第六章：生成器与迭代器

Table of Contents

[1. 序言 2](#_Toc50997854)

[1.1 章节简介 2](#_Toc50997855)

[2. 迭代 2](#_Toc50997856)

[3. 列表生成式 4](#_Toc50997857)

[4. 生成器 6](#_Toc50997858)

[5. 迭代器 10](#_Toc50997859)

[6. 小结 13](#_Toc50997860)

# 序言

## 章节简介

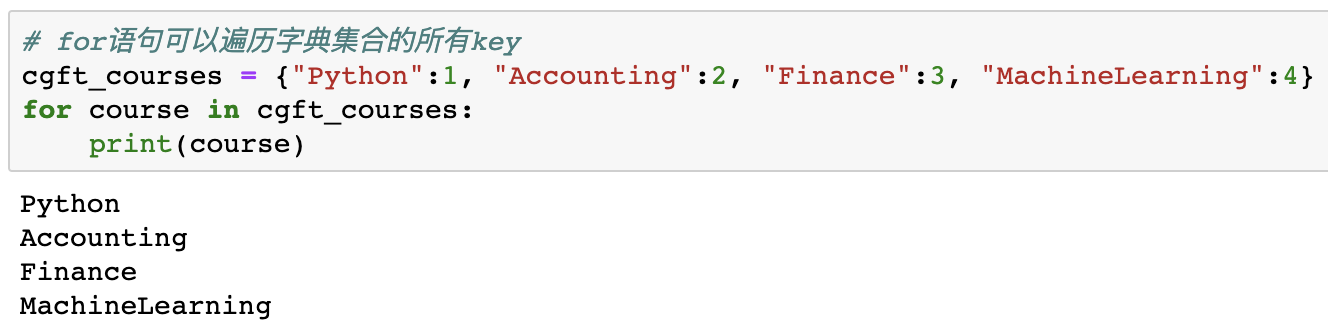
在上一章节里，我们学习了什么是循环语句以及经常使用的for循环和while循环，本章节继续学习跟循环相关的一些高级话题，让我们对如何使用for语句和遍历集合有更好的理解。具体内容为：

