Appendix F

Naming Conventions

You have heard it many times before and we summarized in in **Rule 2**: programs should be readable. Part of that readability is good naming. We have focused in the book on naming variables and functions based on what they are used for. However, there are also conventions on *how* elements should be named and the format of that naming.

As simple as that sounds, the style used for naming can be quite contentious. For example, see `http://en.wikipedia.org/wiki/Naming_conventions_(programming)` for all the different ways that programmers have used to name programming elements.

Whatever the style you prefer, one consideration in choosing a programming style is fitting in with the programming style of the group you are working with. Groups establish naming conventions so that reading a program is even easier. Conventions make it easier to recognize different programming elements just by how they are formatted.

Python has established such a naming convention. Python provides a special process called the *PEP* process, short for Python Enhancement Process for changes proposed to Python. A PEP is described as follows `http://www.python.org/dev/peps/pep-0001/`

> We intend PEPs to be the primary mechanisms for proposing new features, for collecting community input on an issue, and for documenting the design decisions that have gone into Python. The PEP author is responsible for building consensus within the community and documenting dissenting opinions.

You can see the Python is very open to the community at large when it comes to improving the language. One of the early PEPs, PEP-8, was focused on a Style Guide for writing Python programs (see `http://www.python.org/dev/peps/pep-0008/`).

For the most part, we have followed the style guidelines provided in PEP-8. Not all of those guidelines apply to what we have done in this book (there is a *lot* Python you have yet to see), but we did stick with those parts that applied to our book. Some prove difficult to use in a book (where space is at a premium) but we did as much as we felt we could. Also, remember **Rule 4** (which, by the way, shows up prominently in PEP-8). We followed the rules as long as they helped in readability, but we diverged when we thought there was a better way!

F.1 Python Style Elements

If you read all of PEP-8, you can see that it is quite detailed! In fact, there are enough details that Python has a package you may download called *pep8* (`http://pypi.python.org/pypi/pep8`) that will check for you how well you adhere to the style standards. How you write Python code is prescribed to a level that might surprise many people. Remember, if we as Python programmers follow these conventions, then our code is likely to more readable.

These are some of the highlights that we adhered to in the book:

F.2 Naming Conventions

There are a couple of naming conventions in use in Python:

- *lower_with_underscores*: Uses only lower case letters and connects multiple words with underscores.
- *UPPER_WITH_UNDERSCORES*: Uses only upper case letters and connects multiple words with underscores.
- *CapitalWords*: Capitalize the beginning of each letter in a word; no underscores.

With these conventions in mind, here are the naming conventions in use in the book.

Variable Names : lower_with_underscores

Constants : UPPER_WITH_UNDERSCORES

Function Names : lower_with_underscores

Function Parameters : lower_with_underscores

Class Names : CapitalWords

Method Names : lower_with_underscores

Method Parameters and Variables : lower_with_underscores

- Always use *self* as the first parameter to a method
- To indicate privacy, precede name with a single underscore. To invoke the name mangling rules (see Section 11.8.2)

F.2.1 Our Added Naming Conventions

We adopted two other naming conventions that we use throughout the book.

First, we often (though not always) append a type name to the end of a variable, preceded with an underscore. As we know, this does not affect ability to associated whatever type we wish with a variable, but it helps us as programmers remember the types we intended to store with this variable. We violate this rule when the type is clear (making the naming tedious) and usually add a comment in the code to indicate what a name without type is assumed to be.

Second, when we create variable names that are used to demonstrate some attribute of programming, as opposed to performing some operation in a program, we name variables as `my_str` or `my_list` . That is, we preced the name with *my* just to indicate we need a "throw away" variable for some demonstration.

F.3 Other Python Conventions

- Indentation should be done using four spaces per indentation level. If you use IDLE, you get this kind of indentation "for free", that is IDLE does it for you. However, remember that you are using spaces, not tabs, when you do indentation. This difference might matter, if you use a editor that does not understand Python.
- Lines that get too long should be aligned in a "reasonable" fashion, for example aligned with an open delimiter and at a different indentation level than any following indented suites.
- Lines should be less than 80 characters. This rule ensures compatibility with many display formats.
- Blank lines are recommended in many circumstances which we list below. However, writing a book and trying to minimize spacing requires that we violated these rules occasionally.

  - Separate functions and class definitions with two lines.
  - Separate method definitions with one line.
  - Use blank lines to group related elements.

- Do individual imports on a separate line at the top of the file.
- Use of whitespace:
  - White space should be used to separate binary operators and assignment statements from other elements (just one space is sufficient).
  - No white space around default parameter assignments in function definitions.
  - No space between a function call and its argument list's first parenthesis.