Python程序设计

第六讲 函数与模块化
生成器函数

张 华
WHU

生成器函数

生成器函数

- 生成器函数使用yield语句返回一个值，然后保存当前函数整个执行状态，等待下一次调用next()。
- 生成器函数是一个迭代器，是可迭代对象，支持迭代。

```
>>> def int_stream(n):
        for i in range(n):
            yield i


>>> type(int_stream)
<class 'function'>
>>> istream = int_stream(10)
>>> type(istream)
<class 'generator'>
>>> from collections.abc import Iterator
>>> isinstance(istream, Iterator)
True
```

```
>>> next(istream)
0
>>> istream.__next__()
1
>>> [v for v in istream]
[2, 3, 4, 5, 6, 7, 8, 9]
>>> next(istream)
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    next(istream)
StopIteration
```

生成器函数

- 生成器函数和普通函数的执行流程不一样。
  - 普通函数是顺序执行，遇到return语句或者最后一行函数语句就返回。
  - 而改成生成器的函数，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。

```
def odd():
    print('step 1')
    yield 1
    print('step 2')
    yield 3
    print('step 3')
    yield 5
```

```
>>> o = odd()
>>> next(o)
step 1
1
>>> next(o)
step 2
3
>>> next(o)
step 3
5
```

生成器函数

练习：

- 利用生成器函数创建Fibonacci数列。

```
def genFibs():
    a,b = 0,1
    while True:
        a,b = b,a+b
        yield a

for x in genFibs():
    if x<100:
        print(x, end=' ')
    else:
        break
```

```
1 1 2 3 5 8 13 21 34 55 89
```