Universidad Carlos III

Desarrollo de Software

2022-23

Ejercicio guiado II

Definición de normativa de código

Curso 2022-23

Ingeniería Informática, Segundo Curso

Adrián Fernández Galán(NIA: 100472182, e-mail: 100472182@alumnos.uc3m.es)

César López Mantecón (NIA: 100472092, e-mail: 100472092@alumnos.uc3m.es)

Prof. María Natividad Carrero de las Peñas

Grupo: 81

Índice

1. Introducción

Para la definición de nuestra propia normativa de código nos hemos basado en PEP-8, pero hemos incorporado una serie de cambios que nos permitirán tener funciones/métodos y clases más simples y legibles. Además de poder diferenciar a través del nombre las variables, atributos, argumentos u otros elementos.
Los aspectos no contemplados en este documento, seguirán el estándar establecido por PEP-8 (https://peps.python.org/pep-0008/).

2. Nombres de variables y constantes

Las **variables** deben seguir el estilo *camelCase*. Es decir, comenzar por minúscula y diferenciar las palabras poniendo la primera letra mayúscula y sin espacios. Además, las variables deberán tener una longitud mayor a un carácter para facilitar su búsqueda y la legibilidad del código.
Para esto hemos incluido en el archivo *.pylintrc* las líneas:

```
variable-naming-style=camelCase
bad-names-rgxs=^.$
```

Ejemplo correcto:

```
variableTipoEntero = 4
```

Ejemplos incorrectos:

```
variable_tipo_entero = 4
variabletipoentero = 4
VariableTipoEntero = 4
x = “hola”
```

Las **constantes** deben escribirse con letras *mayúsculas* y sin espacios, con el fin de distinguirlas de las variables a simple vista. Se ha modificado la siguiente línea en *.pylintrc*:

```
class-const-naming-style=UPPER_CASE
```

Ejemplo correcto:

```
NOMBREDECONSTANTE = 3.14159265359
```

Ejemplos incorrectos:

```
NombreConstante = 3.14
NOMBRE_CONSTANTE = 3.14
NOMBREcONSTANTE = 3.14
```

3. Nombres de funciones

Las **funciones** deben seguir el estilo *PascalCase*. Es decir, comenzar por mayúscula y diferenciar una nueva palabra con la primera letra mayúscula y sin espacios. Con esto, las funciones serán fáciles de reconocer y diferenciar de otros elementos del programa. Se ha modificado la siguiente línea en *.pylintrc*:

```
function-naming-style=PascalCase
```

Ejemplo correcto:

```
NombreFuncion():
```

Ejemplos incorrecto:

```
nombreFuncion():
NOMBREFUNCION():
Nombre_funcion():
```

4. Nombres de argumentos de funciones

Los nombres de **argumentos** deben seguir el estilo *camelCase,* manteniendo la consistencia con los nombres de variables y constantes. Se ha modificado la siguiente línea en *.pylintrc*:

```
argument-naming-style=camelCase
```

Ejemplo correcto:

```
FuncionConArgumentos(argumentoUno, argumentoDos, argumentoTres):
```

Ejemplos incorrectos:

```
FuncionConArgumentos(arg_1, Arg2, argumento_tres):
```

5. Máximo número de argumentos en una función

Las funciones tendrán, como máximo, **8 argumentos**. Siguiendo el estándar de RISC-V, con el que ya estamos familiarizados. Se ha modificado la siguiente línea en *.pylintrc*:

```
max-args=8
```

Ejemplo correcto:

```
FuncionConArgumentos(argA, argB, argC, argD, argE):
```

Ejemplos incorrectos:

```
FuncionConArgumentos(argA, argB, argC, argD, argE, argF, argG, argH,
argI, argJ, argK, argL):
```

6. Nombres de métodos

Los **métodos** deben seguir el estilo *PascalCase,* al igual que las funciones, con el fin de diferenciarse de otros elementos del código. Se ha modificado la siguiente línea en *.pylintrc*:

```
method-naming-style=PascalCase
```

Ejemplo correcto:

```
objeto.NombreMetodo():
```

Ejemplos incorrecto:

```
objeto.nombreMetodo():
objeto.NOMBREMETODO():
objeto.Nombre_metodo():
```

7. Máximo número de ramas por función

Las funciones tendrán, como máximo, **8 *if 's*** , con el fin de tener funciones cortas y limpias. Se ha modificado la siguiente línea en *.pylintrc*:

```
max-branches=8
```

8. Nombres de Clases

Los nombres de las **clases** solo pueden empezar por una letra mayúscula y seguida de cualquier número, barra baja o letra, tanto minúscula como mayúscula. Para ello utilizaremos la siguiente línea en *.pylintrc*:

```
class-rgx=^[A-Z][a-z_0-9A-Z]*$
```

