Actividad 6.2 Ejercicios de Programación con Pruebas y Análisis Estático

Pruebas de software y aseguramiento de la calidad (Gpo 10)

Presenta:

A01796976 Lucía Guadalupe Macías Morales

Profesor: Dr. Gerardo Padilla Zárate

Tutor: Francisco Solorzano Domínguez

Nuevo León, México

Fecha: 19 de febrero del 2026

Reporte del ejercicio de programación con pruebas y análisis estático

Introducción

El objetivo de esta actividad es implementar un sistema a través del lenguaje Python, aplicar pruebas unitarias, seguir los estándares de codificación PEP-8 y validar el código con herramientas de análisis estático como Flake8 y Pylint. Lo anterior con el objetivo de reforzar las prácticas de calidad del software y los principios de gestión de la configuración.

Para esta práctica es importante contar con el correspondiente registro de "commits" en el repositorio.

Ejercicio 1. Reservation System

Req 1. Implement a set of classes in Python that implements two abstractions:

1. Hotel
2. Reservation
3. Customers

Req 2. Implement a set of methods to handle the next persistent behaviors (stored in files):

1. Hotels
    a. Create Hotel
    b. Delete Hotel
    c. Display Hotel information
    d. Modify Hotel Information
    e. Reserve a Room
    f. Cancel a Reservation
2. Customer
    a. Create Customer
    b. Delete a Customer
    c. Display Customer Information
    d. Modify Customer Information
3. Reservation
    a. Create a Reservation (Customer, Hotel)
    b. Cancel a Reservation

You are free to decide the attributes within each class that enable the required behavior.

Req 3. Implement unit test cases to exercise the methods in each class. Use the unittest module in Python.

Req 4. The code coverage for all unittests should accumulate at least 85% of line coverage.

Req 5. The program shall include the mechanism to handle invalid data in the file. Errors should be displayed in the console and the execution must continue.

Req 6. Be compliant with PEP8.

Req 7. The source code must show no warnings using Fleak and PyLint.

Descripción del sistema implementado

Se elaboró una aplicación de consola que funciona como un sistema de reservación, tomando en cuenta la programación orientada a objetos.

Clases implementadas

- Hotel
  - Identificador
  - Nombre
  - Ubicación
  - Número de habitaciones
- Cliente (Customer)
  - Identificador
  - Nombre
  - Correo
- Reservación (Reservation)
  - Identificador
  - Identificador del Cliente
  - Identificador del Hotel

Funcionalidades principales

- Creación, modificación y eliminación de hoteles
- Gestión de clientes
- Reservaciones y cancelaciones
- Persistencia en archivos JSON

Arquitectura del Proyecto

La estructura del proyecto generado quedo de la siguiente forma:

A01796976_A6.2/

.venv	Entorno aislado para ejecutar el proyecto
data	Carpeta de la estructura de los json.
tests/	Carpeta para las respectivas suites de pruebas.
tests_all.py	**-**Validar la totalidad de la lógica.
tests_customer.py	-Validar la lógica de creación y edición de clientes.
tests_hotel.py	**-**Validar la lógica de creación y edición de hoteles.
tests_reservation.py	**-**Validar la lógica de reservaciones.
customer.py	Clase para administrar hoteles y su persistencia
hotel.py	Clase para administrar clientes y su persistencia.
reservation.py	Clase para administrar reservaciones y su persistencia.
main.py	Programa principal (Main) para ejecutar el Sistema.
requirements.txt	Archivo de requisitos para las librerías

Para validar la funcionalidad del aplicativo realice algunas pruebas básicas de la consola.

