Universidad Carlos III de Madrid

Software Development

Guided Exercise 2 Writeup

Section C9.84.19637-1

Megana Atluri and Sarah Hoover

100504469 and 100504446

March 28, 2023

Table of Contents

1. Introduction

The purpose of this assignment was to practice pair programming and embrace the idea of collective code ownership. Additionally, this assignment allowed exploration of the coding standard, and allowed students to change at least fifteen of the providing coding standard rules. Through this practice, we learned the reasoning behind these rules, the importance of having a coding standard, and we were able to use Pylint to apply the original coding standard and our modifications.

The assignment was to develop a python program capable of detecting the validity of an EAN-13 barcode. A valid EAN-13 barcode is a thirteen-character digit string, where the last digit is a check digit. The first twelve digits are summed, with even index characters receiving a weight of 3 and the odd index characters receiving a weight of 1. The check digit thirteenth character) must equal the amount necessary to round the weighted sum to the next multiple of 10. For example, if the weighted sum is 83, then the check digit must equal 7.

2. Rule Changes

This section includes the rule changes made in the .pylintrc file and examples of these changes on PyCharm.

Naming Conventions

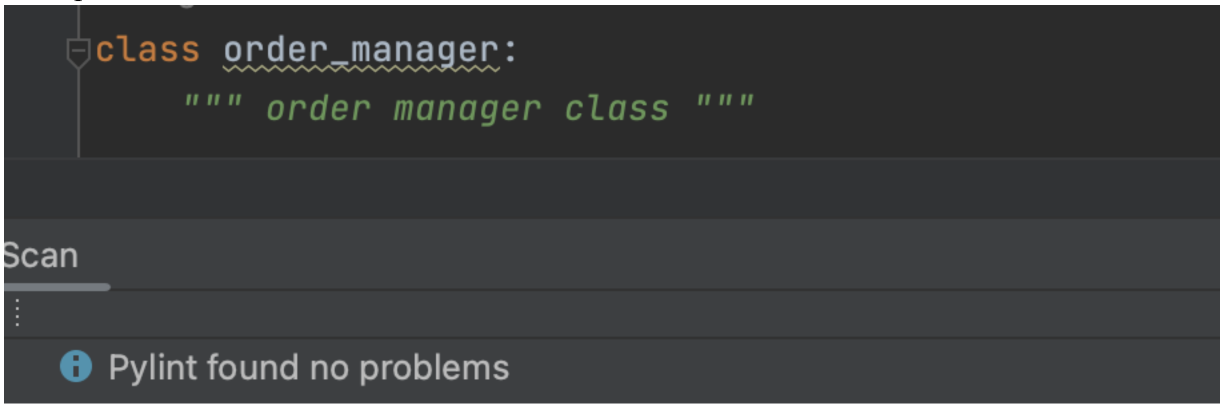
The following rules of the coding standard are in regards to naming conventions. These rules include naming styles, variable names, and naming format.

Naming style matching correct class names.

We changed this rule to ensure that the name matching style is snake_case,

```
# CHANGED CODING STANDARD
# Naming style matching correct class names.
class-naming-style=snake_case
```

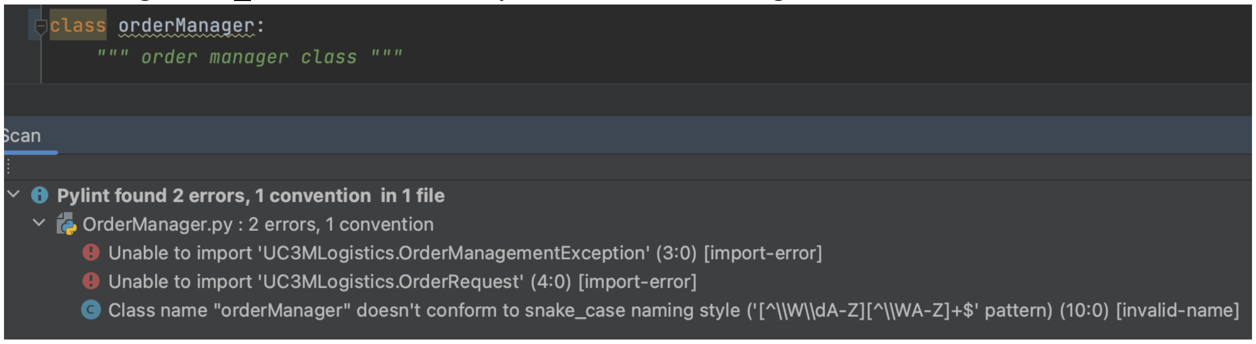
Examples:



```
class order_manager:
    """ order manager class """
```

```
Scan
Pylint found no problems
```

When using snake_case, there are no Pylint errors or warnings related to the class name.

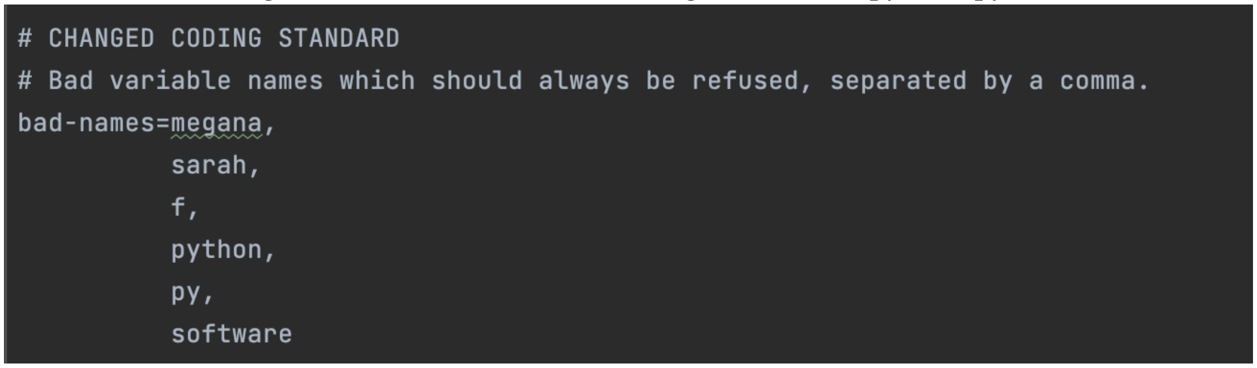


```
class orderManager:
    """ order manager class """
```

```
Scan
Pylint found 2 errors, 1 convention  in 1 file
  OrderManager.py : 2 errors, 1 convention
    Unable to import 'UC3MLogistics.OrderManagementException' (3:0) [import-error]
    Unable to import 'UC3MLogistics.OrderRequest' (4:0) [import-error]
    Class name "orderManager" doesn't conform to snake_case naming style ('[^\\W\\dA-Z][^\\WA-Z]+$' pattern) (10:0) [invalid-name]
```