* 迭代
* 列表生成式
* 生成器
* 迭代器

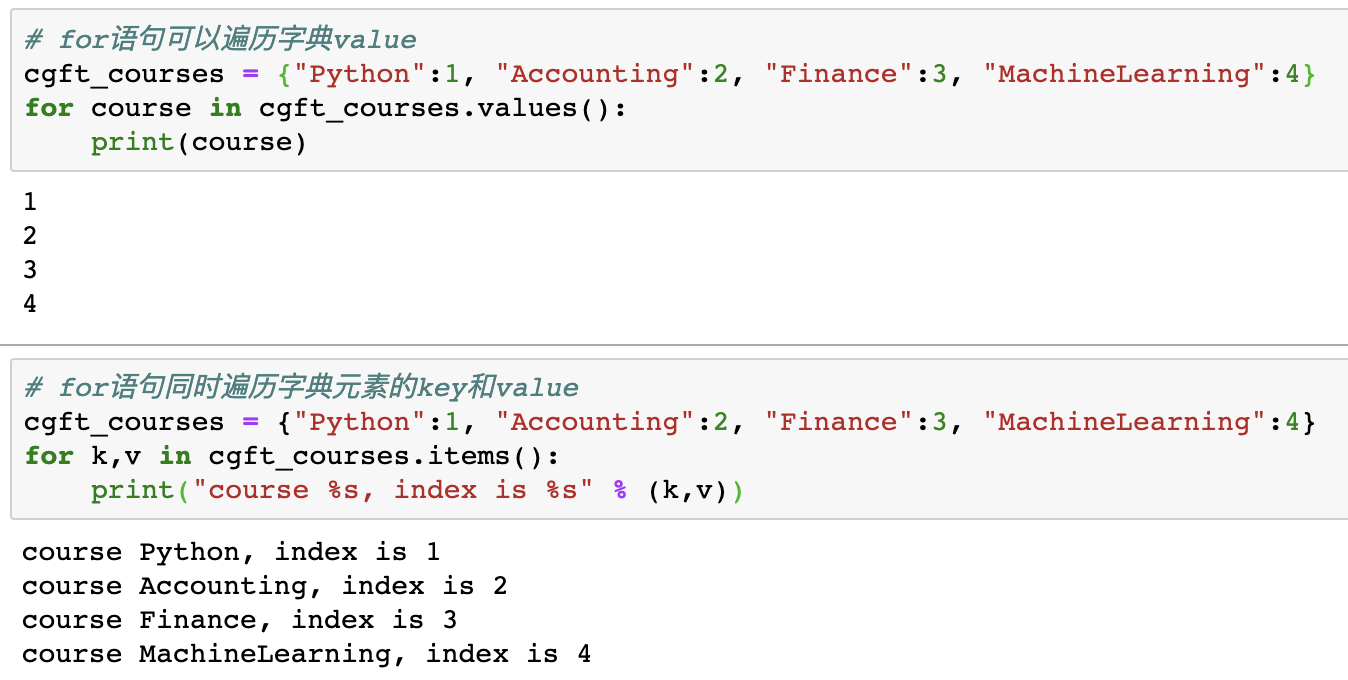
# 迭代

在上一小节我们看到，可以用for循环来访问list集合的每一个元素，我们可以通过for循环来遍历一个list或者dict，这种遍历集合里每一个元素的过程我们称、就叫做迭代(Iteration)。

Python的for循环不仅可以用在list或tuple上，还可以作用在其他可迭代对象上。list这种数据类型虽然有下标，但很多其他数据类型是没有下标的，但是，只要是可迭代对象，无论有无下标，都可以迭代，比如dict就可以迭代：



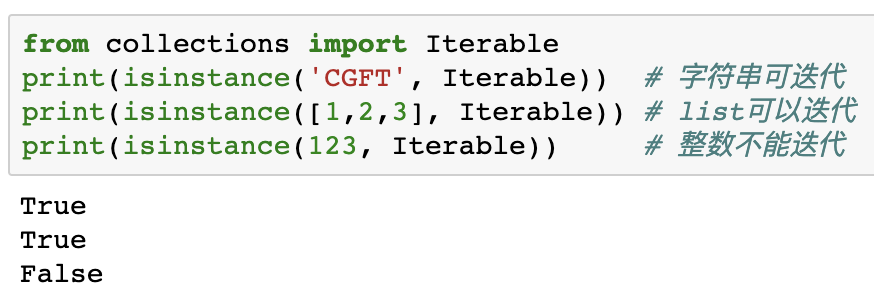
因为dict存储元素时不时顺序排列，所以，有时迭代出的结果顺序很可能不一样，但是所有的元素是一定都会访问到的。默认情况，我们使用for循环迭代dict时访问的是dict的key。如果想迭代value，可以用for value in d.values()，如果要同时迭代key和value，可以用for k, v in d.items()。



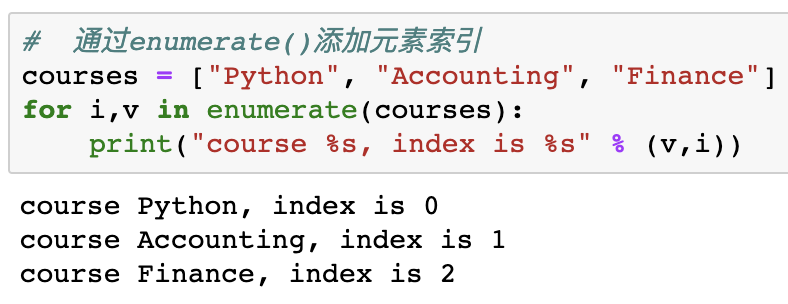
字符串本身也是可迭代对象，所以也可以用for语句访问字符串的每个字符：



所以，当我们使用for循环时，只要这个变量是一个可迭代的对象，for循环就可以正常运行，我们不需要关心该对象究竟是list还是其他数据类型。这里，同学们可能会问，什么样的对象是可迭代对象呢(关于类和对象，这是Python编程里的重点，我们会在第八章里详细的介绍)？方法是通过collections模块的Iterable类型(关于模块我们会在第十一章介绍)判断：