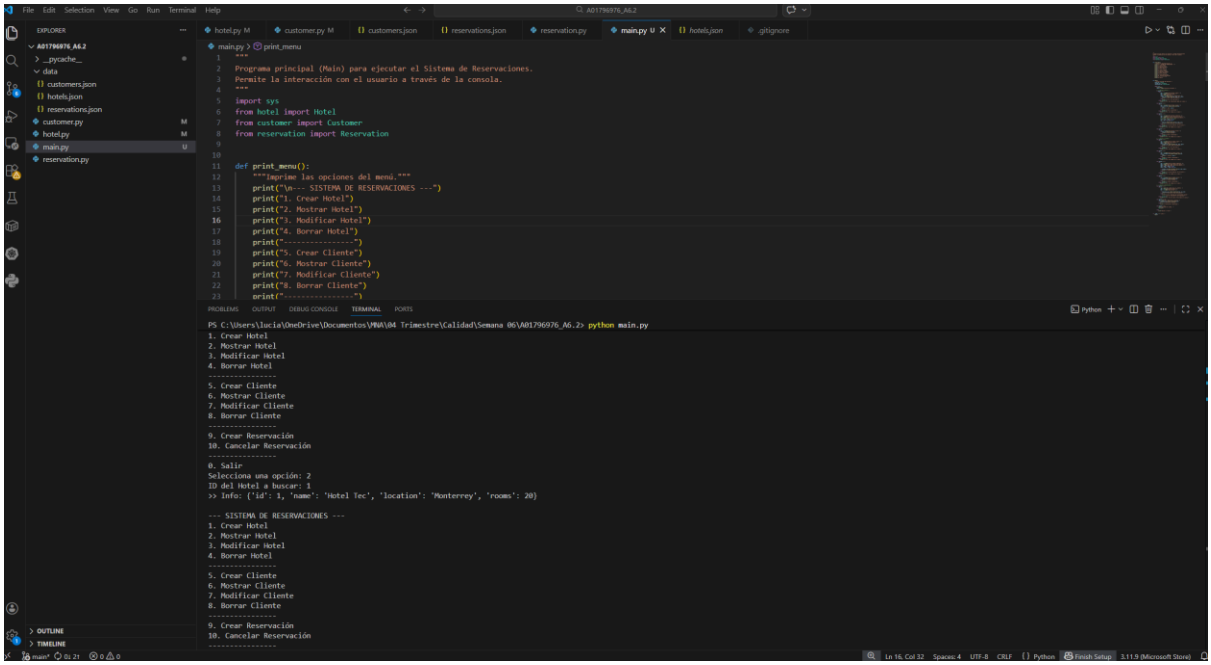

Ilustración 1. Ejecución de la aplicación

Pruebas Unitarias

Para la realización de las pruebas unitarias la estrategia es utilizar “unittest”, para probar ciclos completos (create, modify, delete) y revisar el manejo de errores y datos inválidos.

Además, se realizó la instalación de “Coverage”, implementándolo a través del bash.

```
pip install coverage
```

Una vez lista la instalación se ejecuta el comando de coverage run -m unittest y el generador del reporte para identificar las inconsistencias.

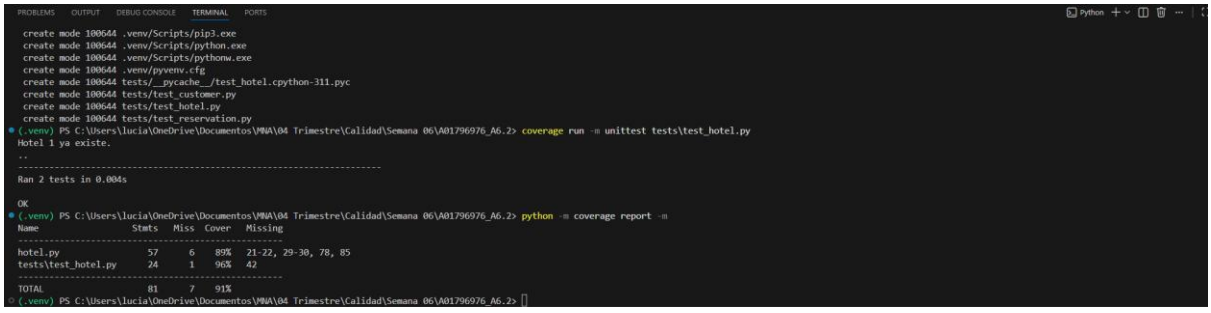

Ilustración 2. Ejecución de pruebas unitarias de la Clase Hotel

Realizando una investigación de las líneas "Missing", para poder corregirlas se muestran en la siguiente tabla.

