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 最近阅读

17 生成器表达式和列表生成式

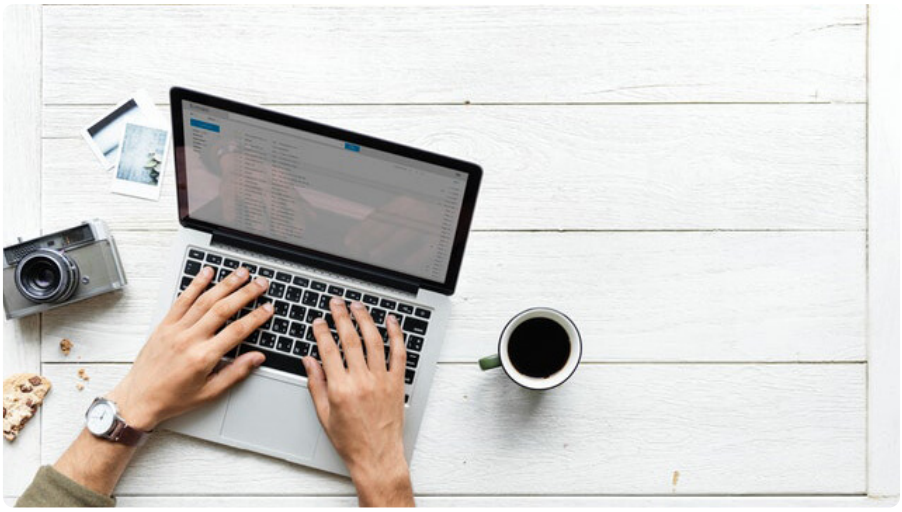
18 把盒子升级为豪宅：函数进阶

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

16 深入理解下迭代器和生成器

更新时间：2019-09-20 10:36:46



> 古之立大事者，不唯有超世之才，亦必有坚韧不拔之志。
>
> ——苏轼

迭代（Iteration）

本章节的主题和迭代密切相关，那什么是迭代呢？在编程中，**迭代**指的是通过重复执行某个操作，不断获取被迭代对象中的数据。这样的每一次操作就是就是一次迭代。

简而言之，迭代是遍历的一种形式。例如我们之前所学习的 for 循环，它能不断从地从列表、元组、字符串、集合、字典等容器中取出新元素，每次一个元素直至所有元素被取完。这种 for 循环操作就是迭代。

```
>>> for item in [1, 2, 3, 4, 5]:
... print(item)
...
1
2
3
4
5
```

迭代器（Iterator）

迭代器是具有迭代功能的对象。我们使用迭代器来进行迭代操作。

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 最近阅读

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

iter() 的使用方法：

```
迭代器 = iter(容器)
```

```
>>> numbers = [1, 2, 3, 4, 5]
>>> iterator = iter(numbers)
>>> iterator
<list_iterator object at 0x1074f34a8>
```

上面的「list_iterator」便是列表的迭代器。这个迭代器可用于迭代列表中的所有元素。

要使用迭代器，只需对迭代器调用内置函数 next()，便可逐一获取其中所有的值。

next() 的使用方法：

```
值 = next(迭代器)
```

对于上面的列表迭代器，可以像这样使用它：

```
>>> next(iterator)
1
>>> next(iterator)
2
>>> next(iterator)
3
>>> next(iterator)
4
>>> next(iterator)
5
>>> next(iterator)
Traceback (most recent call last):
    File “ ”, line 1, in
StopIteration
```

可以看到，每次调用 next() 将依次返回列表中的一个值。直至所有的值被遍历一遍，此时将抛出 StopIteration 异常以表示迭代终止。

for 循环的迭代过程

for 循环的迭代就是通过使用迭代器来完成的。它在背后所做的事情是：

1. 对一个容器调用 iter() 函数，获取到该容器的迭代器
2. 每次循环时对迭代器调用 next() 函数，以获取一个值
3. 若捕获到 StopIteration 异常则结束循环

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 最近阅读

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

并不是所有的对象都可以被 iter() 函数使用。比如整数：

```
>>> iter(123)
Traceback (most recent call last):
    File “ ”, line 1, in
TypeError: ‘int’ object is not iterable
```

这里抛出 TypeError 异常，提示 int 对象不是可迭代的。

什么是**可迭代(的)**？

- 从表面来看，所有可用于 for 循环的对象是可迭代的，如列表、元组、字符串、集合、字典等容器
- 从深层来看，**定义了 __iter__() 方法的类对象就是可迭代的。当这个类对象被 iter() 函数使用时，将返回一个迭代器对象**。如果对象具有 __iter__() 方法，则可以说它支持迭代协议。

判断一个已有的对象是否是可迭代的，有两个方法：

1. 通过内置函数 dir() 获取这个对象所有方法，检查是否有 '__iter__'

```
>>> ‘__iter__’ in dir(list)
True
>>> ‘__iter__’ in dir(int)
False
```

