Instituto Tecnológico y de Estudios Superiores de Monterrey

Pruebas de software y aseguramiento de la calidad

6.2 Ejercicio de programación 3

Pablo Alejandro Colunga Vázquez - A01793671

Actividad 6.2. Ejercicio de programación 3

2.18 Explicar los fundamentos del desarrollo de pruebas unitarias

2.19 Desarrollar pruebas unitarias para fragmentos de programas usando las mejores prácticas recomendadas.

En esta actividad vas a generar un repositorio para los ejercicios de programación en GIT. Revisa las indicaciones para los programas a implementar:

1. Genera un repositorio para los ejercicios de programación
2. Download ejercicios de programación
3. en GIT.
4. Revisa las indicaciones para los programas a implementar.
5. Implementa los programas, indicados al final del documento, usando el lenguaje Python.
6. Sigue el estándar de codificación PEP-8.
7. Verifica la correcta ejecución de tus programas generando pruebas de cada ejercicio usando los recursos indicados. Documente los resultados.
8. Instala el paquete flake8 usando PIP,https://luminousmen.com/post/python-static-analysis-tools
9. Si tienes duda del uso, revisa el tutorial de Flake8: https://flake8.pycqa.org/en/latest/
10. Links to an external site.
11. Verifica que tus programas no generen errores o problemas usando pylint.

    The error code of flake8 are:
      - E***/W***: Errors and warnings of pycodestyle
      - F***: Detections of PyFlakes
      - C9**: Detections of circulate complexity by McCabe-script
12. Arregla todos los detalles que encontró flake8 y verifica que tu programa sigue funcionando correctamente.
13. Al terminar carga tus programas en tu repositorio personal que generaste. El proyecto deberá llamarse: Matrícula de estudiante_Número de actividadA6.2
14. A manera de evidencia de la ejecución y facilitar la revisión de los mismos adjunta capturas de pantalla de la ejecución de los mismos al entregar la actividad.
15. Sube la liga del repositorio en la tarea de Canvas.
16. Sube los archivos fuente de la tarea a Canvas.

Actividad 6.2. Ejercicio de programación 3

Programming Exercise	Description	Practice	Test Cases and Evidence

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.
		• Control structures
		• Console Input output
		• Mathematical computation
		• File management
		• Error handling
			Record the execution. Use files included in the assignment.

Pylint.

Resolved each pylint issue found, the common issue was docstrings, and that message is shared when you don't use a description in some instructions. The other one is about the quantity of characters used in a row. Pylint is running locally in the chapter of the customer script.

```
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> python customer.py
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> pylint customer.py

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

PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> pylint hotel.py
```

Flake8.

Reviewed each python file, and resolved each warning of flake 8, the common warning was "line too long" or "expected X blank lines, found x".

```
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:7:1: E302 expected 2 blank lines, found 1
customer.py:35:5: E301 expected 1 blank line, found 0
customer.py:59:80: E501 line too long (98 > 79 characters)
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:7:1: E302 expected 2 blank lines, found 1
customer.py:35:5: E301 expected 1 blank line, found 0
customer.py:59:80: E501 line too long (98 > 79 characters)
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:36:5: E301 expected 1 blank line, found 0
customer.py:60:80: E501 line too long (98 > 79 characters)
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:62:80: E501 line too long (85 > 79 characters)
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:62:35: W291 trailing whitespace
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:62:35: W291 trailing whitespace
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
customer.py:63:25: E131 continuation line unaligned for hanging indent
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> flake8 customer.py
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\customer> []
```

Notes.

Using both analysers in scripts with a directory embedded.

You can see in the image a bad score using pylint, because you are running the analyser locally and it doesn't enable the connection to another script (unable to import …).

In another example, you can see when running flake8 locally, only see some errors related to "blank lines", and no are considering some warnings about the connection.

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> pylint Test\customer\customer_test.py
************ Module customer_test
Test\customer\customer_test.py:7:0: E0401: Unable to import 'Work.customer.customer' (import-error)