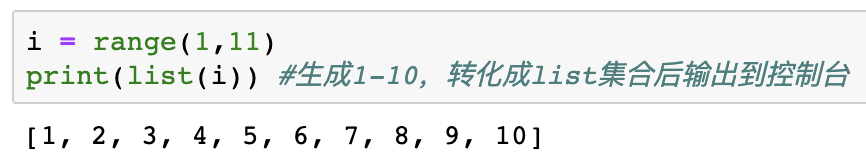


本节最后，我们知道for语句访问list集合是没有下标的，那我们如何可以知道每个元素它的下标(索引)呢，这里我们使用Python的内置函数enumerate()。enumerate在字典上是枚举、列举的意思。对于一个可迭代的（iterable）/可遍历的对象（如列表、字符串），enumerate将其组成一个索引序列，利用它可以同时获得索引和值。enumerate()多用于在for循环中得到计数。

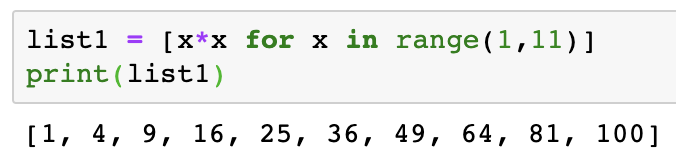


# 列表生成式

列表生成式即是Python内置的非常简单却强大的可以用来创建list的功能。顾名思义，列表生成式就是一个用来生成列表的特定语法形式的表达式。例如我们需要生成一个列表,由0到9组成，我们可以这样写list1 = [0,1,2,3,4,5,6,7,8,9]，没有问题，如果是100个数字呢？有没有一个更简洁的写法表达这样一个列表，这里range()就派上了用场。要生成list [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]可以用list(range(1, 11))：



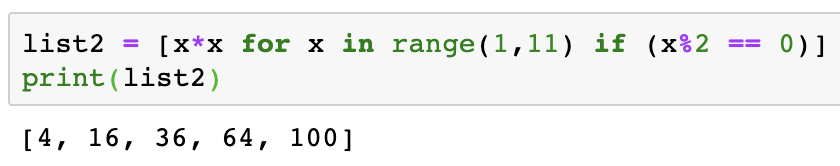
我们继续，如果要生成一个list集合里边包含1到10每个数字的平方，该如何简洁清晰的编写呢？如下代码所示，我们首先使用range()函数生成1到10的数字，那么每一个元素我们用x表示，我们每访问到一个元素x时就计算x\*x，然后将所有的结果生成一个list集合赋给变量list1。



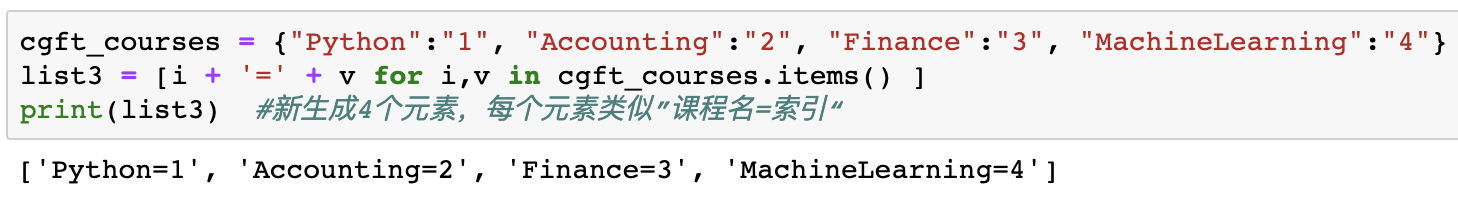
我们再来归纳一下使用列表生成式时的基础语法格式 [exp for iter\_var in iterable]。工作过程：

* + 迭代iterable中的每个元素；
  + 每次迭代都先把结果赋值给iter\_var，然后通过exp得到一个新的计算值；
  + 最后把所有通过exp得到的计算值以一个新列表的形式返回。

