Calidad en el código

El código se considera de calidad si:

- Hace lo que se supone que debe hacer
- No contiene fallos o da problemas.
- **Es sencillo de leer, mantener y modificar para añadir mejoras.**

Es en ese tercer apartado en el que nos centramos en este documento.

Guías de estilo

La fundamental, PEP 8. Se indican cuestiones como:

- Indentation: Use 4 spaces per indentation level.
- Tabs or Spaces? Spaces are the preferred indentation method. Tabs should be used solely to remain consistent with code that is already indented with tabs.Python 3 disallows mixing the use of tabs and spaces for indentation.
- Maximum Line Length: Limit all lines to a maximum of 79 characters.
- Blank Lines:
  - Surround top-level function and class definitions with two blank lines.
  - Method definitions inside a class are surrounded by a single blank line.
- Imports: Imports should usually be on separate lines

- Etc...

Luego adicionalmente hay otras, como por ejemplo la de Google para Python.

- Guía de estilo del código Python. Traducción por Raúl González Duque el día 10 de Agosto de 2007. http://mundogeek.net/traducciones/guia-estilo-python.htm
- Guía de estilo Google para Python: https://google.github.io/styleguide/pyguide.html

Comentarios y documentación

A estas alturas no vamos a insitir en esto. Además hemos repasado el uso de los docstring en otro documento.

Linting y linters

Hay un cierto grupo de errores que pueden ser identificados y reparados antes de que tu código llegue a ser ejecutado, estos pueden ser:

- Errores de sintaxis
- Código poco intuitivo o dificil de mantener
- Uso de "malas practicas"
- O uso de estilos de código inconsistentes.

Las herramientas que revisan tu código para buscar estos errores o malas prácticas se llaman linters. En el caso de Python hay varios disponibles. Nosotros vamos a ver un par de ellos:

- flake8 (realmente hace varias cosas...)
- pylint

Los pasos que daremos a continuación son:

- Usar un fichero con errores: `test_linters.py`.
- Crear un entorno virtual e instalar los linters pylint y flake8
- Probar estas herramientas desde la línea de comandos.
- Probarlas desde Visual Studio Code.
- Aprender lo básico sobre su uso y configuración

Instalación

- Creamos entorno virtual

```
@tos:~/calidad$ python -m venv venv
@tos:~/calidad$ source venv/bin/activate
(venv) @tos:~/calidad$
```

- Instalamos flake8

```
(venv) @tos:~/calidad$ pip install flake8
```

- Instalamos pylint

```
(venv) @tos:~/calidad$ pip install pylin
```

Ejecución en CLI

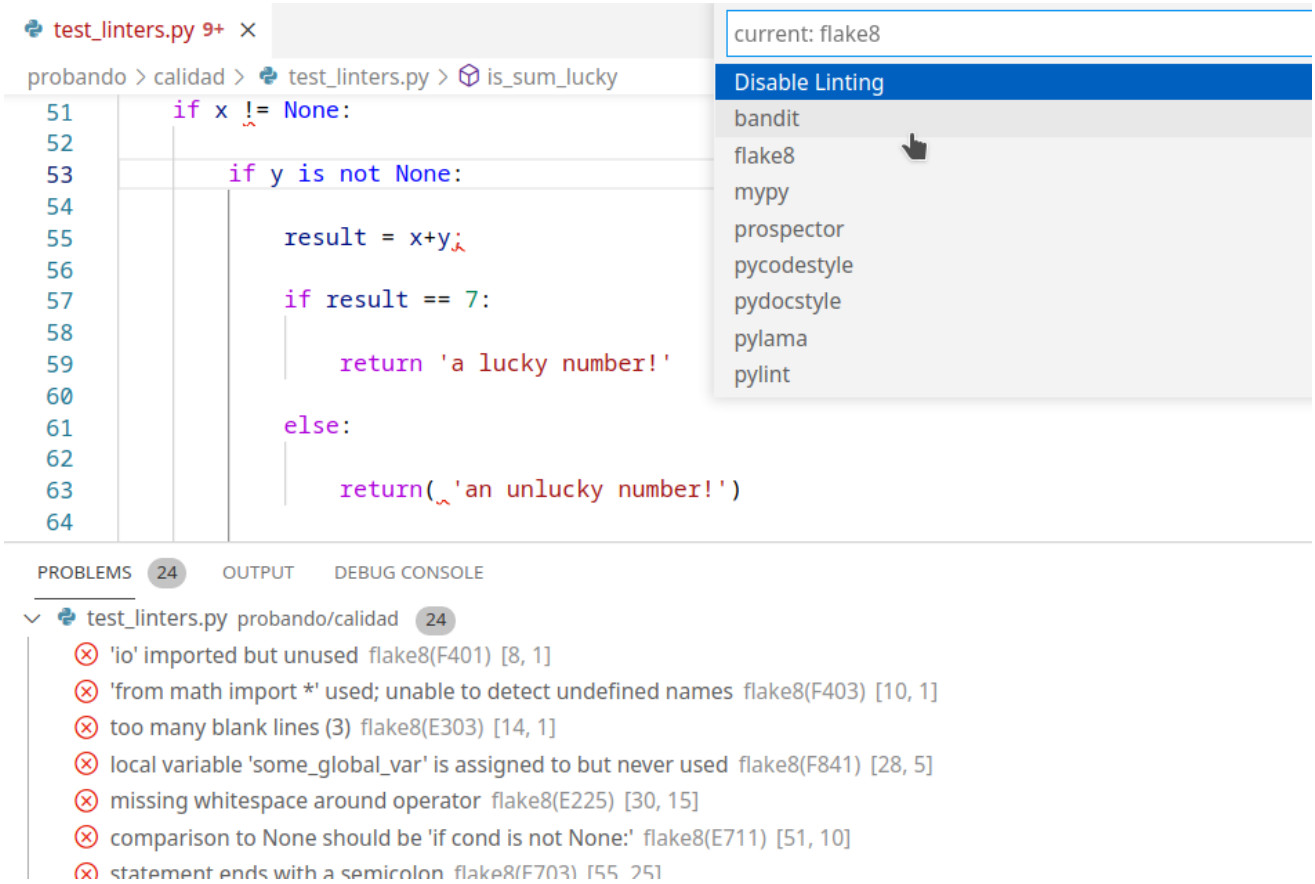
Probamos flake8

```
(venv) @tos:~/calidad$ flake8  test_linters.py
test_linters.py:8:1: F401 'io' imported but unused
test_linters.py:14:1: E303 too many blank lines (3)
test_linters.py:28:5: F841 local variable 'some_global_var' is
assigned to but never used
test_linters.py:63:24: E201 whitespace after '('
test_linters.py:66:13: E303 too many blank lines (2)
test_linters.py:72:58: E251 unexpected spaces around keyword /
parameter equals
test_linters.py:76:31: E222 multiple spaces after operator
test_linters.py:78:9: F841 local variable 'list_comprehension' is
assigned to but never used
test_linters.py:78:80: E501 line too long (83 > 79 characters)
test_linters.py:80:9: F841 local variable 'time' is assigned to
but never used
...
(venv) ango@tos:~/CETI-PPS-ango/bloque_01/probando/calidad$
```

