**Itertools:functions to creat iterators for efficient looping**

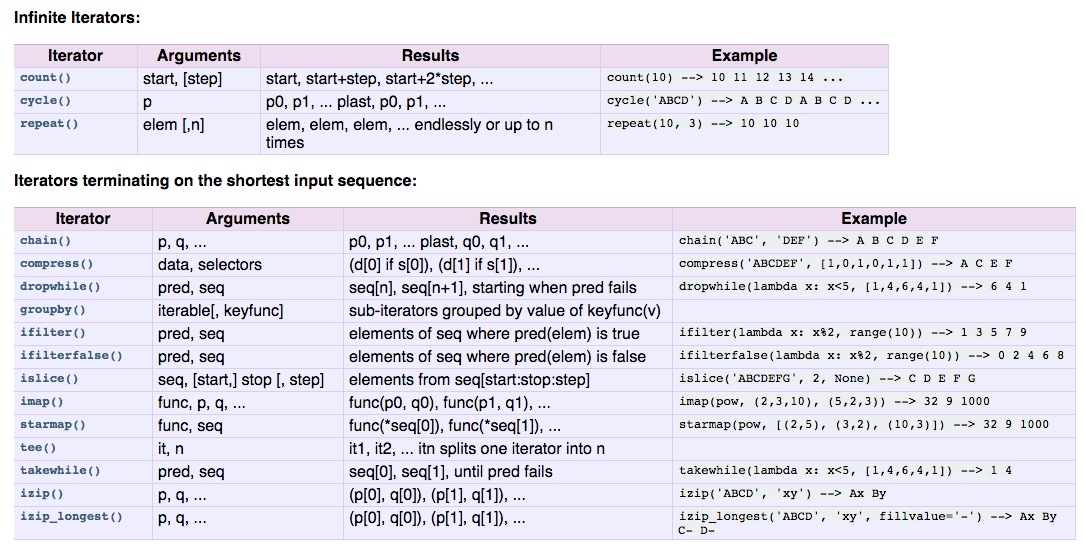
**https://docs.python.org/2/library/itertools.html**

Count(start,[step]) : count(10) -> 10 11 12 13…

Cycle(p) :cycle(‘abcd’) -> a b c d a b c d a…

Repeat(elt,[t]): repeat(10,3) -> 10,10,10

Chain(p,q…) [connect the p ,q …] : chain(‘abc’,’efg’) -> a b c e f g



**dropwhile**: start when the predictions fails.

**Takewhile:** end when the predictions fails.

dropwhile(lambda x: x<5, [1,4,6,4,1]) --> 6 4 1

takewhile(lambda x: x<5, [1,4,6,4,1]) --> 1 4

**izip 和 izip\_longest:** 两者都是将序列一一组合，不过后者一长序列为准，其他的用任意的补全.

**Imap 和 starmap**: 都是返回某种函数操作的iterator,

但前者可以又多个序列，即imap(func,p,q,m…),func()将依次取出每个序列的相应值并返回，

后者是只有一个序列，但序列的每个元素可以是tuple,即starmap(func,[(q0,q1),(q2,q3)…]),但此时func 就要处理tuple argument了。

**Product() 和 permutation() 和combinations()和combinations\_with\_replacement()**

:

输入都是：’ABCD’,2

product:AA,AB,AC,AD,BA,BB,BC,BD,CA,CB,CC,CD,DA,DB,DC,DD

permutation: AB,AC,AD,BA, BC,BD,CA,CB, CD,DA,DB,DC(去掉相同的例子)

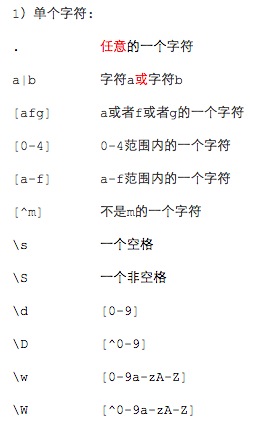
combinations():AB,AC,AD,BC,BD,CD

combinations\_with\_replacement():AB,AC,AD,BC,BD,CD+AA,BB,CC,DD

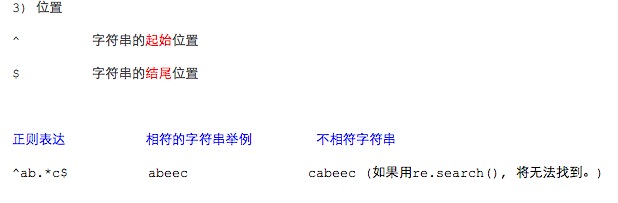
**RE Module: regular expression operations**

**正则表达式(regular expression)**主要功能是从字符串(string)中通过特定的模式(pattern)，搜索想要找到的内容。

A regular expression operation (notice it is **back slash**):