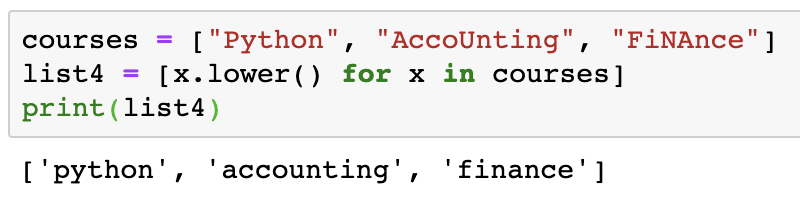
继续生成式的学习，for循环后面还可以加上if判断，这样我们就可以筛选出仅偶数的平方组成的list列表，在这个表达式里，我们遍历range生成的集合里的每一个元素，并且只返回符合if判断条件的元素，这样*for x in range(1,11) if (x%2 == 0)*返回的是[2,4,6,8,10]，然后继续x\*x处理[2,4,6,8,10]，这样就得到一个所有偶数平方的新的集合list2。



因为for循环可以使用两个或者多个变量，所以列表生成式也可以使用两个变量来生成我们想要的list，举例如下：



我们再看一个例子多熟悉一下这样的用法，下面的实例将每个字符串都转换成小写：



# 生成器

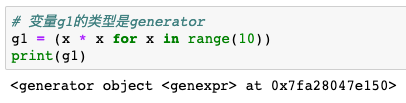
通过列表生成式，我们可以直接创建一个列表。但是如果列表里元素很多，就会占用很大的内存，而一个列表里的元素大多数时候我们并不会同时访问，再比如如果我们仅仅需要访问前面几个元素，那后面绝大多数元素占用的空间都白白浪费了。

所以，如果列表元素可以按照某种算法推算出来，那我们是否可以在循环的过程中不断推算出后续的元素呢？这样就不必创建完整的list，从而节省大量的空间。在Python中，这种一边循环一边计算的机制，称为生成器(generator)。生成器的优点有2个：

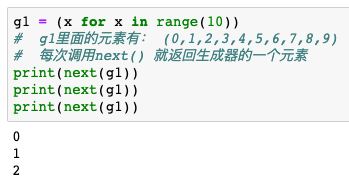
1. 延迟计算，一次返回一个结果。生成器不会一次生成所有的结果，而是一边循环一边计算，这对于大数据量处理，是个非常有用的优势。因为在编程的实际应用中，占用内存量是工程师必须考虑的一个问题。
2. 有效提高代码可读性。使用生成器以后，代码行数更少。

在python中生成器有两种表示形式：（1）生成器表达式（2）生成器函数。本章我们都会介绍。

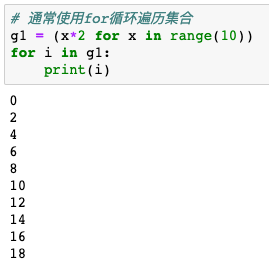
第一种方法很简单，只要把一个列表生成式的[]改成()，就创建了一个generator：



要想访问生成器里的元素，有两种方式，除了我们前面看到过的使用for循环语句，**可以通过next()函数获得generator的下一个返回值**，generator保存的是算法，每次调用next(g1)，就计算出g1的下一个元素的值，直到计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误。



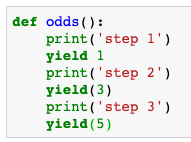
绝大多数时候使用next()获取生成器的下一个元素都很少使用，基本上都是通过for循环来遍历生成器，就像我们之前看到过的。



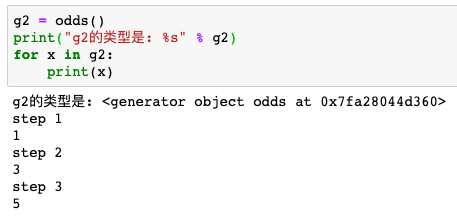
下面我们来介绍第二种表示生成器的方法，也就是生成器函数，生成器函数与python中的函数定义相同，都是使用def 关键字进行定义(关于函数的定义，我们会在第7章详细介绍)。二者的不同在于，生成器使用yield语句而不是return语句返回值。在遍历生成器的某个元素时，程序执行生成器函数的yield语句之后，会返回一次结果，然后将函数挂起，当再次调用查询生成器的下一个元素时，执行生成器函数时从上次返回的yield语句处继续执行，并返回结果后继续将函数挂起，等待程序下一次的调用(延时生成元素，用到时才会计算并返回元素)。

