

México, 18 de febrero de 2024

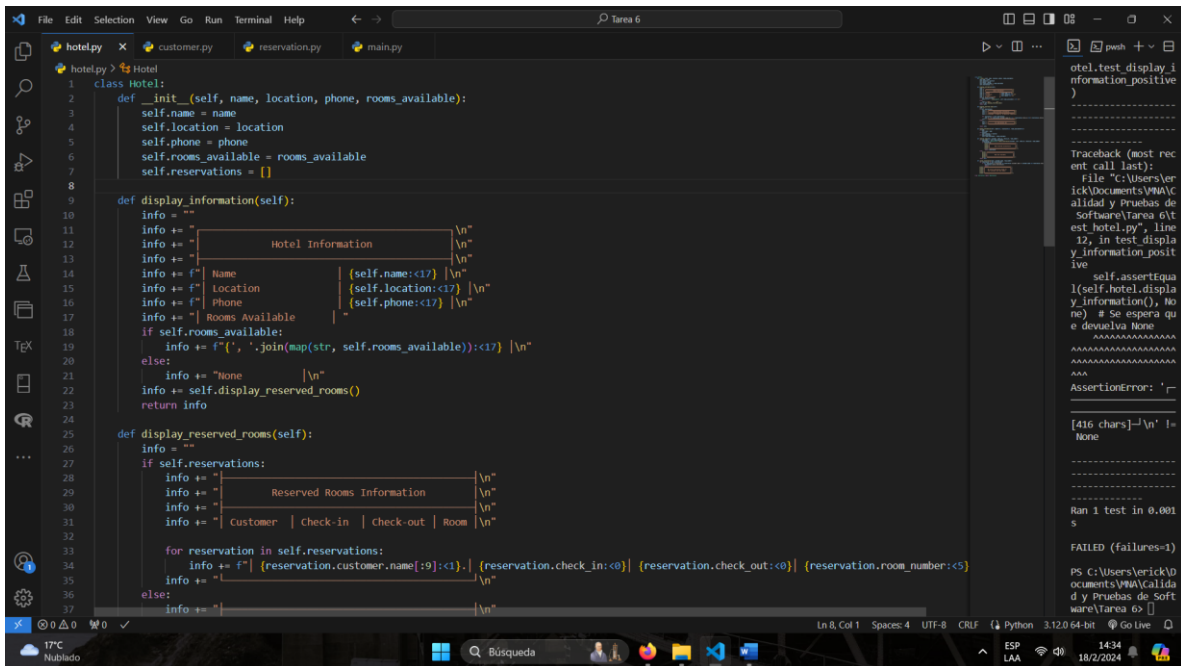
Actividad 6.2 | Ejercicio de programación 3

Enlace al repositorio de GitHub: https://github.com/ErickHCerecedo/A01066428_A6.2

El siguiente entregable se centra en documentar la elaboración de pruebas unitarias de la actividad 6.2, en el cual contiene la creación de las clases *Hotel*, *Customer*, y *Reservation* su revisión de redacción y estilo usando las librerías de PyLint junto al análisis de errores y riesgos con la librería Flake.

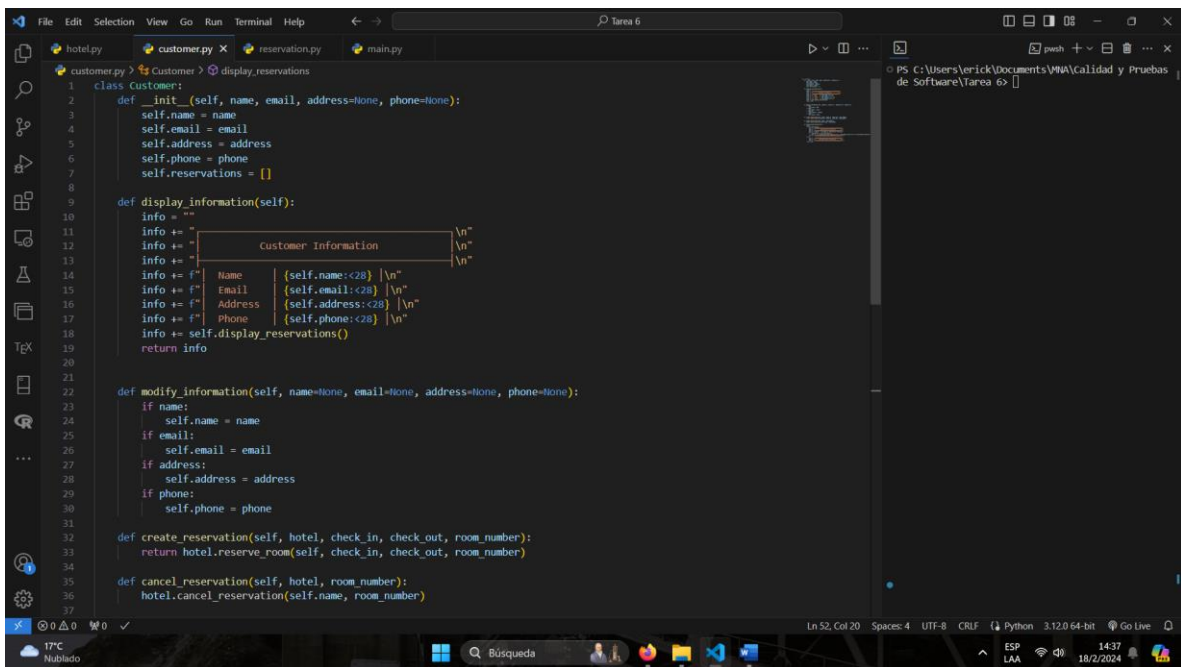
Problem 1: Compute Sales	
Description	<p>Requirement 1. Implement a set of classes in Python that implements two abstractions:</p> <ol style="list-style-type: none">1. Hotel2. Reservation3. Customers <p>Requirement 2. Implement a set of methods to handle the next persistent behaviors (stored in files):</p> <ol style="list-style-type: none">1. Hotels<ol style="list-style-type: none">a. Create Hotelb. Delete Hotelc. Display Hotel informationd. Modify Hotel Informatione. Reserve a Roomf. Cancel a Reservation2. Customer<ol style="list-style-type: none">a. Create Customerb. Delete Customerc. Display Customer Informationd. Modify Customer Information3. Reservation<ol style="list-style-type: none">a. Create a Reservation (Customer, Hotel)b. Cancel a Reservation <p>You are free to decide the attributes within each class that enable the required behavior.</p> <p>Requirement 3. Implement unit test cases to exercise the methods in each class. Use the unittest module in Python.</p> <p>Requirement 4. The code coverage for all unittests should accumulate at least 85% of line coverage.</p> <p>Requirement 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.</p> <p>Requirement 6. Be compliant with PEP8.</p> <p>Requirement 7. The source code must show no warnings using Fleak and PyLint.</p>

1. Creación de las clases hotel.py, customer.py y reservation.py



```
1 class Hotel:
2     def __init__(self, name, location, phone, rooms_available):
3         self.name = name
4         self.location = location
5         self.phone = phone
6         self.rooms_available = rooms_available
7         self.reservations = []
8
9
10    def display_information(self):
11        info = ""
12        info += "
13        info += "
14        info += "
15        info += "
16        info += "
17        info += "
18        if self.rooms_available:
19            info += f"
20        else:
21            info += "
22            info += self.display_reserved_rooms()
23        return info
24
25    def display_reserved_rooms(self):
26        info = ""
27        if self.reservations:
28            info += "
29            info += "
30            info += "
31            info += "
32            for reservation in self.reservations:
33                info += f"
34                info += "
35            else:
36                info += "
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

Traceback (most recent call last):
File "C:\Users\verick\Documents\WMA\Calidad y Pruebas de Software\Tarea 6\test_hotel.py", line 12, in test_display_information:
self.assertEqual(self.hotel.display_information(), None)
Se espera que devuelva None
AssertionError: 'Hotel Information' != None
[416 chars] '\n' != None
Ran 1 test in 0.001s
FAILED (failures=1)
PS C:\Users\verick\Documents\WMA\Calidad y Pruebas de Software\Tarea 6>



