

Every tool here is lazy: it hands out one item at a time and is used up after one pass, so the examples wrap it in `list()` to show what it yields. They start with `from itertools import *`.

ENDLESS: ALWAYS CAP THEM

```
list(islice(count(10, 2), 3))
→ [10, 12, 14]
count(start, step) never stops

list(islice(cycle("ab"), 5))
→ ['a', 'b', 'a', 'b', 'a']
loops the input forever

list(repeat(0, 3))
→ [0, 0, 0]
no count: repeats forever

list(zip(count(1), "ab"))
→ [(1, 'a'), (2, 'b')]
zip stops at the shortest input
```

SLICE AND STOP

```
list(islice("abcdef", 3))
→ ['a', 'b', 'c']
the first 3, from any iterable

list(islice("abcdef", 2, None, 2))
→ ['c', 'e']
start, stop, step; no negatives

next(islice("abc", 1, None))
→ 'b'
the item at index 1

list(takewhile(str.isdigit, "12ab3"))
→ ['1', '2']
stops at the first False

list(dropwhile(str.isdigit, "12ab3"))
→ ['a', 'b', '3']
skips the start, keeps the rest

list(compress("abcd", [1, 0, 1, 0]))
→ ['a', 'c']
keep where the mask is true

list(filterfalse(str.isdigit, "a1b2"))
→ ['a', 'b']
the opposite of filter
```

JOIN AND FLATTEN

```
list(chain("ab", "cd"))
→ ['a', 'b', 'c', 'd']
one after another, no copy

list(chain.from_iterable([[1, 2], [3]]))
→ [1, 2, 3]
flatten one level

list(zip_longest("ab", "xyz",
fillvalue="-"))
→ [('a', 'x'), ('b', 'y'), ('-', 'z')]
pads the shorter one

list(zip("ab", "xyz", strict=True))
→ raises ValueError
strict=True: raise if lengths differ
```

PAIRS AND CHUNKS

```
list(pairwise([1, 4, 9]))
→ [(1, 4), (4, 9)]
neighbours, overlapping

[b - a for a, b in pairwise([1, 4, 9])]
→ [3, 5]
differences between neighbours

list(batched("abcde", 2))
→ [('a', 'b'), ('c', 'd'), ('e',)]
3.12+; the last one can be short

list(batched("abc", 2, strict=True))
→ raises ValueError
3.13+: raise if the last is short
```

COMBINATIONS

```
list(product("ab", "xy"))
→ [('a', 'x'), ('a', 'y'), ('b', 'x'), ('b', 'y')]
nested for loops in one call

list(product([0, 1], repeat=2))
→ [(0, 0), (0, 1), (1, 0), (1, 1)]
every on/off pattern

list(permutations("abc", 2)[:3])
→ [('a', 'b'), ('a', 'c'), ('b', 'a')]
order matters: 6 in all

list(combinations("abc", 2))
→ [('a', 'b'), ('a', 'c'), ('b', 'c')]
order doesn't matter

list(combinations_with_replacement("ab", 2))
→ [('a', 'a'), ('a', 'b'), ('b', 'b')]
an item can be picked again
```

RUNNING TOTALS

```
list(accumulate([1, 2, 3, 4]))
→ [1, 3, 6, 10]
running sum

list(accumulate([3, 1, 5, 2], max))
→ [3, 3, 5, 5]
running maximum

list(accumulate([1, 2, 3], mul))
→ [1, 2, 6]
running product; from operator import mul

list(accumulate([5, -2], initial=100))
→ [100, 105, 103]
start from a balance
```

GROUP RUNS

```
from itertools import groupby
words = ["fig", "kiwi", "plum", "pea"]

[(k, len(list(g))) for k, g in
groupby("aaabca")]
→ [('a', 3), ('b', 1), ('c', 1), ('a', 1)]
groups runs of equal items next to each other

[k for k, g in groupby(sorted("abab"))]
→ ['a', 'b']
sort first for one group per key

words.sort(key=len)
sort on the key you will group by

{n: list(g) for n, g in groupby(words,
len)}
→ {3: ['fig', 'pea'], 4: ['kiwi', 'plum']}
then group: a dict of lists
```

MAP, COPY, FOLD

```
list(starmap(pow, [(2, 3), (3, 2)]))
→ [8, 9]
each tuple becomes the arguments

a, b = tee(iter([1, 2, 3]))
next(b, None)
two independent copies of one iterator

reduce(mul, [2, 3, 4])
→ 24
fold into one value; functools.reduce

sorted(["a", 2], ("b", 1],
key=itemgetter(1))
→ [('b', 1), ('a', 2)]
operator.itemgetter: a fast key function
```

BUILT-IN PARTNERS

```
list(enumerate("ab", start=1))
→ [(1, 'a'), (2, 'b')]
a counter with each item

dict(zip("ab", [1, 2]))
→ {'a': 1, 'b': 2}
two lists into a dict

list(zip(*[(1, "a"), (2, "b")]))
→ [(1, 2), ('a', 'b')]
unzip: rows to columns

list(filter(None, [0, 1, "", "x"]))
→ [1, 'x']
None: keep the truthy items

sum(x * x for x in range(4))
→ 14
a generator: no list is built

any(n > 3 for n in [1, 5, 2])
→ True
stops at the first True

all(n > 0 for n in [])
→ True
empty input gives True

next((n for n in [1, 6, 8] if n > 5),
None)
→ 6
first match, or a default
```

RECIPES

```
list(dict.fromkeys("abcab"))
→ ['a', 'b', 'c']
drop duplicates, keep the order

list(zip(*[iter(range(6))] * 3))
→ [(0, 1, 2), (3, 4, 5)]
chunks before 3.12; drops leftovers

list(zip(s, s[1:], s[2:]))
→ [('a', 'b', 'c'), ('b', 'c', 'd')]
sliding window of 3
```

WATCH OUT

```
rows = map(str, nums)
list(rows)
list(rows)
the second list() is []: an iterator is used up

[k for k, g in groupby("abab")]
→ ['a', 'b', 'a', 'b']
no sort, so the same key comes back

groups = [g for k, g in groupby("aab")]
each g is empty once the loop moves on: list(g) at once

print(chain("ab"))
shows <itertools.chain object ...>: use list()

list(count())
never ends: islice() it or break

chain([1, 2])[0]
TypeError: no indexing; list() it or use islice

list(combinations([1, 1], 2))
→ [(1, 1)]
picks by position, so equal items still pair
```