Línea de Error	Código	Razón	Solución
21-22 (Error de lectura)	except (json.JSONDecodeError, IOError): return []	Las pruebas nunca leyeron un archivo corrupto o inválido.	Agregar una prueba donde crees un archivo hotels.json con texto basura (ej. "{esto no es json") y luego intentes leerlo.
29-30 (Error de escritura):	except IOError: print(...)	Las pruebas nunca fallaron al intentar *guardar* un archivo (ej. disco lleno o permisos denegados).	Es difícil de probar sin herramientas avanzadas (Mocks)
78 (Modificar ubicación):	hotel['location'] = location dentro de modify_hotel_info.	En las pruebas modifique el name (nombre), y el valor estaba mal instanciado.	Adecuar la función.
85 (Modificar no encontrado)	return False al final de modify_hotel_info.	No hay una prueba que intente modificar un hotel que no existe (ej. ID 999).	Crear un método.

Una vez identificado se procede a corroborar que se obtiene más del 85% de "Coverage"

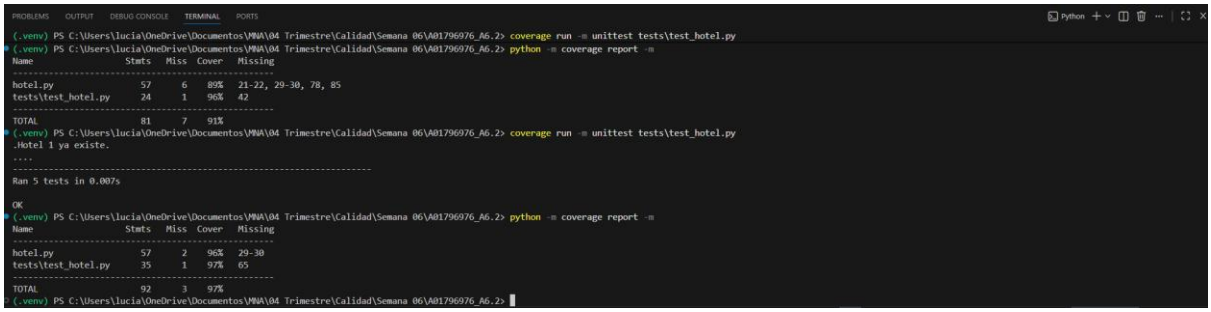

Ilustración 3. Segunda ejecución de la clase Hotel

Posteriormente se procede a la validación de la clase Cliente y se identificaron las líneas y al igual que con la clase Hotel se resuelven para obtener más del 85% de "Coverange"