------------------------------------
Your code has been rated at 7.62/10

PS C:\Users\HP\6.2. Ejercicio de programación 3> flake8 Test\customer\customer_test.py
Test\customer\customer_test.py:9:1: E302 expected 2 blank lines, found 1
Test\customer\customer_test.py:40:1: E305 expected 2 blank lines after class or function definition, found 1
PS C:\Users\HP\6.2. Ejercicio de programación 3> flake8 Test\customer\customer_test.py
Test\customer\customer_test.py:41:1: E305 expected 2 blank lines after class or function definition, found 1
PS C:\Users\HP\6.2. Ejercicio de programación 3> flake8 Test\customer\customer_test.py
PS C:\Users\HP\6.2. Ejercicio de programación 3> []
```

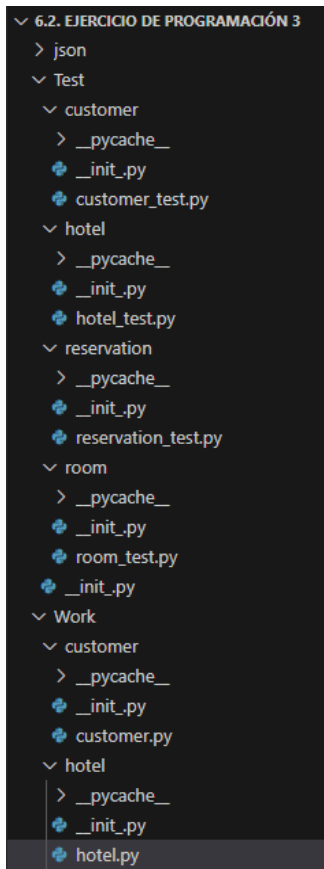Error to run the script in a local folder

```
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\hotel> python -m hotel.py
Traceback (most recent call last):
  File "C:\python310\lib\runpy.py", line 187, in _run_module_as_main
    mod_name, mod_spec, code = _get_module_details(mod_name, _Error)
  File "C:\python310\lib\runpy.py", line 110, in _get_module_details
    __import__(pkg_name)
  File "C:\Users\HP\6.2. Ejercicio de programación 3\Work\hotel\hotel.py", line 6, in <module>
    from Work.room.room import Room
ModuleNotFoundError: No module named 'Work'
PS C:\Users\HP\6.2. Ejercicio de programación 3\Work\hotel> python -m hotel.py
Traceback (most recent call last):
  File "C:\python310\lib\runpy.py", line 187, in _run_module_as_main
    mod_name, mod_spec, code = _get_module_details(mod_name, _Error)
  File "C:\python310\lib\runpy.py", line 110, in _get_module_details
    __import__(pkg_name)
  File "C:\Users\HP\6.2. Ejercicio de programación 3\Work\hotel\hotel.py", line 6, in <module>
    from Work.room.room import Room
ModuleNotFoundError: No module named 'Work'
```

Solution
Run the script as a module within the "Work" package.

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m Work.customer.customer
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m Work.hotel.hotel
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m Work.reservation.reservation
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m Work.room.room
```

Adding an `__init__.py` file to a directory in Python indicates that the directory should be treated as a package.

```
∨ 6.2. EJERCICIO DE PROGRAMACIÓN 3
  > json
  ∨ Test
  ∨ customer
    > __pycache__
    🐍 _init_.py
    🐍 customer_test.py
  ∨ hotel
    > __pycache__
    🐍 _init_.py
    🐍 hotel_test.py
  ∨ reservation
    > __pycache__
    🐍 _init_.py
    🐍 reservation_test.py
  ∨ room
    > __pycache__
    🐍 _init_.py
    🐍 room_test.py
  🐍 _init_.py
  ∨ Work
  ∨ customer
    > __pycache__
    🐍 _init_.py
    🐍 customer.py
  ∨ hotel
    > __pycache__
    🐍 _init_.py
    🐍 hotel.py
```

Finally, test each file with data of five customers, hotels and rooms, previously created as a .json file.

Json files

```
json > {} customer.json > ...
1  [
2      {"name": "John Doe", "email": "john@example.com"},
3      {"name": "Jane Smith", "email": "jane@example.com"},
4      {"name": "Bob Johnson", "email": "bob@example.com"},
5      {"name": "Alice Brown", "email": "alice@example.com"},
6      {"name": "Charlie Wilson", "email": "charlie@example.com"}
7  ]
8
```

To run the script, you need to run the unit of testing individually.

Customer_test

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m unittest Test.customer.customer_test
Customer Data: [{'name': 'John Doe', 'email': 'john@example.com'}, {'name': 'Jane Smith', 'email': 'jane@example.com'}, {'name': 'Bob Johnson', 'email': 'bob@example.com'}, {'name': 'Alice Brown', 'email': 'alice@example.com'}, {'name': 'Charlie Wilson', 'email': 'charlie@example.com'}]
.Customer Data: [{'name': 'John Doe', 'email': 'john@example.com'}, {'name': 'Jane Smith', 'email': 'jane@example.com'}, {'name': 'Bob Johnson', 'email': 'bob@example.com'}, {'name': 'Alice Brown', 'email': 'alice@example.com'}, {'name': 'Charlie Wilson', 'email': 'charlie@example.com'}]
.
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```
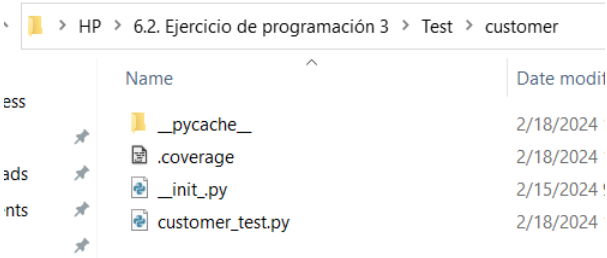
Hotel_test

