Coding Conventions for Python

Shahla Almasri
(salmas1@cs.mcgill.ca)

June 10th, 2005
References

Most of the materials in this presentation are taken directly from:

  http://www.python.org/peps/pep-0008.html
- D. Goodger and G. van Rossum, “Docstring Conventions”,
  http://www.python.org/peps/pep-0257.html
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Naming Conventions

- **Module Names:**
  - Short, lowercase names, without underscores.
  - Example: `myfile.py`

- **Class Names:**
  - CapWords convention.
  - Example: `MyClass`

- **Exception Names:**
  - If a module defines a single exception raised for all sorts of conditions, it is generally called "Error".
  - Otherwise use CapWords convention (i.e. `MyError`).
Method Names and Instance Variables:

- The “Style Guide for Python Code” recommends using lowercase with words separated by underscores (example: my_variable). But since most of our code uses mixedCase, I recommend using this style (example: myVariable)
- Use one leading underscore only for internal methods and instance variables (i.e. protected). Example: _myProtectedVar
- Use two leading underscores to denote class-private names. Example: __myPrivateVar
- Don’t use leading or trailing underscores for public attributes unless they conflict with reserved words, in which case, a single trailing underscore is preferrable (example: class_)
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Organizing Imports

- They should be always put at the top of the file, just after any module comments and docstrings, and before module globals and constants.

- Imports should be on separate lines.

  **Wrong:** import sys, os

  **Right:** import sys

  import os

The following is OK, though:

  from types import StringType, ListType
Organizing Imports (cont.)

- Imports should be grouped in the following order with a blank line between each group of imports:
  - standard library imports
  - related major package imports
  - application specific imports
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Indentions and Line Length

- Indentions:
  - 2 spaces (no tabs!)
  - Avoid using more than five levels of indentation.

- Line length:
  - Maximum of 72 characters (never exceed 79 characters)
  - You can break a long line using “\”.
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Break Lines

- Leave one line between functions in a class.
- Extra blank lines may be used to separate groups of related functions.
- Blank lines may be omitted between a bunch of related one-liners.
- Use blank lines in functions, sparingly, to indicate logical sections.
Outline

- Naming Conventions
- Organizing imports
- Indentations and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
White Space

- Multiple statements on the same line are discouraged.

**WRONG:**

```python
if foo == 'blah': doBlahThing()
```

**CORRECT:**

```python
if foo == 'blah':
    doBlahThing()
```
White Space (cont.)

- No white space immediately before an open parenthesis.

**WRONG:**  spam (1)
**CORRECT:**  spam(1)

**WRONG:**  dict ['key'] = list [index]
**CORRECT:**  dict['key'] = list[index]
White Space (cont.)

- No white space inside parentheses, brackets or braces.
  WRONG: \( \text{spam(ham[1], \{eggs:2\})} \)
  CORRECT: \( \text{spam(ham[1], \{eggs:2\})} \)

- No white space immediately before a comma, semicolon, or colon.
  WRONG:
  \[
  \text{if x == 4:} \\
  \text{print x, y; x, y = y, x}
  \]
  CORRECT:
  \[
  \text{if x == 4:} \\
  \text{print x, y; x, y = y, x}
  \]
White Space (cont.)

- No more than one space around an operator.

**WRONG:**

```plaintext
x = 1
yVal = 2
longVariable = 3
```

**CORRECT:**

```plaintext
x = 1
yVal = 2
longVariable = 3
```
Always surround the following operators with a single space on either side

- assignment (=)
- comparisons (==, <, >, !=, <>, <=, >=, in, not in, is, is not)
- Booleans (and, or, not)
- Arithmetic operators (+, -, *, /, %)

WRONG:

```python
if (x==4)or(x==5):
x=y+5
```

CORRECT:

```python
if (x == 4) or (x == 5):
x = y + 5
```
White Space (cont.)

- Don't use spaces around the '==' sign when used to indicate a keyword argument or a default parameter value.

**WRONG:**

def complex(real, imag = 0.0):
    return magic(r = real, i = imag)

**CORRECT:**

def complex(real, imag=0.0):
    return magic(r=real, i=imag)
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Programming Recommendations

- Comparisons to singletons like None should always be done with 'is' or 'is not'.
  Wrong:
  ```python
  if x:
    y = 6
  ```
  Right:
  ```python
  if x is not None:
    y = 6
  ```

- Don't compare boolean values to True or False.
  Wrong:
  ```python
  if greeting == True:
    y = 6
  ```
  Right:
  ```python
  if greeting:
    y = 6
  ```
Programming Recommendations (cont.)

- Avoid slicing strings when checking for prefixes or suffixes. Use `startswith()` and `endswith()` instead.

Wrong:

```python
if foo[:3] == 'bar':
```

Right:

```python
if foo.startswith('bar'):
```
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Comments

- **Block Comments:**
  - They are indented to the same level as the code they apply to.
  - Each line of a block comment starts with a # and a single space.
  - Paragraphs inside a block comment are separated by a line containing a single #.
  - Block comments are best surrounded by a blank line above and below them

Example:

```plaintext
# Compensate for border. This is done by incrementing x by the same amount

x += 1
```
Comments (cont.)

- **Inline Comments:**
  - They should start with a # and a single space.
  - Should be separated by at least two spaces from the statement they apply to.

Example:

```python
x += 1  # Compensate for border
```
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example
Documentation Strings

- Write docstrings for all public modules, functions, classes, and methods.
- Docstrings are not necessary for non-public methods, but you should have a comment that describes what the method does. This comment should appear after the "def" line.
- Insert a blank line before and after all docstrings that document a class.
One-line Docstrings:
  - The opening and closing """" are on the same line.
  - There is no blank line either before or after the docstring.
  - Describes the function or method's effect as a command ("Do this", "Return that"), not as a description.
Multi-line Docstrings:
- The """" that ends a multiline docstring should be on a line by itself.
- **Script:** The docstring of a script should be usable as its "usage" message. It should document the script's function, the command line syntax, and the environment variables.
- **Module:** The docstring for a module should generally list the classes, exceptions and functions (and any other objects) that are exported by the module, with a one-line summary of each.
– Class:
  • The docstring for a class should summarize its behavior and list the public methods and instance variables.
  • If the class is intended to be subclassed, and has an additional interface for subclasses, this interface should be listed separately.
  • If a class subclasses another class and its behavior is mostly inherited from that class, its docstring should mention this and summarize the differences.
  • The class constructor should be documented in the docstring for its __init__ method.
Function or method:
- The docstring should summarize its behavior and document its arguments, return value, side effects, exceptions raised, and restrictions on when it can be called.
- Optional arguments should be indicated.
- Use the verb "override" to indicate that a subclass method replaces a superclass method and does not call the superclass method; use the verb "extend" to indicate that a subclass method calls the superclass method.
- The docstring should contain a summary line, followed by a blank line, followed by a more elaborate description.
Outline

- Naming Conventions
- Organizing imports
- Indentions and line length
- Break lines
- White space
- Programming Recommendations
- Comments
- Documentation Strings
- Example