So you've got an list, tuple or array and you want to get specific sets of sub-elements from it, without any long, drawn out for loops?

Python has an amazing feature just for that called *slicing*. Slicing can not only be used for lists, tuples or arrays, but custom data structures as well, with the *slice* object, which will be used later on in this article.

**Slicing Python Lists/Arrays and Tuples Syntax**

Let's start with a normal, everyday list.

|  |  |
| --- | --- |
| 1 | >>> a = [1, 2, 3, 4, 5, 6, 7, 8] |

Nothing crazy, just a normal list with the numbers 1 through 8. Now let's say that we really want the sub-elements 2, 3, and 4 returned in a new list. How do we do that?

NOT with a for loop, that's how. Here's the Pythonic way of doing things:

|  |  |
| --- | --- |
| 1  2 | >>> a[1:4]  [2, 3, 4] |

This returns exactly what we want. What the heck does that syntax mean? Good question.

Let me explain it. The 1 means to start at second element in the list (note that the slicing index starts at 0). The 4 means to end at the fifth element in the list, but not include it. The colon in the middle is how Python's lists recognize that we want to use slicing to get objects in the list.

**Advanced Python Slicing (Lists, Tuples and Arrays) Increments**

There is also an optional second clause that we can add that allows us to set how the list's index will increment between the indexes that we've set.

In the example above, say that we did not want that ugly 3 returned and we only want nice, even numbers in our list. Easy peasy.

|  |  |
| --- | --- |
| 1  2 | >>> a[1:4:2]  [2, 4] |

See, it's as simple as that. That last colon tells Python that we'd like to choose our slicing increment. By default, Python sets this increment to 1, but that extra colon at the end of the numbers allows us to specify what we want it to be.

**Python Slicing (Lists, Tuples and Arrays) in Reverse**

Alright, how about if we wanted our list to be backwards, just for the fun of it? Let's try something crazy.

|  |  |
| --- | --- |
| 1  2 | >>> a[::-1]  [8, 7, 6, 5, 4, 3, 2, 1] |

What?! What the heck is that?

Lists have a default bit of functionality when slicing. If there is no value before the first colon, it means to start at the beginning index of the list. If there isn't a value after the first colon, it means to go all the way to the end of the list. This saves us time so that we don't have to manually specify len(a) as the ending index.

Okay. And that -1 I snuck in at the end? It means to increment the index every time by -1, meaning it will traverse the list by going backwards. If you wanted the even indexes going backwards, you could skip every second element and set the iteration to -2. Simple.

**Slicing Python Tuples**

Everything above also works for tuples. The exact same syntax and the exact same way of doing things. I'll give an example just to illustrate that it's indeed the same.

|  |  |
| --- | --- |
| 1  2  3 | >>> t = (2, 5, 7, 9, 10, 11, 12)  >>> t[2:4]  (7, 9) |

And that's it. Simple, elegant, and powerful.

**Using Python Slice Objects**

There is another form of syntax you can use that may be easier to understand. There are objects in Python called slice objects and the can be used in place of the colon syntax above.

Here's an example:

|  |  |
| --- | --- |
| 1  2  3  4 | >>> a = [1, 2, 3, 4, 5]  >>> sliceObj = slice(1, 3)  >>> a[sliceObj]  [2, 3] |

The slice object initialization takes in 3 arguments with the last one being the optional index increment. The general method format is: slice(start, stop, increment), and it is analogous to start:stop:increment when applied to a list or tuple.

That's it!

# [Difference between append vs. extend list methods in Python](https://stackoverflow.com/questions/252703/difference-between-append-vs-extend-list-methods-in-python)

What's the difference between the list methods append() and extend()?





