Python高级特性

切片

- list/tuple/str都支持截取操作，只是tuple切片还是tuple，str切片还是str

```
# a = ['P', 'D', 'Y', 'U', 'R', 'S', 'Q', 'C', 'G', 'K']

a = "hellokitty"
# 指定范围，从1开始，到4结束，但是不取最后1个
print(a[1:4])
# 从头开始
# print(a[:4])
# # 倒数切片，去掉最后一个
print(a[:-1])
# # 取后5个
# print(a[-5:])
# # 从0取到5，每2个取1个
# print(a[0:5:2])
# # 从头取到最后，每5个取1个
# print(a[0::5])
# # 倒顺序
# print(a[::-1])
```

```
ell
hellokitt
```

迭代

- 通过for循环遍历tuple或list，叫迭代
- dict也可以迭代

```
# for i in range(10):
#     print(i)


# # 仅迭代key
d = {"name": "duoceshi", "age": 4, "score": 99}
# for i in d:
#     print(i)

# 仅迭代value
```

```
# for v in d.values():
#     print(v)

# # 同时迭代
# for k, v in d.items():
#     print(k, v)

# for i in d:
#     print(i, d[i])
```

```
name duoceshi
age 4
score 99
```

- 字符串也可以迭代

```
for i in 'hellokitty':
    print(i)
```

- 判断对象是否可迭代

```
from collections import Iterable
# print(isinstance("abc", Iterable))
# print(isinstance([1, 2, 3], Iterable))
print(isinstance(123, Iterable))
```

```
False
```

- 遍历列表时改为元素对

```
a = ['A', 'B', 'C']
# for i in range(len(a)):
#     print(i, a[i])

for i, value in enumerate(a):
    print(i, value)
```

```
0 A
1 B
2 C
```

- 小结： 任何可迭代对象都可以作用于for循环，包括我们自定义的数据类型，只要符合迭代条件，就可以使用for循环。

列表生成式

- 基础用法

```
from collections import Iterable
# range(1, 100)
a = range(10)
print(a, type(a))
# b = list(a)
# print(a, type(a))
# print(isinstance(a, Iterable))
# for i in a:
#     print(i)
# print(b)
```

```
range(0, 10) <class 'range'>
0
1
2
3
4
5
6
7
8
9
```

- 高级用法

```
# 生成1-10的平方
a = []
# for i in range(1, 11):
#     for j in range(1, 11):
#         if i % 2 == 0:
#             a.append(i + j)
#             print(i, j)
# print(a)

# # # 简写
b = [i * i for i in range(1, 11)]
# print(b)
# b = [ i*i for i in range(1, 10)]
# print(b)
# # # 添加判断，偶数的平方
c =  [ i*i for i in range(1, 11) if i%2==0]
# print(c)
# # # 嵌套两层循环
```

```
d = [m + n for m in 'ABC' for n in 'XYZ']
# # 三层
d = [m + n + v  for m in 'ABC' for n in 'XYZ' for v in '123']
print(d)
```

```
['AX1', 'AX2', 'AX3', 'AY1', 'AY2', 'AY3', 'AZ1', 'AZ2', 'AZ3', 'BX1', 'BX2',
'BX3', 'BY1', 'BY2', 'BY3', 'BZ1', 'BZ2', 'BZ3', 'CX1', 'CX2', 'CX3', 'CY1',
'CY2', 'CY3', 'CZ1', 'CZ2', 'CZ3']
```

- 练习1：列出当前文件列表

```
# 列出当前文件列表
import os
e = [d for d in os.listdir('.')]
print(e)
```

```
['re正则.ipynb', '变量与数据结构.py', 'test_script', '02python基础之函数.ipynb',
'01python基础回顾.ipynb', 'test.log', 'output.log', 'regex.ipynb', '04函数式编
程.ipynb', '递归字典', 'beautifulsoup.ipynb', '生成号码.py', 'logging模块.ipynb',
'requests.ipynb', 'yield实战.py', '.ipynb_checkpoints', 'pyquery.ipynb',
'demo.html', 'selenium.ipynb', '03python高级特性.ipynb', '导入练习']
```

- 练习2：字典生成列表

```
a = {"name": "duoceshi", "age": 4}
b = [k + str(v) for k, v in a.items()]
print(b)
```

```
['nameduoceshi', 'age4']
```

- 练习3：是字符串大小写切换

```
a =  ['Hello', 'World', 'IBM', 'Apple']
# 变为小写
b = [ i.lower() for i in a]
print(b)
# # 变为大写

d = [ i.upper() for i in a]
print(d)
```

```
['hello', 'world', 'ibm', 'apple']
['HELLO', 'WORLD', 'IBM', 'APPLE']
```

- 练习4：把生成随机字符的函数改造用列表生成式

```
import random, string
def create_str2(num_int=0, num_letters=0, num_zh=0, num_pun=0):
    a = [random.choice(string.digits) for i in range(int(num_int))]
    b = [random.choice(string.ascii_letters) for i in range(int(num_letters))]
    c = [chr(random.randint(0x4e00, 0x9fbf)) for i in range(int(num_zh))]
    d = [random.choice(string.punctuation) for i in range(int(num_pun))]
    ran_list = a + b + c + d
    random.shuffle(ran_list)
    return ''.join(ran_list)
create_str2(1,2,3,4)
```

```
'A1]緪鄘&蔥Y(='
```

