

`seq[start:stop:step]` makes a new sequence from a run of items: start is included, stop is excluded, and any of the three can be left out. Examples use `xs = [0, 1, 2, 3, 4, 5]` and `s = "python"`.

START AND STOP

`xs = [0, 1, 2, 3, 4, 5]`

`xs[1:4]` → `[1, 2, 3]`
start included, stop excluded

`xs[:3]` → `[0, 1, 2]`
the first 3

`xs[3:]` → `[3, 4, 5]`
from index 3 to the end

`xs[:]` → `[0, 1, 2, 3, 4, 5]`
a copy of the whole list

`xs[:2] + xs[2:]` → `[0, 1, 2, 3, 4, 5]`
split at any index, nothing is lost

COUNT FROM THE END

`xs = [0, 1, 2, 3, 4, 5]`

`xs[-1]` → 5
an index, not a slice: the last item

`xs[-2:]` → `[4, 5]`
the last 2

`xs[:-2]` → `[0, 1, 2, 3]`
all but the last 2

`xs[1:-1]` → `[1, 2, 3, 4]`
drop the first and the last

`xs[-4:-1]` → `[2, 3, 4]`
-1 as stop: up to, not including, the last

STEP

`xs = [0, 1, 2, 3, 4, 5]`

`xs[::2]` → `[0, 2, 4]`
every second item

`xs[1:2]` → `[1, 3, 5]`
every second, starting at index 1

`xs[1:5:2]` → `[1, 3]`
all three parts

`xs[::3]` → `[0, 3]`

BACKWARDS: A NEGATIVE STEP

`xs = [0, 1, 2, 3, 4, 5]`

`xs[::-1]` → `[5, 4, 3, 2, 1, 0]`
a reversed copy

`xs[::-2]` → `[5, 3, 1]`

`xs[4:1:-1]` → `[4, 3, 2]`
start is now the bigger index

`xs[3::-1]` → `[3, 2, 1, 0]`
from index 3 back to the start

`xs[:2:-1]` → `[5, 4, 3]`
from the end back to, not including, 2

STRINGS

`s = "python"`

`s[:2]` → `'py'`

`s[-3:]` → `'hon'`

`s[::-1]` → `'nohtyp'`
reversed

`"level"[::-1] == "level"` → `True`
a palindrome check

`s[0].upper() + s[1:]` → `'Python'`
strings can't change: build a new one

`"report.csv"[:-4]` → `'report'`
drop a known ending; removesuffix is clearer

`[s[i:i + 2] for i in range(0, len(s), 2)]`
→ `['py', 'th', 'on']`
cut into chunks of 2

OTHER SEQUENCES

from `itertools` import `islice`
`squares = (n * n for n in range(9))`

`(1, 2, 3)[1:]` → `(2, 3)`
a tuple slice is a tuple

`range(10)[2:8:2]` → `range(2, 8, 2)`
a range slice is a range: no list built

`list(islice(squares, 3))` → `[0, 1, 4]`
generators can't be sliced: use `islice`

CHANGE A LIST THROUGH A SLICE

Lists only. Each row carries on from the one above.

`xs = [0, 1, 2, 3, 4, 5]`

`xs[1:3] = [7, 8, 9]`
`xs`
→ `[0, 7, 8, 9, 3, 4, 5]`
replace a run; the length can change

`xs[1:4] = []`
`xs`
→ `[0, 3, 4, 5]`
assign `[]` to remove a run

`xs[:0] = [9]`
`xs`
→ `[9, 0, 3, 4, 5]`
an empty slice inserts

`del xs[:2]`
`xs`
→ `[0, 4]`
`del` works on slices too

COPIES

numpy is the exception: an array slice is a view, so call `.copy()`.

`xs = [0, 1, 2, 3, 4, 5]`

`ys = xs[:]`
`ys[0] = 99`
`xs`
→ `[0, 1, 2, 3, 4, 5]`
a slice is a new list: `xs` is untouched

`ys = xs`
`ys[0] = 99`
`xs`
→ `[99, 1, 2, 3, 4, 5]`
no slice: two names for one list

`grid = [[1], [2]]`
`cp = grid[:]`
`cp[0].append(9)`
`grid`
→ `[[1, 9], [2]]`
shallow: the inner lists are shared

PAST THE END

`xs = [0, 1, 2, 3, 4, 5]`

`xs[10]` → **raises `IndexError`**
an index past the end raises

`xs[4:99]` → `[4, 5]`
a slice is clipped to fit

`xs[10:20]` → `[]`
nothing there: empty, no error

`[][:1]` → `[]`
safe on an empty list, unlike `[] [0]`

HANDY PATTERNS

`xs = [0, 1, 2, 3, 4, 5]`
`k = 2`

`xs[k:] + xs[:k]` → `[2, 3, 4, 5, 0, 1]`
rotate left by `k`

`xs[-k:] + xs[:-k]` → `[4, 5, 0, 1, 2, 3]`
rotate right by `k`

`list(zip(xs, xs[1:]))[:3]`
→ `[(0, 1), (1, 2), (2, 3)]`
each item with the next one

WATCH OUT

`xs = [0, 1, 2, 3, 4, 5]`
`s = "python"`

`s[0] = "P"`
TypeError: strings can't change; use `"P" + s[1:]`

`xs[4:1]` → `[]`
start after stop is empty: add a step of -1

`xs[-1:]` → `[5]`
a slice gives a list; `xs[-1]` gives the item

`xs[: -0]` → `[]`
-0 is 0, so `xs[: -k]` is empty when `k` is 0

`xs[::2] = [0]`
`xs`
→ **raises `ValueError`**
a stepped slice needs exactly as many items

`{"a": 1}.keys()[:1]` → **raises `TypeError`**
dicts don't slice: `list(d)[:1]`