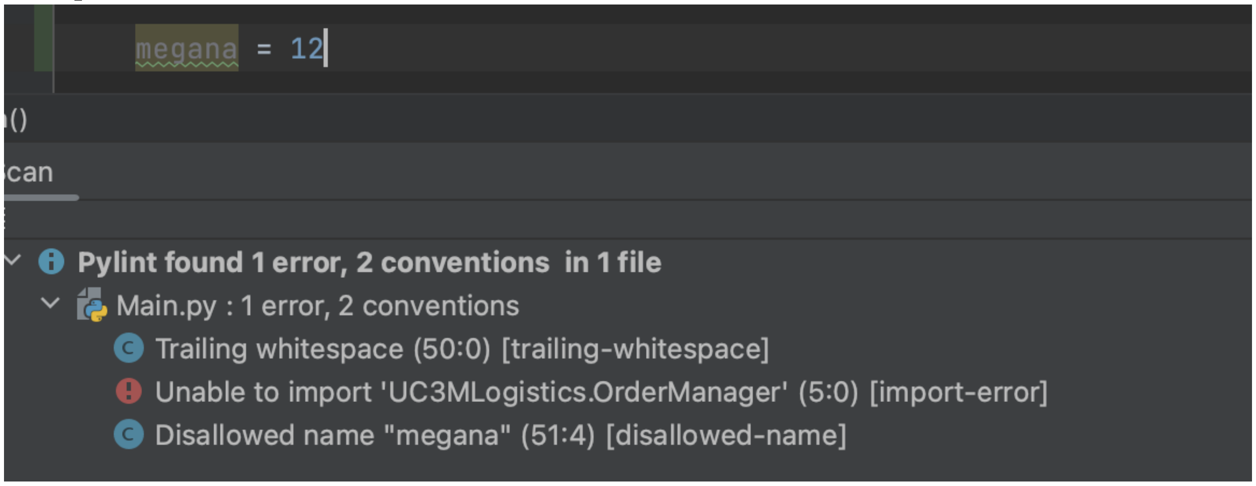
However, when the class name is changed to be in PascalCase, there is an conforming error found as the third line under OrderManager.py in the pylint file (the convention violation line).

Bad variable names which should always be refused, separated by a comma.
We modified the illegal variable names to include megana, sarah, f, python, py, and software



```
# CHANGED CODING STANDARD
# Bad variable names which should always be refused, separated by a comma.
bad-names=megana,
          sarah,
          f,
          python,
          py,
          software
```

Example:

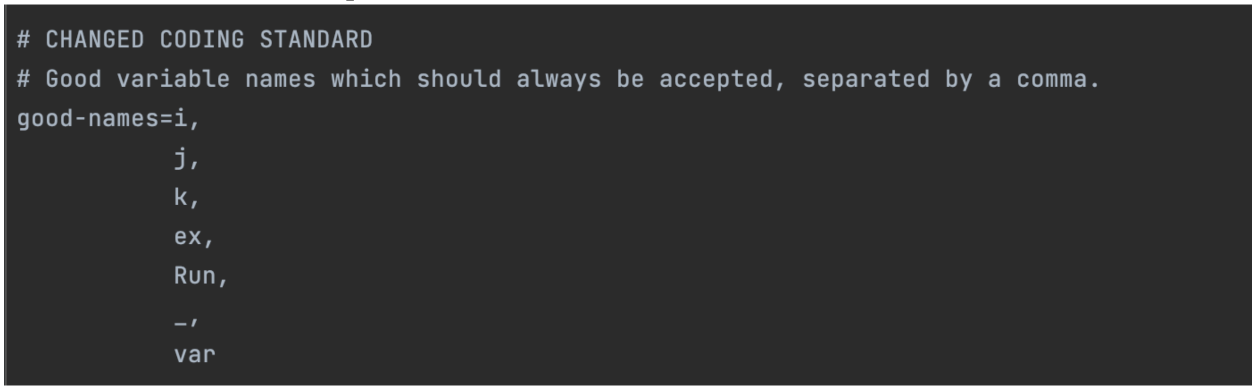


```
    megana = 12
```

```
()
can
Pylint found 1 error, 2 conventions  in 1 file
  Main.py : 1 error, 2 conventions
    Trailing whitespace (50:0) [trailing-whitespace]
    Unable to import 'UC3MLogistics.OrderManager' (5:0) [import-error]
    Disallowed name "megana" (51:4) [disallowed-name]
```

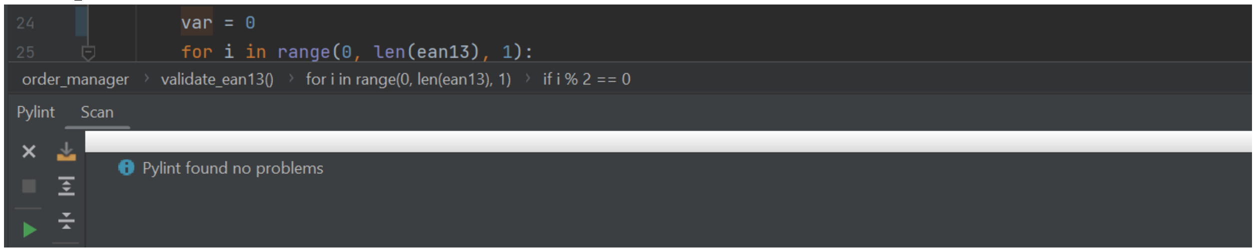
Focusing specifically on the third line of the Pylint errors and conventions, it is clear that “megana” is an illegal name for a variable.

Good variable names which should always be accepted, separated by a comma.
We included var as an acceptable name of a variable in this rule.



```
# CHANGED CODING STANDARD
# Good variable names which should always be accepted, separated by a comma.
good-names=i,
           j,
           k,
           ex,
           Run,
           _,
           var
```

Example:



```
        var = 0
        for i in range(0, len(ean13), 1):
```

order_manager › validate_ean13() › for i in range(0, len(ean13), 1) › if i % 2 == 0

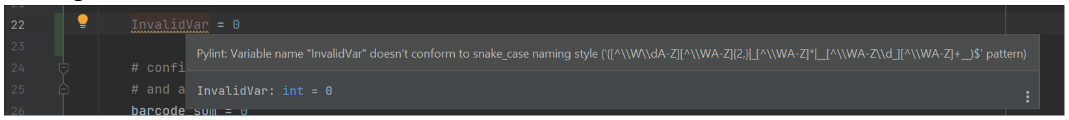
```
Pylint found no problems
```

Var is accepted as a variable name.