- 课后练习：将随机号码生成函数，改成列表生成式

```
import random, string
# 生成指定号段的随机手机号码
t =
'130,131,132,145,146,155,156,166,167,1704,1707,1708,1709,171,175,176,185,186'
t_list = t.split(',')
c = [random.choice(t_list) + ''.join(random.sample(string.digits*10, 11 -
len(random.choice(t_list)))) for i in range(10)]
print(c)
```

```
['1664378540', '170469567412', '15593216572', '13167866498', '170741605345',
'16628849517', '17631954309', '170706426758', '17169366888', '16756268087']
```

生成器

- 引入原因：列表大小受内存限制，初始化的时候就会占用掉固定内存
- 引入生成器：generator，通过算法推演后续元素，一边循环一边使用，达到用多少，取多少的目的，节省内存开支

```
# 直接生成列表
# a = [x * x for x in range(10000000000000)]
# print(a)
# 使用生成器，是一个生成器对象
b =( x * x for x in range(10000000000000000000000000000000))
print(b)
# print(b)
# # 生成器也支持for循环遍历
# for i in b:
#     if i > 1099:
#         break
#     print(i)
```

- 使用yield定义生成器
- 补充：**任何使用yield关键字的函数都称之为生成器**
- 使用yield，可以让函数生成一个序列，该函数返回的对象类型是”generator”，通过该对象连续调用next()方法返回序列值。

```
def odd1():
    print('step 1')
    return 1
    print('step 2')
    return(3)
    print('step 3')
    return(5)

# a = odd1()
# print(a)


def odd2():
#     print('step 1')
    yield(1)
#     print('step 2')
    yield(3)
#     print('step 3')
    yield(5)

h = odd2()
# print(h)
# # 在执行过程中，遇到yield就中断，下次又继续执行
next(h)
next(h)
next(h)
# next(h)
# for i in odd2():
#     print(i)
# # next(h)
```

迭代器

- for循环可以遍历两类数据类型：
    1. 集合数据类型：list/tuple/dict/set/str等
    2. generator：普通生成器/带yield的函数
- 可迭代对象(Iterable)：可直接用于for循环的对象，不要求可以使用next()方法
- 迭代器(Iterator)：可以使用next()方法的对象，表示一个惰性计算的序列，不要求可以循环
- 生成器(generator)：在Python中，一边循环一边计算的机制，称为生成器：generator，是迭代器的一种
    1. 看起来像函数：可以接收参数，可以被调用，但不同于一般的函数一次性返回全部结果，生成器一次只生产1个值，目的是为了防止内存溢出
    2. 生成器的本质上是一个迭代器，保存的是算法，每次调用next()的时候去计算下一个元素值
    3. 生成器同时也是一个可迭代对象，一般使用的时候，不要使用next()，而是直接使用for循环
    4. 所有生成器都是迭代器，但是反过来迭代器不一定是生成器

```
# 判断是否可迭代
print(isinstance([], Iterable))
print(isinstance({}, Iterable))
print(isinstance('abc', Iterable))
print(isinstance((x for x in range(10)), Iterable))
print(isinstance(100, Iterable))
```

```
True
True
True
True
False
```

yield总结：

1. 通常的for..in...循环中，in后面是一个数组，这个数组就是一个可迭代对象，类似的还有链表，字符串，文件。他可以是`a = [1,2,3]`，也可以是`a = [x*x for x in range(3)]`。它的缺点也很明显，就是所有数据都在内存里面，如果有海量的数据，将会非常耗内存。
2. 生成器是可以迭代的，但是只可以读取它一次。因为用的时候才生成，比如a = (x*x for x in range(3))。!!!!注意这里是小括号而不是方括号。
3. 生成器（generator）能够迭代的关键是他有next()方法，工作原理就是通过重复调用next()方法，直到捕获一个异常。
4. 带有yield的函数不再是一个普通的函数，而是一个生成器generator，可用于迭代
5. yield是一个类似return 的关键字，迭代一次遇到yield的时候就返回yield后面或者右面的值。而且下一次迭代的时候，从上一次迭代遇到的yield后面的代码开始执行
6. yield就是return返回的一个值，并且记住这个返回的位置。下一次迭代就从这个位置开始。
7. 带有yield的函数不仅仅是只用于for循环，而且可用于某个函数的参数，只要这个函数的参数也允许迭代参数。

- 实战练习：使用python模拟tail -f 监控日志文件(日志监控程序)

```
import time


def tail(f):
    # 第一个参数：偏移量
    # 第二个参数：0，代表文件开始，1，代表文件当前位置，2，代表文件末尾
    f.seek(0, 2)  # 移动到文件最后一行
    while True:
        line = f.readline()  # 读取文件中新的文本行
        if not line:
            # 没有最新行，则每0.1秒执行一次
            time.sleep(0.1)
            continue
        yield line


def grep(lines, searchtext):
    # 遍历lines生成器
    for line in lines:
        # 判断是否包含关键字
        if searchtext in line:
            # 若包含则返回
            yield line


# 模拟tail -f |grep python
# 获取生成器
flog = tail(open('test.log'))
# 把生成器当作可迭代对象传入，继续yield返回，pylines得到的也是一个生成器
pylines = grep(flog, 'python')
# 再次遍历生成器，将返回结果打印
for line in pylines:
    print(line, )
```

- 实战练习2:通过固定长度的缓冲区不断读文件，防止一次性读取出现内存溢

```python
# 实战2，分段读取
def read_file(path):
    size = 1024  # 每次取1024字节，即1024个字符
    with open(path, 'r') as f:
        while True:
            # 按固定的大小每次去读取
            block = f.read(size)
            if block:
                yield block
            else:
                # 没有内容了则返回空
                return
```