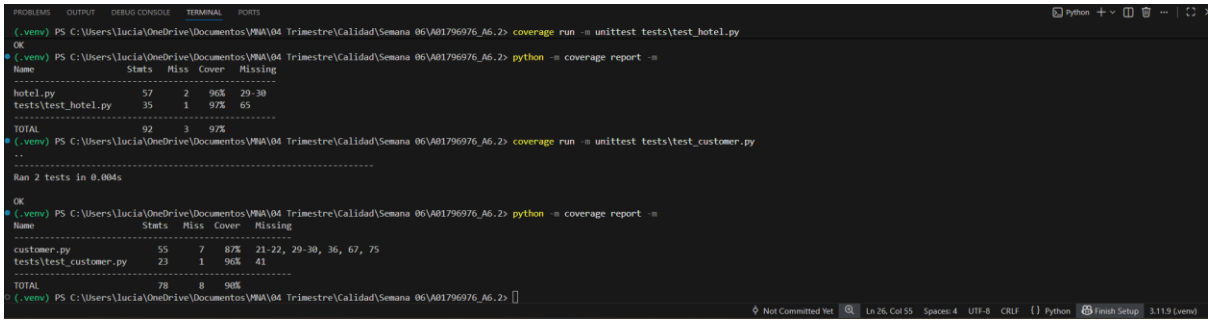

Ilustración 4. Primera ejecución de la clase Cliente

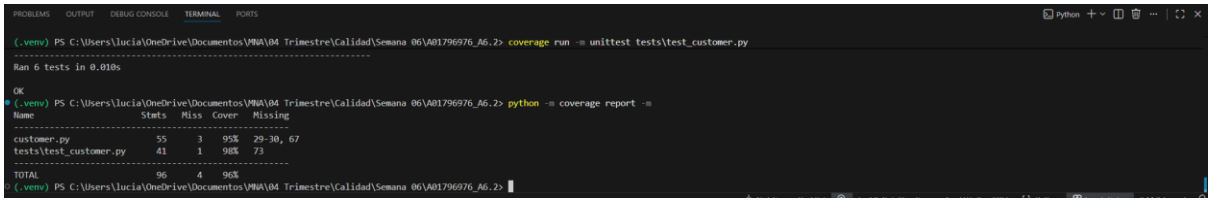

Ilustración 5.Segunda ejecución de la clase Cliente

Para el caso de reservación fue una situación particular

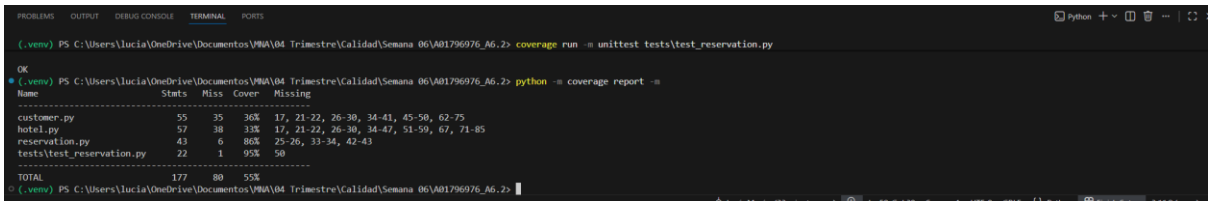

Ilustración 6. Ejecución de la clase Reservation

Debido a que reservation.py necesita instanciar a hotel.py y customer.py, y solo usa sus funciones de lectura/búsqueda, es decir nunca ejecuta las funciones de "Eliminar", "Modificar" o "Crear" de los otros archivos, por eso se obtiene una cobertura en esos archivos es del 50% ya que solo la mitad del código se usa para pruebas.

Es por lo anterior que para llegar a más del 85% opte por generar un archivo de pruebas denominado **test_all.py**, que ejecuta todo junto. Este archivo combina todas las pruebas necesarias (Hotel, Cliente y Reservación) y asegura que compartan los mismos archivos de prueba para evitar errores

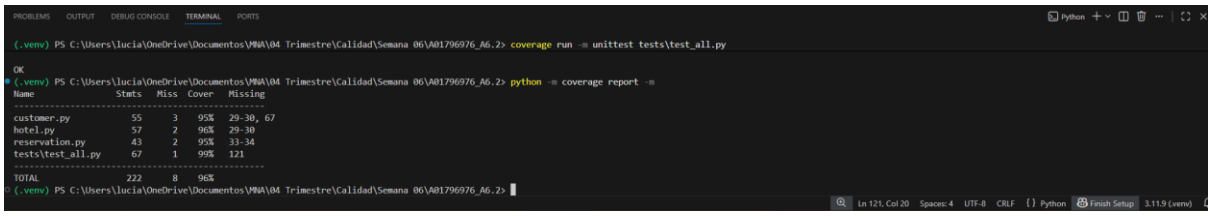

Ilustración 7. Ejecución de archivo de pruebas maestro

Análisis Estático del Código

Para poder realizar las pruebas estáticas y la verificación del código fuente contra el PEP8, se ejecuto el archivo de “requirements,txt” para la instalación de las librerías en el ambiente controlado.

```
pip install -r requirements.txt
```

