



Tecnológico de Monterrey

Maestría en Inteligencia Artificial Aplicada - Programa de Posgrados en Línea

Actividad 6.2 Ejercicio de programación 3 y pruebas de unidad

A01795690. Michelle Alejandro Durán Sánchez

Escuela de Ingeniería y Ciencias, Tecnológico de Monterrey

TC4017.10: Pruebas de software y aseguramiento de la calidad

Dr. Gerardo Padilla Zárate

Mtra. María Mylen Treviño E.

Ciudad de México, 16 de febrero de 2025

Ejercicio de programación

Ejercicio 1: Programming Exercise

Requerimiento:

Requisito 1: Implementar un conjunto de clases en Python que represente tres abstracciones:

1. Hotel
2. Reserva
3. Clientes

Requisito 2: Implementar un conjunto de métodos para gestionar los siguientes comportamientos persistentes (almacenados en archivos):

1. Hoteles
 - a. Crear un hotel
 - b. Eliminar un hotel
 - c. Mostrar información del hotel
 - d. Modificar información del hotel
 - e. Reservar una habitación
 - f. Cancelar una reserva
2. Clientes
 - a. Crear un cliente
 - b. Eliminar un cliente
 - c. Mostrar información del cliente
 - d. Modificar información del cliente
3. Reservas
 - a. Crear una reserva (Cliente, Hotel)
 - b. Cancelar una reserva

Se tiene libertad para definir los atributos dentro de cada clase que permitan el comportamiento requerido.

Requisito 3: Implementar casos de prueba unitarios para evaluar los métodos de cada clase. Usar el módulo “*unittest*” en Python.

Requisito 4: La cobertura de código de todas las pruebas unitarias debe alcanzar al menos un 85% de cobertura de líneas.

Requisito 5: El programa debe incluir un mecanismo para manejar datos inválidos en los archivos. Los errores deben mostrarse en la consola y la ejecución debe continuar.

Requisito 6: El código debe cumplir con PEP8.

Requisito 7: El código fuente no debe mostrar advertencias al ejecutarse con Flake8 y PyLint.

Resultados:

- Ejemplo si hubiera errores con PyLint:

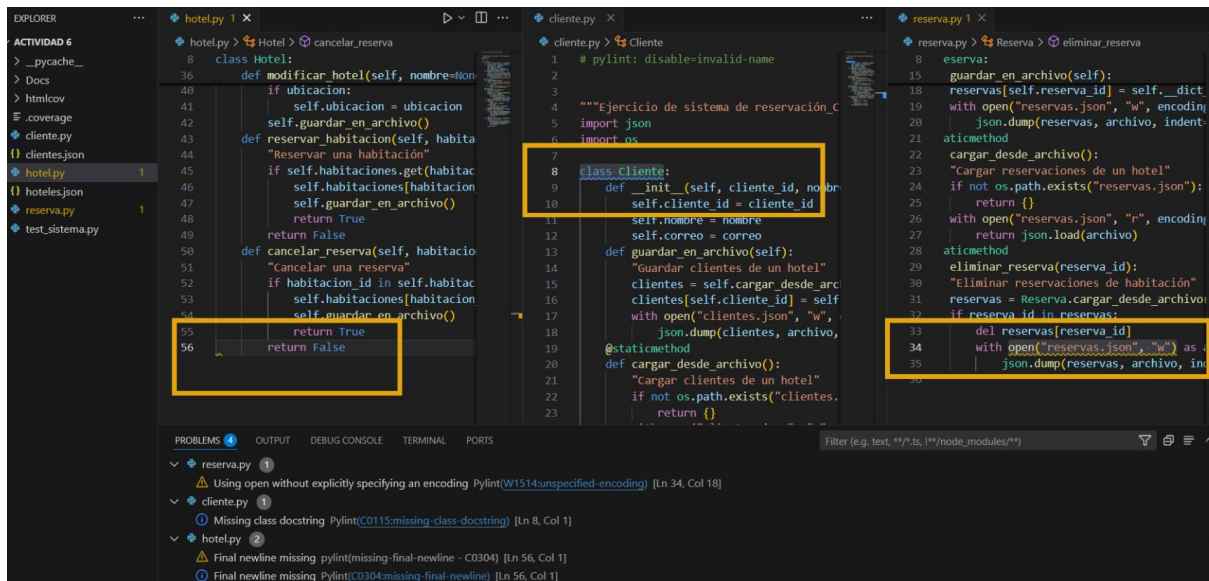


Imagen 1. Resultados con errores en PyLint de hotel.py, cliente.py, reserva.py.

- Archivos sin errores de PyLint:

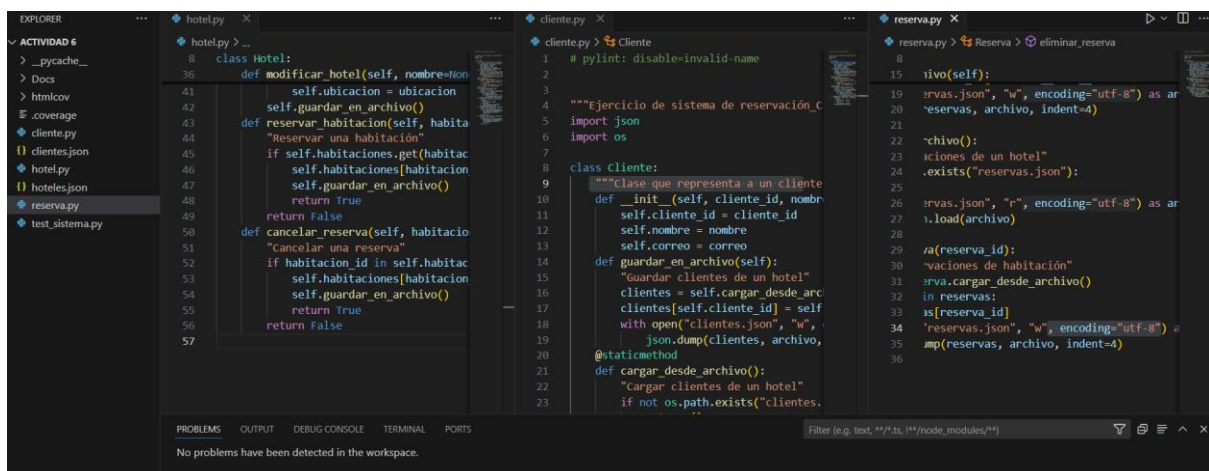


Imagen 2. Resultados sin errores en PyLint de hotel.py, cliente.py, reserva.py.

- Ejemplo si hubiera errores con Flake 8:

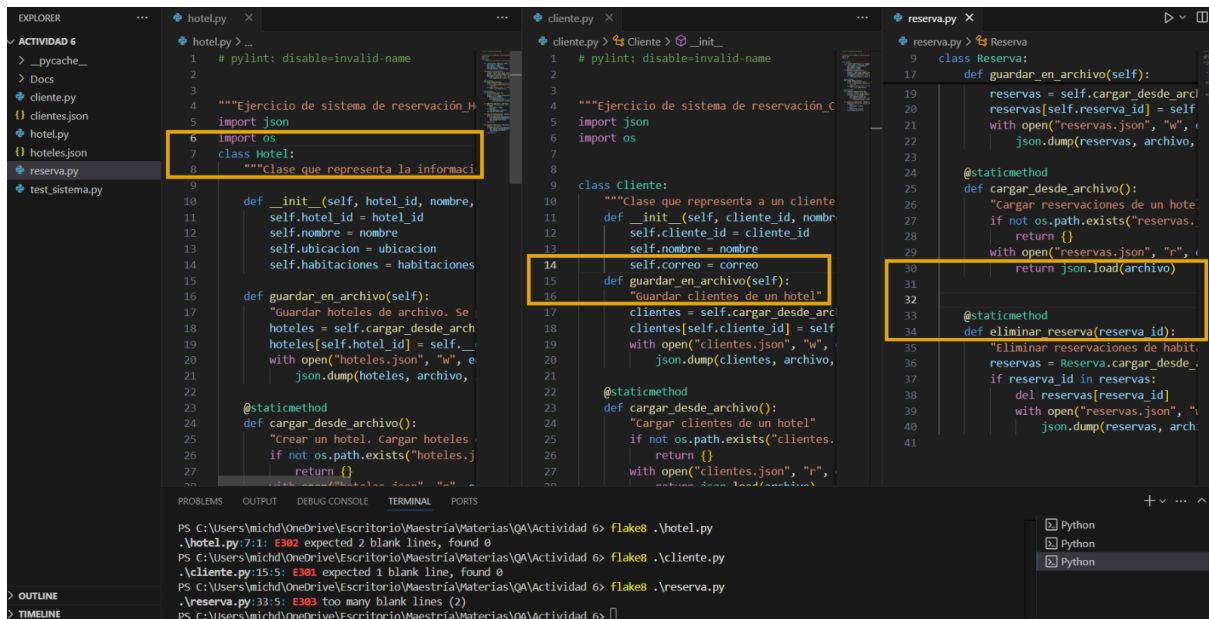


Imagen 3. Resultados con errores en Flake 8 de hotel.py, cliente.py, reserva.py.

- Archivos sin errores de Flake 8:

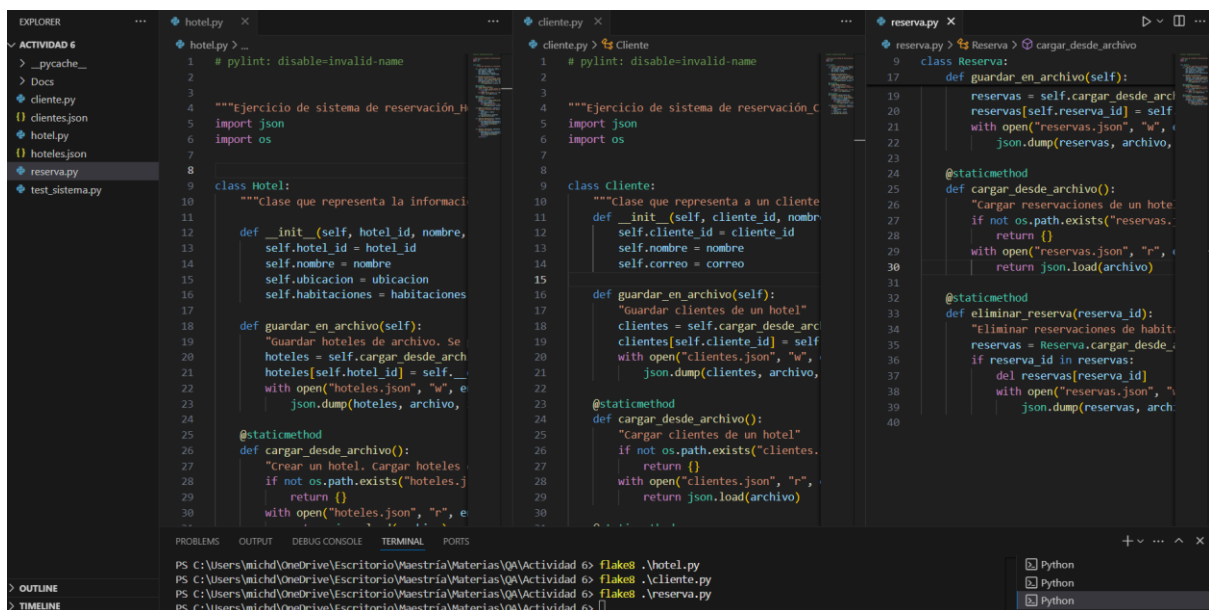


Imagen 4. Resultados sin errores en Flake 8 de hotel.py, cliente.py, reserva.py.