Include a hint for the correct naming format with invalid-name.
We changed this rule in order to include a naming format hint.

```
# CHANGED CODING STANDARD
# Include a hint for the correct naming format with invalid-name.
include-naming-hint=yes
```

Example:



```
InvalidVar = 0

# confi
# and a
barcode_sum = 0
```

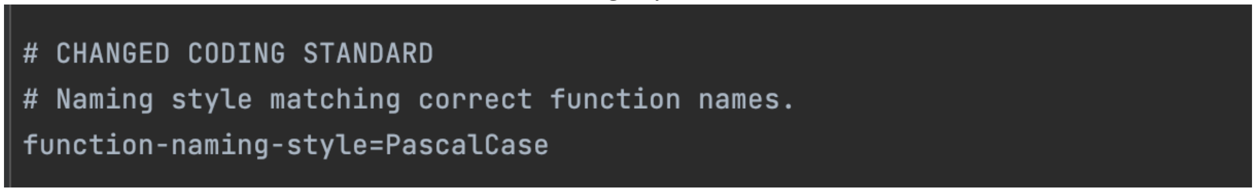
Pylint: Variable name "InvalidVar" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{2,}|_[^\\WA-Z]*|__[^\\WA-Z\\d_][^\\WA-Z]+__)$' pattern)

```
InvalidVar: int = 0
```

This is an example of an invalid variable name and Pycharm has given the user a hint on how to properly name the variable.

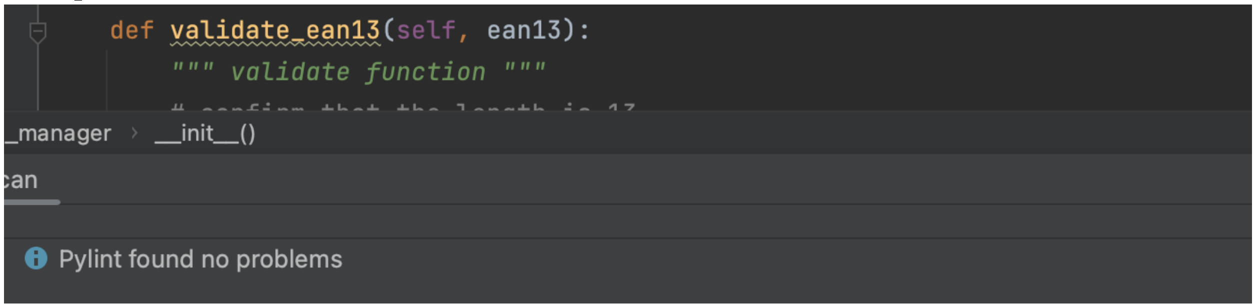
Naming style matching correct function names.

We modified this rule to set the function-naming-style to be PascalCase.



```
# CHANGED CODING STANDARD
# Naming style matching correct function names.
function-naming-style=PascalCase
```

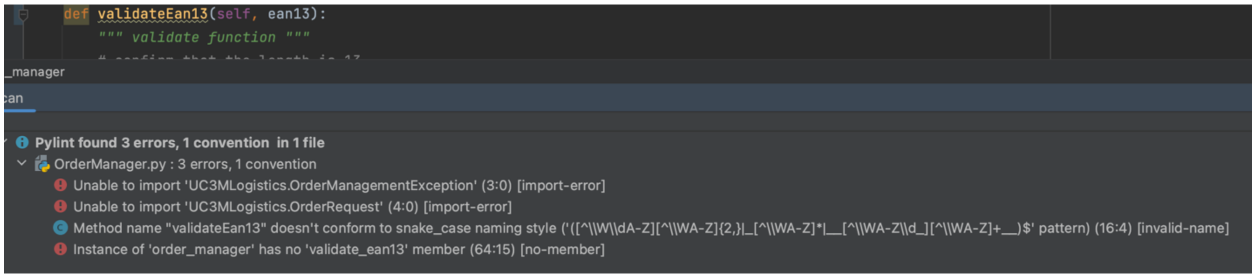
Example:



```
    def validate_ean13(self, ean13):
        """ validate function """
```

```
_manager › __init__()
can

Pylint found no problems
```

Obviously, Pylint has no problems with the naming convention of snake_case related to function names.



```
    def validateEan13(self, ean13):
        """ validate function """
```

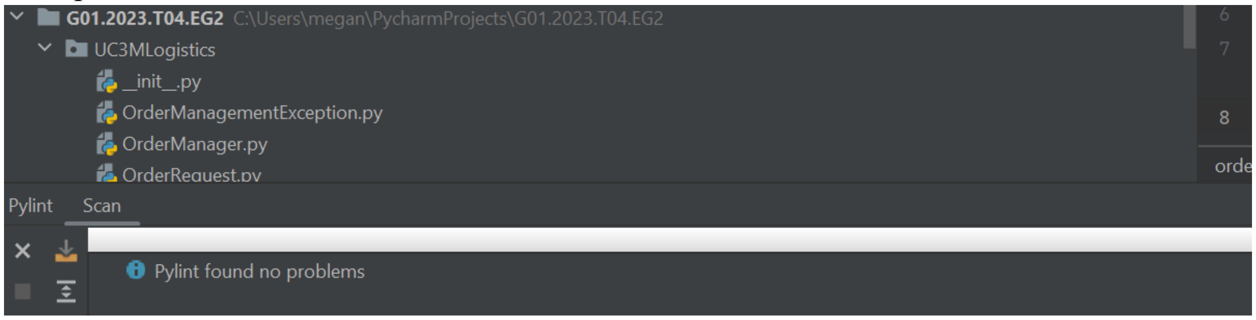
```
_manager
can

Pylint found 3 errors, 1 convention  in 1 file
  OrderManager.py : 3 errors, 1 convention
    Unable to import 'UC3MLogistics.OrderManagementException' (3:0) [import-error]
    Unable to import 'UC3MLogistics.OrderRequest' (4:0) [import-error]
    Method name "validateEan13" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{2,}|_[^\\WA-Z]*|__[^\\WA-Z\\d_][^\\WA-Z]+__)$' pattern) (16:4) [invalid-name]
    Instance of 'order_manager' has no 'validate_ean13' member (64:15) [no-member]
```

Focusing specifically on the third line of the Pylint file under the OrderManager.py, Pylint states that a convention is violated, specifically that validateEan13 does not conform to snake_case naming style.

