Coding Standard Report

José María Duro Agea - NIA: 100522306

100522306@alumnos.uc3m.es

Juan Macías Romero - NIA: 100472858

100472858@alumnos.uc3m.es

Grado en Ingeniería Informática

Project G89.2026.T10.GE1

Introduction

This document defines the specific deviations from the PEP8 standard that we have introduced into our project. These modifications were done in the aims of giving the project a shared technical language that is more adapted to its specific needs. To ensure that the new standards are met consistently, we modified the pylintrc file to include the modifications chosen.

Rule Documentation

[MAIN] - Environment & Analysis

Source-roots

- **Description:** This parameter defines the root directory for the imports used in the files of the project
- **Configuration:** "source-roots=."
- **Justification:** It allows pylint to properly resolve the UC3MConsulting package from the project root, preventing problems with the import-error messages.

[MESSAGES CONTROL] - Rule Overrides

C0103: invalid name

- **Description:** It enforces a specific naming convention for variables and functions
- **Action:** Disabled
- **Justification:** It allows naming of variables like "x" or "y" that otherwise would be considered too short, or the use of camelCase names if necessary.
- **Negative example:** cif_validation_result = True
- **Positive example:** CIFResult = True

C0114, C0115, C0116: missing docstrings

- **Description:** Documentation strings are necessary for modules, classes and functions
- **Action:** Disabled
- **Justification:** It reduces the cluttering created by innecessary dosctrings for each module, class and function, making them optional.
- **Negative example:** "class EnterpriseRequest:" (would give an error for no usage of docstring)
- **Positive example:** "class EnterpriseRequest:" (doesn't raise errors)

C0304: missing-final-newline

- **Description:** It requires a blank line at the end of the files
- **Action:** Disabled
- **Justification:** Prevents errors for simple visibility decisions

C0121: singleton-comparison

- **Description:** Checks for comparisons to singleton objects lije “True” or “False” or “None”
- **Action:** Disabled
- **Justification:** Allows the use of explicit comparisons like “==True” for better readability
- **Negative example:** “if CIFResult:”
- **Positive example:** “if CIFResult == True"

C0411: wrong-import-order

- **Description:** Enforces alphabetical ordering and grouping of imports
- **Action:** Disabled
- **Justification:** Gives flexibility to the ordering of the imports
- **Negative example:** import json
  import datetime (raises error as they’re not in order)
- **Positive example:** import json
  import datetime (no error of ordering)

R1705: no-else-return

- **Description:** Prevents the use of an else after a return statement
- **Action:** Disabled
- **Justification:** Allows else to maintain visual symmetry and logical clarity when in a complex conditional statement
- **Negative example:** “if Condititon:
  return True
  return False” (else was forced to be removed)
- **Positive example:** “if Condititon:
  return True
  else:
  return False” (more symmetry with else)

C0325: superfluous-parens

- **Description:** Checks for unnecessary parentheses
- **Action:** Disabled
- **Justification:** Allows extra parentheses for mathematical functions or clarity
- **Negative example:** “(x*2)+2” (it would previously raise an error)
- **Positive example:** “(x*2)+2” (it doesn’t raise errors)

W0511: fixme

- **Description:** Doesn’t allow the use of notes such as TODO or FIXME
- **Action:** Disabled
- **Justification:** Allows the use of these notes so the developers can communicate with each other during the development process
- **Negative example:** “#TODO:” (This would raise an error)
- **Positive example:** “#TODO:” (It won’t raise errors)

W0613: unused-argument

- **Description:** Doesn’t allow unused arguments
- **Action:** Disabled
- **Justification:** Allows unsued arguments for later implementations
- **Negative example:** “def validate(self, data, context):
  return self.check(data)” (Error: context is unused)
- **Positive example:** “def validate(self, data, context):
  return self.check(data)” (no error)

W0238: unused-private-member

- **Description:** Doesn’t allow private arguments to be defined and unread
- **Action:** Disabled
- **Justification:** Allows for attributes like timestamps for future logging
- **Negative example:** “def __init__(self):
  self.__internal_id=123” (Error if never read)
- **Positive example:** “def __init__(self):
  self.__internal_id=123” (Stored for later audit)

[DESIGN] - Structural Limits

R0914: max-locals

- **Description:** sets the maximum number of local variables in a function
- **Configuration: "**max-locals=25"
- **Justification:** The function that validates the CIF has more than 7 steps that require many intermediate variables and we don't want to reuse variables or to have to refactor functions in many more small ones

R0913: max-args

- **Description:** Limits the number of arguments in a function
- **Configuration:** "max-args=8"
- **Justification:** Necessary for business classes like EnterpriseRequest
- **Negative example:** "def __init__(self, a, b, c, d, e, f):" (exceeds limit of 5)
- **Positive example:** "def __init__(self, cif, phone, name, user, status):"

R0903: min-public-methods

- **Description:** Requires a minimum number of public methods for a class
- **Configuration:** "min-public-methods=0"
- **Justification:** Allows for Data Transfer Objects that store states of objects
- **Negative example:** "class Data:
  def __init__(self, val):
  self.val=val" (min public methods is 2)
- **Positive example:** "class EnterpriseManagementException(Exception):
  def __init__(self, message):
  self.__message=message"

[FORMAT] - Layout & Style

C0301: max-line-length

- **Description:** Limits the number of characters per line
- **Configuration:** "max-line-length=120"
- **Justification:** Allows the use of detailed legal comments, or the use of long mathematical strings without affecting readibility

Use of Generative AI

In this project, AI was mainly used during the brainstorming phase, where it was highly helpful in choosing which rules were the best to modify. Furthermore, we used AI during the debugging phase to clarify how to approach specific errors. Essentially, it was used as an advanced search tool to validate our ideas and to determine which was the best way to resolve the issues that came up during the development of this project.