The Combinatoric Generators

The itertools library contains four iterators that can be used for creating combinations and permutations of data. We will be covering these fun iterators in this section.

combinations(iterable, r)

If you have the need to create combinations, Python has you covered with **itertools.combinations**. What combinations allows you to do is create an iterator from an iterable that is some length long. Let’s take a look:

```
from itertools import combinations
print (list(combinations('WXYZ', 2)))
#[('W', 'X'), ('W', 'Y'), ('W', 'Z'), ('X', 'Y'), ('X', 'Z'), ('Y', 'Z')]
```

When you run this, you will notice that combinations returns tuples. To make this output a bit more readable, let’s loop over our iterator and join the tuples into a single string:

```
from itertools import combinations
for item in combinations('WXYZ', 2):
    print(''.join(item))

#WX
#WY
#WZ
#XY
#XZ
#YZ
```

Now it’s a little easier to see all the various combinations. Note that the combinations function does its combination in lexicographic sort order, so if you the iterable is sorted, then your combination tuples will also be sorted. Also worth noting is that combinations will not produce repeat values in the combinations if all the input elements are unique.

combinations_with_replacement(iterable, r)

The **combinations_with_replacement** with iterator is very similar to **combinations**. The only difference is that it will actually create combinations where elements do repeat. Let's try an example from the previous section to illustrate:

```
from itertools import combinations_with_replacement
for item in combinations_with_replacement('WXYZ', 2):
    print(''.join(item))

#WW
#WX
#WY
#WZ
#XX
#XY
#XZ
#YY
#YZ
#ZZ
```

As you can see, we now have four new items in our output: WW, XX, YY and ZZ.

product(*iterables, repeat=1)

The itertools package has a neat little function for creating Cartesian products from a series of input iterables. Yes, that function is **product**. Let's see how it works!

```
from itertools import product
arrays = [(-1,1), (-3,3), (-5,5)]
cp = list(product(*arrays))
print (cp)
#[(-1, -3, -5),
# (-1, -3, 5),
# (-1, 3, -5),
# (-1, 3, 5),
# (1, -3, -5),
# (1, -3, 5),
# (1, 3, -5),
# (1, 3, 5)]
```

Here we import product and then set up a list of tuples which we assign to the variable **arrays**. Next we call product with those arrays. You will notice that we call it using ***arrays**. This will cause the list to be "exploded" or applied to the product function in sequence. It means that you are passing in 3 arguments instead of one. If you want, try calling it with the asterisk pre-pended to arrays and see what happens.

permutations

The **permutations** sub-module of itertools will return successive *r* length permutations of elements from the iterable you give it. Much like the combinations function, permutations are emitted in lexicographic sort order. Let's take a look:

```
from itertools import permutations
for item in permutations('WXYZ', 2):
    print(''.join(item))

#WX
#WY
#WZ
#XW
#XY
#XZ
#YW
#YX
#YZ
#ZW
#ZX
#ZY
```

You will notice that the output it quite a bit longer than the output from combinations. When you use permutations, it will go through all the permutatations of the string, but it won't do repeat values if the input elements are unique.

Wrapping Up

The itertools is a very versatile set of tools for creating iterators. You can use them to create your own iterators all by themselves or in combination with each other. The Python documentation has a lot of great examples that you can study to give you ideas of what can be done with this valuable library.