Naming style matching correct module names.
We modified this rule to set the module-naming-style to be PascalCase.

```
# CHANGED CODING STANDARD
# Naming style matching correct module names.
module-naming-style=PascalCase
```

Example:



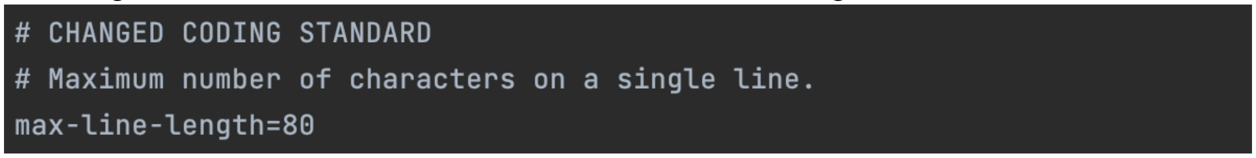
```
G01.2023.T04.EG2  C:\Users\megan\PycharmProjects\G01.2023.T04.EG2
  UC3MLogistics
    __init__.py
    OrderManagementException.py
    OrderManager.py
    OrderRequest.py
```

```
Pylint found no problems
```

Module name is in PascalCase; therefore, Pylint found no errors.

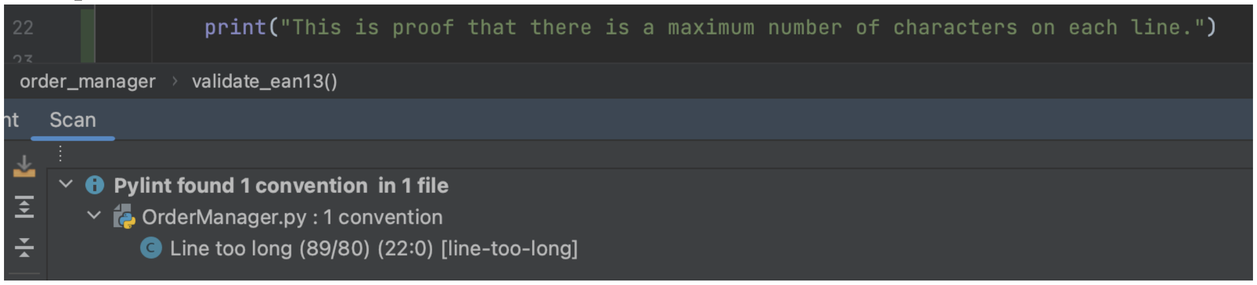
Format
The following rules of the coding standard are in regards to the code's format.

Maximum number of characters on a single line.
We changed the minimum number of characters allowed on a single line to be 80 instead of 100.



```
# CHANGED CODING STANDARD
# Maximum number of characters on a single line.
max-line-length=80
```

Example:

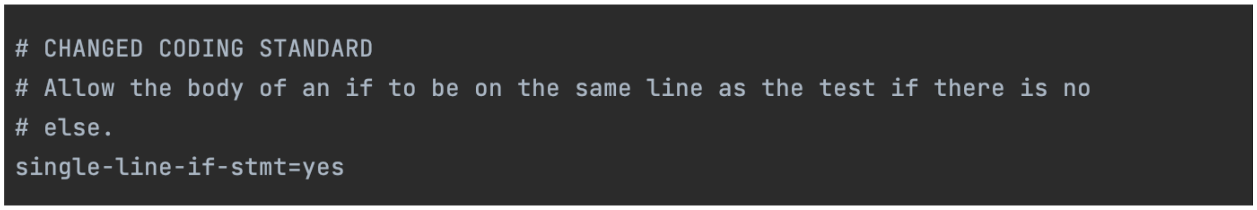


```
22        print("This is proof that there is a maximum number of characters on each line.")
```

```
Pylint found 1 convention in 1 file
  OrderManager.py : 1 convention
    Line too long (89/80) (22:0) [line-too-long]
```

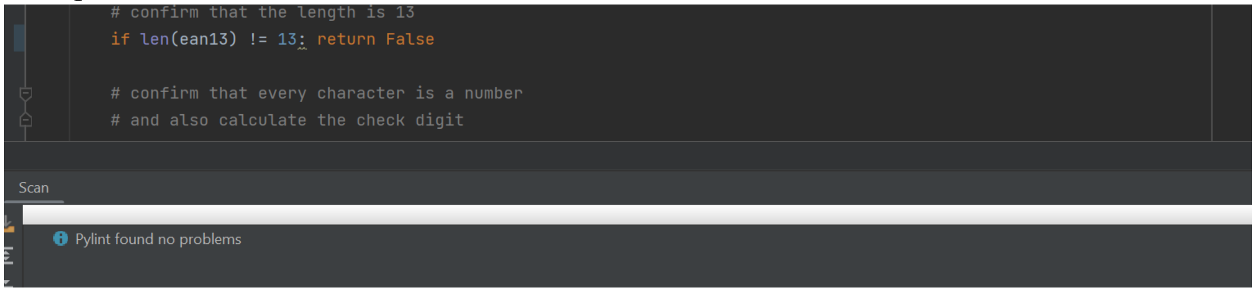
Obviously, line 22 is a violation of the line length rule.

Allow the body of an if to be on the same line as the test if there is no else.
We modified this rule to allow the body of the if statement to be on the same line as the "if" in the event that there is no else block.



```
# CHANGED CODING STANDARD
# Allow the body of an if to be on the same line as the test if there is no
# else.
single-line-if-stmt=yes
```

Example:



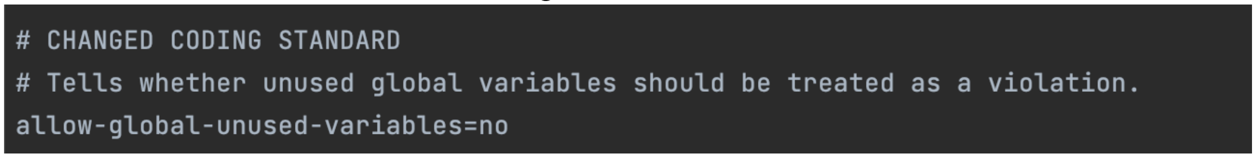
```
    # confirm that the length is 13
    if len(ean13) != 13: return False

    # confirm that every character is a number
    # and also calculate the check digit
```

```
Pylint found no problems
```

Here is an example of the body of an if statement being on the same line as the "if" because there is no else block.

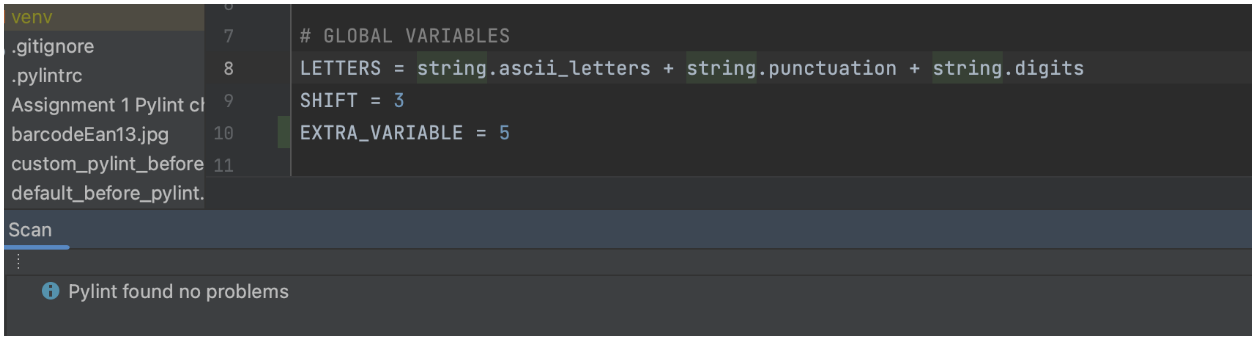
Variables
The following rules of the coding standard are in regards to the variables in the code.

