9. Module itertools — iterators for efficient looping

```
import itertools
```

Fast, memory-efficient **iterator building blocks.** Together they form an "iterator algebra" in pure Python. Often used iterators obtained from iterables as arguments: `iter(iterable)` or `range()`.

1 Infinite iterators

```
count(start [,step])
count(10) --> 10 11 12 13 14 ...

cycle(p)
cycle('ABCD') --> A B C D A B C D ...

repeat(elem [, N])
repeat(10, 3) --> 10 10 10
```

2 Finite iterators

```
accumulate(p [,func of 2 args])
accumulate([1,2,3,4,5]) --> 1 3 6 10 15

chain(p, q)
chain('ABC', 'DEF') --> A B C D E F

chain.from_iterable(iterable)
chain.from_iterable(['ABC', 'DEF']) --> A B C D E F

compress(data, selectors)
compress('ABCDEF', [1,0,1,0,1,1]) --> A C E F

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

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

filter(lambda, seq) - builtin function

filterfalse(lambda, seq)
filterfalse(lambda x: x%2, range(10)) --> 0 2 4 6 8

groupby(iterable [, key])
for i in itertools.groupby([6, 6, 8, 8, 11]): print(i[0]) --> 6, 8, 11

islice(seq, [start,] stop [, step])
islice('ABCDEFG', 2, None) --> C D E F G

starmap(func, seq)
starmap(pow, [(2,5), (3,2), (10,3)]) --> 32 9 1000
```

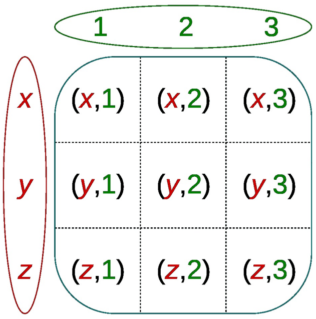
```
tee(iterable, n) - returns n independent iterators from a single iterable, copy iterators
for i in itertools.tee([6, 6, 8, 8, 11], 3): print(list(i)) -->
[6, 6, 8, 8, 11]
[6, 6, 8, 8, 11]
[6, 6, 8, 8, 11]

zip_longest('ABCD', 'xy', fillvalue='-') --> Ax By C- D-

zip(p,q,...) - built-in function
zip('abcd', '1234') --> [('a', '1'), ('b', '2'), ('c', '3'), ('d', '4')]
```

3 Combinatoric iterators

```
product(p,q,..., [repeat=1]) -- Cartesian product;
'repeat' is equivalent to the number of nested loops:
product('AB', 2) --> ('A','A'), ('A','B'), ('B','A'), ('B','B')

permutations(p [,length])
permutations('ABC', 2) --> ('A','B'), ('A','C'), ('B','A'), ('B','C'), ('C','A'), ('C','B')

combinatios(p, length)
combinations('ABC', 2) --> ('A','B'), ('A','C'), ('B','C')

combinations_with_replacement(p, length)
combinations_with_replacement('AB', 2) --> ('A','A'), ('A','B'), ('B','B')
```

4 Cartesian product

In set theory (and, usually, in other parts of mathematics), a Cartesian product is a mathematical operation that returns a set (or product set or simply product) from multiple sets. That is, for sets A and B, the Cartesian product $A \times B$ is the set of all ordered pairs (a, b) where $a \in A$ and $b \in B$:



```
p = itertools.product('AB', (1,2))
list(p)

> <itertools.product at 0x10387b948>
> [('A', 1), ('A', 2), ('B', 1), ('B', 2)]
```