Ejemplo correcto

```
class Nombre_Clase01:
```

Ejemplos incorrecto:

```
class nombre_Clase01:
class 01nombre_Clase01
```

9. Número de atributos

Los **atributos** de las clases no pueden ser más de 5, con el objetivo de tener clases más compactas. Para ello utilizamos la siguiente línea de *.pylintrc*:

```
max-attributes=5
```

10. Nombres de atributos

Los nombres de los **atributos** deben seguir el estilo, en general, *camelCase;* a excepción de los atributos privados, que empezarán por dos guiones bajos (‘__atributoPrivado’). Para ello utilizaremos la siguiente línea de *.pylintrc*:

```
attr-rgx=^[__]|[a-z][a-zA-Z]*$
class-attribute-rgx=^[__]|[a-z][a-zA-Z]*$
```

Ejemplo correcto

```
self.atributoA
self.atributoB
self.__atributoPrivado
```

Ejemplos incorrecto:

```
self.atributo_a
self.Atributo_B
self.ATRIBUTO_C
```

11. Máximo número de sentencias por función

El número máximo de **sentencias** por función será de 30, con el objetivo de tener funciones simples. Esto lo conseguimos a través de la siguiente línea de *.pylintrc*:

```
max-statements=30
```

12. Máximo número de returns por función

Una función deberá tener, a lo sumo, 5 *returns* distintos, para facilitar la comprensión y legibilidad del código. Para ello, hemos modificado la siguiente línea del fichero *.pylintrc*:

```
max-returns=5
```

13. Nombres de módulos

Los nombres de los **módulos** seguirán el estilo *PascalCase*. Para ello utilizaremos la siguiente línea de *.pylintrc*:

```
module-naming-style=PascalCase
```

Ejemplo correcto

```
ModuloA.py
```

Ejemplos incorrecto:

```
modulo_a.py
MODULO_B.py
moduloC.py
```

14. Máxima longitud de línea

Las líneas de código deberán ajustarse a 69 caracteres. Para ello utilizamos la siguiente línea de *.pylintrc*:

```
max-line-length=69
```

Ejemplo correcto

```
raise OrderManagementException \
   ("Wrong file or file path") from exception
```

Ejemplos incorrecto:

```
raise OrderManagementException("Wrong file or file path") from exception
```

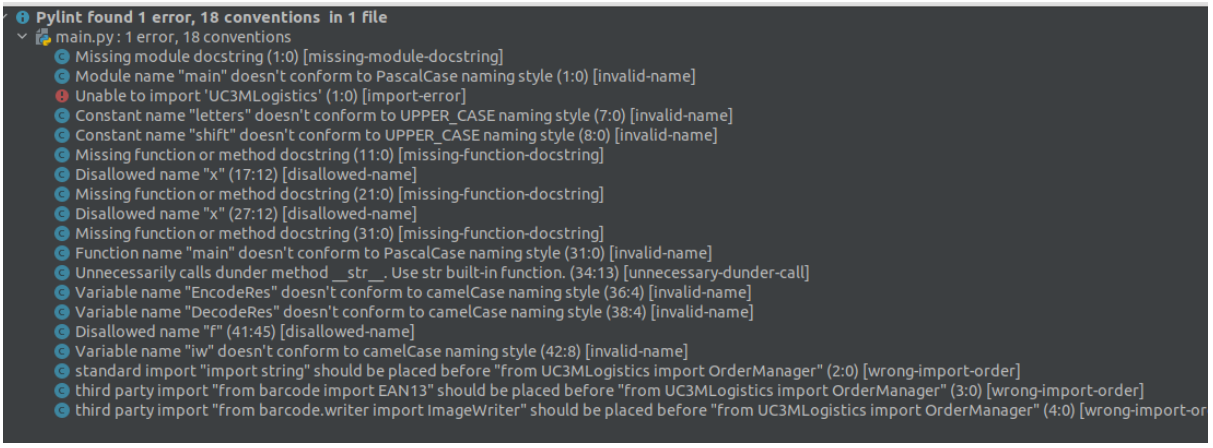
15. Conclusión

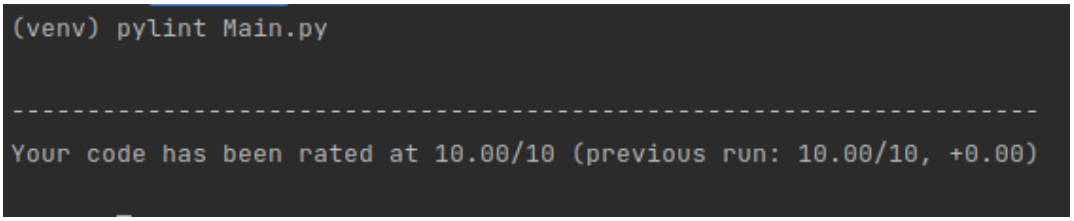
A través de estas normas de estilo logramos mantener un código limpio, legible y fácil de comprender. Además, establecemos un formato y lenguaje común a ambos miembros del equipo de trabajo, evitando inconsistencias y faltas de coherencia en el código. También, al ser un código discutido desde todas las partes, se genera un ambiente donde prima la comodidad de todos los miembros del equipo.