Tells whether unused global variables should be treated as a violation.
We decided to allow the use of an unused global variable and to not treat it as a violation.



```
# CHANGED CODING STANDARD
# Tells whether unused global variables should be treated as a violation.
allow-global-unused-variables=no
```

Example:



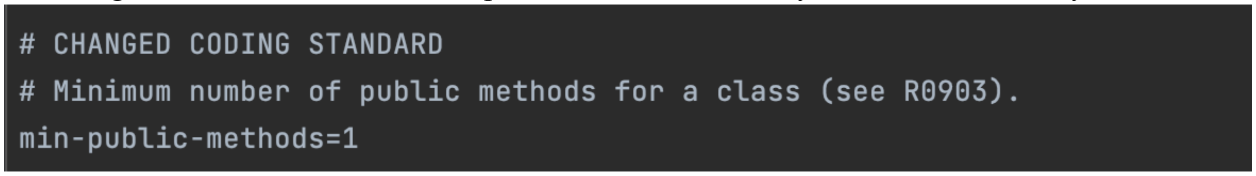
```
# GLOBAL VARIABLES
LETTERS = string.ascii_letters + string.punctuation + string.digits
SHIFT = 3
EXTRA_VARIABLE = 5
```

```
Pylint found no problems
```

Even though the EXTRA_VARIABLE global variable is completely unused within the code, Pylint has no problems and raises no issues related to this variable.

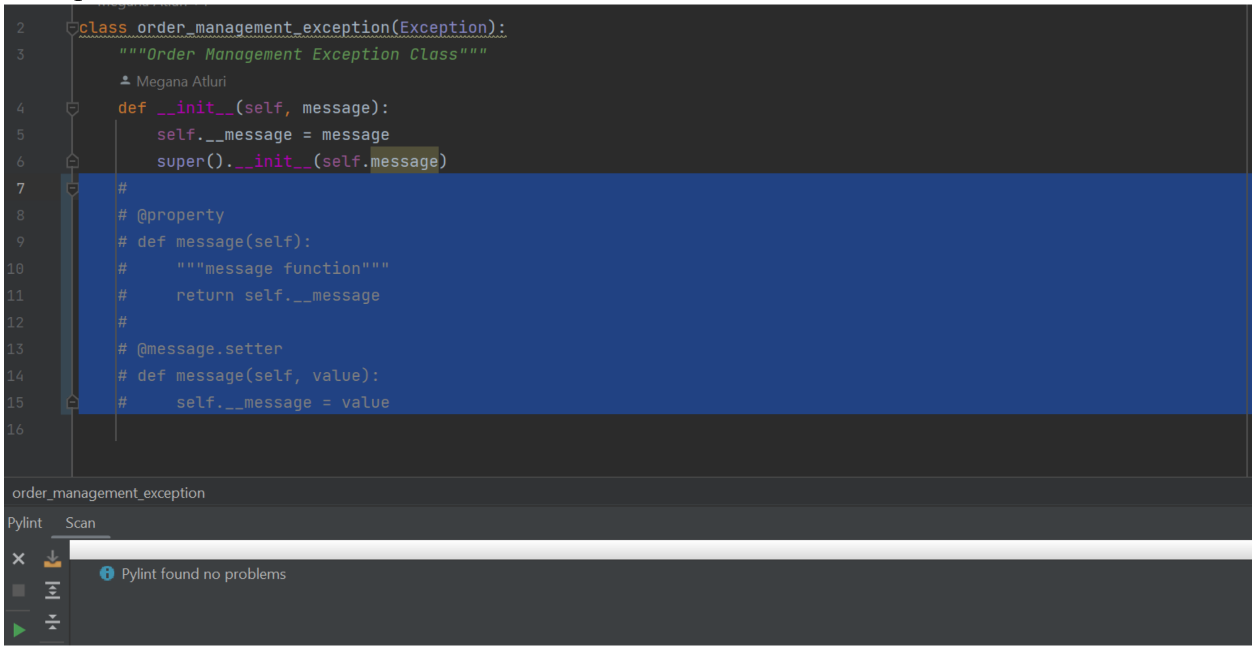
Design
The following rules of the coding standard are in regards to the design of the program.

Minimum number of public methods for a class (see R0903).
We changed the minimum number of public methods necessary for a class to be only 1.



```
# CHANGED CODING STANDARD
# Minimum number of public methods for a class (see R0903).
min-public-methods=1
```

Example:



```
class order_management_exception(Exception):
    """Order Management Exception Class"""
    Megana Atluri
    def __init__(self, message):
        self.__message = message
        super().__init__(self.message)
    #
    # @property
    # def message(self):
    #     """message function"""
    #     return self.__message
    #
    # @message.setter
    # def message(self, value):
    #     self.__message = value
```