```
1 class Customer:
2     def __init__(self, name, email, address=None, phone=None):
3         self.name = name
4         self.email = email
5         self.address = address
6         self.phone = phone
7         self.reservations = []
8
9
10    def display_information(self):
11        info = ""
12        info += "
13        info += "
14        info += "
15        info += "
16        info += "
17        info += "
18        info += self.display_reservations()
19        return info
20
21
22    def modify_information(self, name=None, email=None, address=None, phone=None):
23        if name:
24            self.name = name
25        if email:
26            self.email = email
27        if address:
28            self.address = address
29        if phone:
30            self.phone = phone
31
32    def create_reservation(self, hotel, check_in, check_out, room_number):
33        return hotel.reserve_room(self, check_in, check_out, room_number)
34
35    def cancel_reservation(self, hotel, room_number):
36        hotel.cancel_reservation(self.name, room_number)
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
```

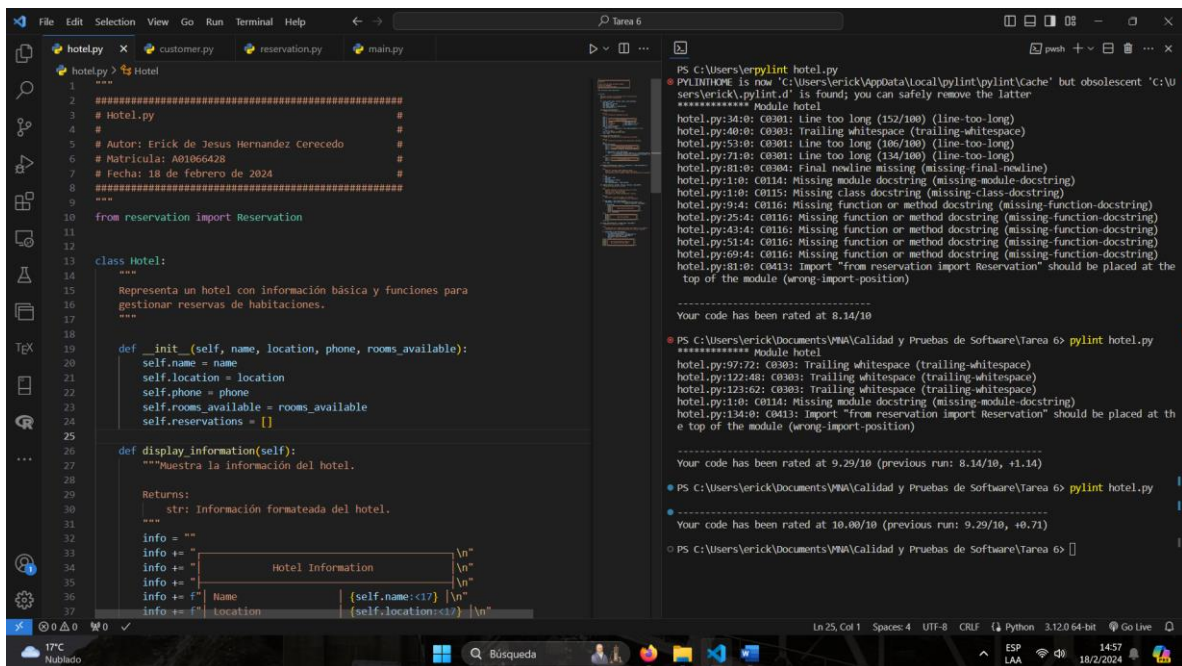
PS C:\Users\verick\Documents\WMA\Calidad y Pruebas de Software\Tarea 6>

```
File Edit Selection View Go Run Terminal Help Tarea 6
reservation.py customer.py reservation.py x main.py

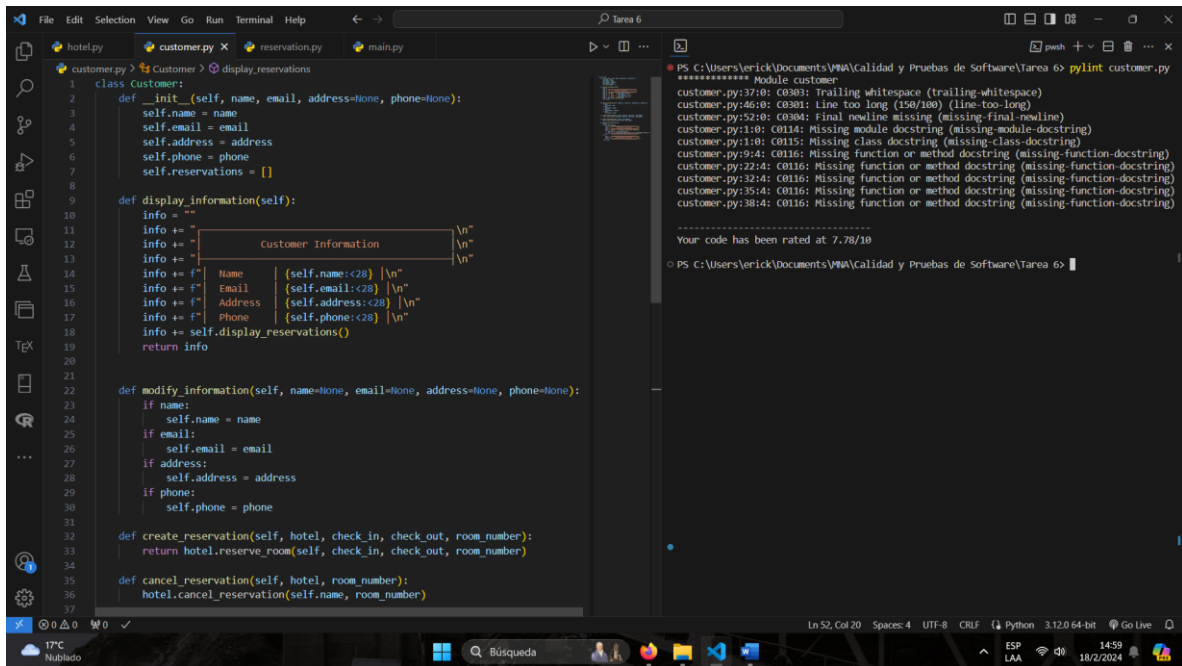
1 class Reservation:
2     def __init__(self, customer, hotel, check_in, check_out, room_number):
3         self.customer = customer
4         self.hotel = hotel
5         self.check_in = check_in
6         self.check_out = check_out
7         self.room_number = room_number
8         self.is_cancelled = False
9
10    @classmethod
11    def create_reservation(cls, customer, hotel, check_in, check_out, room_number):
12        if room_number in hotel.rooms_available:
13            reservation = cls(customer, hotel, check_in, check_out, room_number)
14            hotel.rooms_available.remove(room_number)
15            hotel.reservations.append(reservation)
16            customer.reservations.append(reservation)
17            print("
18            print("
19            print("      Reservation created successfully!
20            print("
21            print("\n")
22            return reservation
23        else:
24            print("
25            print("
26            print("      Room not available for reservation.
27            print("
28            print("\n")
29            return None
30
31    def cancel_reservation(self):
32        if not self.is_cancelled:
33            self.hotel.rooms_available.append(self.room_number)
34            self.hotel.reservations.remove(self)
35            self.customer.reservations.remove(self)
36            self.is_cancelled = True
37            print("

Ln 16, Col 17 Spaces: 4 UTF-8 CRLF Python 3.12.0 64-bit Go Live
17°C Nublado Búsqueda 14:37 18/2/2024
```

Inspección inicial de la clase Hotel con PyLint:



Inspección inicial de la clase Customer con PyLint:



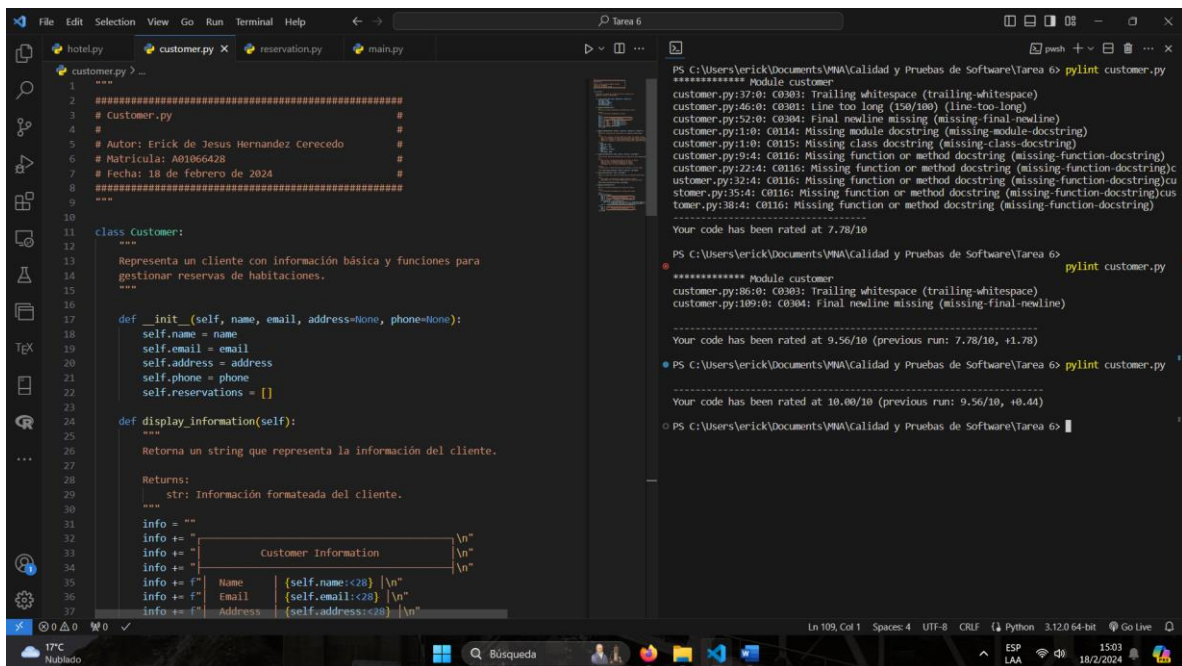
```
1 class Customer:
2     def __init__(self, name, email, address=None, phone=None):
3         self.name = name
4         self.email = email
5         self.address = address
6         self.phone = phone
7         self.reservations = []
8
9     def display_information(self):
10        info = ""
11        info += "
12        info += "
13        info += "
14        info += f" Name {self.name:<28} \n"
15        info += f" Email {self.email:<28} \n"
16        info += f" Address {self.address:<28} \n"
17        info += f" Phone {self.phone:<28} \n"
18        info += self.display_reservations()
19        return info
20
21
22    def modify_information(self, name=None, email=None, address=None, phone=None):
23        if name:
24            self.name = name
25        if email:
26            self.email = email
27        if address:
28            self.address = address
29        if phone:
30            self.phone = phone
31
32    def create_reservation(self, hotel, check_in, check_out, room_number):
33        return hotel.reserve_room(self, check_in, check_out, room_number)
34
35    def cancel_reservation(self, hotel, room_number):
36        hotel.cancel_reservation(self.name, room_number)
```

PS C:\Users\erick\Documents\VM\Calidad y Pruebas de Software\Tarea 6> pylint customer.py

