Universidad Carlos III

Desarrollo de Software

2022-23

EG2 Desarrollo de Software

Fecha: **04/02/22**

NIA: **100472277 / 100472252 //** Grupo: **82**

Grado: **Grado en Ingeniería Informática**

Alumnos: **Miguel Lucena Belmonte/Javier Pallarés de Bonrostro**

TABLA DE CONTENIDO

Argumentos y Atributos

→ argument-naming-style=camelCase
→ class-attribute-naming-style=camelCase
→ attr-rgx=^(__)?([a-zA-Z]+[a-z0-9]*)+$

```python
# Correct:
def add_numbers(firstNumber, secondNumber):
    sum = firstNumber + secondNumber
    print('Sum:', sum)

add_numbers(2, 3)
# Wrong:
def add_numbers(first_number, second_number):
    sum = first_number + second_number
    print('Sum:', sum)

add_numbers(2, 3)
```

```python
#Correct:

class Vehiculo:

    def __init__(self, marcaVehiculo, colorVehiculo):
        self.marcaVehiculo = marcaVehiculo
        self.colorVehiculo = colorVehiculo

#Wrong:

class Vehiculo2:

    def __init__(self, marca_vehiculo, color_vehiculo):
        self.marca_vehiculo = marca_vehiculo
        self.color_vehiculo = color_vehiculo
```

Descripción

Se ha decidido cambiar el estilo de escritura para los nombres de los argumentos y los atributos de clases. El estilo predeterminado era snake_case, que ha sido sustituido por camelCase. Con la expresión regular, conseguimos que Pylint acepte los atributos con el estilo camelCase, incluyendo aquellos que son privados (comienzan por '__').

Funciones

→ docstring-min-length=3

```python
#Correct:
def square(n):
    '''Takes in a number n, returns the square of n if n value is less than variable value'''
    variable = 35
    if n < variable:
        return n**2

#Wrong:
def square_2(n):
    '''Takes in a number n, returns the square of n'''
    return n**2
```

Descripción

Para que una función necesite tener un docstring debe tener un tamaño de al menos 3 líneas. De esta manera, evitamos tener que crear un docstring en funciones muy sencillas que no necesitan información extra para ser comprendidas.

→ include-naming-hint=yes

Descripción

Se ha decidido obligar el incluir naming hint, es decir, se debe llevar a cabo en el código un tipado de los parámetros a utilizar.

→ max-args=7

Descripción

Se ha decidido que el número máximo de argumentos a utilizar en una función o método sea siete. La inspiración de esta decisión viene de la arquitectura RISC-V, utilizada por los integrantes del equipo en el pasado.

→ max-attributes=10

Descripción

Se ha decidido cambiar el número máximo de atributos para clases; ahora, el nuevo límite será de 10 atributos.

→ max-returns=7

Descripción

Se ha decidido que el número máximo de retornos o devoluciones de la máquina en una función o método sea siete, con la idea de basarse en la arquitectura RISC-V, utilizada por los integrantes del equipo en el pasado, con el objetivo de facilitar el proceso.

→ max-nested-blocks=3

```
#Correct:

def ejemplo(firstValue, secondValue):

    if firstValue > secondValue:
        counter = 2
        if counter > secondValue:
            secondValue = counter

    return secondValue


#Wrong:

def ejemplo_2(firstValue, secondValue):
    if firstValue > secondValue:
        counter = 2
        if counter > secondValue:
            secondValue = counter
            if secondValue > firstValue:
                firstValue = secondValue
                secondCounter = 35
                if firstValue > secondCounter:
                    return firstValue

    return secondValue
```

Descripción

Se ha decidido reducir el número máximo de anidamientos dentro de una función o método, con el objetivo de aclarar el código para el lector de éste.

VARIABLES

→ bad names= (añadimos) var, aux y temp

```
#Correct:
counter = 0

#Wrong:
aux = 0
```

Descripción

Se ha hecho una elección de nombres no ideales para las variables que se escriban en el código, con el objetivo de que éste sea formal y legible. De esta manera, con el uso de nombres significativos para las variables, se consigue que el código sea también explicativo.

→ bad-names-rgxs=\w{1,2}$

Descripción

Se ha hecho una elección de nombres no ideales para las expresiones regulares que se escriban en el código, con el objetivo de que éste sea formal y legible. Toda expresión con menos de 3 caracteres será rechazada (aunque existen excepciones como ‘i’).

→ variable-naming-style=camelCase

```
#Correct

variableEjemplo = 0
segundaVariable = variableEjemplo

#Wrong:

variable_ejemplo = 0
segunda_variable = variable_ejemplo
```

Descripción

Se ha decidido establecer un estilo de escritura
para las variables a declarar. El estilo será camelCase.

