Python对延迟提供的支持：在需要时产生结果，而不是立即产生结果，这样不用把所有的结果都存入内存中。

生成器函数：

例：程序function\_detail/function\_detail11.py

from collections import Iterable  
  
  
# 生成器，使用yield关键字  
# 调用函数时，返回一个迭代器对象  
def gensquares(n):  
for i in range(n):  
yield i \*\* 2  
  
  
# gensquares每次循环产生一个值，通过yield  
# 返回给调用者，并暂停，其上一个状态保存下来，  
for i in gensquares(5):  
print(i, end=':') # 0:1:4:9:16:  
print()  
G = 88  
  
  
def gen(n):  
global G  
 G = 99  
for i in range(n):  
yield i \*\* 3  
  
  
# 与普通函数不同，直接调用生成器函数时，  
# 不会立即执行函数体的语句，其返回一个迭代器对象，  
# 通过next()执行，遇到yield返回，再次通过next()执行时，  
# 从上次返回的yield语句处继续执行  
x = gen(4)  
print(isinstance(x, Iterable)) # True  
print("G =", G) # G = 88 # 表明函数gen未执行  
print(next(x)) # 0  
print("G =", G) # G = 99 # 表函数gen执行了  
print(next(x)) # 1  
print(next(x)) # 8  
print()  
  
  
# 通过for循环遍历生成器与通过next()执行生成器是类似的  
# 但如果生成器有return语句时，会有所不同

#生成器不会返回结果，如果有return语句，则返回值会作为StopIteration异常的说明

# for循环会自动过滤StopIteration  
def fib(number):  
 n, a, b = 0, 0, 1  
while n < number:  
yield b  
 a, b = b, a + b  
 n += 1  
return 'Done'  
  
  
f = fib(4)  
while True:  
try:  
print(next(f))  
except StopIteration as e:  
print("Generator return message:", e.value)  
break  
print()  
  
# 通过for循环遍历生成器对象，无法得到return的内容  
for i in fib(4):  
print(i)

输出为：

0:1:4:9:16:

True

G = 88

0

G = 99

1

8

1

1

2

3

Generator return message: Done

1

1

2

3

生成器的一个应用：程序function\_detail/function\_detail11-1.py

def all\_pinyin():  
 for \_ in \_pinyin:  
 yield \_  
 print("pinyin done!")  
  
  
strs = "zhuang"  
  
# if in 作用在生成器上  
# 如果条件不为True，会一直迭代生成器，直到生成器结束  
# 判断strs是否在生成器上，其实就是判断strs是否在\_pinyin里  
# 使用生成器，不用耗费大量的内存

if strs in all\_pinyin():  
 print(strs, ": ok")  
  
strs = "la"  
# 调用一次all\_pinyin()会产生一个新的生成器  
if strs in all\_pinyin():  
 print(strs, ": ok")  
  
strs = "ben"  
if strs in all\_pinyin():  
 print(strs, ": ok")

输出为：

pinyin done!

pinyin done!

ben : ok