```
***** Module customer
customer.py:37:0: C0303: Trailing whitespace (trailing-whitespace)
customer.py:46:0: C0301: Line too long (150/100) (line-too-long)
customer.py:52:0: C0304: Final newline missing (missing-final-newline)
customer.py:1:0: C0114: Missing module docstring (missing-module-docstring)
customer.py:1:0: C0115: Missing class docstring (missing-class-docstring)
customer.py:9:4: C0116: Missing function or method docstring (missing-function-docstring)
customer.py:22:4: C0116: Missing function or method docstring (missing-function-docstring)
customer.py:32:4: C0116: Missing function or method docstring (missing-function-docstring)
customer.py:35:4: C0116: Missing function or method docstring (missing-function-docstring)
customer.py:38:4: C0116: Missing function or method docstring (missing-function-docstring)

Your code has been rated at 7.78/10
```

Solución de la clase Customer con PyLint:



```
1 """
2     """
3     # Customer.py
4     #
5     # Autor: Erick de Jesus Hernandez Cerecedo
6     # Matricula: A01066428
7     # Fecha: 18 de febrero de 2024
8     """
9
10
11 class Customer:
12     """
13     Representa un cliente con información básica y funciones para
14     gestionar reservas de habitaciones.
15     """
16
17     def __init__(self, name, email, address=None, phone=None):
18         self.name = name
19         self.email = email
20         self.address = address
21         self.phone = phone
22         self.reservations = []
23
24     def display_information(self):
25         """
26         Retorna un string que representa la información del cliente.
27
28         Returns:
29             str: Información formateada del cliente.
30         """
31         info = ""
32         info += "
33         info += "
34         info += "
35         info += f" Name {self.name:<28} \n"
36         info += f" Email {self.email:<28} \n"
37         info += f" Address {self.address:<28} \n"
```

PS C:\Users\erick\Documents\VM\Calidad y Pruebas de Software\Tarea 6> pylint customer.py

```
***** Module customer
customer.py:37:0: C0303: Trailing whitespace (trailing-whitespace)
customer.py:109:0: C0304: Final newline missing (missing-final-newline)

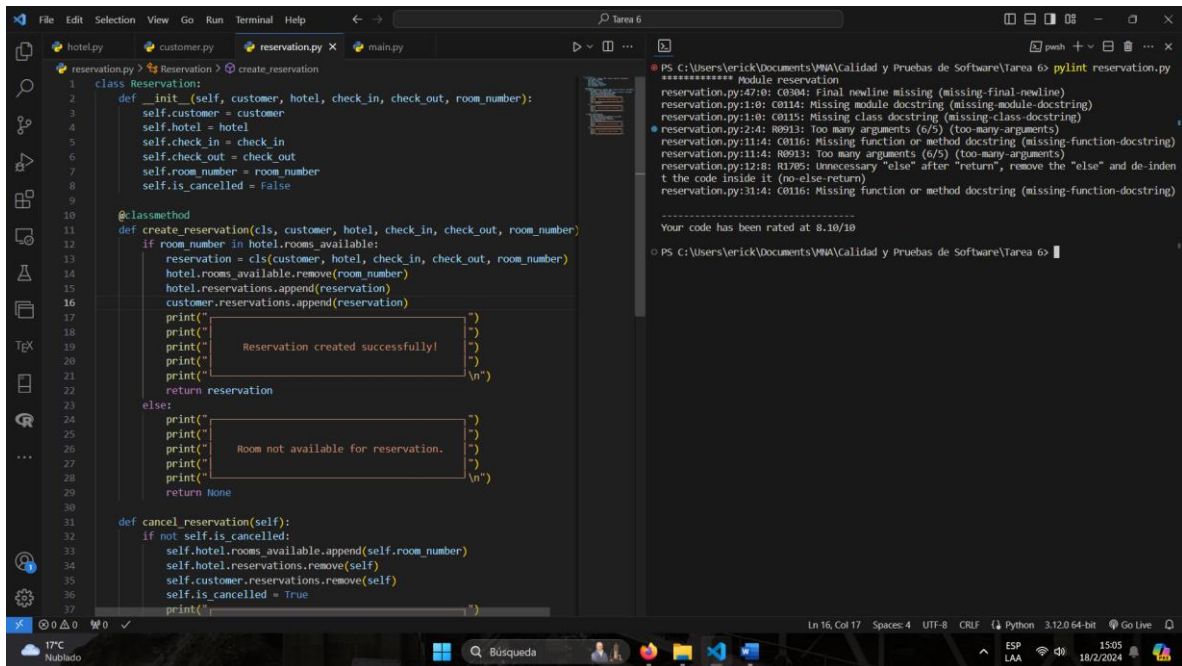
Your code has been rated at 9.56/10 (previous run: 7.78/10, +1.78)
```

PS C:\Users\erick\Documents\VM\Calidad y Pruebas de Software\Tarea 6> pylint customer.py

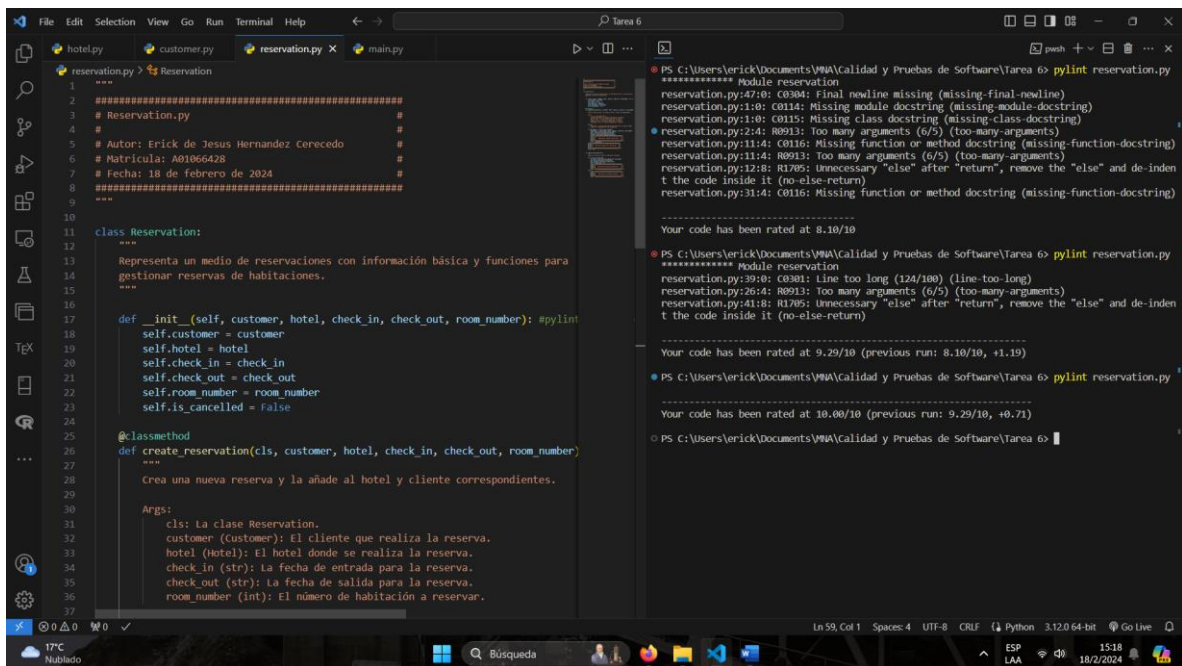
```
***** Module customer
customer.py:109:0: C0304: Final newline missing (missing-final-newline)

Your code has been rated at 10.00/10 (previous run: 9.56/10, +0.44)
```


Inspección inicial de la clase Reservation con PyLint:



Solución de la clase Reservation con PyLint:

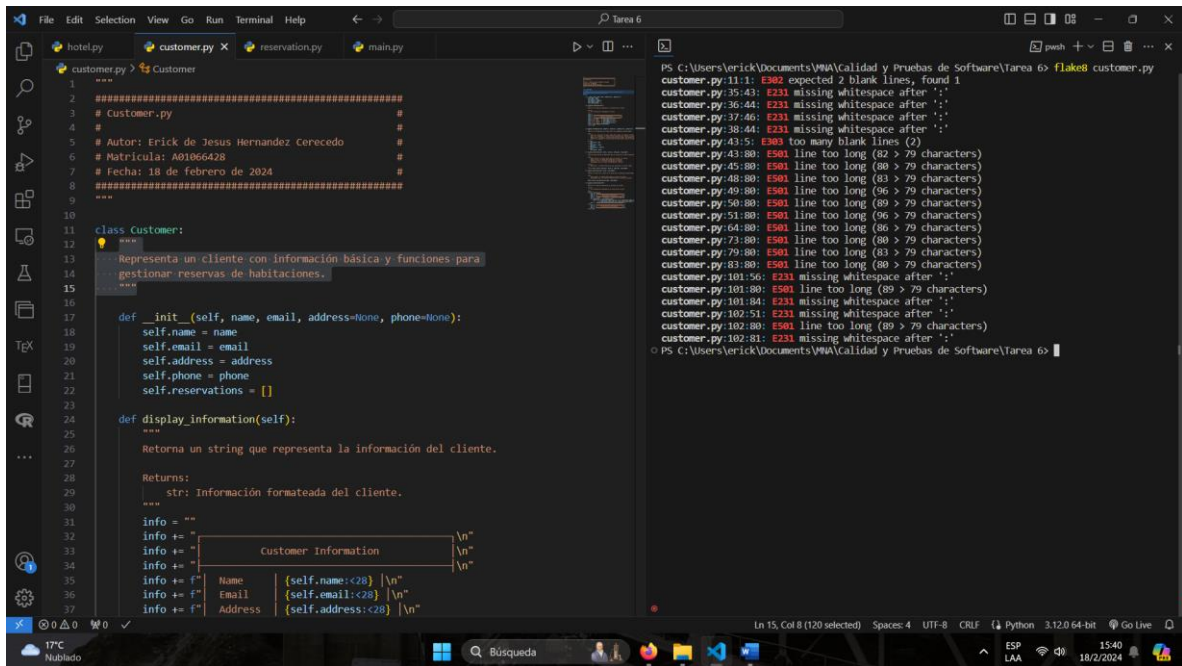


Inspección inicial de la clase Hotel con Flake:

Solución de problemas de la clase Hotel con Flake:

```
File Edit Selection View Go Run Terminal Help
hotelpy.py x customer.py reservation.py main.py
hotelpy.py > Hotel > cancel_reservation
1 """
2 #####
3 # Hotel.py
4 #
5 # Autor: Erick de Jesus Hernandez Cerecedo
6 # Matrícula: A01866428
7 # Fecha: 18 de febrero de 2024
8 #####
9
10 from reservation import Reservation
11
12
13 class Hotel:
14     """
15     Representa un hotel con información básica y funciones para
16     gestionar reservas de habitaciones.
17     """
18
19     def __init__(self, name, location, phone, rooms_available):
20         self.name = name
21         self.location = location
22         self.phone = phone
23         self.rooms_available = rooms_available
24         self.reservations = []
25
26     def display_information(self):
27         """Muestra la información del hotel.
28
29         Returns:
30             str: Información formateada del hotel.
31         """
32         info = ""
33         info += "\n"
34         info += "Hotel Information\n"
35         info += "\n"
36         info += f"Name {self.name: <17}\n"
37         info += f"Location {self.location: <17}\n"
```

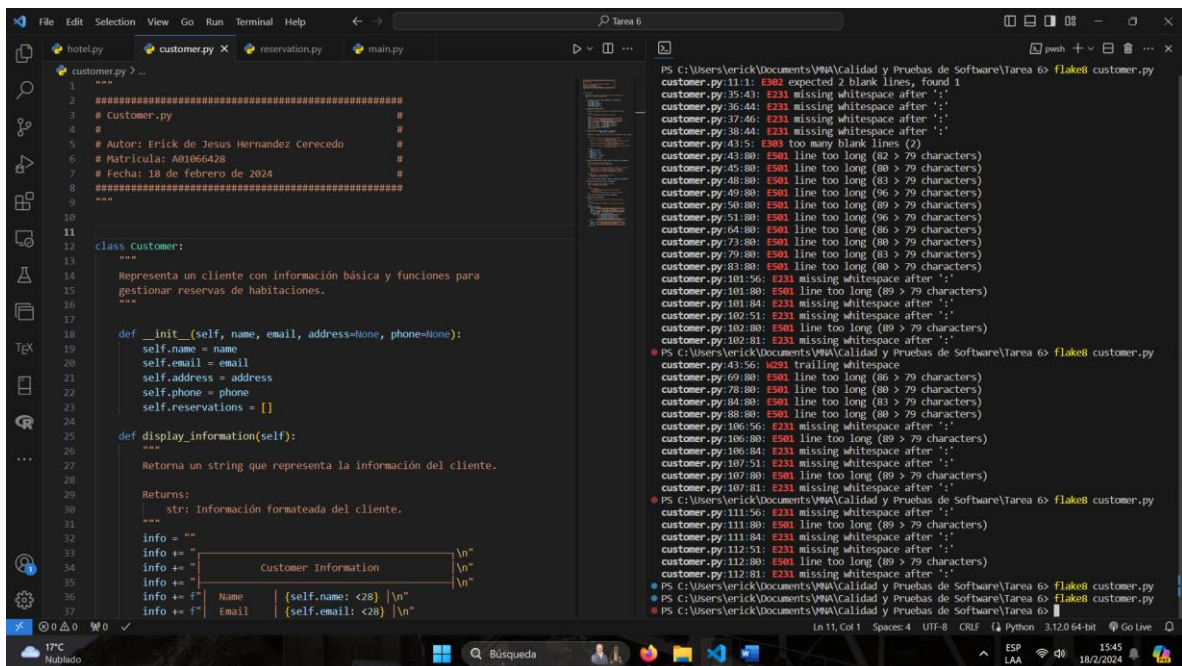
Inspección inicial de la clase Customer con Flake:



```
customer.py
1 """
2 =====
3 # Customer.py
4 #
5 # Autor: Erick de Jesus Hernandez Cerecedo
6 # Matricula: A01066428
7 # Fecha: 18 de febrero de 2024
8 # =====
9
10
11 class Customer:
12     """
13     Representa un cliente con información básica y funciones para
14     gestionar reservas de habitaciones.
15     """
16
17     def __init__(self, name, email, address=None, phone=None):
18         self.name = name
19         self.email = email
20         self.address = address
21         self.phone = phone
22         self.reservations = []
23
24     def display_information(self):
25         """
26         Retorna un string que representa la información del cliente.
27
28         Returns:
29             str: Información formateada del cliente.
30         """
31         info = ""
32         info += "\n"
33         info += "Customer Information\n"
34         info += "\n"
35         info += f"Name: {self.name}<28> \n"
36         info += f"Email: {self.email:<28> } \n"
37         info += f"Address: {self.address:<28> } \n"
```