→ const-rgx=[A-Z0-9_]{2,40}$

Descripción

El estilo de nombramiento de las constantes sólo admite mayúsculas, de esta manera también admitirá números y '_'.

CONDICIONALES

→ max-bool-expr=3

```
#Correct:

if azul == True and rojo == False and amarillo == True:
    print("OK!")

#Wrong:

if azul == True and rojo == False and amarillo == True and verde == False:
    print("COOL!")
```

Descripción

Se ha decidido reducir el límite superior de uso de expresiones booleanas de 5 a 3. El objetivo principal es facilitar la comprensión del código. En el caso de necesitar más de 3 cláusulas, se recurriría a condiciones anidadas, que son más legibles para el usuario.

→ single-line-if-stmt=yes

```
# Correct:
value = 5
if x >= value: value = x
# Alternative:
value = 5
if x >= value:
    value = x
```

Descripción

Se ha decidido que en una condición, el statement if se desarrolle en una sola línea, es decir, la condición y consecuencia se encuentran en la misma línea de código. Esto permitirá facilitar la lectura en condiciones simples.

FORMATO

→ max-line-length=80
→ max-module-lines=750

Descripción

Se ha decidido reducir tanto el número máximo de líneas como el tamaño máximo de línea, con el objetivo de facilitar la lectura del código.

MÓDULOS

→ module-naming-style=PascalCase

```python
# MiModulo.py
def suma_valores(a, b):
    return a + b

def resta_valores(a, b):
    return a - b
```

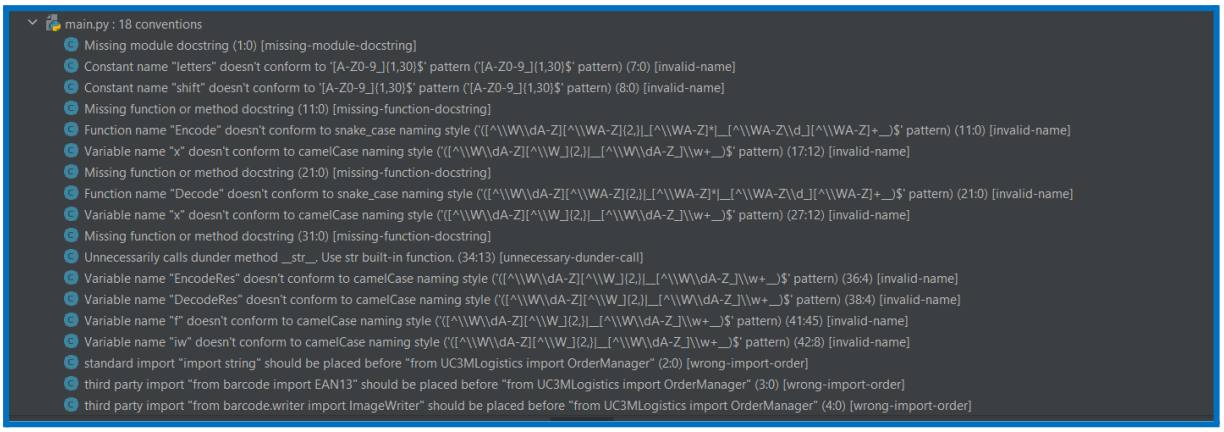
```python
#OtroModulo.py
import MiModulo

print(MiModulo.suma_valores(4, 3))
print(MiModulo.resta_valores(10, 9))
```

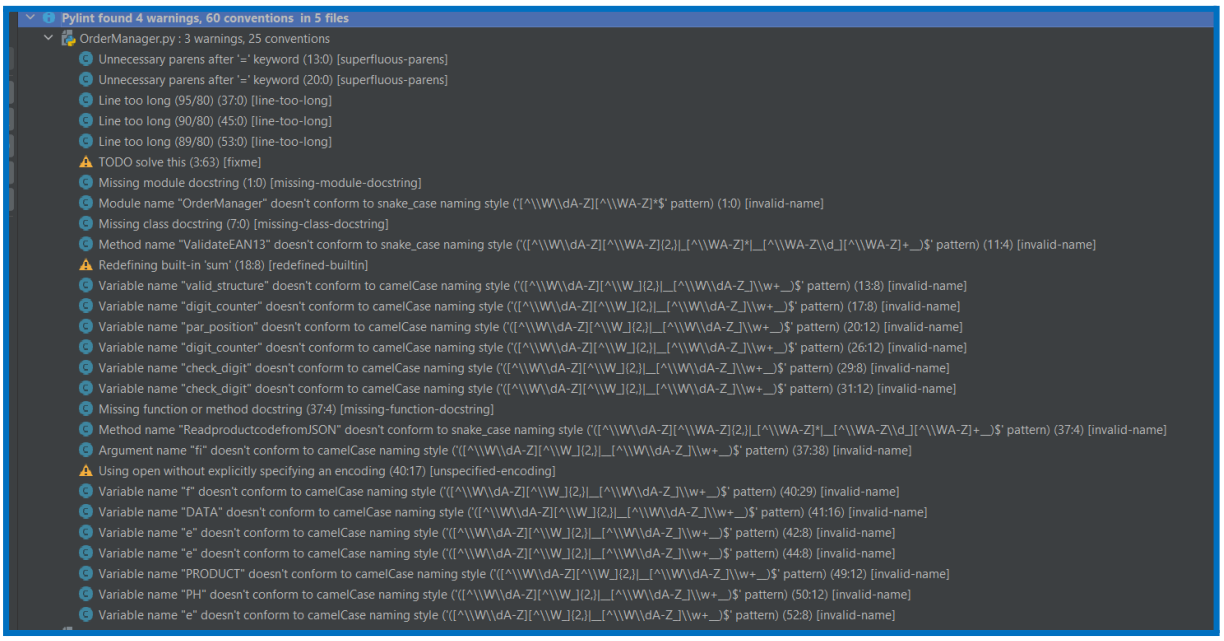
Se ha decidido establecer un estilo de escritura para los nombres de los módulos. El estilo será PascalCase.

