Python

Generators

by laike9m laike9m@gmail.com https://github.com/laike9m

学习编程最难的是什么?

You don't know what you don't know

听说过,但不理解>> 没听说过

Generator function

当你看见 yield,就 说明这成一数它的 大生不作数 不作数 不作数 的函数

- When invoked, returns a generator object
- Generator objects implement the iterator interface:

 next (.__next__ in Python 3)

```
Turing.com.br
```

```
>>> def gen_123():
        yield 1
        yield 2
        yield 3
>>> for i in gen_123(): print(i)
>>> g = gen_123()
<generator object gen_123 at ...>
>>> next(g)
>>> next(g)
>>> next(g)
>>> next(q)
Traceback (most recent call last):
StopIteration
```

Generator behavior

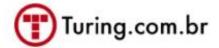
 Invoking a generator function builds the generator object but does not execute the body of the function

```
>>> def gen_ab():
        print('starting...')
        yield 'A'
        print('here comes B:')
        yield 'B'
        print('the end.')
>>> for s in gen_ab(): print(s)
starting...
here comes B:
the end.
>>> g = gen_ab()
>>> next(g)
starting...
>>> next(g)
here comes B:
'B'
>>> next(q)
Traceback (most recent call last):
StopIteration
```

Generator behavior

 The body is executed only when next is called, and only up to the following yield

```
>>> def gen_ab():
        print('starting...')
        yield 'A'
        print('here comes B:')
        yield 'B'
        print('the end.')
>>> for s in gen_ab(): print(s)
starting...
here comes B:
the end.
>>> g = gen_ab()
>>> next(q)
starting...
>>> next(g)
here comes B:
>>> next(g)
Traceback (most recent call last):
StopIteration
```



到底什么意思?

```
>>> def gen_ab():

print('starting...')

yield 'A'

print('here comes B:')

yield 'B'

print('the end.')

第二次调用next(), 停在这里
第二次调用next(), 停在这里
```

生成器函数可以看成一串事件,yield暂停执行,next恢复执行 "yield 是 具有 暂停功能的 return"

```
def gen_ab():
        print('starting...')
        yield 'A'
        print('here comes B:')
        yield 'B'
        print('the end.')
>>> for s in gen_ab(): print(s)
starting...
here comes B:
the end.
>>> g = gen_ab()
>>> next(g)
starting...
>>> next(g)
here comes B:
'B'
>>> next(g)
Traceback (most recent call last):
StopIteration
```

斐波那契生成器(Fibonacci)

斐波那契数列:

0,1,1,2,3,5,8,11,19,...

前两个数是0,1,后一个数是前两个数之和

斐波那契数列生成器

```
def fib(max):
```

$$a, b = 0, 1$$

(1

2

$$a, b = b, a + b$$
 ③

```
>>> def fib(max):
         a, b = 0, 1
         while a < max:
                 yield a
                  a, b = b, a+b
>>> for fib number in fib(100):
         print(fib number)
0 1 1 2 3 5 8
13
21
34
55
89
```

for 循环 每次都会调用next()

更深入的讨论: interator(迭代器)

每次都把返回的a输出 注意yield是"具有暂停功能的return" return a (= yield a) fib_number = a 然后打印出来

不要害怕生成器函数里的循环

• 循环就是一个事件流,只不过里面包含了一些条件判断

def fib(max):

a, b = 0, 1

while a < max:

yield a

a, b = b, a+b

等价于

```
def fib(max):
    a, b = 0, 1
    if a < max:
       yield a
       a, b = b, a+b
    if a < max:
       yield a
       a, b = b, a+b
    if a < max:
       yield a
       a, b = b, a+b
```

练习

- 一起写一个斐波那契生成器
- enumerate 函数

内置enumerate函数

enumerate(iterable, start=0)

Return an enumerate object. *iterable* must be a sequence, an *iterator*, or some other object which supports iteration. The __next__() method of the iterator returned by enumerate() returns a tuple containing a count (from *start* which defaults to 0) and the values obtained from iterating over *iterable*.

```
>>> seasons = ['Spring', 'Summer', 'Fall', 'Winter']
>>> list(enumerate(seasons))
[(0, 'Spring'), (1, 'Summer'), (2, 'Fall'), (3, 'Winter')]
>>> list(enumerate(seasons, start=1))
[(1, 'Spring'), (2, 'Summer'), (3, 'Fall'), (4, 'Winter')]
```

list():调用next()直到不能调用为止,并且把返回值存入列表,实质就是**列表解析** [i for i in enumerate(seasons)]

自己写enumerate函数

```
def enumerate(sequence, start=0):
   n = start
   for elem in sequence:
      yield n, elem
   n += 1
```

生成器表达式

Generator expression >>> g = (n for n in [1, 2, 3]) >>> for i in g: print i

 When evaluated, returns a generator object

```
g.next()是Python2.X
的写法,对应
Python3.X中的next(g)
```

```
>>> g = (n for n in [1, 2, 3])
>>> q
<generator object <genexpr> at
0x109a4deb0>
>>> g.next()
>>> g.next()
>>> g.next()
>>> g.next()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
StopIteration
```



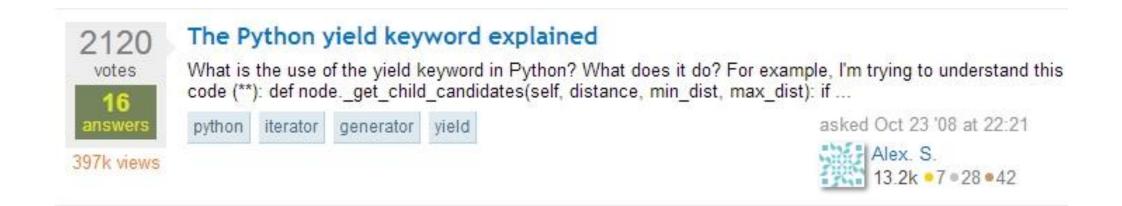


总结

- 有yield的函数: 生成器函数
- yied是带暂停功能的return
- for, list 本质上是在调用next(obj)
- 把生成器函数看成事件流,yield暂停,next继续执行
- 生成器函数内部通常包含循环,循环也是事件流
- 把列表解析的中括号换成小括号就是生成器表达式

为什么要用生成器

- 快
- 内存占用小 需要的时候才会产生
- 语义上的含义
 - 一次性使用
- 保存函数的当前执行环境,包括所有局部变量等



http://stackoverflow.com/questions/231767/the-python-yield-keyword-explained

Q&A