- Se ejecuta el test_sistema.py para las pruebas unitarias a través del módulo unittest. Como se observa en la imagen de abajo, se corren las pruebas sin errores y se generan los .json para el cliente creado (ID del cliente, nombre y correo) así como el .json para la reserva de la habitación asociada al hotel (ID del hotel, nombre del hotel, ubicación y habitaciones disponibles) en donde la reserva también está asociada al cliente generado:

The screenshot shows the VS Code interface with the following content:

- test_sistema.py:**

```

1 # pylint: disable=invalid-name
2
3 """Ejercicio de sistema de reservación Tests"""
4 import unittest
5
6 # Ignorar errores de importación en pylint
7 from hotel import Hotel # pylint: disable=imp
8 from cliente import Cliente # pylint: disable
9 from reserva import Reserva # pylint: disable
10
11 class TestSistema(unittest.TestCase):
12     """Clase para pruebas unitarias"""
13     def setUp(self):
14         self.hotel = Hotel("H001", "Hotel Cent
15         self.cliente = Cliente("C001", "Michel
16         self.reserva = Reserva("R001", "C001",
17
18     def test_crear_hotel(self):
19         "crear hotel"
20         self.hotel.guardar_en_archivo()
21         hoteles = Hotel.cargar_desde_archivo()
22         self.assertEqual(hoteles, hoteles)
23     def test_crear_cliente(self):
24         "crear cliente"
25         self.cliente.guardar_en_archivo()

```
- clientes.json:**

```

1 {
2     "C001": {
3         "cliente_id": "C001",
4         "nombre": "Michelle Duran",
5         "correo": "michelle@example.com"
6     }
7 }

```
- hoteles.json:**

```

1 {
2     "H001": {
3         "hotel_id": "H001",
4         "nombre": "Hotel Central",
5         "ubicacion": "CDMX",
6         "habitaciones": {
7             "101": false,
8             "102": true
9         }
10 }
11 }

```
- Terminal:**

```

PS C:\Users\michd\OneDrive\Escritorio\Maestría\Materias\QA\Actividad 6> & C:/Users/michd/AppData/Local/Programs/Python/Python312/python.exe
"c:/Users/michd/OneDrive/Escritorio\Maestría\Materias\QA\Actividad 6/test_sistema.py"
.....
Ran 4 tests in 0.005s
OK

```

Imagen 5. Resultados de las pruebas unitarias OK.

- Se ejecutan las pruebas de cobertura a través del módulo coverage para saber si la cobertura de código de todas las pruebas unitarias alcanza al menos un 85% de cobertura de líneas (imagen 6) y se obtiene el reporte tanto en la terminal (imagen 7) como en un html (imagen 8) obteniendo un 93% de cobertura:

The screenshot shows the VS Code interface with the following content:

- test_sistema.py:**

```

12 class TestSistema(unittest.TestCase):
13     """Clase para pruebas unitarias"""
14     def setUp(self):
15         self.hotel = Hotel("H001", "Hotel Central", "CDMX",
16         self.cliente = Cliente("C001", "Michelle Duran", "
17         self.reserva = Reserva("R001", "C001", "H001", "10
18     def test_crear_hotel(self):
19         "crear hotel"
20         self.hotel.guardar_en_archivo()
21         hoteles = Hotel.cargar_desde_archivo()
22         self.assertEqual(hoteles, hoteles)
23     def test_eliminar_hotel(self):
24         "Eliminar hotel"
25         self.hotel.guardar_en_archivo()
26         hotel_eliminar_hotel("H001")
27         hoteles = Hotel.cargar_desde_archivo()
28         self.assertEqual(hoteles, hoteles)
29     def test_modificar_hotel(self):
30         "Modificar información del hotel"
31         self.hotel.guardar_en_archivo()
32         self.hotel.modificar_hotel(nombre="Hotel Modificado
33         hoteles = Hotel.cargar_desde_archivo()
34         self.assertEqual(hoteles["H001"]["nombre"], "Hotel
35         self.assertEqual(hoteles["H001"]["ubicacion"], "Nue
36     def test_crear_cliente(self):
37         "crear cliente"
38         self.cliente.guardar_en_archivo()