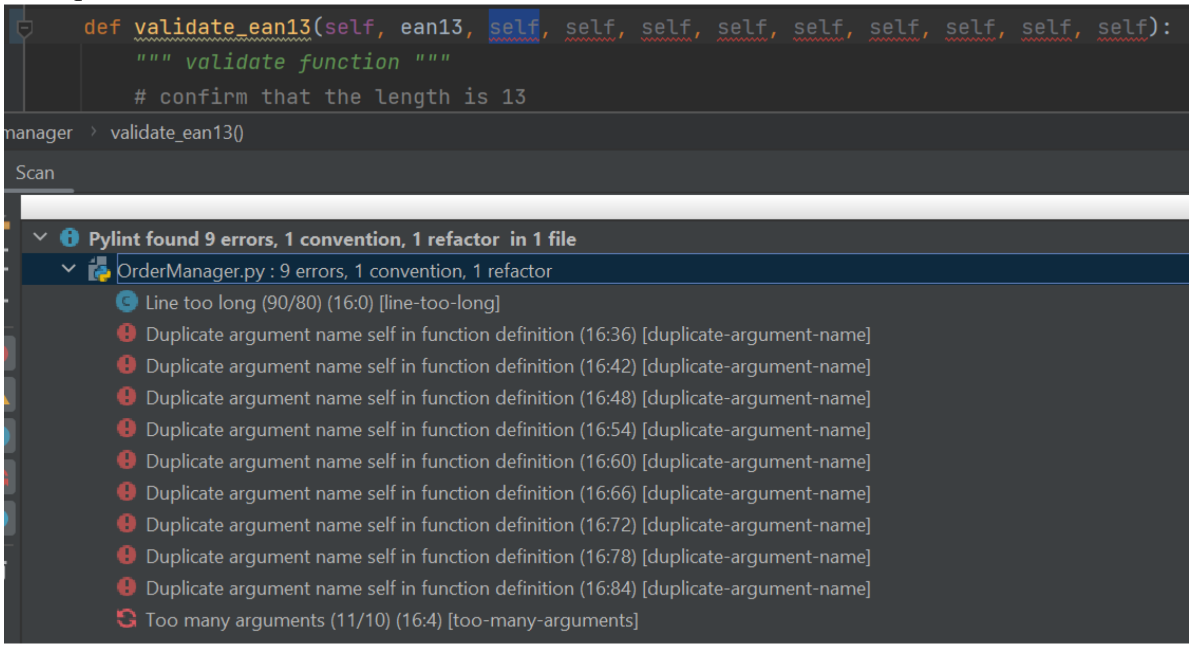
```
order_management_exception
Pylint    Scan
Pylint found no problems
```

There is only one public method in this class, and Pylint does not find any errors.

Maximum number of arguments for function / method.
We decided to increase the maximum allowed number of arguments for a function or method to 10.

```
# CHANGED CODING STANDARD
# Maximum number of arguments for function / method.
max-args=10
```

Example:



```
def validate_ean13(self, ean13, self, self, self, self, self, self, self, self, self):
    """ validate function """
    # confirm that the length is 13
```

manager › validate_ean13()

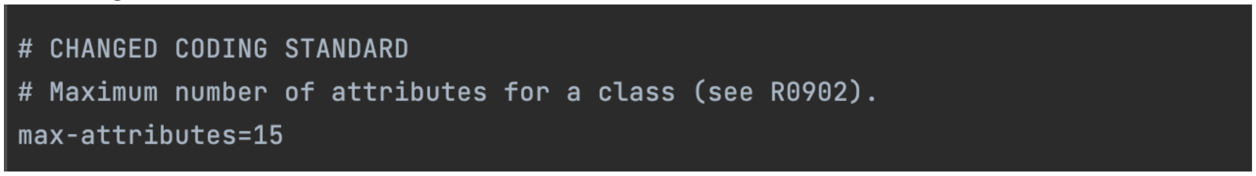
Scan

```
Pylint found 9 errors, 1 convention, 1 refactor  in 1 file
  OrderManager.py : 9 errors, 1 convention, 1 refactor
    Line too long (90/80) (16:0) [line-too-long]
    Duplicate argument name self in function definition (16:36) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:42) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:48) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:54) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:60) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:66) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:72) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:78) [duplicate-argument-name]
    Duplicate argument name self in function definition (16:84) [duplicate-argument-name]
    Too many arguments (11/10) (16:4) [too-many-arguments]
```

The last line of this screenshot shows how Pylint throws an error in regards to the number of arguments in the function when there are 11 arguments.

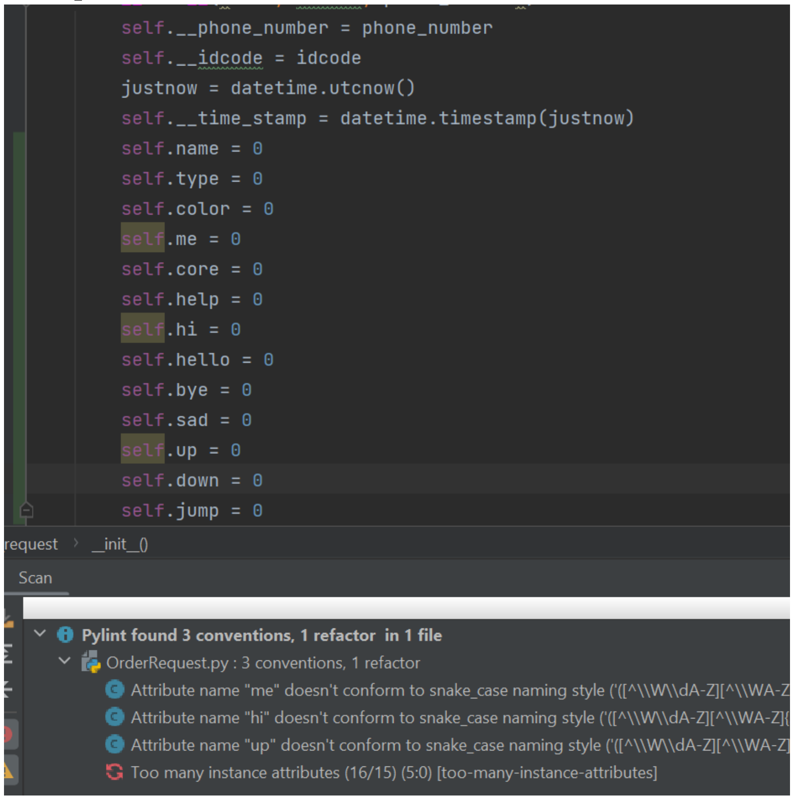
Maximum number of attributes for a class (see R0902).

We changed the maximum number of attributes for a class to 15.



```
# CHANGED CODING STANDARD
# Maximum number of attributes for a class (see R0902).
max-attributes=15
```

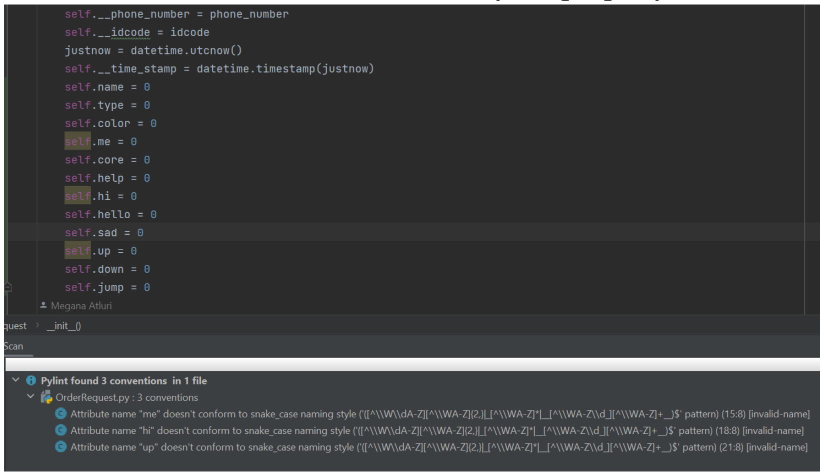
Example:



```
        self.__phone_number = phone_number
        self.__idcode = idcode
        justnow = datetime.utcnow()
        self.__time_stamp = datetime.timestamp(justnow)
        self.name = 0
        self.type = 0
        self.color = 0
        self.me = 0
        self.core = 0
        self.help = 0
        self.hi = 0
        self.hello = 0
        self.bye = 0
        self.sad = 0
        self.up = 0
        self.down = 0
        self.jump = 0
```

```
Pylint found 3 conventions, 1 refactor  in 1 file
  OrderRequest.py : 3 conventions, 1 refactor
    Attribute name "me" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z
    Attribute name "hi" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{
    Attribute name "up" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]
    Too many instance attributes (16/15) (5:0) [too-many-instance-attributes]
```