2. 使用内置函数 isinstance() 判断其是否为 Iterable 的对象

```
from collections.abc import Iterable
isinstance(对象, Iterable)
```

```
>>> from collections.abc import Iterable
>>> isinstance([1, 2, 3], Iterable)
True
```

自定义迭代器

我们可以自己来定义迭代器类，只要在类中定义 __next__() 和 __iter__() 方法即可。如：

```
class MyIterator:
    def __next__(self):
        代码块

    def __iter__(self):
        return self
```

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 最近阅读

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

```
class PowerOfTwo:
    def __init__(self):
        self.exponent = 0                           # 将每次的指数记录下来

    def __next__(self):
        if self.exponent > 10:
            raise StopIteration
        else:
            result = 2 ** self.exponent          # 以 2 为底数求指数幂
            self.exponent += 1
            return result

    def __iter__(self):
        return self
```

每次对迭代器使用内置函数 next() 时， next() 将在背后调用迭代器的 __next__() 方法。所以迭代器的重点便是 __next__() 方法的实现。在这个 __next__() 方法中，我们将求值时的指数记录在对象属性 self.exponent 中，求值结束时指数加 1，为下次求值做准备。

对于方法 __iter__() 的实现，我们直接返回迭代器对象自身即可。有了这个方法，迭代器对象便是可迭代的，可直接用于 for 循环。

> 扩展：如果对象具有 __iter__() 和 __next__() 方法，则可以说它支持**迭代器协议**。

迭代器 PowerOfTwo 使用示例：

```
>>> p = PowerOfTwo()
>>> next§
1
>>> next§
2
>>> next§
4
>>> next§
8
>>> next§
16
>>> next§
32
>>> next§
64
>>> next§
128
>>> next§
256
>>> next§
```

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 最近阅读

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

```
1024
>>> next§
Traceback (most recent call last):
    File “ ”, line 1, in
    File “ ”, line 6, in next
StopIteration
```

这个迭代器当然也可用于 for 循环：

```
>>> p = PowerOfTwo()
>>> for item in p:
··· print(item)
...
1
2
4
8
16
32
64
128
256
512
1024
```

迭代器的好处

- 一方面，迭代器可以提供迭代功能，当我们需要逐一获取数据集合中的数据时，使用迭代器可以达成这个目的
- 另一方面，数据的存储是需要占用内存的，数据量越大所占用的内存就越多。如果我们使用列表这样的结构来保存大批量的数据，并且数据使用频率不高的话，就十分浪费资源了。而迭代器可以不保存数据，它的数据可以在需要时被计算出来（这一特性也叫做**惰性计算**）。在合适的些场景下使用迭代器可以节省内存资源。

生成器（Generator）

刚才我们自定义了迭代器，其实创建迭代器还有另一种方式，就是使用**生成器**。

生成器是一个函数，这个函数的特殊之处在于它的 return 语句被 yield 语句替代。

如刚才用于生成 2 的指数幂的迭代器，可以通过生成器来实现：

```
def power_of_two():
    for exponent in range(11): # range(11) 表示左开右闭区间 [0, 11)，不包含 11
        yield 2 ** exponent          # 以 2 为底数求指数幂
```

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 最近阅读

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

```
p = power_of_two()                    # 以函数调用的方式创建生成器对象
next(p)                                # 同样使用 next() 来取值
```

生成器的关键在于 yield 语句。yield 语句的作用和 return 语句有几分相似，都可以将结果返回。不同在于，生成器函数执行至 yield 语句，返回结果的同时记录下函数内的状态，下次执行这个生成器函数，将从上次退出的位置（ yield 的下一句代码）继续执行。当生成器函数中的所有代码被执行完毕时，自动抛出 StopIteration 异常。

我们可以看到，生成器的用法和迭代器相似，都使用 next() 来进行迭代。这是因为生成器其实就是创建迭代器的便捷方法，生产器会在背后自动定义 __iter__() 和 __next__() 方法。

生成器表达式（Generator Expression）

可以用一种非常简便的方式来创建生成器，就是通过**生成器表达式**。生成器的写法非常简单，但是灵活性也有限，所能表达的内容相对简单。

生成器表达式的写法如下：

```
生成器 = (针对项的操作 for 项 in 可迭代对象)
```

如：

```
>>> letters = (item for item in ‘abc’)
>>> letters
<generator object at 0x1074a8228>
>>> next(letters)
‘a’
>>> next(letters)
‘b’
>>> next(letters)
‘c’
```

```
>>> letters = (i.upper() * 2 for i in ‘abc’)
>>> next(letters)
‘AA’
>>> next(letters)
‘BB’
>>> next(letters)
‘CC’
```

← 15 Python大法初体验：内置函数 17 生成器表达式和列表生成式 →

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 最近阅读

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

欢迎在这里发表留言，作者筛选后可公开显示

sunzhenyang

老师您好，请问这一段 next 后面跟的那个符号是什么意思，我复制到编辑器里也报错呢 p = PowerOfTwo() next§

0 回复 2019-12-16

千学不如一看，千看不如一练