WARNINGS PREVIA MODIFICACIÓN

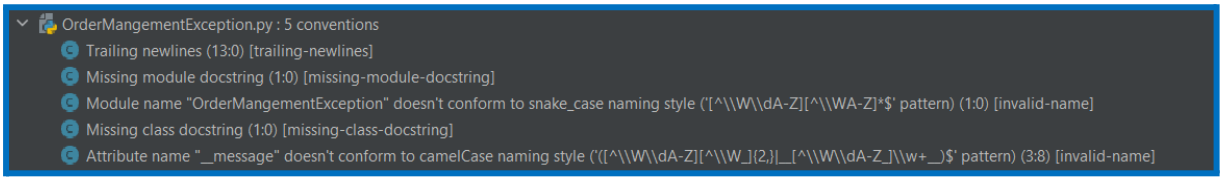
Main



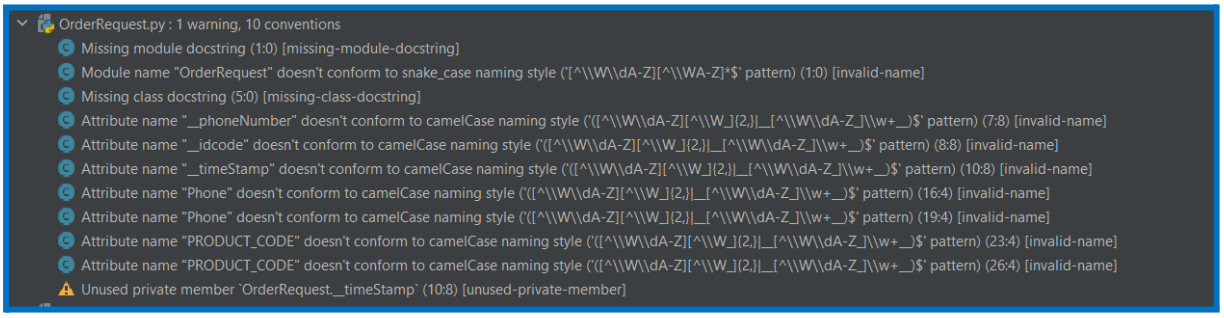
Order Manager



Order Management Exception



Order Request



WARNINGS POST MODIFICACIÓN

Main

```
(venv) C:\Users\bale2\PycharmProjects\DSoftware\G81.2023.T02.EG2>pylint Main.py

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


(venv) C:\Users\bale2\PycharmProjects\DSoftware\G81.2023.T02.EG2>
```

Order Manager

```
(venv) C:\Users\bale2\PycharmProjects\DSoftware\G81.2023.T02.EG2\UC3MLogistics>pylint OrderManager

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

Order Management Exception

```
(venv) C:\Users\bale2\PycharmProjects\DSoftware\G81.2023.T02.EG2\UC3MLogistics>pylint OrderMangementException

--------------------------------------------------------------------
Your code has been rated at 10.00/10 (previous run: 8.75/10, +1.25)
```

Order Request

```
(venv) C:\Users\bale2\PycharmProjects\DSoftware\G81.2023.T02.EG2\UC3MLogistics>pylint OrderRequest
************* Module OrderRequest
OrderRequest.py:19:8: W0238: Unused private member `OrderRequest.__timeStamp` (unused-private-member)

-------------------------------------------------------------------
Your code has been rated at 9.44/10 (previous run: 9.44/10, +0.00)
```