PYTHON SNIPPETS

Python Itertools Part 2 - Combinations & Permutations

Phil Best
2 min read · Jun 17

In our last snippet post we a quick look at the `product` function found in the `itertools` module. Today we're going to look at a few more combinatoric iterators from the `itertools` module: `permutations`, `combinations`, and `combinations_with_replacement`.

First, let's look at `permutations`. `permutations` is concerned with finding all of the possible orderings for a given collection of items. For example, if we have the string `"ABC"`, permutations will find all of the ways we can reorder the letters in this string, so that each order is unique.

```
from itertools import permutations

p_1 = permutations("ABC")
# ('A', 'B', 'C') ('A', 'C', 'B') ('B', 'A', 'C') ('B', 'C', 'A')
# ('C', 'A', 'B') ('C', 'B', 'A')
```

By default, `permutations` returns different orderings for the entire collection, but we can use the optional `r` parameter to limit the function to finding shorter permutations.

```
p_2 = permutations("ABC", r=2)
# ('A', 'B') ('A', 'C') ('B', 'A') ('B', 'C') ('C', 'A') ('C', 'B')
```

Providing an `r` value greater than the length of the collection passed into `permutations` will yield an empty permutations object.

Now let's take a look at `combinations`. `combinations` returns an iterable object containing unique combinations of elements from a provided collection. Note that `combinations` isn't concerned with the order of elements, so `combinations` will treat `('A', 'B')` as being identical to `('B', 'A')` in its results.

The length of the resulting combinations is controlled by the `r` parameter once again, but in the case of `combinations`, this argument is mandatory.

```
from itertools import combinations

c_1 = combinations("ABC", r=2)
# ('A', 'B') ('A', 'C') ('B', 'C')

c_2 = combinations("ABC", r=3)
# ('A', 'B', 'C')
```

It is possible to get duplicate elements back from `combinations`, but

only if the provided iterable contains multiple instances of a given element. `(1, 2, 3, 1)`, for example.

```
c_3 = combinations((1, 2, 3, 1), r=2)
# (1, 2) (1, 3) (1, 1) (2, 3) (2, 1) (3, 1)
```

In this case `(1, 2)` and `(2, 1)` are not simply the same elements in a different order, the 1s are in fact different elements in the original collection.

It's possible to include instances where an item is paired with itself using the `combinations_with_replacement` function. It works just like `combinations`, but will also match every element to itself.

```
from itertools import combinations, combinations_with_replacement

c_4 = combinations((1, 2, 3), r=2)
# (1, 2) (1, 3) (2, 3)

c_5 = combinations_with_replacement((1, 2, 3), r=2)
# (1, 1) (1, 2) (1, 3) (2, 2) (2, 3) (3, 3)
```

That wraps up the combinatoric iterators! I hope you learnt something new, and be sure to check out the official documentation for more details.

If you're looking to upgrade your Python skills even further, we'd love to have you on our Complete Python Course. You can follow the link in this post to get the course for just $9.99, and there's also a 30 day

money back guarantee.

We'll be back next Monday with another snippet post, this time covering a very interesting type of collection. Follow us on Twitter, or sign up to our mailing list below so you don't miss out!

Phil Best
I'm a freelance developer, mostly working in web development. I also create course content for Teclado!

Read More

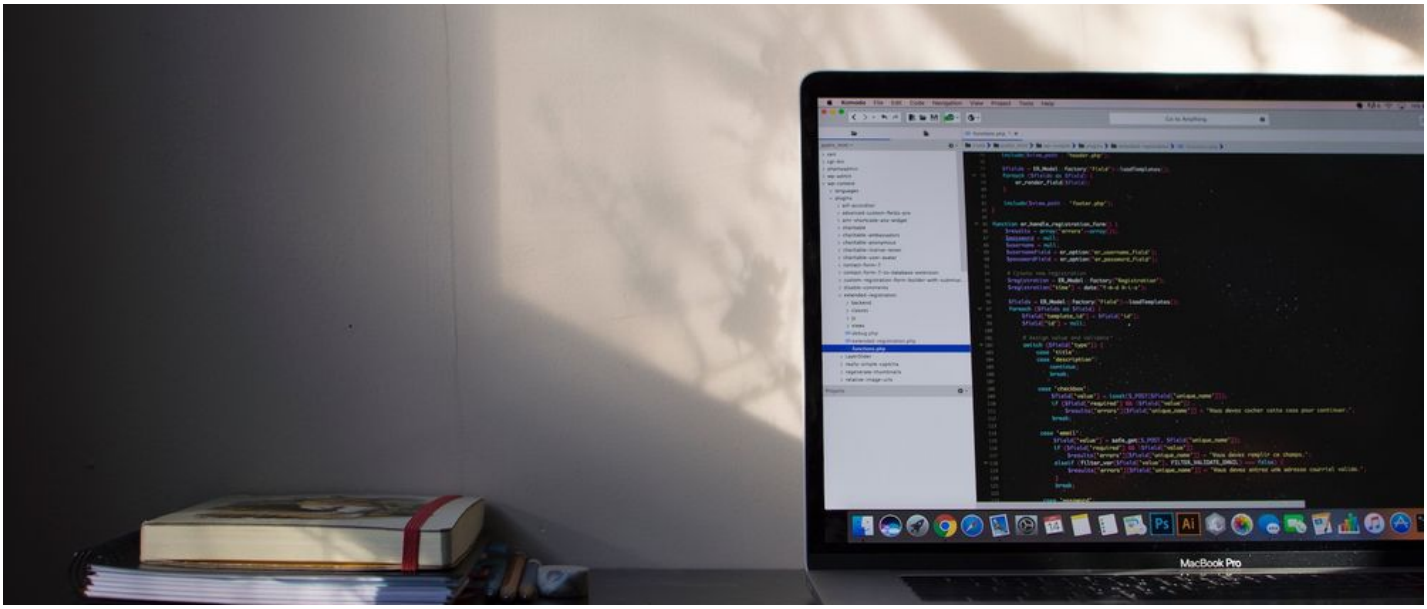
Python Snippets

Assignment expressions in Python →

Python's namedtuples →

Python's divmod Function →

→ See all 26 posts



FLASK

Simple JWT Authentication with Flask-JWT

Adding authentication in Flask applications can be really straightforward. Learn more about token-based authentication with Flask-JWT in this post.

Jose Salvatierra
5 min read · Jun 20