**Group:**

output\_(\d{4})

该正则表达式用括号()包围了一个小的正则表达式，\d{4}。 这个小的正则表达式被用于从结果中筛选想要的信息（在这里是四位数字）。这样被括号圈起来的正则表达式的一部分，称为群(group)。

若要将group 命名，可以使用**(?P<name>...)，**例如：

****

**Functions:**

**re.search(pattern, string):** 搜索整个字符串，知道找到符合的子 字符串

**re.match(pattern, string):** 搜从头检查字符串是否符合正则表达式，**必须从字符串的第一个字符**开始就相符。

**re.sub(pattern,replacement, string):** 将string按照pattern 搜索，将搜索后的字符串用replacement 代替，返回替换后的字符串

**re.split():** 将字符根据re 分割字符串，将所有子字符串放在一个list中返回。其中匹配的pattern 在列表中就变成了’’,例如：

m=re.split(‘\d{5}’,’199213dsd’),

m->[‘’,’3dsd’]

**re.findall():** 将字符串根据re 搜索，所有符合的字符串放在list中返回

为了提高效率，配合re.search(),re.match(),可以先将pattern compile一下

**pro=re.compile(pattern)**

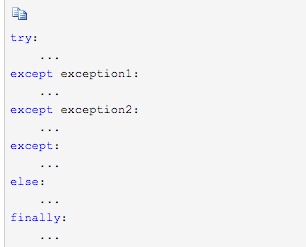
**result=pro.match(string)#or result=pro.search(string)**

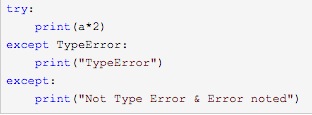
练习  
有一个文件，文件名为output\_1981.10.21.txt 。下面使用Python： 读取文件名中的日期时间信息，并找出这一天是周几。将文件改名为output\_YYYY-MM-DD-W.txt (YYYY:四位的年，MM：两位的月份，DD：两位的日，W：一位的周几，并假设周一为一周第一天)

(要配合os module 和datatime 或time包)

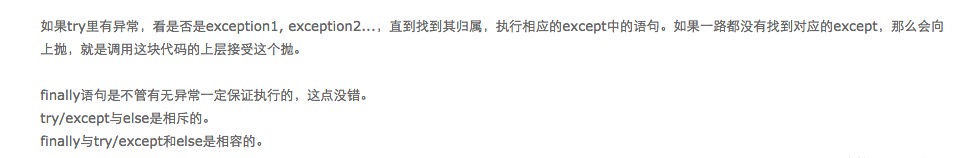
**Exception 处理：**

完整的结构：如果try 中有异常，执行异常的归属，执行except,异常层层比较，看是否是exception1,exception2…,若exception后没有任何参数，则将所有exception交给此段程序执行，





同样如果没有将异常交给合适的对象，将异常继续向上抛出直至被捕捉或者造成主程序报错



see the example at ***exception.py***

**文件和路径OS：**

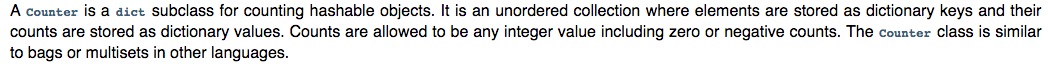
os.path包主要是处理路径字符串，比如说'/home/vamei/doc/file.txt'，提取出有用信息。

glob包最常用的方法只有一个, glob.glob()。该方法的功能与Linux中的ls相似(参看[Linux文件管理命令](http://www.cnblogs.com/vamei/archive/2012/09/13/2682519.html))，接受一个Linux式的文件名格式表达式(filename pattern expression)，列出所有符合该表达式的文件（与正则表达式类似）

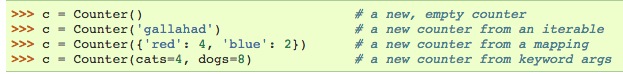
**Collections**

**High-performance container datatypes**

**Counter()**[注意C 必须大写],里面的对象必须是iterable 或者是mapping.



常见的counter 初始化：



Counter({'11': 1, '22': 1}) -> counter 的形式,是dict 的子类。一般我们要对对象的元素进行计数处理，可以优先考虑Counter()［注意C是大写］

**elements**()

**most\_common**([*n*])

**subtract**([*iterable-or-mapping*])

**namedtuple()**:将tuple 里面的元素命名化了，这样我们可以不用数字index 来访问tuple了，以便于我们访问tuple.

里例如：

from collections import namedtuple

point=namedtuple(‘point’,’x y’) [注意x 和y 之间又一个空格，用来识别这个tuple 里有几个参数]

