SOFTWARE DEVELOPMENT

Guided Exercise 2

Collective Code Ownership and Coding Standards

100472280@alumnos.uc3m.es | 100472315@alumnos.uc3m.es
Víctor Carnicero Príncipe | María Isabel Fernández Barrio

CODING STANDARD:

Introduction:

In this file we try to present our coding standard following the principles presented in class. Since defining a whole coding standard can be a challenging task we will present our principles as a modified version of the PEP-8 standard for coding in python. And present it through the pylint tool as modifications to the .pylintrc file provided for the PEP-8 standard. The changes are not very significant since the tool is difficult to use for beginners in software development.

Standard for names and variables:

Since this was the most accessible and easiest thing to change in pylint it is the part that has the most changes.

1. Variable naming:

We decided to use a variable naming convention similar to snake_case that is ruled by the following regular expression:

[a-zA-Z_]{1}([a-z0-9_]*){0,60}

Variable names can start with Uppercase or lowercase but cannot have uppercase letters anywhere else. This first character may be followed by lowercase letters or numbers. Character _ is also accepted. The maximum length for a variable name is 61, and the minimum is 1.

Acceptable variable names:

```
letters = ...
inside_drift7 = ...
_x = ...
A = ...
b = ...
Lett2ers = ...
Inside_drift = ...
```

Not accepted variable names:

```
lEtters = ...
insideDrift7 = ...
_A = ...
Inside_Drift = ...
```

2. Function naming:

The convention we established to determine the name of our functions is camelCase.

Acceptable function names:

```
def wordsToday(): ...
def interpolate(): ...
```

Not acceptable function names:

```
def words_today(): ...
def InterPolate(): ...
```

3. Method naming:

The convention we established to determine the name of methods is camelCase.

Acceptable method names:

```
    def wordsToday(): ...
    def interpolate(): ...
```

Not acceptable function names:

```
    def words_today(): ...
    def InterPolate(): ...
```

4. Constant naming:

The convention we decided to use to name the constant is snake_case.

Accepted constant names

```
letters = ...
inside_drift7 = ...
b = ...
```

Not accepted constant names:

```
Ltters = ...
InsideDrift7 = ...
_A = ...
Inside_Drift = ...
```

5. Variables always accepted

We learned that it is possible to make some exceptions to the naming rules and add variable names that we want to be accepted for sure. We added some accepted labels like “x” and “y”, additionally to some that were already accepted like “i”, “j” and “k”.

```
good-names=i,
           j,
           k,
           x,
           y,
           ex,
```

```
          Run,
          _
```

6. Bad variable names

Finally, we discovered that it is also possible to fix any string you want not to be a variable name. Since we found ourselves recurring to these names for naming variables we did not yet know what we would be using them for. We established that the labels “pepe” and “juan” cannot be the name of a variable.

```
bad-names=foo,
          bar,
          baz,
          toto,
          tutu,
          tata,
          pepe,
          juan
```

Document lines standards:

7. Characters per line:

It is established that the maximum number of characters allowed in a line is 100, but we find this to be too restricting and small, compared to screen sizes. That is why we have set the maximum number of characters in a line to 150 characters.

```
max-line-length=150
```

8. Lines per module

The current maximum recommended sizes for modules is 1000 lines, but we find this to be too small, that is why we have decided to increase it to 2000 lines, so that we can write huge pieces of code in a single module.

```
max-module-lines=2000
```

Single line statements:

9. Single line class

We allow writing a class in a single line as we considered that this can be handful when coding simple classes.

```
single-line-class-stmt=yes
```

10. Single line if statement

We think in some cases if statements in the same line are very useful and more legible than if done in multiple lines, so we allow it.

```
single-line-if-stmt=yes
```

Additional:

11. Unused imports:

We fixed pylint to raise an alarm to tell us if there are any unused imports in __init__ files in some point of the code, in order to make our code more clear and understandable.

```
init-import=yes
```

12. Arguments for functions:

We fixed the maximum number of arguments for functions and methods to 7. We learned that 5 is the number of arguments commonly allowed but thought that some more arguments would be helpful for coding functions in several cases.

```
max-args=7
```

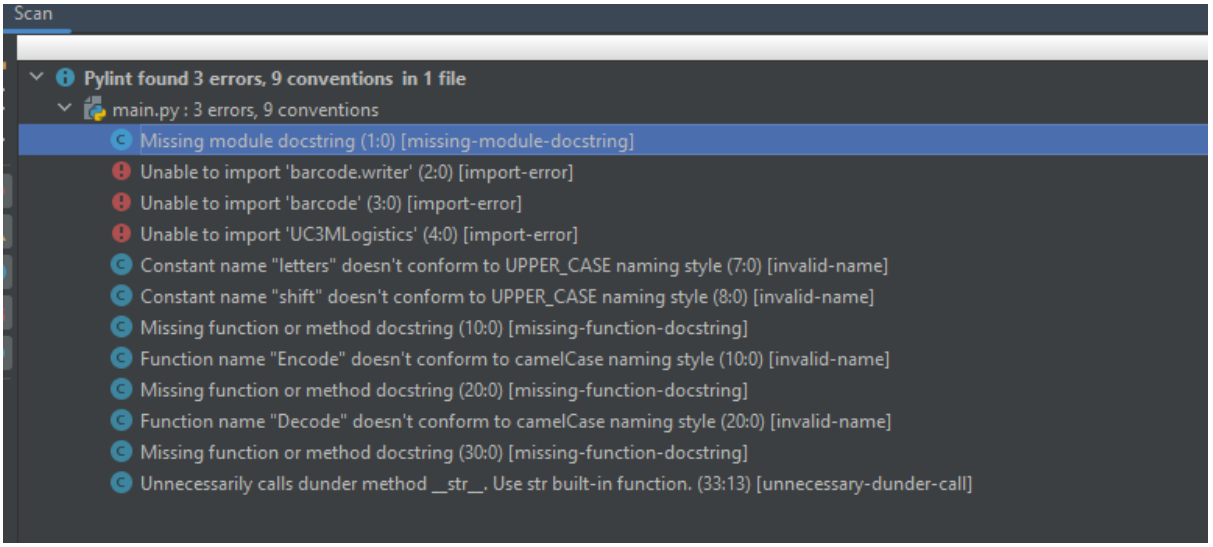
13. Note tags

We learned that pylint offers the possibility of adding personal note tags to highlight remarkable pieces of code by writing those words in our code, so we added the tag “IMPORTANT” to quickly find a line that we find particularly relevant.

```
notes=FIXME,
      XXX,
      TODO,
      IMPORTANT
```

Pylint Progress:

Before implementing the coding standard:



After implementing the coding standard: