# Expanding, Unpacking, or ... Splatting?

*Posted by Scot Clausing on March 13, 2013*

Whatever you call it (http://stackoverflow.com/questions/2322355/proper-name-for-python-operator), a few wonderfully useful features in Python are

- the ability to bind the items in an Iterable to local names: `a, b = 1, 2`
- or pass them as positional args to a function: `foo(*args)`,
- and similarly, to pass Dictionary items as keyword args: `foo(**kwargs)`.

And, with a similar syntax (`*args` and `**kwargs`), functions may be defined to accept an unknown number of positional or keword arguments!

## Unpacking into local names with `=`

Unpack any fixed length iterable into the same number of names:

```
>>> a, b, c = (1, 2, 3)
>>> a, b, c
(1, 2, 3)

>>> d, e, f = [1, 2, 3]
>>> d, e, f
(1, 2, 3)

>>> g, h, i = '123'
>>> g, h, i
('1', '2', '3')
```

But, what happens with a dictionary since key iteration is not predictably ordered?

```
>>> j, k, l = {'j': 1, 'k': 2, 'l': 3}
>>> j, k, l
('k', 'j', 'l')
```

That's awesome! And, remember that generators are iterables (http://pynash.org/2013/02/27/comprehensions-and-generators.html), too:

```
>>> def gen():
...     for x in range(3):
...         yield x

>>> m, n, o = gen()
>>> m, n, o
(0, 1, 2)
```

But don't forget that the number of names must equal the number of items being expanded:

```
>>> blow, up = 3, 2, 1
Traceback (most recent call last):
    ...
ValueError: too many values to unpack

>>> blow, up = 3,
Traceback (most recent call last):
    ...
ValueError: need more than 1 value to unpack
```

- This restriction goes away in Python 3 with the awesome PEP-3132: Extended Iterable Unpacking. (http://www.python.org/dev/peps/pep-3132/)

## Unpacking with function arguments with `*args` and `**kwargs`

First, we'll need a function to play with and some values to unpack into it.

```
>>> def foo(a, b, c): return c, b, a

>>> tuple123 = (1, 2, 3)
>>> list123 = [1, 2, 3]
>>> str123 = '123'
>>> dict123 = {'a': 1, 'b': 2, 'c': 3}
```

Excellent; let's unpack! Everything from the previous section still applies, except that instead of the iterable following `=`, it follows the `*` operator *(splat!)*.

```
>>> foo(*tuple123)
(3, 2, 1)
>>> foo(*list123)
(3, 2, 1)
>>> foo(*str123)
('3', '2', '1')
>>> foo(*dict123)
('b', 'c', 'a')

>>> # and unpacking generators? of course.
>>> foo(*gen())
(2, 1, 0)
```

Let's revisit `foo(*dict123)` ... what if we wanted to apply the dictionary items as keyword arguments?

```
>>> foo(**dict123)
(3, 2, 1)
```

Nice! And, since we're passing values as keyword arguments, it doesn't matter that the iteration order is unpredictable.

## Defining functions with `*args` and `**kwargs`

Now let's go the opposite direction. Instead of expanding arguments when calling functions, let's define functions that collect positional arguments into an iterable and keyword arguments into a dictionary!

```
>>> def bar(*args): return args
>>> def baz(**kwargs): return kwargs

>>> bar(1, 2, 3)
(1, 2, 3)

>>> baz(a=1, b=2, c=3)
{'a': 1, 'c': 3, 'b': 2}
```

That's enough for now. Check out the official Python tutorial for more on defining functions (http://docs.python.org/2/tutorial/controlflow.html#more-on-defining-functions).

## Swapping Values

A common use for unpacking is the ability to neatly swap values between names, or within in a dictionary or list, without a temporary variable:

```
>>> a, b = 1, 2
>>> a, b
(1, 2)

# now swap their values
>>> a, b = b, a
>>> a, b
(2, 1)

# swap positions in a list:
>>> list123 = [1, 2, 3]
>>> list123[0], list123[2] = list123[2], list123[0]
>>> list123
[3, 2, 1]

# swap keys in a dictionary
>>> dict123 = {'a': 1, 'b': 2, 'c': 3}
>>> dict123['a'], dict123['c'] = dict123['c'], dict123['a']
>>> dict123
{'a': 3, 'c': 1, 'b': 2}
```

And that's all I've got for this week. With a little luck, next week I'll talk about slicing ... but PyCon 2013 is right around the corner and it might inspire a completely different topic!

**Related Posts**

- 27 Oct 2016 » October 2016 PyNash - Beck Cronin-Dixon on scikit-learn (/2016/10/27/Beck-on-Sciki-Learn/)
- 23 Jun 2016 » June 2016 PyNash - Scott Burns on Django Channels (/2016/06/23/Scott-Burns-on-Django-Channels/)
- 08 Jul 2014 » PyNash Coffee and Code Environments (/2014/07/08/PyNash-CNC-Environments/)

← PREVIOUS POST (/2013/03/07/LOGGING-INTRO/)

NEXT POST → (/2013/03/18/PYPI-PACKAGE-STATS/)

(/rss.xml) (https://twitter.com/pynash) (https://github.com/pynashorg)