或者是：

point=namedtuple(‘point’,[‘x’,’y’])

a=point(1,2),or a=point(x=1,y=2)

a.x,a.y 也可以访问tuple里面的元素了。

其他的方法：

**somenamedtuple.\_fields**

**somenamedtuple.\_make(*iterable*)**

**Deque():**Deque支持线程安全的，经过优化的append和pop操作，在队列两端的相关操作都能够达到近乎O(1)的时间复杂度。虽然list也支持类似的操作，但是它是对定长列表的操作表现很不错，而当遇到pop(0)和insert(0, v)这样既改变了列表的长度又改变其元素位置的操作时，其复杂度就变为O(n)了。

下面的例子证明了：

import time

from collections import deque

num = 100000

def append(c):

    for i in range(num):

        c.append(i)

def appendleft(c):

    if isinstance(c, deque):

        for i in range(num):

            c.appendleft(i)

    else:

        for i in range(num):

            c.insert(0, i)

def pop(c):

    for i in range(num):

        c.pop()

def popleft(c):

    if isinstance(c, deque):

        for i in range(num):

            c.popleft()

    else:

        for i in range(num):

            c.pop(0)

for container in [deque, list]:

    for operation in [append, appendleft, pop, popleft]:

        c = container(range(num))

        start = time.time()

        operation(c)

        elapsed = time.time() - start

        print "Completed {0}/{1} in {2} seconds: {3} ops/sec".format(

              container.\_\_name\_\_, operation.\_\_name\_\_, elapsed, num / elapsed)

# Completed deque/append in 0.0250000953674 seconds: 3999984.74127 ops/sec

# Completed deque/appendleft in 0.0199999809265 seconds: 5000004.76838 ops/sec

# Completed deque/pop in 0.0209999084473 seconds: 4761925.52225 ops/sec

# Completed deque/popleft in 0.0199999809265 seconds: 5000004.76838 ops/sec

# Completed list/append in 0.0220000743866 seconds: 4545439.17637 ops/sec

# Completed list/appendleft in 21.3209998608 seconds: 4690.21155917 ops/sec

# Completed list/pop in 0.0240001678467 seconds: 4166637.52682 ops/sec

# Completed list/popleft in 4.01799988747 seconds: 24888.0046791 ops/sec

**Defaultdict():**Defaultdict()可以设置默认的dict 的value的默认形式，这样我们在未初始化的情况下可以对value 进行一些操作，如我们可以实施append 的操作。

**Random: generate the pseudo random numbers**

Random.seed():

Random.range(start,stop,step):跟choice(range(start, stop, step)) ，但并没有建立range 对象。

random.**randint**(*a*, *b*) ：Return a random integer *N* such that a <= N <= b.

random.**choice**(*seq*)：Return a random element from the non-empty sequence *seq*. If *seq* is empty, raises [**IndexError**](https://docs.python.org/2/library/exceptions.html#exceptions.IndexError)

random.**shuffle**(*x*[, *random*])：

random.**sample**(*population*, *k*)：在polulation 中选择k个数，如果population 不是unique的话，可能被重复选择

random.**random**()：随机选择（0～1）间的数

some other distributions like : random.**uniform**(*a*, *b*), random.**triangular**(*low*, *high*, *mode*), random.**betavariate**(*alpha*, *beta*)

random.**expovariate**(*lambd*)

random.**gammavariate**(*alpha*, *beta*)….

**Notice:**

The pseudo-random generators of this module should not be used for security purposes. Use [os.urandom()](https://docs.python.org/2/library/os.html#os.urandom) or [SystemRandom](https://docs.python.org/2/library/random.html#random.SystemRandom) if you require a cryptographically secure pseudo-random number generator

**Queue (一般和multiprocess 和multithreads 一起用)**

**From Queue import Queue**

**Python Queue模块有三种队列及构造函数:**  
1、Python Queue模块的FIFO队列先进先出。 class Queue.Queue(maxsize)  
2、LIFO类似于堆，即先进后出。 class Queue.LifoQueue(maxsize)  
3、还有一种是优先级队列级别越低越先出来。 class Queue.PriorityQueue(maxsize)

**此包中的常用方法(q = Queue.Queue()):**  
q.qsize() 返回队列的大小  
q.empty() 如果队列为空，返回True,反之False  
q.full() 如果队列满了，返回True,反之False  
q.full 与 maxsize 大小对应  
q.get([block[, timeout]]) 获取队列，timeout等待时间  
q.get\_nowait() 相当q.get(False)  
非阻塞 q.put(item) 写入队列，timeout等待时间  
q.put\_nowait(item) 相当q.put(item, False)  
q.task\_done() 在完成一项工作之后，q.task\_done() 函数向任务已经完成的队列发送一个信号  
q.join() 实际上意味着等到队列为空，再执行别的操作

**两个多线程的例子：**

[**http://blog.csdn.net/bravezhe/article/details/8588437**](http://blog.csdn.net/bravezhe/article/details/8588437)