Y ahora `pylint`

```
(venv) @tos:~/calidad$ pylint test_linters.py
************* Module test_linters
test_linters.py:55:0: W0301: Unnecessary semicolon (unnecessary-
semicolon)
test_linters.py:63:0: C0325: Unnecessary parens after 'return'
keyword (superfluous-parens)
test_linters.py:66:0: C0325: Unnecessary parens after 'return'
keyword (superfluous-parens)
...
-------------------------------------------------------------------
---
Your code has been rated at -2.22/10 (previous run: -2.22/10,
+0.00)
```

En el editor



Uso básico y configuración flake8

> *Flake8 is a Python library that wraps PyFlakes, pycodestyle and Ned Batchelder's McCabe script. It is a great toolkit for **checking your code base against coding style (PEP8)**, programming errors (like "library imported but unused" and "Undefined name") and to check cyclomatic complexity.*

Si solamente queremos revisar un error en particular usamos `--select` : :

```
flake8 --select E123,W503 path/to/code/
```

Si queremos ignorar alguno `--ignore` :

```
flake8 --ignore E24,W504 path/to/code/
```

Flake8 supports storing its configuration in your project in one of `setup.cfg`, `tox.ini`, or `.flake8`. Values set at the command line have highest priority, then those in the project configuration file, and finally there are the defaults. However, there are additional command line options which can alter this.

Ejemplo de fichero tox.ini

```
[flake8]
ignore = D203
exclude = .git,__pycache__,docs/source/conf.py,old,build,dist
max-complexity = 10
```

Para tener más información

- Opciones: https://flake8.pycqa.org/en/latest/user/options.html
- Errores:
  - https://flake8.pycqa.org/en/latest/user/error-codes.html
  - https://www.flake8rules.com/
- Codificación de los errores: https://flake8.pycqa.org/en/2.0/warnings.html

> Default If the ignore option is not in the configuration and not in the arguments, only the error codes E123/E133, E226 and E241/E242 are ignored (see the warning and error codes).

Uso básico y configuración pylint

https://pylint.pycqa.org/en/latest/user_guide/run.html

https://learn.adafruit.com/improve-your-code-with-pylint/pylintrc

Specifying all the options suitable for your setup and coding standards can be tedious, so it is possible to use a configuration file to specify the default values. You can specify a configuration file on the command line using the `--rcfile` option. Otherwise, Pylint searches for a configuration file in the following order and uses the first one it finds:

1. `pylintrc` in the current working directory
2. `.pylintrc` in the current working directory
3. `pyproject.toml` in the current working directory, providing it has at least one `tool.pylint.` section. The `pyproject.toml` must prepend section names with `tool.pylint.`, for example `[tool.pylint.'MESSAGES CONTROL']`. They can also be passed in on the command line.
4. `setup.cfg` in
5. ....

```
pylint --generate-rcfile > $HOME/.pylintrc
```

Enlaces

- Escribiendo código de alta calidad en Python (2020) https://medium.com/@gonzaloandres.diaz/fundamentos-y-automatizacion-codigo-de-alta-calidad-en-python-2020-671706a7f09b
- Python Code Quality: Tools & Best Practices https://realpython.com/python-code-quality/
- Linters en VSCode: https://code.visualstudio.com/docs/python/linting
- PEP 8
    - PEP 8 -- Style Guide for Python Code: https://www.python.org/dev/peps/pep-0008/

    - PEP 8 — the Style Guide for Python Code (This stylized presentation of the well-established PEP 8 was created by Kenneth Reitz (for humans)). https://pep8.org/
    - Guía de estilo del código Python. Por Guido van Rossum, Barry Warsaw. Traducción al castellano por Raúl González Duque el día 10 de Agosto de 2007. http://mundogeek.net/traducciones/guia-estilo-python.htm
- Guía de estilo Google para Python: https://google.github.io/styleguide/pyguide.html