If you don't want the default to be shared between subsequent calls, you can write the function like this instead:

```
def f(a, L=None):
    if L is None:
        L = []
    L.append(a)
    return L
```

4.7.2 Keyword Arguments

Functions can also be called using *keyword arguments* of the form `kwarg=value`. For instance, the following function:

```
def parrot(voltage, state='a stiff', action='voom', type='Norwegian Blue'):
    print("-- This parrot wouldn't", action, end=' ')
    print("if you put", voltage, "volts through it.")
    print("-- Lovely plumage, the", type)
    print("-- It's", state, "!")
```

accepts one required argument (`voltage`) and three optional arguments (`state`, `action`, and `type`). This function can be called in any of the following ways:

```
parrot(1000)                                          # 1 positional argument
parrot(voltage=1000)                                  # 1 keyword argument
parrot(voltage=1000000, action='VOOOOOM')             # 2 keyword arguments
parrot(action='VOOOOOM', voltage=1000000)             # 2 keyword arguments
parrot('a million', 'bereft of life', 'jump')         # 3 positional arguments
parrot('a thousand', state='pushing up the daisies')  # 1 positional, 1 keyword
```

but all the following calls would be invalid:

```
parrot()                     # required argument missing
parrot(voltage=5.0, 'dead')  # non-keyword argument after a keyword argument
parrot(110, voltage=220)     # duplicate value for the same argument
parrot(actor='John Cleese')  # unknown keyword argument
```

In a function call, keyword arguments must follow positional arguments. All the keyword arguments passed must match one of the arguments accepted by the function (e.g. `actor` is not a valid argument for the `parrot` function), and their order is not important. This also includes non-optional arguments (e.g. `parrot(voltage=1000)` is valid too). No argument may receive a value more than once. Here's an example that fails due to this restriction:

```
>>> def function(a):
...     pass
...
>>> function(0, a=0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: function() got multiple values for keyword argument 'a'
```

When a final formal parameter of the form `**name` is present, it receives a dictionary (see typesmapping) containing all keyword arguments except for those corresponding to a formal parameter. This may be combined with a formal parameter of the form `*name` (described in the next subsection) which receives a *tuple* containing the positional arguments beyond the formal parameter list. (`*name` must occur before `**name`.) For example, if we define a function like this:

```
def cheeseshop(kind, *arguments, **keywords):
    print("-- Do you have any", kind, "?")
    print("-- I'm sorry, we're all out of", kind)
    for arg in arguments:
        print(arg)
```

(continues on next page)