[**http://www.jb51.net/article/58004.htm**](http://www.jb51.net/article/58004.htm)

**heapq （堆）:最小堆**

**heapify(heap):将heap 属性强制引用到任意一个列表heap**

**heappush(heap,x):将x入堆**

**heappop(heap):将最小元素弹出**

**heapreplace(heap,x):将最小元素弹出后并将x将加入堆**

**nlargest(n,iter):返回iter中前n大的元素**

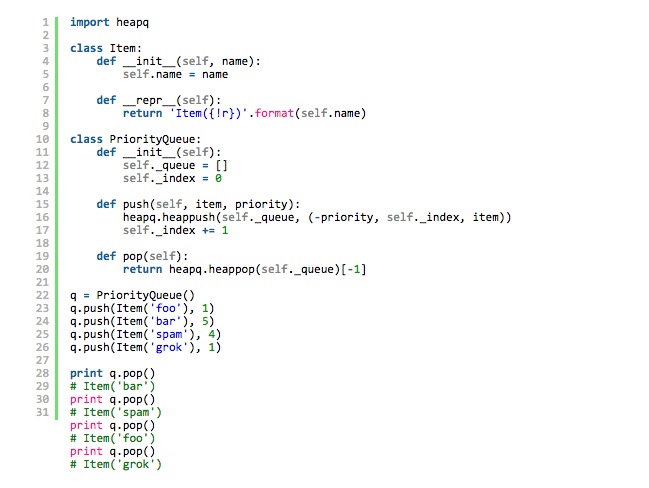
**nsmallest(n,iter):返回iter中前n小的元素 ，**

**这两者的输入里面还可以有key参数：**

****

**后两个函数可以用sorted 加分片功能实现，但是heap的属性更快而且有效使用内存。**

**而且，heapq模块可以实现优先队列，**每次pop操作都返回优先级最高的元素的队列例子，如：



**Bisect: array bisection algorithm**

bisect模块能够提供保持list元素序列的支持。它使用了二分法完成大部分的工作。它在向一个list插入元素的同时维持list是有序的。在某些情况下，这比重复的对一个list进行排序更为高效，并且对于一个较大的list来说，对每步操作维持其有序也比对其排序要高效

bisect.bisect(seq,item,lo=0,hi=len(seq)) => 返回索引，类似的还有

bisect.bisect\_left(seq,item,lo=0,hi=len(seq)), bisect.bisect\_right(seq,item,lo=0,hi=len(seq))

bisect.insort\_left(seq,item，lo=0,hi=len(seq)) => 将item 插入有序的列表内。类似的还有：bisect.insort\_right(seq,item，lo=0,hi=len(seq))

对于item已经在seq的情况下，bisect.bisect(seq,item,lo=0,hi=len(seq)) 将元素插在存在元素后，而bisect.bisect\_left(seq,item,lo=0,hi=len(seq)), 插在存在元素前

**weakref:不会阻止对象的撤销**

weakref模块能够帮助我们创建Python引用，却不会阻止对象的销毁操作。这一节包含了weak reference的基本用法，并且引入一个代理类。

class Foo(object):

    def \_\_init\_\_(self):

        self.obj = None

        print 'created'

    def \_\_del\_\_(self):

        print 'destroyed'

    def show(self):

        print self.obj

    def store(self, obj):

        self.obj = obj

a = Foo() # created

b = a

del a

del b # destroyed

上述例子只有当我们同时删除a 和b时候foo对象才删除，因为a,b 都是strong reference,

只删除a或者b并不会将引用计数化为0，这样就不会被垃圾所回收。

**而weak reference** 则是对对象的引用计数器不会产生影响。当一个对象存在weak reference时，并不会影响对象的撤销。这就说，如果一个对象仅剩下weak reference，那么它将会被销毁。

**weakref.ref：**

>>> import weakref

>>> a = Foo()

created

>>> b = weakref.ref(a)

>>> b

>>> del a

destroyed

>>> b() is None

True

注意当我们删除a的时候，对象已经被回收了，b已经是空引用了。

weakref.proxy： 用法基本和上面基本一样，不过当我们对weak ref 对象进行操作时，会抛出referenceError,。

**Notice:**

能被weakref的对象只有class instances、functions、methods、sets、frozen sets、files、generators、type objects和certain object types(例如sockets、arrays和regular expression patterns)，**而内建的类型list,dict,strings和numbers不支持**。

**inspect : 关于python 活动的对象的信息，源代码，栈帧等**

getclasstree(classes,[,unique]):classes 是一个 classes 列表，根据继承关系将类排序为层次结构。

getdoc(object)

getfile(object):返回定义object 的文件名。

getmembers(object[,predicate])

…