Coding Style Guidelines for Python

This document was adapted from Guido's original Python Style Guide essay "PEP8 – Python Enhancement Proposals" [i]. It is recommended to you studying these guidelines before starting coding.

The guidelines provided in PEP8 are intended to improve the readability of code and make it consistent, recognizing that code is read much more often than it is written. However, this is only a guideline and sometimes these recommendations won't be applicable.

As we are going to jump in existing codes, we should also intent to improve those codes and return them as maintainable, consistent and stables routines. Not necessarily we should change their styles, but we could try to refactor, to reduce lines and to use better methods or python's libraries, and of course, we should document these existing codes.

Likewise, we should always be thinking in our cloud-computed based architecture. For that reason, we should build component-based codes in which each one of them could not be heavier than 50 Mb including all their dependencies (this limit could be exceed but will depend on the AWS service we will use).

Some PEP guidelines as indentation, maximum line length, line breaks before binaries operators, blank lines, whitespaces in expressions and statements, etc., could be carried out using python's formatting extensions and some IDE's default settings (the most used for python are Visual studio code and Black formatter).

In this document will be a summary of the main rules for coding as imports, documentation strings (docstrings) and naming conventions.

Importing packages and libraries

Imports should be on separates lines, not by commas, this applies only for different libraries, when we are importing methods or functions within a library it is able to import them with a single line:

```
# Correct:
import os
import sys

# Wrong:
import sys, os

# Correct:
from subprocess import Popen, PIPE
```

Order is also important. Imports are always put at the top of the file, just after module's docstring and before module's global variables and constants. Import should be grouped in the following order:

1. Standard library imports (python packages, i.e., NumPy, sys, os, etc.)
2. Related third party imports
3. Local library specific imports (our own modules)

It's recommended to use absolute imports because they are more readable and have better error messages. However, when dealing with complex package layouts or components it's acceptable using explicit relative imports:

```
from . import sibling
from .sibling import example
```

We should avoid importing all methods from modules (i.e., "from <module> import *), these imports make if unclear which name are presents in the namespace, confusing both readers and many automated tools.

Documentation Strings

Always write docstrings for all modules, functions, classes, and methods. Comments should appear after the def line for functions, methods, or classes, and after imports for module's docstrings.

```
"""Return a foobang

Optional plotz says to frobnicate the bizbaz first.
"""
```

Naming conventions

In the following, it will be recommended naming standards for constants, variables, functions, methods, and classes for our codes. These recommendations will help the code to be able to be recognized and consistent with a unified style regardless of the developer who made it.

It will be also described special forms for handling codes, using leading or trailing underscores for avoiding conflicts, naming class attributes and function's mangling.

Package and Module Names

Modules should have short, all-lowercase names. Underscores can be used in the module name if it improves readability. Python packages should also have short, all-lowercase names, although the use of underscores is discouraged.

In example, a valid package name could be "*component1*" and a valid module name could be "*processing.py*" or "*processing_geometry.py*"

Function and Variable Names

Function names should be lowercase, with words separated by underscores as necessary to improve readability. Variable names follow the same convention as function names.

Class Names

Class names should normally use the CapWords convention (i.e., FileGeometry). The naming convention for functions may be used instead in cases where the class is used as a callable.

Type Variable Names

Names of type variables should normally use CapWords preferring short names: Num, AnyStr, Obj, Flt, etc.

For example, take variable:

```
one_text = "hello world"
IsString = type(one_text)
```

Global Variable Names

The conventions are about the same as those for functions.

Function and Method arguments

Always use `self` for the first argument to instance methods and use `cls` for the first argument to class methods.

If a function argument's name clashes with a reserved keyword, it is better to append a single trailing underscore rather than use an abbreviation or spelling corruption. Thus `class_` is better than `clss`. (Perhaps better is to avoid such clashes by using a synonym.)

Constants

Constants are usually defined on a module level and written in all capital letters with underscores separating words. Examples include `MAX_OVERFLOW` and `TOTAL`.

Special Forms for handling code

The following special forms using leading or trailing underscores usually combined with any case naming convention:

- `_single_leading_underscore`: weak "internal use" indicator. This is used when importing all functions from a module, python does not import objects whose names start with an underscore.
- `single_trailing_underscore_`: used by convention to avoid conflicts.
- `__double_leading_underscore`: when naming a class attribute, invokes name mangling.
- `__double_leading_and_trailing_underscore__`: "magic" objects or attributes that live in user-controlled namespaces. For example, `__init__`, `__import__` or `__file__`. Never invent such names; only use them as documented.

[i] https://peps.python.org/pep-0008/