Arguments and parameters

A lot of people—not just novices—mix up parameters and arguments, especially when it comes to things like how default-valued parameters and keyword arguments, or argument unpacking and variable parameters.

The FAQ has a section called What is the difference between arguments and parameters [http://docs.python.org/3/faq/programming.html#faq-argument-vs-parameter] , which explains the basics:

> *Parameters* are defined by the names that appear in a function definition, whereas *arguments* are the values actually passed to a function when calling it. Parameters define what types of arguments a function can accept.

PEP 3102 [http://www.python.org/dev/peps/pep-3102/] explains the current design. The glossary entries for argument [http://docs.python.org/3/glossary.html#term-argument] and parameter [http://docs.python.org/3/glossary.html#term-parameter] are also helpful. And the tutorial also has nice sections on Defining Functions [http://docs.python.org/3/tutorial/controlflow.html#defining-functions] , which explain things in loose terms. However, to actually understand how it works, you need to read the reference documentation on Calls [http://docs.python.org/3/reference/expressions.html#calls] and Function definitions [http://docs.python.org/3/reference/compound_stmts.html#function] , and some of the terminology is only defined in the middle of the inspect module documentation [http://docs.python.org/3.3/library/inspect.html#inspect.Parameter] .

However, putting it all together is a little difficult.

The explanations below are all for Python 3.x, but there is a brief overview of the differences between 3.x and 2.x at the end.

PEP 3102 uses different terminology from the reference documentation and the inspect module; I've chosen to go with the latter.

Parameters

Parameters are part of a function definition (def or lambda). There are five different kinds of parameters:

- positional-or-keyword: Normal parameters in a function definition, with or without default values.
  - Each has a name and an index, and can accept a positional argument with the same index, or a keyword argument with the same name, or (if it has a default value) nothing.
  - Technically, every parameter before the first bare *, var-positional, or var-keyword is a positional-or-keyword parameter.
- positional-only: Only found in builtin/extension functions.

    - Each has a name and an index, but only accepts positional arguments with the same index.
- var-positional: This is the *args.
    - This accepts a sequence consisting of all positional arguments whose index is larger than any positional-or-keyword or positional-only parameter. (Note that you can also specify a bare * here. In that case, you don't take variable positional arguments. You do this to set off keyword-only from positional-or-keyword parameters.)
- keyword-only: These are parameters that come after a * or *args, with or without default values.
    - Each has a name only, and accepts only keyword arguments with the same name.
    - Technically, every parameter after the first bare * or var-positional, but before the var-keyword (if any), is a keyword-only parameter.
- var-keyword: This is the **args.
    - This accepts a mapping consisting of all keyword arguments whose name does not match any positional-or-keyword or keyword-only parameter.

Arguments

Arguments are part of a function call. There are four different kinds of arguments:

- positional: Arguments without a name.
    - Each is matched to the positional-or-keyword or positional-only parameter with the same index, or to the var-positional parameter if there is no matching index (or, if there is no var-positional parameter, it's an error if there is no match).
- keyword: Arguments with a name.
    - Each is matched to the postional-or-keyword or keyword-only parameter with the same name, or to the var-keyword parameter if there is no matching name (or, if there is no var-keyword parameter, it's an error if there is no match).
- packed positional: An iterator preceded by *.
    - The iterator is unpacked, and the values treated as separate positional arguments.
- packed keyword: A mapping preceded by **.
    - The mapping is iterated, and the key-value pairs treated as separate keyword arguments.

There is no direct connection between parameters with a default value and keyword arguments. You can pass keyword arguments to parameters without default values, or positional arguments to parameters with them.

Similarly, there is no direct connection between a *args in a function call and a *args in a function definition. A packed positional argument with three values might be unpacked into the last three positional-or-keyword parameters, or all three may be extended to the last two normal positional arguments and passed together to the

var-positional, or its first two values may be unpacked into positional-or-keyword parameters and the last into the var-positional.

Examples

In the following example:

```
>>> def func(spam, eggs=1, *args, foo, bar=2, **kwargs):
...     print(spam, eggs, args, foo, bar, kwargs)
>>> func(1, 2, 3, 4, foo=5, bar=6, baz=7)
1 2 (3, 4) 5 6 {'baz': 7}
>>> func(baz=1, bar=2, foo=3, eggs=4, spam=5)
5 4 () 3 2 {'baz': 1}
>>> func(foo=1, spam=2)
2 1 () 1 2 {}
```

spam is a positional-or-keyword parameter with no default value. This means any call must pass either at least one positional argument (as in the first call), or a keyword argument with name "spam" (as in the other two).

eggs is a positional-or-keyword parameter with a default value. This means a call may pass either at least two positional arguments (as in the first call), or a keyword argument with name "eggs" (as in the second), or neither (as in the third).

args is a var-positional parameter. Because there is such a parameter, it's legal to pass more than two positional arguments, in which case it will get all of the excess (as in the first call), but of course it's not required (as in the other two).

foo is a keyword-only parameter with no default value. This means any call must pass a keyword argument with the name "foo" (as in all three calls).

bar is a keyword-only argument with a default value. This means a call may pass a keyword argument with the name "bar" (as in the first two calls), or not (as in the third).

kwargs is a var-keyword parameter. Because there is such a parameter, it's legal to pass keyword arguments whose names don't match any parameter (as in the first two calls), but of course it's not required (as in the third).

With the same function from above:

```
>>> a = [2,3,4]
>>> d = {'foo': 5, 'baz': 6}
>>> func(1, *a, **d)
1 2 (3, 4) 5 2 {'baz': 6}
>>> func(1, 2, 3, 4, foo=5, baz=6)
1 2 (3, 4) 5 2 {'baz': 6}
```

In the first call, the a iterable is unpacked into three positional arguments, and the **d is unpacked into keyword arguments named "foo" and "baz". So, this has the exact same effect as the second call.

Python 2.x

Most of the above is true for Python 2.x as well, except:

- There are no keyword-only arguments between *args and **kwargs.
- You cannot use a bare * to mean no variable-positional parameter.

However, the documentation is not as thorough, and some edge cases may be slightly different.

Posted 12th August 2013 by barnert

Labels: tutorial

Das süchtigma