# 列表解析

　　根据已有列表，高效创建新列表的方式。

　　列表解析是Python迭代机制的一种应用，它常用于实现创建新的列表，因此用在[]中。

语法：

　　[expression for iter\_val in iterable]

　　[expression for iter\_val in iterable if cond\_expr]

要求：列出1~10所有数字的平方

####################################################

1、普通方法：

>>> L = []

>>> for i in range(1,11):

... L.append(i\*\*2)

...

>>> print L

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

####################################################

2、列表解析

>>>L = [ i\*\*2 for i in range(1,11)]

>>>print L

[1, 4, 9, 16, 25, 36, 49, 64, 81, 100]

[这个列表每次循环并满足条件后想要添加的元素 for ... in ... [if ....] ]

num = [i for i in range(10) if i == 2 or i == 5 ] # [2, 5]

print(num)

num = [i\*\*i for i in range(10) if i == 2 or i == 5] #[4, 3125]

print(num)

Range(begin， end， 步长) 生成一个整数列表，一般用在循环中

print(range(10)) #[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

print(range(1,10,2)) #[1, 3, 5, 7, 9]

# 迭代器

把一个类作为一个迭代器使用需要在类中实现两个方法 \_\_iter\_\_() 与 \_\_next\_\_() 。

\_\_iter\_\_() 方法返回一个特殊的迭代器对象， 这个迭代器对象实现了 \_\_next\_\_() 方法并通过 StopIteration 异常标识迭代的完成。

\_\_next\_\_() 方法（Python 2 里是 next()）会返回下一个迭代器对象。

在 Python 中，使用了 yield 的函数被称为生成器（generator）。

跟普通函数不同的是，生成器是一个返回迭代器的函数，只能用于迭代操作，更简单点理解生成器就是一个迭代器。

在调用生成器运行的过程中，每次遇到 yield 时函数会暂停并保存当前所有的运行信息，返回 yield 的值, 并在下一次执行 next() 方法时从当前位置继续运行。

import sys

def fib(n):

a, b, counter = 0, 1, 0

while True:

if counter > n:

return

yield a

a, b = b, a + b

counter += 1

f = fib(10)

while True:

try:

print(next(f))

except StopIteration:

sys.exit()

# 生成器

带yield的函数是一个生成器，而不是一个函数了，这个生成器有一个函数就是next函数，next就相当于“下一步”生成哪个数，这一次的next开始的地方是接着上一次的next停止的地方执行的，所以调用next的时候，生成器并不会从foo函数的开始执行，只是接着上一步停止的地方开始，然后遇到yield后，return出要生成的数，此步就结束

def foo():

i = 1

for j in range(0,10):

res = yield i + j

print res

a = foo()

print next(a)

print a.send(10)

'''

foo()就是一个生成器

next（）会调用一次，res不会被赋值，为None

send（）也会调用一次 并将 将10 发送给了res

'''

相关博客

<https://blog.csdn.net/mieleizhi0522/article/details/82142856>