(continued from previous page)

```
    raise ValueError('Invalid log level: %s' % loglevel)
logging.basicConfig(level=numeric_level, ...)
```

The call to `basicConfig()` should come *before* any calls to `debug()`, `info()` etc. As it's intended as a one-off simple configuration facility, only the first call will actually do anything: subsequent calls are effectively no-ops.

If you run the above script several times, the messages from successive runs are appended to the file *example.log*. If you want each run to start afresh, not remembering the messages from earlier runs, you can specify the *filemode* argument, by changing the call in the above example to:

```
logging.basicConfig(filename='example.log', filemode='w', level=logging.DEBUG)
```

The output will be the same as before, but the log file is no longer appended to, so the messages from earlier runs are lost.

1.4 Logging from multiple modules

If your program consists of multiple modules, here's an example of how you could organize logging in it:

```
# myapp.py
import logging
import mylib

def main():
    logging.basicConfig(filename='myapp.log', level=logging.INFO)
    logging.info('Started')
    mylib.do_something()
    logging.info('Finished')

if __name__ == '__main__':
    main()
```

```
# mylib.py
import logging

def do_something():
    logging.info('Doing something')
```

If you run *myapp.py*, you should see this in *myapp.log*:

```
INFO:root:Started
INFO:root:Doing something
INFO:root:Finished
```

which is hopefully what you were expecting to see. You can generalize this to multiple modules, using the pattern in *mylib.py*. Note that for this simple usage pattern, you won't know, by looking in the log file, *where* in your application your messages came from, apart from looking at the event description. If you want to track the location of your messages, you'll need to refer to the documentation beyond the tutorial level – see *Advanced Logging Tutorial*.