Re-raising Exceptions

After reading Chris McDonough's What Not To Do When Writing Python Software, it occurred to me that many people don't actually know *how* to properly re-raise exceptions. So a little mini-tutorial for Python programmers, about exceptions…

First, this is bad:

```
try:
    some_code()
except:
    revert_stuff()
    raise Exception("some_code failed!")
```

It is bad because all the information about how `some_code()` failed is lost. The traceback, the error message itself. Maybe it was an expected error, maybe it wasn't.

Here's a modest improvement (but still not very good):

```
try:
    some_code()
except:
    import traceback
    traceback.print_exc()
    revert_stuff()
    raise Exception("some_code failed!")
```

`traceback.print_exc()` prints the original traceback to stderr. Sometimes that's the best you can do, because you really want to recover from an unexpected error. But if you aren't recovering, *this* is what you should do:

```
try:
    some_code()
except:
    revert_stuff()
    raise
```

Using `raise` with no arguments re-raises the last exception. Sometimes people give a blank *never use "except:"* statement, but this particular form (`except:` + `raise`) is okay.

There's another form of `raise` that not many people know about, but can also be handy. Like `raise` with no arguments, it can be used to keep the traceback:

```
try:
    some_code()
except:
    import sys
    exc_info = sys.exc_info()
    maybe_raise(exc_info)

def maybe_raise(exc_info):
    if for some reason this seems like it should be raised:
        raise exc_info[0], exc_info[1], exc_info[2]
```

This can be handy if you need to handle the exception in some different part of the code from where the exception happened. But usually it's not that handy; it's an obscure feature for a reason.

Another case when people often clobber the traceback is when they want to add information to it, e.g.:

```
for lineno, line in enumerate(file):
    try:
        process_line(line)
    except Exception, exc:
        raise Exception("Error in line %s: %s" % (lineno, exc))
```

You keep the error message here, but lose the traceback. There's a couple ways to keep that traceback. One I sometimes use is to retain the exception, but change the message:

```
except Exception, exc:
    args = exc.args
    if not args:
        arg0 = ''
    else:
        arg0 = args[0]
    arg0 += ' at line %s' % lineno
    exc.args = arg0 + args[1:]
    raise
```

It's a little awkward. Technically (though it's deprecated) you can raise *anything* as an exception. If you use `except Exception:` you won't catch things like string exceptions or other weird types. It's up to you to decide if you care about these cases; I generally ignore them. It's also possible that an exception won't have `.args`, or the string message for the exception won't be derived from those arguments, or that it will be formatted in a funny way (`KeyError` formats its message differently, for instance). So this isn't foolproof. To be a bit more robust, you can get the exception like this:

```
except:
```

```
    exc_class, exc, tb = sys.exc_info()
```

`exc_class` will be a string, if someone does something like `raise "not found"`. There's a reason why that style is deprecated. Anyway, if you really want to mess around with things, you can then do:

```
new_exc = Exception("Error in line %s: %s"
                    % (lineno, exc or exc_class))
raise new_exc.__class__, new_exc, tb
```

The confusing part is that you've changed the exception class around, but you have at least kept the traceback intact. It can look a little odd to see `raise ValueError(...)` in the traceback, and `Exception` in the error message.

Anyway, a quick summary of proper ways to re-raise exceptions in Python. May your tracebacks prosper!

Update: Kumar notes the problem of errors in your error handler. Things get more long winded, but here's the simplest way I know of to deal with that:

```
try:
    code()
except:
    exc_info = sys.exc_info()
    try:
        revert_stuff()
    except:
        # If this happens, it clobbers exc_info, which is why we had
        # to save it above
        import traceback
        print >> sys.stderr, "Error in revert_stuff():"
        traceback.print_exc()
    raise exc_info[0], exc_info[1], exc_info[2]
```