16. Errores antes y después de la depuración

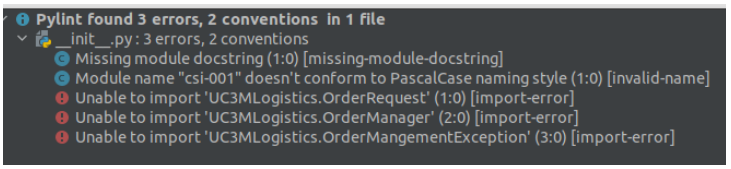
Todos los errores de pylint fueron corregidos a lo largo de las diferentes iteraciones del proyecto.

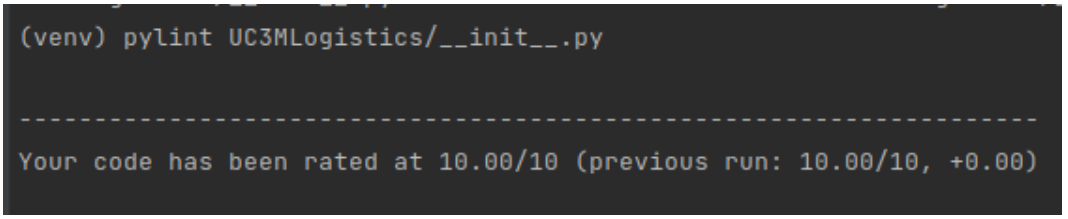
- *Main.py*





- *__init__.py*





- *OrderManager.py*

```
Pylint found 2 errors, 1 warning, 13 conventions in 1 file
  OrderManager.py : 2 errors, 1 warning, 13 conventions
    Unnecessary parens after 'if' keyword (26:0) [superfluous-parens]
    Missing module docstring (1:0) [missing-module-docstring]
    Attempted relative import beyond top-level package (2:0) [relative-beyond-top-level]
    Attempted relative import beyond top-level package (3:0) [relative-beyond-top-level]
    Missing class docstring (6:0) [missing-class-docstring]
    Missing function or method docstring (10:4) [missing-function-docstring]
    Missing function or method docstring (31:4) [missing-function-docstring]
    Argument name "fi" doesn't conform to camelCase naming style (31:38) [invalid-name]
    Using open without explicitly specifying an encoding (34:17) [unspecified-encoding]
    Disallowed name "f" (34:29) [disallowed-name]
    Variable name "DATA" doesn't conform to camelCase naming style (35:16) [invalid-name]
    Disallowed name "e" (36:8) [disallowed-name]
    Disallowed name "e" (38:8) [disallowed-name]
    Variable name "PRODUCT" doesn't conform to camelCase naming style (42:12) [invalid-name]
    Variable name "PH" doesn't conform to camelCase naming style (43:12) [invalid-name]
    Disallowed name "e" (45:8) [disallowed-name]
```

```
(venv) pylint UC3MLogistics/OrderManager.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

- *OrderMangementException.py*

```
Pylint found 4 conventions in 1 file
  OrderMangementException.py : 4 conventions
    Trailing newlines (13:0) [trailing-newlines]
    Missing module docstring (1:0) [missing-module-docstring]
    Missing class docstring (1:0) [missing-class-docstring]
    Missing function or method docstring (7:4) [missing-function-docstring]
```

```
(venv) pylint UC3MLogistics/OrderMangementException.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```

- *OrderRequest.py*

```
Pylint found 1 warning, 8 conventions in 1 file
  OrderRequest.py : 1 warning, 8 conventions
    Missing module docstring (1:0) [missing-module-docstring]
    Missing class docstring (5:0) [missing-class-docstring]
    Missing function or method docstring (16:4) [missing-function-docstring]
    Attribute name "Phone" doesn't conform to '^[__][a-z][a-zA-Z]*$' pattern (16:4) [invalid-name]
    Attribute name "Phone" doesn't conform to '^[__][a-z][a-zA-Z]*$' pattern (19:4) [invalid-name]
    Missing function or method docstring (23:4) [missing-function-docstring]
    Attribute name "PRODUCT_CODE" doesn't conform to '^[__][a-z][a-zA-Z]*$' pattern (23:4) [invalid-name]
    Attribute name "PRODUCT_CODE" doesn't conform to '^[__][a-z][a-zA-Z]*$' pattern (26:4) [invalid-name]
    Unused private member `OrderRequest.__timeStamp` (10:8) [unused-private-member]
```

```
(venv) pylint UC3MLogistics/OrderRequest.py

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
```