Here there are 16 instance attributes. Pylint properly throws an error

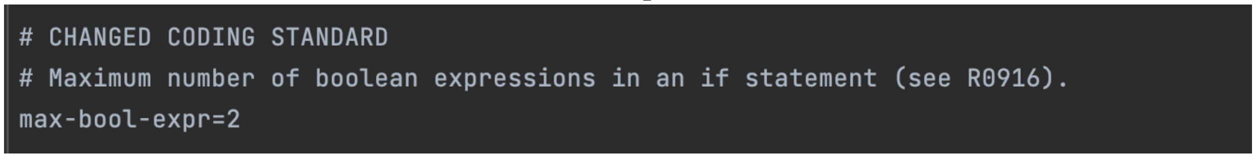


```
        self.__phone_number = phone_number
        self.__idcode = idcode
        justnow = datetime.utcnow()
        self.__time_stamp = datetime.timestamp(justnow)
        self.name = 0
        self.type = 0
        self.color = 0
        self.me = 0
        self.core = 0
        self.help = 0
        self.hi = 0
        self.hello = 0
        self.sad = 0
        self.up = 0
        self.down = 0
        self.jump = 0
```

```
Pylint found 3 conventions  in 1 file
  OrderRequest.py : 3 conventions
    Attribute name "me" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{2,}|_[^\\WA-Z]*|__[^\\WA-Z\\d_][^\\WA-Z]+__)$' pattern) (15:8) [invalid-name]
    Attribute name "hi" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{2,}|_[^\\WA-Z]*|__[^\\WA-Z\\d_][^\\WA-Z]+__)$' pattern) (18:8) [invalid-name]
    Attribute name "up" doesn't conform to snake_case naming style ('([^\\W\\dA-Z][^\\WA-Z]{2,}|_[^\\WA-Z]*|__[^\\WA-Z\\d_][^\\WA-Z]+__)$' pattern) (21:8) [invalid-name]
```

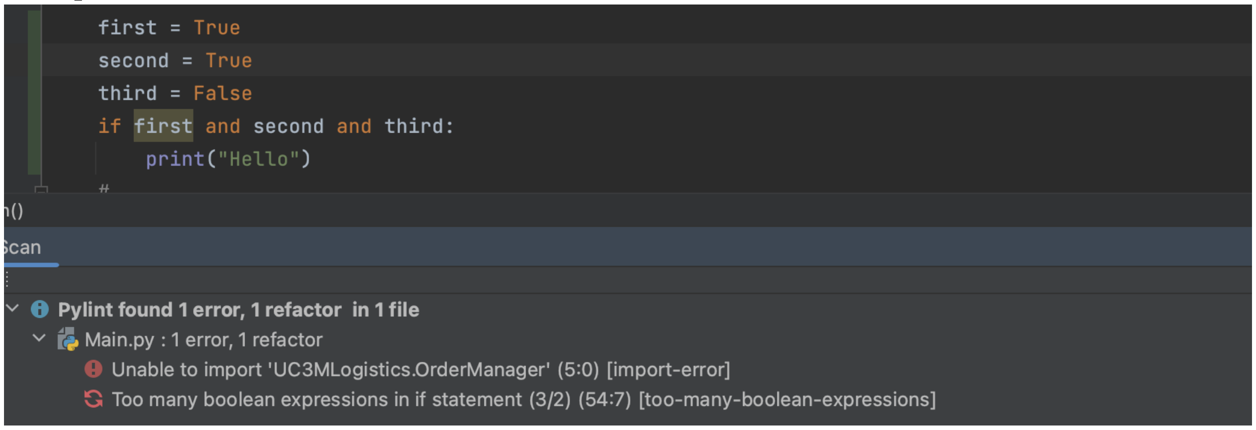
When there are 15 attributes Pylint does not throw an error.

Maximum number of boolean expressions in an if statement (see R0916).
We modified the maximum number of Boolean expressions allowed in an if statement to.



```
# CHANGED CODING STANDARD
# Maximum number of boolean expressions in an if statement (see R0916).
max-bool-expr=2
```

Example:



```
    first = True
    second = True
    third = False
    if first and second and third:
        print("Hello")
```

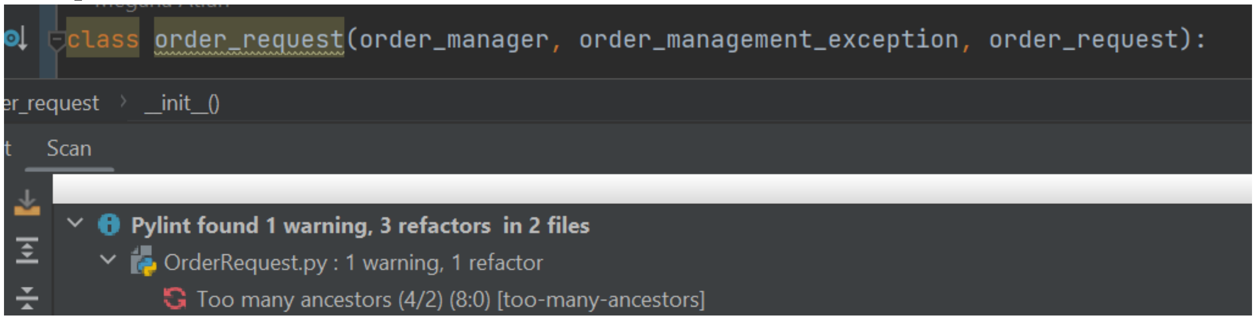
```
Pylint found 1 error, 1 refactor  in 1 file
  Main.py : 1 error, 1 refactor
    Unable to import 'UC3MLogistics.OrderManager' (5:0) [import-error]
    Too many boolean expressions in if statement (3/2) (54:7) [too-many-boolean-expressions]
```

Focusing on the second error line in the Pylint file, there are too many Boolean expressions in the specified if statement, meaning that there are 3 instead of 2 (the maximum).

