# Understanding slice notation



2640

I need a good explanation (references are a plus) on Python's slice notation.

To me, this notation needs a bit of picking up.

It looks extremely powerful, but I haven't quite got my head around it.

1258

python list slice

edited Jan 15 at 23:36

martineau
**69.8k** 10 92 186

asked Feb 3 '09 at 22:31

Simon
**32.5k** 23 78 114

## 31 Answers

1 2 next

3612

It's pretty simple really:

```
a[start:stop]  # items start through stop-1
a[start:]      # items start through the rest of the arra
a[:stop]       # items from the beginning through stop-1
```

There is also the `step` value, which can be used with any of the above:

```
a[start:stop:step] # start through not past stop, by step
```

The key point to remember is that the `:stop` value represents the first value that is *not* in the selected slice. So, the difference between `stop` and `start` is the number of elements selected (if `step` is 1, the default).

The other feature is that `start` or `stop` may be a *negative* number, which means it counts from the end of the array instead of the beginning. So:

```
a[-1]    # last item in the array
a[-2:]   # last two items in the array
a[:-2]   # everything except the last two items
```

Similarly, `step` may be a negative number:

```
a[::-1]    # all items in the array, reversed
a[1::-1]   # the first two items, reversed
a[:-3:-1]  # the last two items, reversed
a[-3::-1]  # everything except the last two items, reversed
```

Python is kind to the programmer if there are fewer items than you ask for. For example, if you ask for `a[:-2]` and `a` only contains one element, you get an empty list instead of an error. Sometimes you would prefer the error, so you have to be aware that this may happen.

### Relation to `slice()` object

The slicing operator `[]` is actually being used in the above code with a `slice()` object using the `:` notation (which is only valid within `[]`), i.e.:

is equivalent to:

```
a[slice(start, stop, step)]
```

Slice objects also behave slightly differently depending on the number of arguments, similarly to `range()`, i.e. both `slice(stop)` and `slice(start, stop[, step])` are supported. To skip specifying a given argument, one might use `None`, so that e.g. `a[start:]` is equivalent to `a[slice(start, None)]` or `a[::-1]` is equivalent to `a[slice(None, None, -1)]`.

While the `:`-based notation is very helpful for simple slicing, the explicit use of `slice()` objects simplifies the programmatic generation of slicing.

edited Feb 24 at 19:26

answered Feb 3 '09 at 22:48

Greg Hewgill
**679k** 147 1019 1172

74 Slicing builtin types returns a copy but that's not universal. Notably, slicing NumPy arrays returns a view that shares memory with the original. – Beni Cherniavsky-Paskin Sep 23 '13 at 0:13

1 @RodriKing It means that your start and end are empty, and your step is -2. So you reverse (beause it's negative) and take by 2 elements, for the whole list because start and empty are not defined. – mbh86 Aug 9 '18 at 9:54

2 Another example: `a = list(range(100)) # [0,1,2, .... ,99]` and `a[20::-3]` It means that you will go reverse 3 by 3. You start from position 20 and go 3 backwards ... 20, 17,

of the step has priority to the list ends. I was expecting to get `[99, 96, 93, ..., 24, 21]` . – Ciprian Tomoiagă Aug 31 '18 at 10:39

3 @nodakai: Strings are immutable; lists are not. `b[:]` must be a different object from `b` , but `a[:]` and `a` can be the same. – Greg Hewgill Sep 13 '18 at 3:46

461

The Python tutorial talks about it (scroll down a bit until you get to the part about slicing).

The ASCII art diagram is helpful too for remembering how slices work:

```
 +---+---+---+---+---+---+
 | P | y | t | h | o | n |
 +---+---+---+---+---+---+
 0   1   2   3   4   5   6
-6  -5  -4  -3  -2  -1
```

> One way to remember how slices work is to think of the indices as pointing *between* characters, with the left edge of the first character numbered 0. Then the right edge of the last character of a string of *n* characters has index *n*.

edited Sep 18 '17 at 11:02

kenorb
**70.6k** 30 412 418

answered Feb 3 '09 at 22:49

Hans Nowak
**5,085** 1 15 13