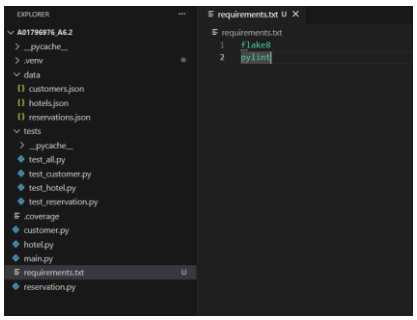

Ilustración 8. requirements.txt

Se inicio con la validación a través de Flake8

```
flake8 reservation.py tests\test_reservation.py
```

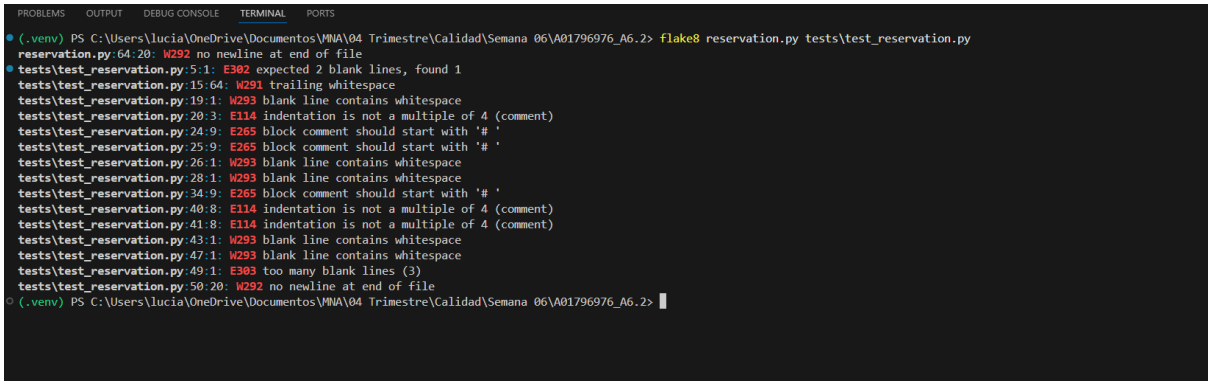

Ilustración 9. Validación de Clases referentes a reservación

La gran mayoría de las observaciones fueron por espacios y blancos, una vez subsanados se continuo con las otras clases.

Ilustración 10. Validación de Clases referentes a hotel

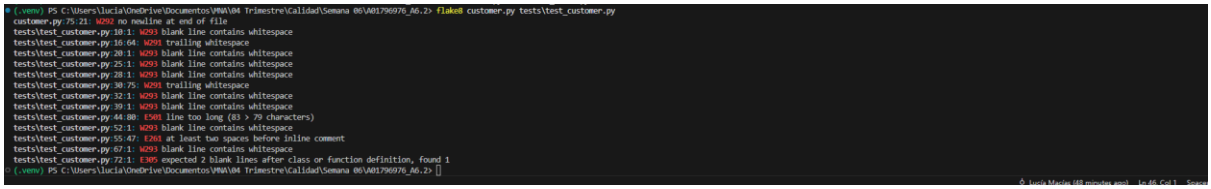

Ilustración 11.Validación de Clases referentes a clientes

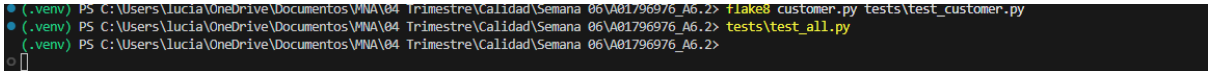

Ilustración 12. Validación de clase de archivo maestro de pruebas

Posteriormente se continuo con la validación a través del análisis con **Pylint**

```
pylint reservation.py tests\test_reservation.py
```

