itertools — Functions creating iterators for efficient looping

**Infinite iterators:**

| **Iterator** | **Arguments** | **Results** | **Example** |
| --- | --- | --- | --- |
| count() | start, [step] | start, start+step, start+2\*step, … | count(10) --> 10 11 12 13 14... |
| cycle() | p | p0, p1, … plast, p0, p1, … | cycle('ABCD') --> A B C D AB C D ... |
| repeat() | elem [,n] | elem, elem, elem, … endlessly or up to n times | repeat(10, 3) --> 10 10 10 |

**Iterators terminating on the shortest input sequence:**

| **Iterator** | **Arguments** | **Results** | **Example** |
| --- | --- | --- | --- |
| accumulate() | p [,func] | p0, p0+p1, p0+p1+p2, … | accumulate([1,2,3,4,5]) --> 13 6 10 15 |
| chain() | p, q, … | p0, p1, … plast, q0, q1, … | chain('ABC', 'DEF') --> A B CD E F |
| chain.from\_iterable() | iterable | p0, p1, … plast, q0, q1, … | chain.from\_iterable(['ABC','DEF']) --> A B C D E F |
| compress() | data, selectors | (d[0] if s[0]), (d[1] if s[1]), … | compress('ABCDEF',[1,0,1,0,1,1]) --> A C E F |
| dropwhile() | pred, seq | seq[n], seq[n+1], starting when pred fails | dropwhile(lambda x: x<5,[1,4,6,4,1]) --> 6 4 1 |
| filterfalse() | pred, seq | elements of seq where pred(elem) is false | filterfalse(lambda x: x%2,range(10)) --> 0 2 4 6 8 |
| groupby() | iterable[, key] | sub-iterators grouped by value of key(v) |  |
| islice() | seq, [start,] stop [, step] | elements from seq[start:stop:step] | islice('ABCDEFG', 2, None) -->C D E F G |
| starmap() | func, seq | func(\*seq[0]), func(\*seq[1]), … | starmap(pow, [(2,5), (3,2),(10,3)]) --> 32 9 1000 |
| takewhile() | pred, seq | seq[0], seq[1], until pred fails | takewhile(lambda x: x<5,[1,4,6,4,1]) --> 1 4 |
| tee() | it, n | it1, it2, … itn splits one iterator into n |  |
| zip\_longest() | p, q, … | (p[0], q[0]), (p[1], q[1]), … | zip\_longest('ABCD', 'xy',fillvalue='-') --> Ax By C- D- |

**Combinatoric iterators:**

| **Iterator** | **Arguments** | **Results** |
| --- | --- | --- |
| product() | p, q, … [repeat=1] | cartesian product, equivalent to a nested for-loop |
| permutations() | p[, r] | r-length tuples, all possible orderings, no repeated elements |
| combinations() | p, r | r-length tuples, in sorted order, no repeated elements |
| combinations\_with\_replacement() | p, r | r-length tuples, in sorted order, with repeated elements |
| product('ABCD', repeat=2) |  | AA AB AC AD BA BB BC BD CA CB CC CD DADB DC DD |
| permutations('ABCD', 2) |  | AB AC AD BA BC BD CA CB CD DA DB DC |
| combinations('ABCD', 2) |  | AB AC AD BC BD CD |
| combinations\_with\_replacement('ABCD',2) |  | AA AB AC AD BB BC BD CC CD DD |