段就是函数的返回值。												
08 将代码放进盒子—函数 <span>最近阅读</span>	这里虽然有多个 <code>return</code> 语句，但是实际上每次函数使用时，只会有一个 <code>return</code> 语句被执行。												
09 知错能改—错误处理、异常机制	副作用												
10 定制一个模子—类	上面这个示例中，给定一个参数 <code>age</code> ，便返回对应的人生阶段。函数内部只是做了一个映射，并没有对程序和系统的状态作出影响，这样的函数是纯函数。												
11 更大的代码盒子—模块和包	纯函数是函数的一个特例，更普遍的情况是，函数包含一些会引起程序或系统状态变化的操作，如修改全局变量、命令行输入输出、读写文件等，这样的变化叫做函数的副作用。												
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													
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													

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	最近阅读
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	
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	

因为有了副作用，函数就不必完全遵从 **输入 -> 映射 -> 输出** 这种模式，函数可以在没有参数或返回值的情况下，拥有其功能。如果你看到一个函数没有参数或返回值，要自然的想到，那是副作用在发挥作用。

没有参数没有返回值：

```
def say_hello():  
    print('hello')
```

有参数没有返回值：

```
def say_words(words):  
    print(words)
```

没有参数有返回值：

```
def pi():  
    return 3.14159
```

函数的调用

函数定义完成后，就可以在后续的代码中使用它了，对函数的使用叫做函数调用。

以前我们调用过 `int()`、`len()`，它们是内置在 Python 语言中的函数，也就是**内置函数**。现在我们自己定义了 `stage_of_life`，调用的方式是一样的：

```
def stage_of_life(age):  
    if age <= 6:  
        return '童年'  
    elif 7 <= age <=17:  
        return '少年'  
    elif 18 <= age <= 40:  
        return '青年'  
    elif 41 <= age <= 65:  
        return '中年'  
    else:  
        return '老年'  
  
stage = stage_of_life(18)  
print(stage)
```

用参数的形式将数据传递给函数，用赋值语句来接收返回值。

需要说明的是

- 函数有多个参数时，参数是有顺序的，要按对应位置将参数传递进去。

```
>>> def minus(x, y):  
...     return x - y  
... 
```

<div><div>← 慕课专栏</div><div>☰ 你的第一本Python基础入门书 / 08 将代码放进盒子—函数</div></div>	
目录	<pre>&gt;&gt;&gt; minus(3, 1) 2</pre>
第 1 章 入门准备	
01 开篇词：你为什么要学 Python？	<ul style="list-style-type: none"><li>函数需要先经过定义，之后才能被调用，否则解释器将找不到这个函数名，也就无法调用它。</li></ul>
02 我会怎样带你学 Python？	未定义函数便直接调用，解释器将报出「名字未定义」的错误：
03 让 Python 在你的电脑上安家落户	
04 如何运行 Python 代码？	<pre>&gt;&gt;&gt; stage = abc(18) Traceback (most recent call last): File “”, line 1, in NameError: name ‘abc’ is not defined</pre>
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 <a href="#">最近阅读</a>	
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	
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	<pre>number = -33 if number &gt; 0:     print(number) else:     print(-number)  number = 456 if number &gt; 0:     print(number) else:     print(-number)  number = -0.03 if number &gt; 0:</pre>

← 慕课专栏	:三 你的第一本Python基础入门书 / 08 将代码放进盒子—函数
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 <span>最近阅读</span>	显然有大量重复的代码，这些重复代码是可以避免的。用函数修改后如下：
09 知错能改—错误处理、异常机制	<pre>def print_absolute(number):     if number &gt; 0:         print(number)     else:         print(-number)  print_absolute(-33) print_absolute(456) print_absolute(-0.03)</pre>
10 定制一个模子—类	代码量减少很多，也能应对未来的相同需求，这就是复用的好处。
11 更大的代码盒子—模块和包	什么时候用函数
12 练习—密码生成器	看了上面函数的好处之后，想必你已经知道在什么时候用函数了吧？
第 3 章 Python 进阶语言特性	<ul style="list-style-type: none"><li>• 需要通过函数的形式把这些复杂性隔离出来，之后在使用时只需调用这个函数即可，清晰且优雅。</li><li>• 需要复用。一段相似甚至完全一致的代码在多处被使用，可以考虑将这段代码定义为函数，然后在各处去调用它。</li></ul>
13 这么多的数据结构（一）：列表、元祖、字符串	总结
14 这么多的数据结构（二）：字典、集合	函数的主要作用是抽象和代码复用。
15 Python大法初体验：内置函数	Python 中函数的定义方法：
16 深入理解下迭代器和生成器	<pre>def 函数名(参数1, 参数2, ...):     代码块</pre>
17 生成器表达式和列表生成式	返回值通过 <code>return</code> 语句传递到函数外部。
18 把盒子升级为豪宅：函数进阶	
19 让你的模子更好用：类进阶	多语言比较
20 从小独栋升级为别墅区：函数式编程	Java 中所有的函数都需要定义在类中，类中的函数也叫做方法。
	Java 中定义函数：
	<pre>int add(int x, int y) {     return x + y }</pre>
	C/C++ 中定义函数：
	<pre>int add(int x, int y) {     return x + y</pre>

← 慕课专栏

☰ 你的第一本Python基础入门书 / 08 将代码放进盒子—函数

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 最近阅读

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

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

Go 中定义函数：

```
func max(x, y int) int {  
    return x + y  
}
```

← 07 不只有一条路—分支和循环

09 知错能改—错误处理、异常机制 →

精选留言 1

欢迎在这里发表留言，作者筛选后可公开显示

sunzhenyang

老师感觉这种写法有些赘余，既然elif了 age<=6，那后面 7 <= age 是不是可以省略 def st age\_of\_life(age): if age <= 6: return '童年' elif 7 <= age <=17: ...

👍 0

回复

2019-12-10

千学不如一看，千看不如一练

www.imooc.com/read/46/article/816

6/6