29-itertools

April 24, 2016

1 itertools module

- advanced iteration tools
- doc

```
In [1]: # like linux 'uniq' command

        from itertools import *

        for k, g in groupby(sorted([1,2,3,1,1,2,1,3,7,3])):
            print(k , list(g))
```

```
1 [1, 1, 1, 1]
2 [2, 2]
3 [3, 3, 3]
7 [7]
```

```
In [2]: # iterates over each arg, from left to right

        list(chain('foo', [1,2,3], 'bar'))
```

```
Out[2]: ['f', 'o', 'o', 1, 2, 3, 'b', 'a', 'r']
```

```
In [3]: # cute way to make power sets

        x = [1, 2, 3]
        print(list(map(list, (combinations(x, r) for r in range(len(x)+1)))))
        list(chain.from_iterable(combinations(x, r) for r in range(len(x)+1)))
```

```
[[()], [(1,), (2,), (3,)], [(1, 2), (1, 3), (2, 3)], [(1, 2, 3)]]
```

```
Out[3]: [(), (1,), (2,), (3,), (1, 2), (1, 3), (2, 3), (1, 2, 3)]
```

```
In [4]: # no replacements

        list(combinations(range(4), 3))
```

```
Out[4]: [(0, 1, 2), (0, 1, 3), (0, 2, 3), (1, 2, 3)]
```

```
In [5]: list(combinations_with_replacement(range(3), 3))
```

```
Out[5]: [(0, 0, 0),
         (0, 0, 1),
         (0, 0, 2),
         (0, 1, 1),
```

```
 (0, 1, 2),
 (0, 2, 2),
 (1, 1, 1),
 (1, 1, 2),
 (1, 2, 2),
 (2, 2, 2)]
```

In [13]:
```
# similiar to numpy boolean indexing

list(compress(range(5), [1,0,0,1,0]))
```

Out[13]:
```
[0, 3]
```

In [6]:
```
# repeats indefinitely

c =cycle('larry')

[ next(c) for j in range(13) ]
```

Out[6]:
```
['l', 'a', 'r', 'r', 'y', 'l', 'a', 'r', 'r', 'y', 'l', 'a', 'r']
```

In [7]:
```
# repeat generates infinite sequence of one value

g = repeat(2)
for e in range(4):
    print(next(g))
```

```
2
2
2
2
```

In [39]:
```
# can use repeat with zip, because zip terminates when one sequence terminates

 [b**e for b,e in zip(g, range(4))]
```

Out[39]:
```
[1, 2, 4, 8]
```

In [8]:
```
# another way to do a padded dot product

list(zip_longest([1,2,3,4], [1], [4,5], fillvalue=10))
```

Out[8]:
```
[(1, 1, 4), (2, 10, 5), (3, 10, 10), (4, 10, 10)]
```

In [9]:
```
# count produces an infinite sequence
# count is lazy

for j,c in enumerate(count(start=3, step=5)):
    if j > 10:
        break
    print(j, c)
```

```
0 3
1 8
2 13
3 18
4 23
```

```
5 28
6 33
7 38
8 43
9 48
10 53
```

In [10]:
```
# takewhile takes elements from begining of a squence until predicate fails

g = takewhile(lambda x: x < 30, count(start=3, step=5))
list(g)
```

Out[10]:
```
[3, 8, 13, 18, 23, 28]
```

In [21]:
```
# dropwhile drops some number of items at the begining of a sequence

g = dropwhile(lambda x: x < 30, count(start=3, step=5))
[ next(g) for j in range(20) ]
```

Out[21]:
```
[33,
 38,
 43,
 48,
 53,
 58,
 63,
 68,
 73,
 78,
 83,
 88,
 93,
 98,
 103,
 108,
 113,
 118,
 123,
 128]
```

In [20]:
```
# since count is infinite, g is infinite

next(g)
```

Out[20]:
```
113
```

In [12]:
```
# only keep iterables that meet a criteria

list(filter(lambda x : x % 3 ==0, range(10)))
```

Out[12]:
```
[0, 3, 6, 9]
```

In [13]:
```
# map func can take more than one arg

list(map(lambda x, y: x + 5 * y, range(10), range(10,20)))
```

Out[13]:
```
[50, 56, 62, 68, 74, 80, 86, 92, 98, 104]
```

```
In [11]: # lets you take a slice of a generator

         list(islice(count(start=100), 4, 10, 2 ))
```

```
Out[11]: [104, 106, 108]
```

```
In [12]: list(permutations(range(3)))
```

```
Out[12]: [(0, 1, 2), (0, 2, 1), (1, 0, 2), (1, 2, 0), (2, 0, 1), (2, 1, 0)]
```

```
In [16]: list(permutations(range(3),2))
```

```
Out[16]: [(0, 1), (0, 2), (1, 0), (1, 2), (2, 0), (2, 1)]
```

```
In [13]: # cartesian product

         list(product(['jack','jill'], ['hill', 'up', 'water']))
```

```
Out[13]: [('jack', 'hill'),
          ('jack', 'up'),
          ('jack', 'water'),
          ('jill', 'hill'),
          ('jill', 'up'),
          ('jill', 'water')]
```

```
In [23]: # sort of a running total
         # lazy

         list(accumulate([1,4,7,4,3,1,2,9]))
```

```
Out[23]: [1, 5, 12, 16, 19, 20, 22, 31]
```

```
In [18]: # make N generators

         gs = tee(range(10), 5)
         gs
```

```
Out[18]: (<itertools._tee at 0x107466a48>,
          <itertools._tee at 0x107466f88>,
          <itertools._tee at 0x107468048>,
          <itertools._tee at 0x107468148>,
          <itertools._tee at 0x107468188>)
```

```
In [19]: # each generator is independent

         list(map(list, gs))
```

```
Out[19]: [[0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9],
          [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]]
```

```
In [48]: # pull out parts of a list

         t1, t2, t3 = tee(range(20),3)

         list(map(list, [filter(lambda x : 0 == x % 2, t1), filter(lambda x : x >10, t2),
                   filter(lambda x : x < 7, t3)]))
```

```
Out[48]: [[0, 2, 4, 6, 8, 10, 12, 14, 16, 18],
          [11, 12, 13, 14, 15, 16, 17, 18, 19],
          [0, 1, 2, 3, 4, 5, 6]]
```

2 operators

- functions that correspond to operators

```
In [24]: import operator
         [operator.add(2,3), operator.mul(2,3), operator.pow(2,3)]
```

```
Out[24]: [5, 6, 8]
```

```
In [25]: # yet another way to do dot product

         sum(map(operator.mul, [1,2,3], [4,5,6]))
```

```
Out[25]: 32
```

```
In [34]: # starmap(func, seq) => func(*seq[0], *seq[1])

         list(starmap(operator.pow, [[2,4], [5,3], [3,2]]))
```

```
Out[34]: [16, 125, 9]
```