这就是定义generator的另一种方法。如果一个函数定义中包含**yield**关键字，那么这个函数就不再是一个普通函数，而是一个generator。这里，最难理解的就是generator和函数的执行流程不一样。函数是顺序执行，遇到return语句或者最后一行函数语句就返回。而变成generator的函数，在每次调用next()的时候执行，遇到yield语句返回，再次执行时从上次返回的yield语句处继续执行。

举个简单的实例，定义一个generator，依次返回数字1，3，5：



调用该generator时，首先要生成一个generator对象，然后用for循环遍历每一个元素：



同样，既然是一个生成器，我们也可以使用next()访问生成器的每一个元素，当计算到最后一个元素，没有更多的元素时，抛出StopIteration的错误。(关于错误和异常，我们会在第十章进行详细介绍。在执行过程中，遇到yield就中断，下次又继续执行。执行3次yield后，已经没有yield可以执行了，所以，第4次调用next(g3)就报错。

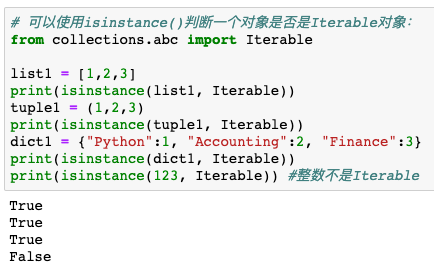


# 迭代器

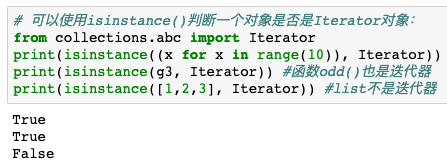
通过目前为止的学习我们做一下小结：都有以下类型可以用于for循环遍历元素：

1. 集合类型的变量，list、tuple、dict、set、str。
2. 通过生成器generator，生成generator的方式有两种(1)生成器表达式;(2)生成器函数。

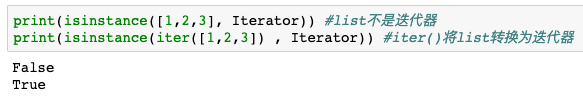
我们把所有这些可以直接用于for循环，被循环语句遍历元素的的对象抽象为一个统一的名称，叫做可迭代对象：Iterable。



我们再来回顾什么是生成器(generator)：遍历生成器的元素不但可以用for语句，还可以不断调用next()函数每次返回一个元素，直到最后抛出StopIteration错误表示生成器已经没有元素可以返回。我们把可以被next()函数调用并不断返回下一个值的对象称为迭代器：Iterator。



如果我们想把list,tuple,set,dict这些结合转换成迭代器使用，可以使用iter()函数：



那么为什么Python规定list、dict、str等数据类型不是Iterator？这是因为Python的Iterator对象表示的是一个数据流，Iterator对象可以被next()函数调用并不断返回下一个数据，直到没有数据时抛出StopIteration错误。可以把这个数据流看做是一个有序序列，但我们却不能提前知道序列的长度，只能不断通过next()函数实现按需计算下一个数据，所以Iterator的计算是惰性的，只有在需要返回下一个数据时它才会计算。Iterator甚至可以表示一个无限大的数据流，例如全体自然数。而使用list是永远不可能存储全体自然数的，list只能表示一个有限的集合。

# 小结

1. 可以被for循环便利的对象都是Iterable类型；
2. 可以作用于next()函数的对象来遍历集合的都是Iterator类型，它们表示一个惰性计算的序列；
3. 集合数据类型如list、dict、str等是Iterable但不是Iterator，不过可以通过iter()函数获得一个Iterator对象。
4. Python的for循环本质上就是通过不断调用next()函数实现的。

关于这些关系我们可以用2个更直观的简图表示：