```
PS C:\Users\verick\Documents\WVA\Calidad y Pruebas de Software\Tarea 6> flake8 customer.py
customer.py:11:1: E302 expected 2 blank lines, found 1
customer.py:35:43: E231 missing whitespace after ':'
customer.py:36:44: E231 missing whitespace after ':'
customer.py:37:46: E231 missing whitespace after ':'
customer.py:38:44: E231 missing whitespace after ':'
customer.py:43:5: E300 too many blank lines (2)
customer.py:43:80: E501 line too long (82 > 79 characters)
customer.py:45:80: E501 line too long (80 > 79 characters)
customer.py:48:80: E501 line too long (83 > 79 characters)
customer.py:49:80: E501 line too long (96 > 79 characters)
customer.py:50:80: E501 line too long (89 > 79 characters)
customer.py:51:80: E501 line too long (96 > 79 characters)
customer.py:64:80: E501 line too long (86 > 79 characters)
customer.py:73:80: E501 line too long (80 > 79 characters)
customer.py:79:80: E501 line too long (83 > 79 characters)
customer.py:83:80: E501 line too long (80 > 79 characters)
customer.py:101:56: E231 missing whitespace after ':'
customer.py:101:80: E501 line too long (89 > 79 characters)
customer.py:101:84: E231 missing whitespace after ':'
customer.py:102:51: E231 missing whitespace after ':'
customer.py:102:80: E501 line too long (89 > 79 characters)
customer.py:102:81: E231 missing whitespace after ':'
```

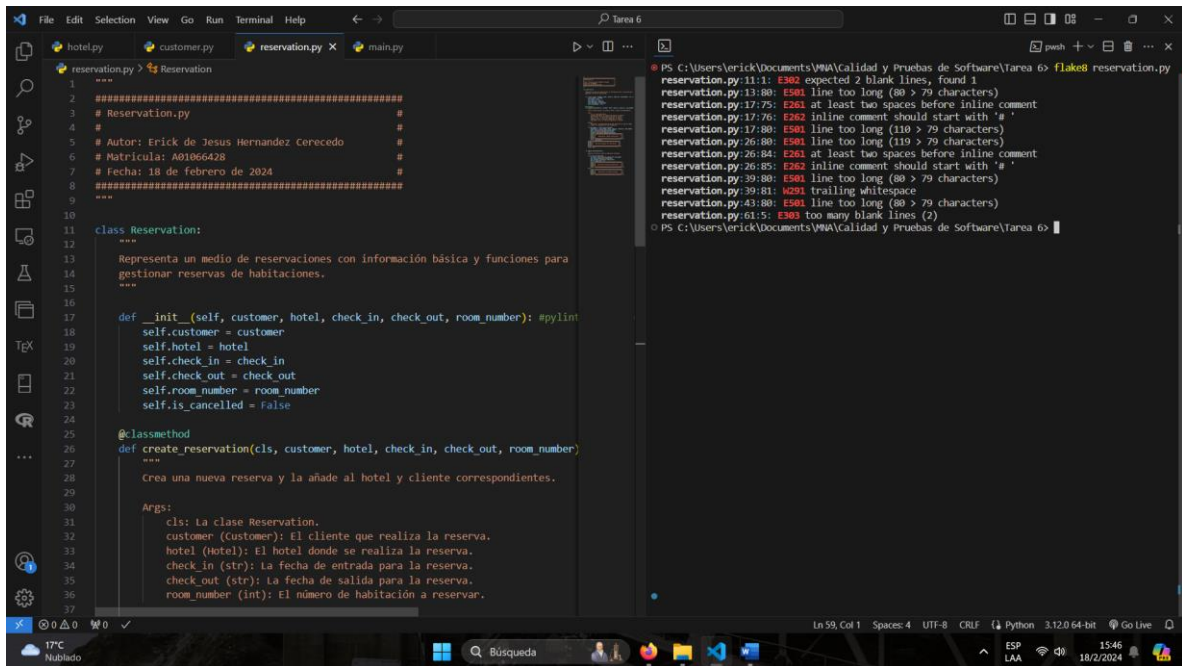
Solución de problemas de la clase Customer con Flake:



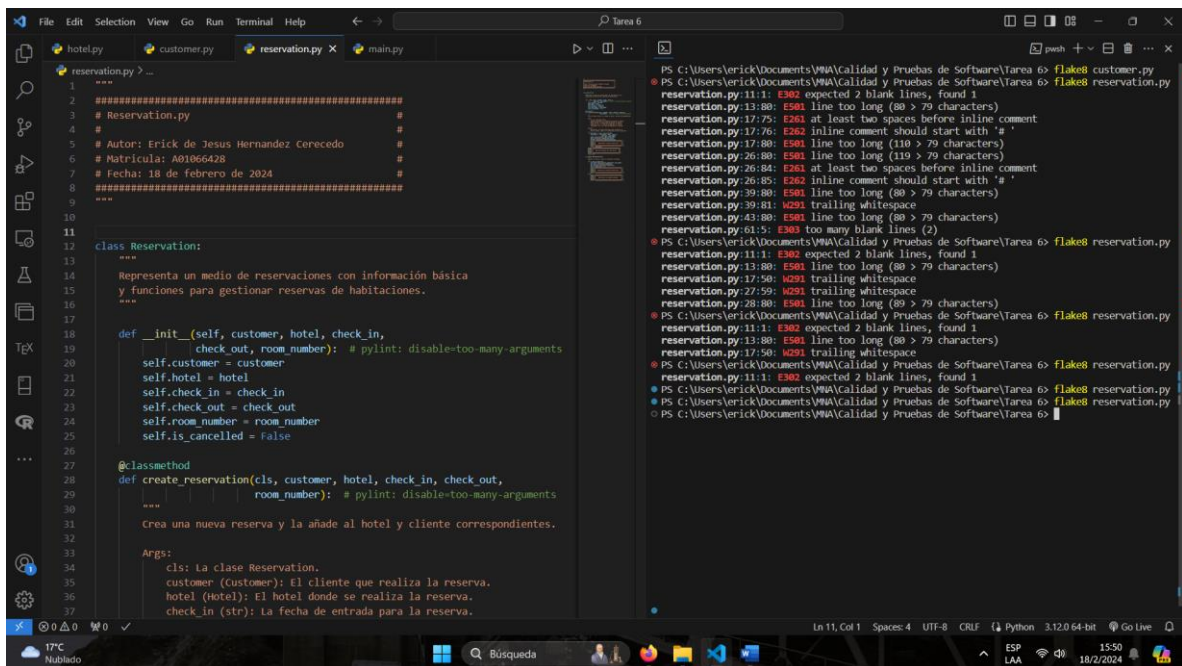
```
customer.py
1 """
2 =====
3 # Customer.py
4 #
5 # Autor: Erick de Jesus Hernandez Cerecedo
6 # Matricula: A01066428
7 # Fecha: 18 de febrero de 2024
8 # =====
9
10
11 class Customer:
12     """
13     Representa un cliente con información básica y funciones para
14     gestionar reservas de habitaciones.
15     """
16
17     def __init__(self, name, email, address=None, phone=None):
18         self.name = name
19         self.email = email
20         self.address = address
21         self.phone = phone
22         self.reservations = []
23
24     def display_information(self):
25         """
26         Retorna un string que representa la información del cliente.
27
28         Returns:
29             str: Información formateada del cliente.
30         """
31         info = ""
32         info += "\n"
33         info += "Customer Information\n"
34         info += "\n"
35         info += f"Name: {self.name:<28> } \n"
36         info += f"Email: {self.email:<28> } \n"
37         info += f"Address: {self.address:<28> } \n"
```

```
PS C:\Users\verick\Documents\WVA\Calidad y Pruebas de Software\Tarea 6> flake8 customer.py
customer.py:11:1: E302 expected 2 blank lines, found 1
customer.py:35:43: E231 missing whitespace after ':'
customer.py:36:44: E231 missing whitespace after ':'
customer.py:37:46: E231 missing whitespace after ':'
customer.py:38:44: E231 missing whitespace after ':'
customer.py:43:5: E300 too many blank lines (2)
customer.py:43:80: E501 line too long (82 > 79 characters)
customer.py:45:80: E501 line too long (80 > 79 characters)
customer.py:48:80: E501 line too long (83 > 79 characters)
customer.py:49:80: E501 line too long (96 > 79 characters)
customer.py:50:80: E501 line too long (89 > 79 characters)
customer.py:51:80: E501 line too long (96 > 79 characters)
customer.py:64:80: E501 line too long (86 > 79 characters)
customer.py:73:80: E501 line too long (80 > 79 characters)
customer.py:79:80: E501 line too long (83 > 79 characters)
customer.py:83:80: E501 line too long (80 > 79 characters)
customer.py:101:56: E231 missing whitespace after ':'
customer.py:101:80: E501 line too long (89 > 79 characters)
customer.py:101:84: E231 missing whitespace after ':'
customer.py:102:51: E231 missing whitespace after ':'
customer.py:102:80: E501 line too long (89 > 79 characters)
customer.py:102:81: E231 missing whitespace after ':'
customer.py:103:56: W291 trailing whitespace
customer.py:69:80: E501 line too long (86 > 79 characters)
customer.py:78:80: E501 line too long (80 > 79 characters)
customer.py:84:80: E501 line too long (83 > 79 characters)
customer.py:88:80: E501 line too long (80 > 79 characters)
customer.py:106:56: E231 missing whitespace after ':'
customer.py:106:80: E501 line too long (89 > 79 characters)
customer.py:106:84: E231 missing whitespace after ':'
customer.py:107:51: E231 missing whitespace after ':'
customer.py:107:80: E501 line too long (89 > 79 characters)
customer.py:107:81: E231 missing whitespace after ':'
customer.py:111:56: E231 missing whitespace after ':'
customer.py:111:80: E501 line too long (89 > 79 characters)
customer.py:111:84: E231 missing whitespace after ':'
customer.py:112:51: E231 missing whitespace after ':'
customer.py:112:80: E501 line too long (89 > 79 characters)
customer.py:112:81: E231 missing whitespace after ':'
```


Inspección inicial de la clase Reservation con Flake:

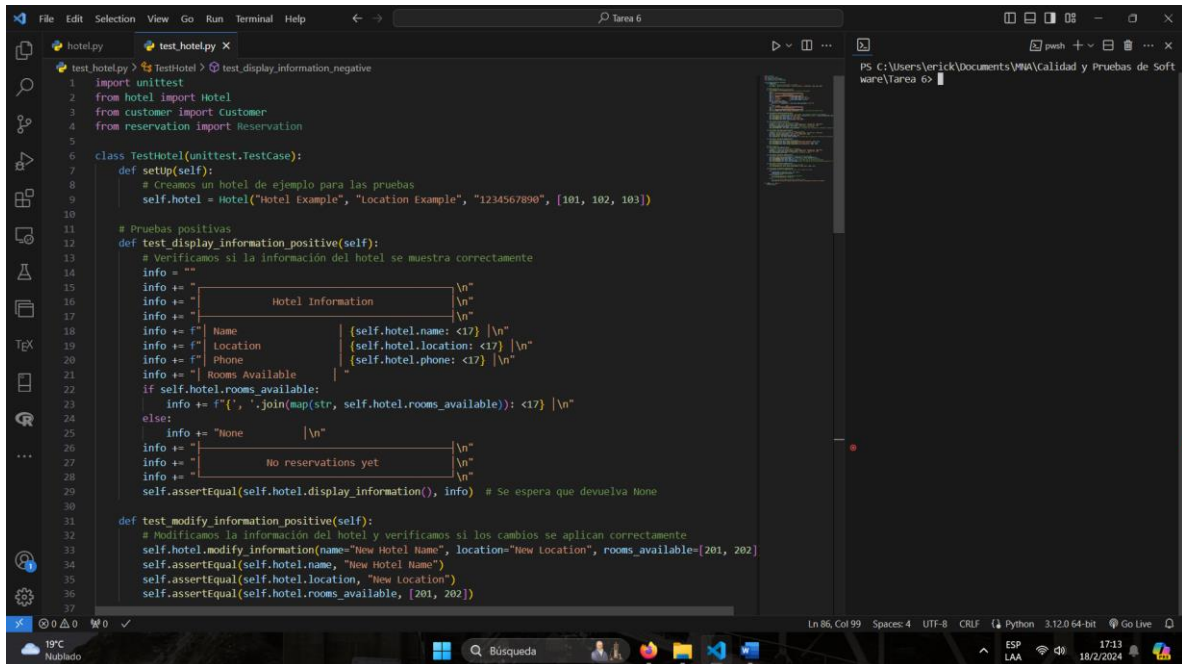


Solución de problemas de la clase Reservation con Flake:



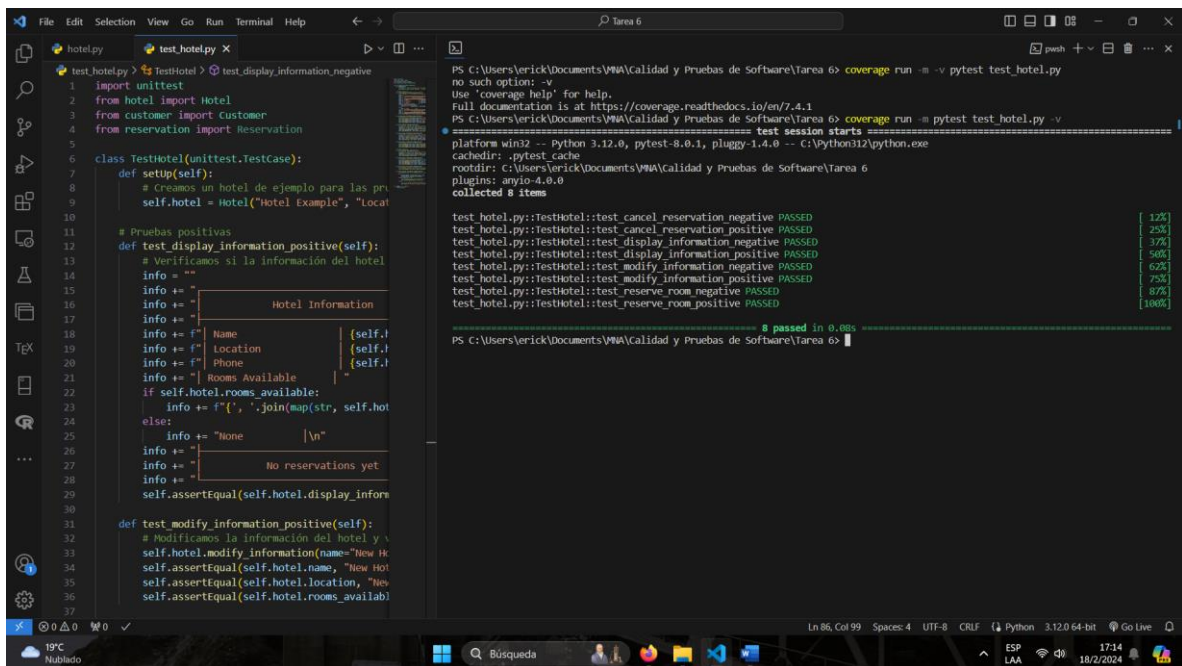
Unit test

Documento de pruebas para la clase Hotel:



```
1 import unittest
2 from hotel import Hotel
3 from customer import Customer
4 from reservation import Reservation
5
6 class TestHotel(unittest.TestCase):
7     def setUp(self):
8         # Creamos un hotel de ejemplo para las pruebas
9         self.hotel = Hotel("Hotel Example", "Location Example", "1234567890", [101, 102, 103])
10
11     # Pruebas positivas
12     def test_display_information_positive(self):
13         # Verificamos si la información del hotel se muestra correctamente
14         info = ""
15         info += "\n"
16         info += "Hotel Information\n"
17         info += "\n"
18         info += f"Name: {self.hotel.name}\n"
19         info += f"Location: {self.hotel.location}\n"
20         info += f"Phone: {self.hotel.phone}\n"
21         info += f"Rooms Available: {self.hotel.rooms_available}\n"
22         if self.hotel.rooms_available:
23             info += f"Rooms Available: {self.hotel.rooms_available}\n"
24         else:
25             info += "None\n"
26         info += "\n"
27         info += "No reservations yet\n"
28         info += "\n"
29         self.assertEqual(self.hotel.display_information(), info) # Se espera que devuelva None
30
31     def test_modify_information_positive(self):
32         # Modificamos la información del hotel y verificamos si los cambios se aplican correctamente
33         self.hotel.modify_information(name="New Hotel Name", location="New Location", rooms_available=[201, 202])
34         self.assertEqual(self.hotel.name, "New Hotel Name")
35         self.assertEqual(self.hotel.location, "New Location")
36         self.assertEqual(self.hotel.rooms_available, [201, 202])
```

Pruebas unitarias aprobadas:



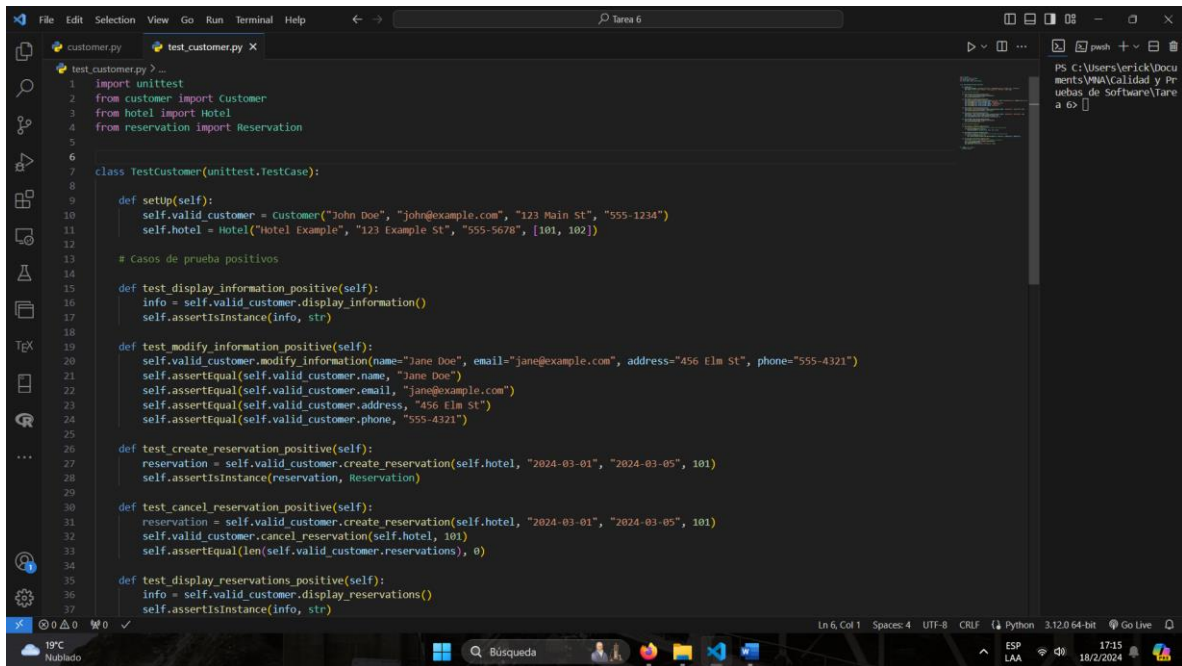
```
1 import unittest
2 from hotel import Hotel
3 from customer import Customer
4 from reservation import Reservation
5
6 class TestHotel(unittest.TestCase):
7     def setUp(self):
8         # Creamos un hotel de ejemplo para las pruebas
9         self.hotel = Hotel("Hotel Example", "Location Example", "1234567890", [101, 102, 103])
10
11     # Pruebas positivas
12     def test_display_information_positive(self):
13         # Verificamos si la información del hotel se muestra correctamente
14         info = ""
15         info += "\n"
16         info += "Hotel Information\n"
17         info += "\n"
18         info += f"Name: {self.hotel.name}\n"
19         info += f"Location: {self.hotel.location}\n"
20         info += f"Phone: {self.hotel.phone}\n"
21         info += f"Rooms Available: {self.hotel.rooms_available}\n"
22         if self.hotel.rooms_available:
23             info += f"Rooms Available: {self.hotel.rooms_available}\n"
24         else:
25             info += "None\n"
26         info += "\n"
27         info += "No reservations yet\n"
28         info += "\n"
29         self.assertEqual(self.hotel.display_information(), info) # Se espera que devuelva None
30
31     def test_modify_information_positive(self):
32         # Modificamos la información del hotel y verificamos si los cambios se aplican correctamente
33         self.hotel.modify_information(name="New Hotel Name", location="New Location", rooms_available=[201, 202])
34         self.assertEqual(self.hotel.name, "New Hotel Name")
35         self.assertEqual(self.hotel.location, "New Location")
36         self.assertEqual(self.hotel.rooms_available, [201, 202])
```

```
PS C:\Users\verick\Documents\VNA\Calidad y Pruebas de Software\Tarea 6> coverage run -- -v pytest test_hotel.py
no such option: -v
Use "coverage help" for help.
Full documentation is at https://coverage.readthedocs.io/en/7.4.1
PS C:\Users\verick\Documents\VNA\Calidad y Pruebas de Software\Tarea 6> coverage run -- pytest test_hotel.py -v
test session starts
platform win32 -- Python 3.12.0, pytest-8.0.1, pluggy-1.4.0 -- C:\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\verick\Documents\VNA\Calidad y Pruebas de Software\Tarea 6
plugins: anyio-4.0.0
collected 8 items

test_hotel.py::TestHotel::test_cancel_reservation_negative PASSED [ 12%]
test_hotel.py::TestHotel::test_cancel_reservation_positive PASSED [ 25%]
test_hotel.py::TestHotel::test_display_information_negative PASSED [ 37%]
test_hotel.py::TestHotel::test_display_information_positive PASSED [ 50%]
test_hotel.py::TestHotel::test_modify_information_negative PASSED [ 62%]
test_hotel.py::TestHotel::test_modify_information_positive PASSED [ 75%]
test_hotel.py::TestHotel::test_reserve_room_negative PASSED [ 87%]
test_hotel.py::TestHotel::test_reserve_room_positive PASSED [100%]