Principalmente se obtuvieron advertencias referentes a la documentación, que se solventaron añadiendo una breve descripción entre triples comillas """ al inicio del archivo y justo debajo de la definición de la clase o método.

```
(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint tests\test_all.py
************* Module test_all
tests\test_all.py:8:0: E0401: Unable to import 'hotel' (import-error)
tests\test_all.py:9:0: E0401: Unable to import 'customer' (import-error)
tests\test_all.py:10:0: E0401: Unable to import 'reservation' (import-error)
tests\test_all.py:47:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:66:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:68:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
tests\test_all.py:75:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:93:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:94:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
tests\test_all.py:101:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:120:4: C0116: Missing function or method docstring (missing-function-docstring)
tests\test_all.py:122:13: W1514: Using open without explicitly specifying an encoding (unspecified-encoding)
tests\test_all.py:7:0: W0611: Unused import json (unused-import)
```

Ilustración 13. Validación con pylint

```
PROBLEMS  OUTPUT  DEBUG CONSOLE  TERMINAL  PORTS

(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint tests\test_all.py

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

(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint reservation.py tests\test_reservation.py

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

(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint hotel.py tests\test_hotel.py

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

(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint customer.py tests\test_customer.py

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

(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2>
```

Ilustración 14.Corroboración de pylint

Extras

Como extra obtuve la calificación de Pylint a través del comando

```
pylint tests/test_all.py --reports=y
```

```
(.venv) PS C:\Users\lucia\OneDrive\Documentos\MNA\04 Trimestre\Calidad\Semana 06\A01796976_A6.2> pylint tests/test_all.py --reports=y


Report
======
69 statements analysed.

Statistics by type
------------------

+---------+-------+-----------+-----------+------------+---------+
|type     |number |old number |difference |%documented |%badname |
+=========+=======+===========+===========+============+=========+
|module   |1      |1          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|class    |1      |1          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|method   |8      |8          |=          |100.00      |0.00     |
+---------+-------+-----------+-----------+------------+---------+
|function |0      |NC         |NC         |0           |0        |
+---------+-------+-----------+-----------+------------+---------+



External dependencies
---------------------
::

    customer (test_all)
    hotel (test_all)
    reservation (test_all)



141 lines have been analyzed

Raw metrics
-----------

+----------+-------+------+---------+-----------+
|type      |number |%     |previous |difference |
+==========+=======+======+=========+===========+
|code      |81     |57.45 |NC       |NC         |
+----------+-------+------+---------+-----------+
|docstring |15     |10.64 |NC       |NC         |
+----------+-------+------+---------+-----------+
```

Ilustración 15. Reporte pylint (1 de 2)

