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

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

20 从小独栋升级为别墅区：函数式编程

更新时间：2019-10-09 09:35:32

> 天才免不了有障碍，因为障碍会创造天才。
>
> ——罗曼·罗兰

函数赋值给变量

在 Python 中，所有的对象都可以赋值给变量，**包括函数**。这可能有点出乎意料，我们不妨来试一试：

```
def say_hello(name):
    return name + ', hello!'

f = say_hello
```

```
>>> f(‘开发者’)
‘开发者, hello!’

>>> f
<function say_hello at 0x10befec80>
```

注意，这里被赋值的是函数本身，而不是函数的结果。赋值后，变量 `f` 与函数 `say_hello` 绑定，`f` 也就相当于是 `say_hello` 的别名，完全可以用调用 `say_hello` 的方式来调用 `f`。

> 扩展：类也可以赋值给变量。如：

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

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

```
banana = Apple
```

```
>>> banana.who_am_i
’ apple ’
```

注意，被赋值的是类本身，而不是类实例化后的对象。赋值后，变量 banana 与类 Apple 绑定， banana 也就相当于是 Apple 的别名，使用 banana 就相当于使用 Apple 。

函数作为函数参数

一切对象都可以作为函数的参数，包括另一个函数。接受函数作为参数的函数，称为高阶函数。这和数学中的高阶函数有些相似。

来看一个函数作为参数的例子。

这个例子中，我们实现了一个函数，它从给定的数字列表中筛选数字，而具体的筛选策略由另一个函数决定并以参数的形式存在：

```
def filter_nums(nums, want_it):
    return [n for n in nums if want_it(n)]
```

函数 filter_nums 用来筛选数字，它接受两个参数， nums 是包含所有待筛选数字的列表， want_it 是一个函数，用来决定某个数字是否保留。

我们选定一个简单的策略来实现下 want_it 参数所对应的函数（其函数名不必为 want_it ）：

```
def want_it(num):
    return num % 2 == 0
```

这里 want_it 接受一个数字作为参数，如果这个数字是 2 的倍数，则返回 True ，否则返回 False 。

调用一下 filter_nums 试试：

```
>>> def filter_nums(nums, want_it):
…     return [n for n in nums if want_it(n)]
...
>>> def want_it(num):
…     return num % 2 == 0
...
>>> filter_nums([11, 12, 13, 14, 15, 16, 17, 18], want_it)
[12, 14, 16, 18]
```

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

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

lambda 表达式

在 Python 中，可以通过 lambda 表达式来便捷地定义一个功能简单的函数，这个函数只有实现没有名字，所以叫作匿名函数。

lambda 表达式的写法如下：

```
lambda 参数1, 参数2, 参数N: 函数实现
```

使用上述表达式将定义一个匿名函数，这个匿名函数可接受若干参数，参数写在冒号前（：），多个参数时用逗号分隔，其实现写在冒号后（：）。

举个例子：

```
f = lambda x: x ** 2
```

这个 lambda 表达式定义了一个匿名函数，这个匿名函数接受一个参数 `x`，返回 `x ** 2` 的计算结果。同时赋值语句将这个匿名函数赋值给了变量 `f`。**注意 `f` 保存的是函数，而不是函数结果。**

```
>>> f
<function at 0x10bcba0d0>

>>> f(4)
16
>>> f(9)
81
```

通过观察上述示例可以发现，lambda 表达式中并没有 `return` 关键字，但结果被返回出来。是的，匿名函数的 `函数实现` 的执行结果就会作为它的返回值，无需使用 `return` 关键字。

从功能上来看，`lambda x: x ** 2` 等同于：

```
def no_name(x):
    return x ** 2
```

```
>>> no_name(4)
16
```

一般情况下，我们不会像 `f = lambda x: x ** 2` 这样直接将匿名函数赋值给变量，然后去用这个变量。而是在需要将函数作为参数时，才去使用 lambda 表达式，这样就无需在函数调用前去定义另外一个函数了。

如我们刚才写的函数 `filter_nums`：

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

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

它的 want_it 参数需要是一个函数，这时用 lambda 表达式便能方便的解决问题。可以像这样来使用：

```
>>> filter_nums([11, 12, 13, 14, 15, 16, 17, 18], lambda x: x % 2 == 0)
[12, 14, 16, 18]
```

以前讲内置函数的时候，我们介绍过排序函数 sorted()。它有一个参数 key，用来在排序复杂元素时，指定排序所使用的字段，这个参数需要是个函数，同样可以用 lambda 表达式来解决：

```
>>> codes = [('上海', '021'), ('北京', '010'), ('成都', '028'), ('广州', '020')]
>>> sorted(codes, key=lambda x: x[1]) # 以区号字典来排序
[('北京', '010'), ('广州', '020'), ('上海', '021'), ('成都', '028')]
```

map() 和 filter()

Python 内置有两个非常好用的高阶函数 map() 和 filter()。

filter() 用于从可迭代对象中筛选元素。用法如下：

```
filter(筛选函数, 可迭代对象)
```

filter() 依次对 可迭代对象 中的每个元素调用 筛选函数，如果返回值为 True，则当前这个元素会被保留，否则被排除。最终返回一个包含所有被保留元素的迭代器。

显然这里的 筛选函数 可以用 lambda 表达式来创建。

例如，从 ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr'] 筛选出长度为奇数的字符串。可以这样写：

```
filter(lambda x: len(x) % 2 == 1, ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr'])
```

```
>>> list(filter(lambda x: len(x) % 2 == 1, ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr']))
['a', 'b', 'efg', 'hig']
```

这里我们用 list() 将迭代器转换为列表。

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

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

```
map(处理函数, 可迭代对象)
```

map() 依次对 可迭代对象 中的每个元素调用 处理函数 ，最终返回一个包含所有被处理过后的元素的迭代器。

显然这里的 处理函数 也可以用 lambda 表达式来创建。

例如，将 ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr'] 全部转换为大写。可以这样来写：

```
map(lambda x: x.upper(), ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr'])
```

```
>>> list(map(lambda x: x.upper(), ['a', 'b', 'cd', 'efg', 'hig', 'klmn', 'opqr']))
['A', 'B', 'CD', 'EFG', 'HIG', 'KLMN', 'OPQR']
```

这里我们用 list() 将迭代器转换为列表。

← 19 让你的模子更好用：类进阶

21 给凡人添加超能力：入手装饰器 →

精选留言 0

欢迎在这里发表留言，作者筛选后可公开显示

目前暂无任何讨论