Maximum number of parents for a class (see R0901).
We changed the maximum number of parents for a class to 2.

```
# CHANGED CODING STANDARD
# Maximum number of parents for a class (see R0901).
max-parents=2
```

Example:



```
class order_request(order_manager, order_management_exception, order_request):
```

er_request › __init__()

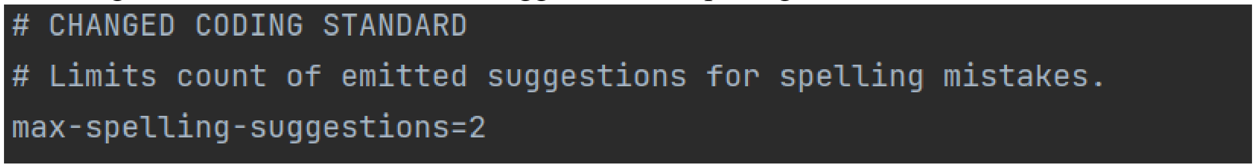
Scan

```
Pylint found 1 warning, 3 refactors  in 2 files
  OrderRequest.py : 1 warning, 1 refactor
    Too many ancestors (4/2) (8:0) [too-many-ancestors]
```

Pylint throws an error when a class has three parents.

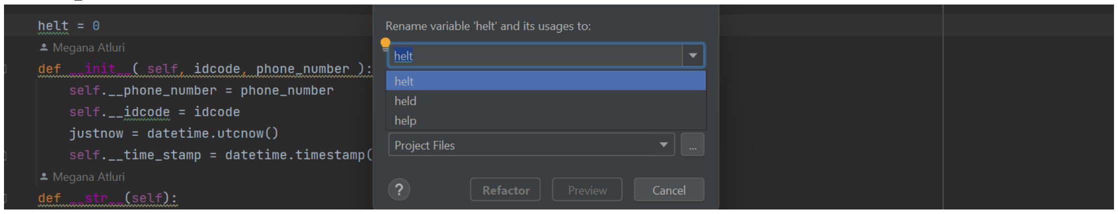
Spelling
The following rules of the coding standard are in regards to spelling.

Limits count of emitted suggestions for spelling mistakes.
We changed the maximum number of suggestions for spelling mistakes to be 2.



```
# CHANGED CODING STANDARD
# Limits count of emitted suggestions for spelling mistakes.
max-spelling-suggestions=2
```

Example:



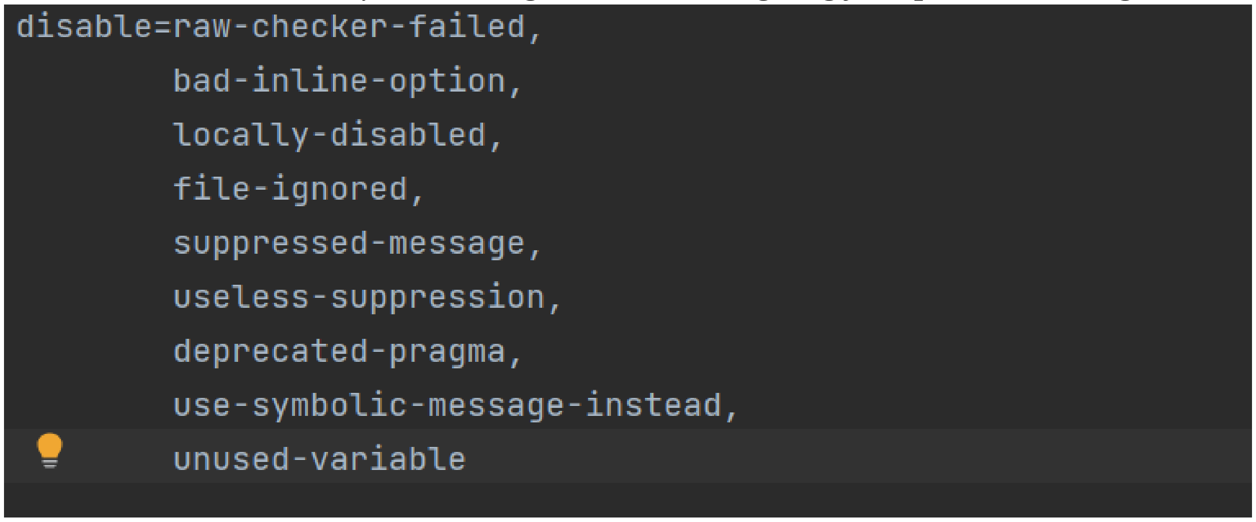
```
helt = 0
Megana Atluri
def __init__( self, idcode, phone_number ):
    self.__phone_number = phone_number
    self.__idcode = idcode
    justnow = datetime.utcnow()
    self.__time_stamp = datetime.timestamp(
Megana Atluri
def __str__(self):
```

Rename variable 'helt' and its usages to:

Here you can see there are two spelling suggestions for helt.

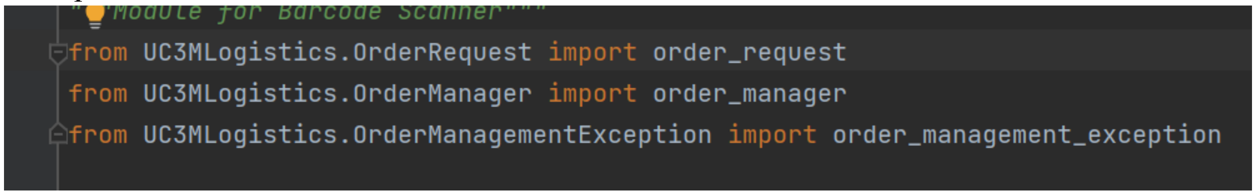
Messages Control
The following rules of the coding standard are in regards to message control.

Disabled unused-variable Pylint warnings because it was giving false positive warnings.



```
disable=raw-checker-failed,
        bad-inline-option,
        locally-disabled,
        file-ignored,
        suppressed-message,
        useless-suppression,
        deprecated-pragma,
        use-symbolic-message-instead,
        unused-variable
```

Example:



```
"""Module for Barcode Scanner"""
from UC3MLogistics.OrderRequest import order_request
from UC3MLogistics.OrderManager import order_manager
from UC3MLogistics.OrderManagementException import order_management_exception
```

This screenshot shows the code which was first getting caught by Pylint. This is why we decided to update the disable rule in the .pylintrc.

3. Conclusion

This assignment was designed to focus on two key aspects of software development, especially group software development: pair programming and collective code ownership, and coding standards. Through the program development of this project, we learned to work together as a team and gained hands-on experience both as drivers and observers. Through the modification of the existing coding standards we were able to explore the importance of documentation and also the tool of Pylint.