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m unittest Test.hotel.hotel_test
Hotel Data: [{'name': 'Sample Hotel 1', 'location': 'City Center', 'rooms': [101, 102, 103]}, {'name': 'Sample Hotel 2', 'location': 'Suburb', 'rooms': [201, 202, 203]}, {'name': 'Sample Hotel 3', 'location': 'Downtown', 'rooms': [301, 302, 303]}, {'name': 'Sample Hotel 4', 'location': 'Beachfront', 'rooms': [401, 402, 403]}, {'name': 'Sample Hotel 5', 'location': 'Mountain View', 'rooms': [501, 502, 503]}]
.Hotel Data: [{'name': 'Sample Hotel 1', 'location': 'City Center', 'rooms': [101, 102, 103]}, {'name': 'Sample Hotel 2', 'location': 'Suburb', 'rooms': [201, 202, 203]}, {'name': 'Sample Hotel 3', 'location': 'Downtown', 'rooms': [301, 302, 303]}, {'name': 'Sample Hotel 4', 'location': 'Beachfront', 'rooms': [401, 402, 403]}, {'name': 'Sample Hotel 5', 'location': 'Mountain View', 'rooms': [501, 502, 503]}]
.
----------------------------------------------------------------------
Ran 2 tests in 0.002s

OK
```

Reservation_test

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m unittest Test.reservation.reservation_test
Reservation Data: [{'customer': 1, 'hotel': 1, 'room': 101}, {'customer': 2, 'hotel': 2, 'room': 202}, {'customer': 3, 'hotel': 3, 'room': 303}, {'customer': 4, 'hotel': 4, 'room': 404}, {'customer': 5, 'hotel': 5, 'room': 505}]
.
----------------------------------------------------------------------
Ran 1 test in 0.002s

OK
```

Room_test

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> python -m unittest Test.room.room_test
Room Data: [{'number': 101}, {'number': 202}, {'number': 303}, {'number': 404}, {'number': 505}]
.
----------------------------------------------------------------------
Ran 1 test in 0.001s

OK
```

Coverage with unittest

Customer

At the beginning of the test, the score of customer_test.py was 95% at the line 43, then I modified the code on the code and verify again, the results was this one:

You can modify the code until have the score that you want, an example.

You can see a file was created on the directory:

Room

```
Test\room\room_test.py       15     1    93%    32
Work\room\room.py            11     4    64%    36-40
------------------------------------------------------
TOTAL                        26     5    81%
PS C:\Users\HP\6.2. Ejercicio de programación 3> coverage run -m unittest Test.room.room_test
Room Data: [{'number': 101}, {'number': 202}, {'number': 303}, {'number': 404}, {'number': 505}]
.
------------------------------------------------------
Ran 1 test in 0.002s

OK
PS C:\Users\HP\6.2. Ejercicio de programación 3> coverage report -m
Name                       Stmts   Miss  Cover   Missing
------------------------------------------------------
Test\room\room_test.py       15     1    93%    32
Work\room\room.py             4     0   100%
------------------------------------------------------
TOTAL                        19     1    95%
```

More examples

```
PS C:\Users\HP\6.2. Ejercicio de programación 3> coverage report -m
Name                          Stmts   Miss  Cover   Missing
------------------------------------------------------
Test\hotel\hotel_test.py        26     2    92%    41, 47
Work\customer\customer.py        9     2    78%    42-43
Work\hotel\hotel.py             17     1    94%    41
Work\reservation\reservation.py  6     0   100%
Work\room\room.py                4     0   100%
------------------------------------------------------
TOTAL                           62     5    92%
```

Final comments

It's important to understand the benefits of making individual tests, it helps to simplify the script, and be more understandable. Also with this exercise I can understand the benefits of using the analysers flake8 and pylint, both have great benefits, but each one can be used for specific error detections. The other thing that I discovered is about running with packages (modules) and directories.

The coverage report helps to find quickly if you have inside your script extra code.

Github repository link:

https://github.com/PColunga87/MNA-Pruebas_de_Software_Aseguramiendo_de_Calidad/tree/96c4602bd530ff37fc557c3830c06fa661e4fa20/6.2.%20Ejercicio%20de%20programaci%C3%B3n%203

Bibliography:
- Flake8: Your Tool For Style Guide Enforcement — flake8 7.0.0 documentation. (s. f.). https://flake8.pycqa.org/en/latest/
- unittest — Unit testing framework. (s. f.). Python Documentation. https://docs.python.org/3/library/unittest.html