Code Conventions for the Python Programming Language

File Suffixes

File Type	Suffix
.py	Python source
.pyc	Python byte code
.pyd	Python extension modules (shared libraries)

File Structure

1. File header describing encoding / authors / copyright / etc.
2. Module documentation string.
3. Import statements.
4. Module properties, at least *__version__*.
5. Definition of constants.
6. Definition of module level variables, functions, classes.

Separate these parts with two blank lines.
Limit the file length to 600 lines.

Source Code Encoding

- Use ASCII encoding
- If non-ASCII characters are required include those using \x, \u or \U escape sequences.
  Exception: Author names:
  - Use the UTF-8 encoding.
  - Indicate this encoding (PEP 263) in the first line as follows:

```
# coding: utf-8
```

Imports

- Each line should usually contain at most one import statement.
- Import packages and modules only.
- Import grouping:
  - Standard library imports.
  - Third party imports.
  - Application-specific imports.
- Separate import groups with a blank line.
- No relative imports.

Indentation

- **Indent:** 4 spaces.
- Don't indent module level variables, classes, functions.
- When an expression will not fit on a single line (max. 120), break it according to these general principles:
  - Use implicit line continuation within (), [], {} for a line break.
  - Break after a comma.
  - Break before an operator.
  - Align the new line with the beginning of the expression at the same level on the previous line **or** use an indent of 4 spaces.

Whitespace

- Use a blank line between related parts of the code and between methods of a class.
- Use two blank lines between module level functions and classes.
- Use blank spaces:
  - Between a keyword and a parenthesis.
  - After commas in argument lists.
  - Before and after binary operators.
- Don't surround the = operator with blanks if it is used to indicate default parameter values or keyword arguments.

Comments

- Don't explain what's obvious from the code.
- Write comments in English.
- Use inline comments sparingly.
- Keep the comments up-to-date!

```
#
# Here is a block comment.
#

x = x + 2  # This is an inline comment.
```

Documentation Strings

- Required for all public modules, functions, classes, methods.
- Non-public stuff should have at least a one-line documentation string after the definition.
- Separate documentation string and implementation with a blank line.
- Use Sphinx mark-up to document parameters, return values and exceptions.

```
""" One-line documentation string. """

"""
Multi-line documentation string.

Another comment.
"""
```

Statements

- Each line should contain at most one statement.
- Limit the line length to 120 characters.
- Compound statements are multiple statements on the same line and should NEVER be used.

The if-else class of statements should have the following form:

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

Please avoid long if-else statements and use dictionaries instead.
Please avoid unnecessary brackets around conditions.

Loop statements should have the following form:

```
for item in target_list:
    statements

for key, value in target_dict.iteritems():
    statements

while condition:
    statements
```

Exceptions

- Use class-based exceptions and inherit from the built-in class *Exception*.
- Don't simply catch *Exception*.
- Don't use the empty *except* statement.
- A *try-except* statement should have the following format:

```
try:
    statements
except ExceptionClass1, error:
    statements
except ExceptionClass2, error:
    statements
```

- Use the *error.args* to access the exception arguments.
- Use the *with* statement to encapsulate clean-up behaviour (available since Python 2.5):

```
with open("/file_path", "r") as file_object:
    print(file_object.read())
```

- Use *except ExceptionClass as error:* (available since Python 2.6).

Classes

- Define all instance variables in *__init__*, *__new__* or *setup* (unit tests).
- First parameter of instance methods is *self*.
- First parameter of class methods is *cls*.
- Don't document the first parameter of methods or class methods.
- Structure of classes
  - Class definition.
  - Class documentation string.
  - *__init__* method.
  - *property* statements
  - Methods (functionally grouped).

Naming Conventions

Rules for Naming	Example
Package and module names: short, lowercase, words separated by underscore, no dashes	common, sdk, logger_utils
Class names: nouns, camel case Use one leading underscore to indicate an internal class.	_Raster, ImageSprite, RasterDelegate
Test classes: see conventions for class names but use the *TestCase* suffix.	TestRaster, TestImageSprite
Exception names: see convention for class names but use the *Error* suffix.	_InternalError, ApplicationError
Methods: verbs, lowercase, words separated by underscore Use one leading underscore for non-public methods. Use two leading underscores to prevent use in subclasses. Use the *test* prefix for test methods.	run, run_fast, get_background, _run_slow, __run_slower, test_run, test_get_background
Instance variable names: see methods but use nouns instead.	width, _parent_frame, __secret_width
Function names: see methods Use a leading underscore to indicate an internal function. Use trailing underscore to solve naming conflicts.	read_file, _calculate_width, print_
Module variable, variable, argument names: lowercase, words separated by underscore Use trailing underscore to solve naming conflicts.	new_width, property_
Constants have to be declared on module level or class level and are written in capital letters separating words by underscores.	TOTAL, AND_OPERATOR

Sample Code

```
#
# Author(s): Name user@dlr.de>
#
# Copyright (c) 2008-2011, (DLR)
# All rights reserved.
#
# http://www.dlr.de/datafinder/
#


""" Module documentation string. """


import os

import ldap

from webdav.test.example import application


__version__ = "$LastChangedRevision$"


AND_OPERATOR = "and"
OR_OPERATOR = "or"


def return_something(parameter=4):
    """
    Here goes the description.

    :param parameter: parameter.
    :type newParameter: int

    :return: A simple integer value.
    :rtype: int
    """

   return new_parameter


class Multiply(object):
    """ Class documentation string. """

    def __init__(self):
        self.multiply_with = 3

    def multiply(self, value):
        """ Performs the calculation. """

        return value * self._multiply_with
```