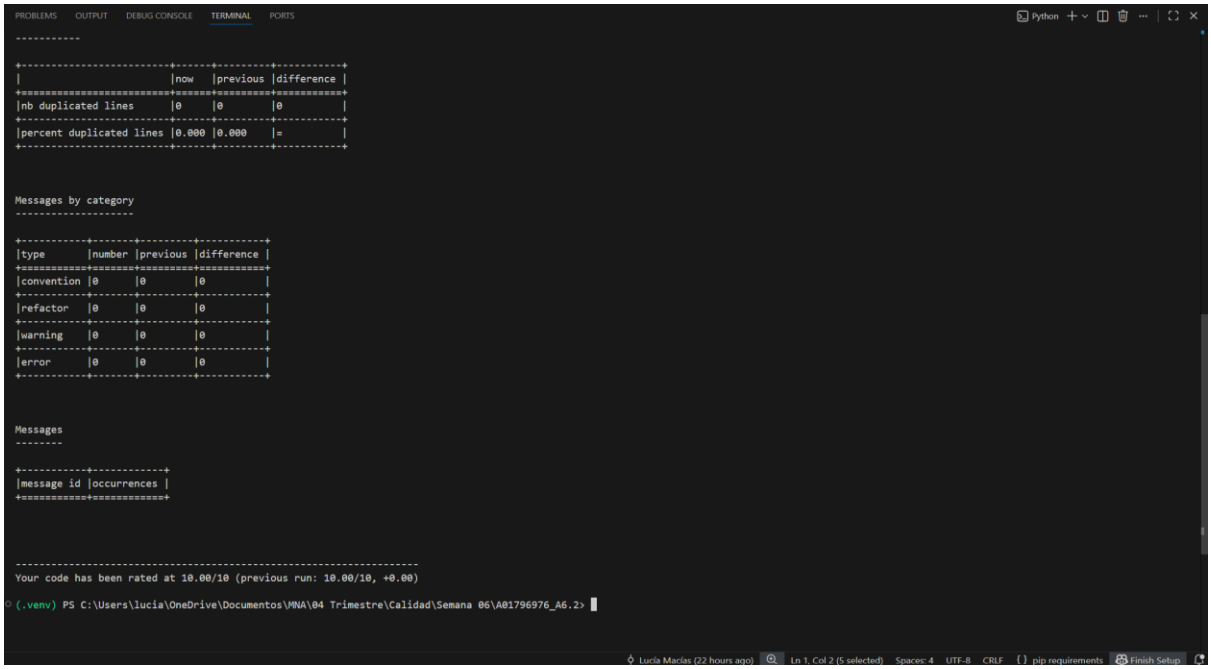

Ilustración 16. Reporte pylint (2 de 2)

Además, como soy más visual genere un reporte para Pytest, para el cual realice lo siguiente:

1. Instale: pip install pytest-html
2. Ejecute: pytest --html=reporte.html --self-contained-html tests/test_all.py
3. Lo cual genero el reporte.html, él cual está en el git.

Conclusiones

Esta actividad demostró la importancia de las pruebas automatizadas y el análisis estático de código para garantizar un software mantenible y de alta calidad. El uso de Flake8 y Pylint ayudó a identificar errores de código y a aplicar las mejores prácticas.

Como reflexión reforcé la importancia de las pruebas unitarias y los beneficios del análisis estático, que ayudan a mejorar la calidad del software, pero la verdad fue algo frustrante en cierto punto que haciendo unas correcciones de pylint se movía, y tenía que volver a corregir lo de Flake8. Poniéndome el sombrero programadora no me gusta mucho la documentación, se lo necesario y relevante que es, pero muchas veces la operación y las urgencias limitan mucho el tiempo que podemos dejar para esas actividades.

Referencias Bibliográficas

Python Software Foundation. (s.f.). unittest — Unit testing framework. Python 3.12.2 documentation. https://docs.python.org/3/library/unittest.html

Van Rossum, G., Warsaw, B., & Coghlan, N. (2001, 5 de julio). PEP 8 – Style Guide for Python Code. Python.org. https://peps.python.org/pep-0008/

PyNative. (s.f.). Python JSON programming: Master JSON processing with Python. https://pynative.com/python/json/

Python Software Foundation. (s.f.). The Python Tutorial. Python 3.12.2 documentation. https://docs.python.org/3/tutorial/index.html

PyCQA. (s.f.). Flake8: Your Tool For Style Guide Enforcement. https://flake8.pycqa.org/en/latest/

Luminousmen. (s.f.). Python Static Analysis Tools. https://luminousmen.com/post/python-static-analysis-tools

Batchelder, N., & colaboradores. (s.f.). Coverage.py: Code coverage measurement for Python. PyPI. https://pypi.org/project/coverage/