```
- clientes.json:**

```

1 {
2     "C001": {
3         "cliente_id": "C001",
4         "nombre": "Nuevo Nombre",
5         "correo": "nuevo@example.com"
6     }
7 }

```
- hoteles.json:**

```

1 {
2     "H001": {
3         "hotel_id": "H001",
4         "nombre": "Hotel Central",
5         "ubicacion": "CDMX",
6         "habitaciones": {
7             "101": false,
8             "102": true
9         }
10 }
11 }

```
- Terminal:**

```

PS C:\Users\michd\OneDrive\Escritorio\Maestría\Materias\QA\Actividad 6> coverage run --source=. -m unittest test_sistema.py
.....
Ran 8 tests in 0.018s
OK

```

Imagen 6. Ejecución de coverage OK.

```

class TestSistema(unittest.TestCase):
    """Clase para pruebas unitarias"""
    def setUp(self):
        self.hotel = Hotel("H001", "Hotel Central", "CDMX")
        self.cliente = Cliente("C001", "Michelle Duran", " ")
        self.reserva = Reserva("R001", "C001", "H001", "10")
    def test_crear_hotel(self):
        "Crear hotel"
        self.hotel.guardar_en_archivo()
        hoteles = Hotel.cargar_desde_archivo()
        self.assertIn("H001", hoteles)
    def test_eliminar_hotel(self):
        "Eliminar hotel"
        self.hotel.guardar_en_archivo()
        Hotel.eliminar_hotel("H001")
        hoteles = Hotel.cargar_desde_archivo()
        self.assertNotIn("H001", hoteles)
    def test_modificar_hotel(self):
        "Modificar información del hotel"
        self.hotel.guardar_en_archivo()

coverage report -m
>>
Name           Stmts   Miss  Cover   Missing
-----
cliente.py       31      1    97%      27
hotel.py         44      7    84%      29, 56, 60-64
reserva.py       26      1    96%      28
test_sistema.py  50      1    98%      67
-----
TOTAL             151     10    93%
PS C:\Users\michd\OneDrive\Escritorio\Maestría\Materias\QA\Actividad 6> coverage html
>>
Wrote HTML report to htmlcov\index.html
PS C:\Users\michd\OneDrive\Escritorio\Maestría\Materias\QA\Actividad 6>

```

Imagen 7. Cobertura del 93% en terminal.

Coverage report: 93%

Files Functions Classes

coverage.py v7.6.12, created at 2025-02-16 10:43 -0600

File	statements	missing	excluded	coverage
cliente.py	31	1	0	97%
hotel.py	44	7	0	84%
reserva.py	26	1	0	96%
test_sistema.py	50	1	0	98%
Total	151	10	0	93%

coverage.py v7.6.12, created at 2025-02-16 10:43 -0600

Imagen 8. Cobertura del 93% en index.html.

Liga de GitHub para ver ejercicios:

- https://github.com/MichelleDuranTEC/A01795690_A6.2.git

Referencias Bibliográficas

- Píldoras de programación. (2023, 26 de marzo). *Flake8 en Python, rápido y sencillo 2023*. [Video]. YouTube. <https://www.youtube.com/watch?v=G3tSdlBnSgA>
- zeroToMasters. (2020, 14 de mayo). *Aprende a generar el CODE COVERAGE paso a paso*. [Video]. YouTube. <https://www.youtube.com/watch?v=I3zyO3iPpOE>