



A comprehension is a for loop that builds a new collection: read `[n * 2 for n in nums if n > 0]` as "for each n in nums, if n > 0, keep n * 2". Examples use `nums = [3, -1, 4, 0, -2]`.

THE FOUR FORMS

```
nums = [3, -1, 4, 0, -2]

[n * n for n in nums] → [9, 1, 16, 0, 4]
list: square brackets

{n % 2 for n in nums} → {0, 1}
set: braces, duplicates dropped

{n: n * n for n in nums[:2]}
→ {3: 9, -1: 1}
dict: braces with key: value

sum(n * n for n in nums) → 30
generator: lazy, no list is built

tuple(n * 2 for n in nums[:3])
→ (6, -2, 8)
no tuple form: pass a generator to tuple()
```

IT IS A LOOP, IN ONE LINE

```
nums = [3, -1, 4, 0, -2]

result = []
for n in nums:
    if n > 0:
        result.append(n * 2)
the same as [n * 2 for n in nums if n > 0]

[n * 2 for n in nums if n > 0] → [6, 8]
for and if read left to right, like the loop
```

FILTER: IF AT THE END

```
nums = [3, -1, 4, 0, -2]

[n for n in nums if n > 0] → [3, 4]
keep only the items that pass

[n for n in nums if n % 2 == 0]
→ [4, 0, -2]

[n for n in nums if n > 0 if n % 2] → [3]
two ifs: both must pass, like and

[y for n in nums if (y := n * n) > 5]
→ [9, 16]
:= names a value so you filter and keep it
```

TRANSFORM: IF/ELSE BEFORE FOR

```
nums = [3, -1, 4, 0, -2]

[n if n > 0 else 0 for n in nums]
→ [3, 0, 4, 0, 0]
every item comes out, changed or not

["odd" if n % 2 else "even" for n in
nums]
→ ['odd', 'odd', 'even', 'even', 'even']
a label for each item

[abs(n) for n in nums if n]
→ [3, 1, 4, 2]
transform the ones that pass the filter
```

TEXT

```
words = ["Apple", "", "fig", "Kiwi "]

[w.strip().lower() for w in words if w]
→ ['apple', 'fig', 'kiwi']
tidy a list; "" is falsy, so it's skipped

[len(w) for w in words] → [5, 0, 3, 5]

[int(x) for x in "3,1,2".split(",")]
→ [3, 1, 2]
numbers from text

[r.split(",") for r in ["a,1", "b,2"]]
→ [['a', '1'], ['b', '2']]
split each line into fields

[c for c in "a1b2" if c.isdigit()]
→ ['1', '2']
a string loops one character at a time

[i for i, w in enumerate(words) if w]
→ [0, 2, 3]
positions of the ones you want
```

NESTED

```
grid = [[1, 2, 3], [4, 5, 6]]

[x for row in grid for x in row]
→ [1, 2, 3, 4, 5, 6]
flatten: the for clauses nest left to right

[x for row in grid for x in row if x % 2]
→ [1, 3, 5]
flatten and filter

[[x * 10 for x in row] for row in grid]
→ [[10, 20, 30], [40, 50, 60]]
one inside another keeps the rows

[list(col) for col in zip(*grid)]
→ [[1, 4], [2, 5], [3, 6]]
rows become columns

[(a, b) for a in "xy" for b in (1, 2)]
→ [('x', 1), ('x', 2), ('y', 1), ('y', 2)]
every pair

[[0] * 3 for _ in range(2)]
→ [[0, 0, 0], [0, 0, 0]]
a grid of separate rows; _ means unused
```

DICT AND SET

```
menu = {"tea": 3, "cake": 5, "pie": 4}

{k: v * 2 for k, v in menu.items()}
→ {'tea': 6, 'cake': 10, 'pie': 8}
change every value

{k: v for k, v in menu.items() if v > 3}
→ {'cake': 5, 'pie': 4}
keep some pairs

{v: k for k, v in menu.items()}
→ {3: 'tea', 5: 'cake', 4: 'pie'}
swap keys and values

{name: len(name) for name in menu}
→ {'tea': 3, 'cake': 4, 'pie': 3}
looping a dict gives its keys

{k: v for k, v in zip("ab", [1, 2])}
→ {'a': 1, 'b': 2}
or simply dict(zip(...))

sorted({len(name) for name in menu})
→ [3, 4]
a set keeps distinct values; sorted() orders them
```

GENERATORS FEEDING A FUNCTION

```
words = ["fig", "kiwi", "lemon"]

max(len(w) for w in words) → 5
no extra brackets as the only argument

any(w.startswith("k") for w in words)
→ True
stops at the first True

all(w.islower() for w in words) → True

sum(1 for w in words if len(w) > 3) → 2
count matches without building a list

next(w for w in words if "m" in w)
→ 'lemon'
the first match; next(gen, None) if none

max((len(w) for w in [1]), default=0) → 0
default when nothing comes out
```

HANDY PATTERNS

```
s = "abcde"

[a + b for a, b in zip([1, 2], [10, 20])]
→ [11, 22]
walk two lists side by side

[s[i:i + 2] for i in range(0, len(s), 2)]
→ ['ab', 'cd', 'e']
cut into chunks of 2

list(dict.fromkeys([3, 1, 3, 2]))
→ [3, 1, 2]
remove duplicates, keep the order

[str(n) for n in range(3)]
→ ['0', '1', '2']
convert every item
```

WATCH OUT

```
nums = [3, -1, 4, 0, -2]

[n for n in nums if n > 0 else 0]
SyntaxError: with else, the if goes before for

(n * 2 for n in nums)
a generator, not a tuple: use tuple(...)

g = (n for n in nums[:2])
list(g), list(g)
→ ([3, -1], [])
a generator is used up after one pass

g = [[0] * 2] * 2
g[0][0] = 1
g
→ [[1, 0], [1, 0]]
the same row twice: build rows in a comprehension

[k for k in range(3)]
k
→ raises NameError
the loop variable stays inside the brackets

[print(n) for n in nums]
builds a useless list of None: use a for loop

[x for a in b for c in a if c for x in c
if x]
too clever to read: write the loop out
```