===== 8 passed in 0.08s =====
PS C:\Users\verick\Documents\VNA\Calidad y Pruebas de Software\Tarea 6>
```

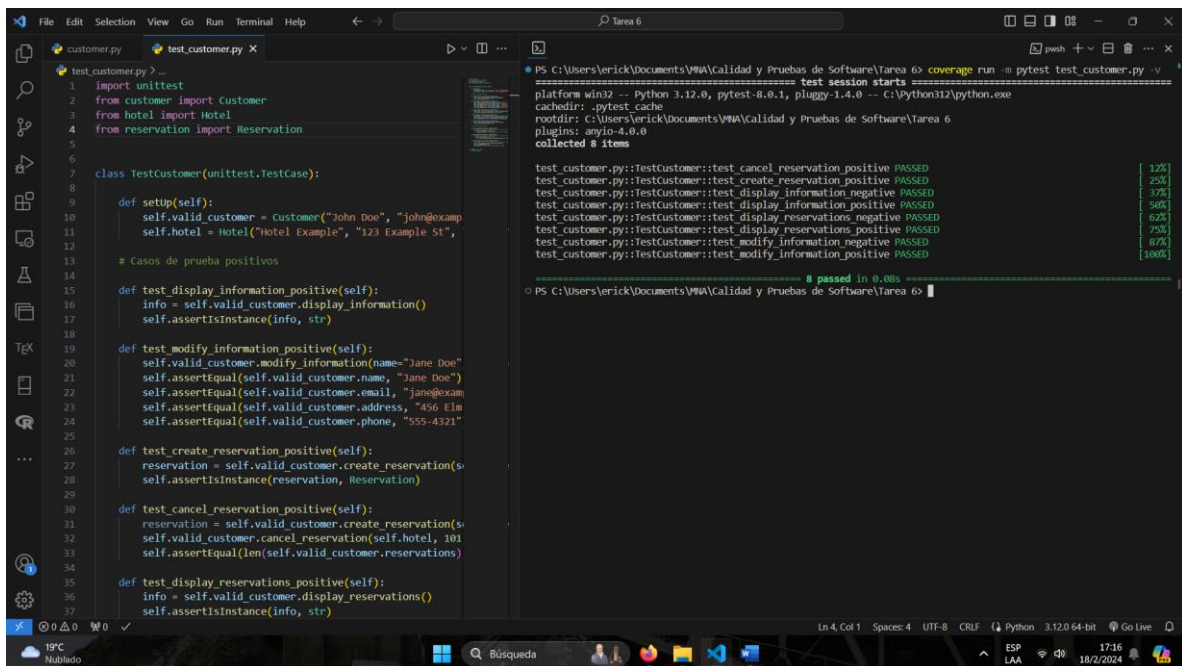
Documento de pruebas de la clase Customer:



The screenshot shows a code editor with a file named `test_customer.py`. The code defines a `TestCustomer` class that inherits from `unittest.TestCase`. It includes a `setUp` method that initializes a `valid_customer` and a `hotel`. There are several test methods: `test_display_information_positive`, `test_modify_information_positive`, `test_create_reservation_positive`, `test_cancel_reservation_positive`, and `test_display_reservations_positive`. Each test method uses `assert` to verify the behavior of the `Customer` class methods.

```
1 import unittest
2 from customer import Customer
3 from hotel import Hotel
4 from reservation import Reservation
5
6
7 class TestCustomer(unittest.TestCase):
8
9     def setUp(self):
10         self.valid_customer = Customer("John Doe", "john@example.com", "123 Main St", "555-1234")
11         self.hotel = Hotel("Hotel Example", "123 Example St", "555-5678", [101, 102])
12
13     # Casos de prueba positivos
14
15     def test_display_information_positive(self):
16         info = self.valid_customer.display_information()
17         self.assertIsInstance(info, str)
18
19     def test_modify_information_positive(self):
20         self.valid_customer.modify_information(name="Jane Doe", email="jane@example.com", address="456 Elm St", phone="555-4321")
21         self.assertEqual(self.valid_customer.name, "Jane Doe")
22         self.assertEqual(self.valid_customer.email, "jane@example.com")
23         self.assertEqual(self.valid_customer.address, "456 Elm St")
24         self.assertEqual(self.valid_customer.phone, "555-4321")
25
26     def test_create_reservation_positive(self):
27         reservation = self.valid_customer.create_reservation(self.hotel, "2024-03-01", "2024-03-05", 101)
28         self.assertIsInstance(reservation, Reservation)
29
30     def test_cancel_reservation_positive(self):
31         reservation = self.valid_customer.create_reservation(self.hotel, "2024-03-01", "2024-03-05", 101)
32         self.valid_customer.cancel_reservation(self.hotel, 101)
33         self.assertEqual(len(self.valid_customer.reservations), 0)
34
35     def test_display_reservations_positive(self):
36         info = self.valid_customer.display_reservations()
37         self.assertIsInstance(info, str)
```

Resultado de pruebas unitarias aprobadas:

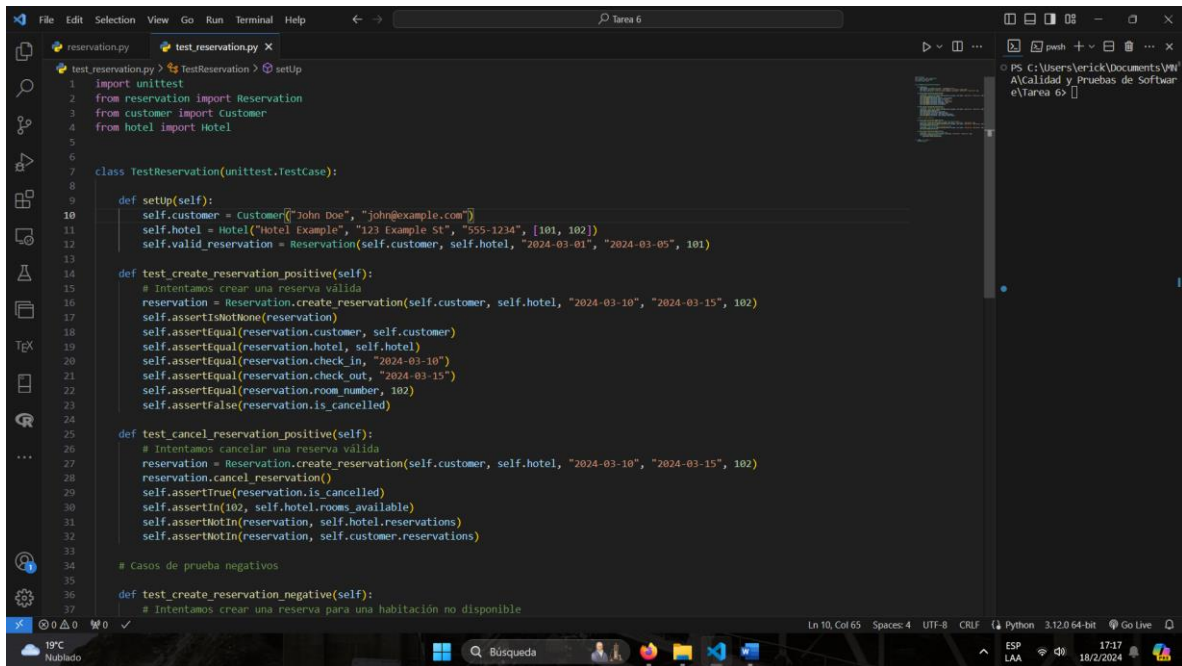


The screenshot shows a terminal window with the output of a `coverage run` command. The output indicates that all tests passed. The test results are as follows:

Test Case	Result	Coverage
test_customer.py::TestCustomer::test_cancel_reservation_positive	PASSED	12%
test_customer.py::TestCustomer::test_create_reservation_positive	PASSED	25%
test_customer.py::TestCustomer::test_display_information_negative	PASSED	37%
test_customer.py::TestCustomer::test_display_information_positive	PASSED	50%
test_customer.py::TestCustomer::test_display_reservations_negative	PASSED	62%
test_customer.py::TestCustomer::test_display_reservations_positive	PASSED	75%
test_customer.py::TestCustomer::test_modify_information_negative	PASSED	87%
test_customer.py::TestCustomer::test_modify_information_positive	PASSED	100%

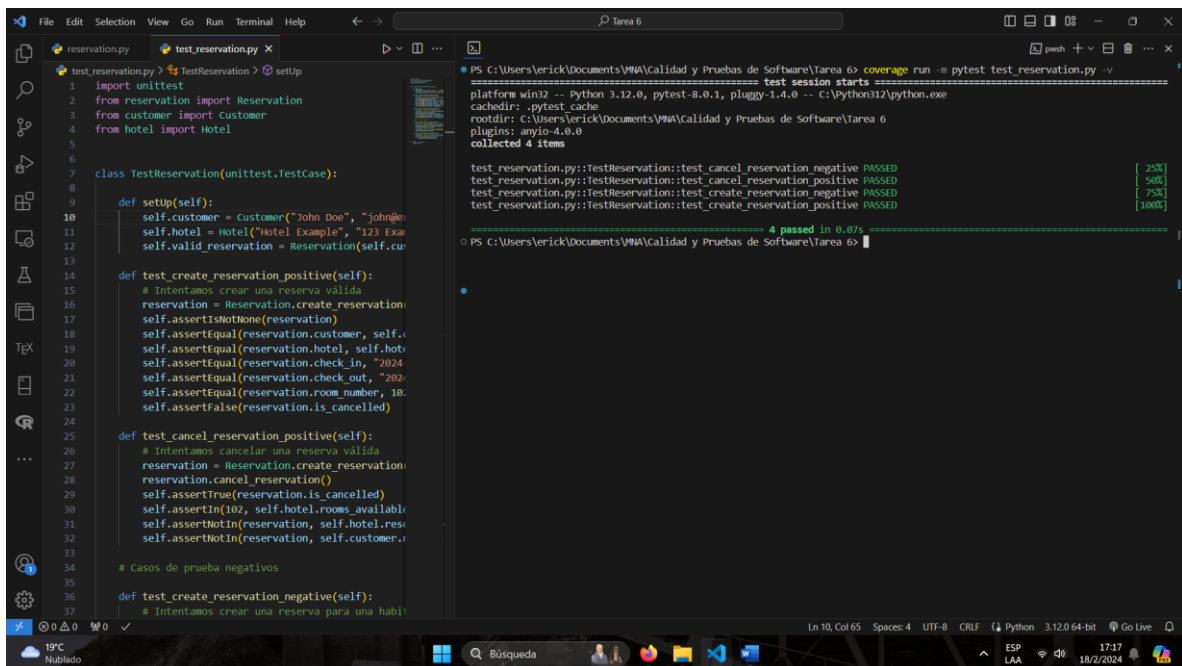
8 passed in 0.00s

Documento de pruebas para la clase Reservation:



```
1 import unittest
2 from reservation import Reservation
3 from customer import Customer
4 from hotel import Hotel
5
6
7 class TestReservation(unittest.TestCase):
8
9     def setUp(self):
10         self.customer = Customer("John Doe", "john@example.com")
11         self.hotel = Hotel("Hotel Example", "123 Example St", "555-1234", [101, 102])
12         self.valid_reservation = Reservation(self.customer, self.hotel, "2024-03-01", "2024-03-05", 101)
13
14     def test_create_reservation_positive(self):
15         # Intentamos crear una reserva valida
16         reservation = Reservation.create_reservation(self.customer, self.hotel, "2024-03-10", "2024-03-15", 102)
17         self.assertIsNotNone(reservation)
18         self.assertEqual(reservation.customer, self.customer)
19         self.assertEqual(reservation.hotel, self.hotel)
20         self.assertEqual(reservation.check_in, "2024-03-10")
21         self.assertEqual(reservation.check_out, "2024-03-15")
22         self.assertEqual(reservation.room_number, 102)
23         self.assertFalse(reservation.is_cancelled)
24
25     def test_cancel_reservation_positive(self):
26         # Intentamos cancelar una reserva valida
27         reservation = Reservation.create_reservation(self.customer, self.hotel, "2024-03-10", "2024-03-15", 102)
28         reservation.cancel_reservation()
29         self.assertTrue(reservation.is_cancelled)
30         self.assertIn(102, self.hotel.rooms_available)
31         self.assertNotIn(reservation, self.hotel.reservations)
32         self.assertNotIn(reservation, self.customer.reservations)
33
34     # Casos de prueba negativos
35
36     def test_create_reservation_negative(self):
37         # Intentamos crear una reserva para una habitación no disponible
```

Casos de prueba unitarios aprobados:



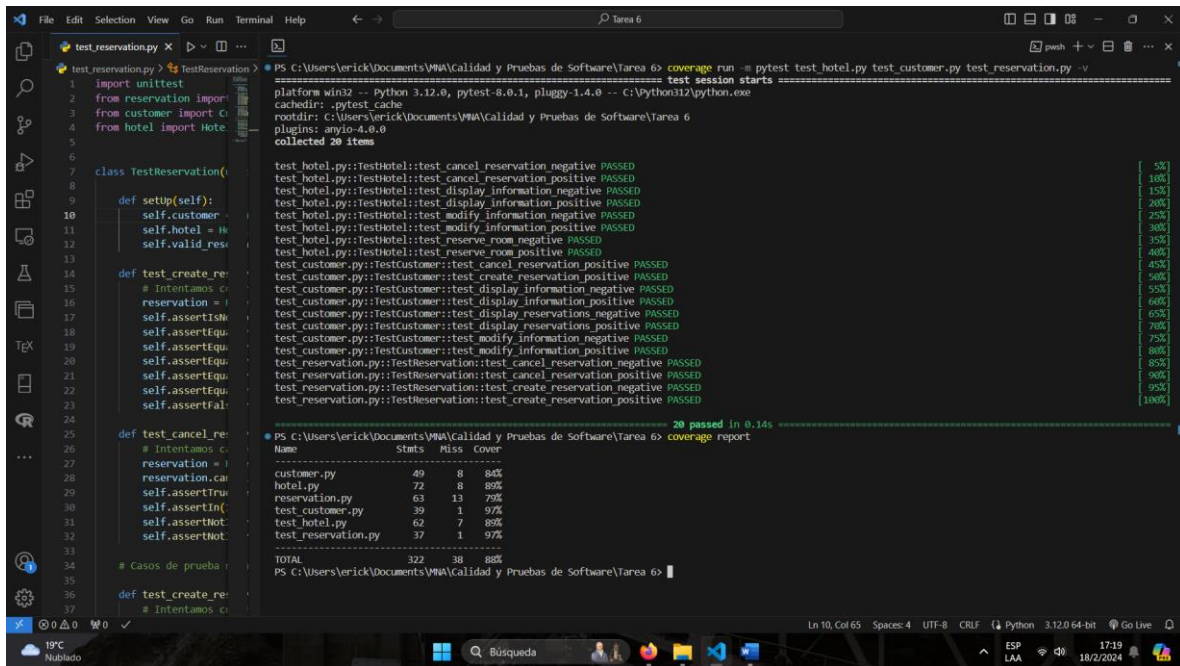
```
1 import unittest
2 from reservation import Reservation
3 from customer import Customer
4 from hotel import Hotel
5
6
7 class TestReservation(unittest.TestCase):
8
9     def setUp(self):
10         self.customer = Customer("John Doe", "john@example.com")
11         self.hotel = Hotel("Hotel Example", "123 Example St", "555-1234", [101, 102])
12         self.valid_reservation = Reservation(self.customer, self.hotel, "2024-03-01", "2024-03-05", 101)
13
14     def test_create_reservation_positive(self):
15         # Intentamos crear una reserva valida
16         reservation = Reservation.create_reservation(self.customer, self.hotel, "2024-03-10", "2024-03-15", 102)
17         self.assertIsNotNone(reservation)
18         self.assertEqual(reservation.customer, self.customer)
19         self.assertEqual(reservation.hotel, self.hotel)
20         self.assertEqual(reservation.check_in, "2024-03-10")
21         self.assertEqual(reservation.check_out, "2024-03-15")
22         self.assertEqual(reservation.room_number, 102)
23         self.assertFalse(reservation.is_cancelled)
24
25     def test_cancel_reservation_positive(self):
26         # Intentamos cancelar una reserva valida
27         reservation = Reservation.create_reservation(self.customer, self.hotel, "2024-03-10", "2024-03-15", 102)
28         reservation.cancel_reservation()
29         self.assertTrue(reservation.is_cancelled)
30         self.assertIn(102, self.hotel.rooms_available)
31         self.assertNotIn(reservation, self.hotel.reservations)
32         self.assertNotIn(reservation, self.customer.reservations)
33
34     # Casos de prueba negativos
35
36     def test_create_reservation_negative(self):
37         # Intentamos crear una reserva para una habitación no disponible
```

```
PS C:\Users\erick\Documents\VMWA\Calidad y Pruebas de Software\Tarea 6> coverage run --pytest test_reservation.py -v
===== test session starts =====
platform win32 -- Python 3.12.0, pytest-8.0.1, pluggy-1.4.0 -- C:\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\erick\Documents\VMWA\Calidad y Pruebas de Software\Tarea 6
plugins: anyio-4.0.0
collected 4 items

test_reservation.py::TestReservation::test_cancel_reservation_negative PASSED [ 25%]
test_reservation.py::TestReservation::test_cancel_reservation_positive PASSED [ 50%]
test_reservation.py::TestReservation::test_create_reservation_negative PASSED [ 75%]
test_reservation.py::TestReservation::test_create_reservation_positive PASSED [100%]

===== 4 passed in 0.07s =====
PS C:\Users\erick\Documents\VMWA\Calidad y Pruebas de Software\Tarea 6>
```


Cobertura final de los casos de prueba:



The screenshot shows a VS Code editor with a Python test file on the left and its execution output on the right. The test file, `test_reservation.py`, contains a `TestReservation` class with methods for creating, canceling, and displaying reservations. The output window shows the results of running the tests using `coverage run` and `coverage report`.

```
test_reservation.py > TestReservation >
1 import unittest
2 from reservation import Reservation
3 from customer import Customer
4 from hotel import Hotel
5
6
7 class TestReservation(unittest.TestCase):
8
9     def setUp(self):
10         self.customer = Customer()
11         self.hotel = Hotel()
12         self.reservation = Reservation()
13
14     def test_create_reservation(self):
15         # Intentamos crear una reserva
16         reservation = self.reservation.create_reservation(
17             self.customer, self.hotel, "2024-01-01", "2024-01-05")
18         self.assertEqual(reservation, self.reservation)
19         self.assertEqual(reservation.status, "confirmed")
20         self.assertEqual(reservation.hotel, self.hotel)
21         self.assertEqual(reservation.customer, self.customer)
22         self.assertEqual(reservation.start_date, "2024-01-01")
23         self.assertEqual(reservation.end_date, "2024-01-05")
24
25     def test_cancel_reservation(self):
26         # Intentamos cancelar una reserva
27         reservation = self.reservation.create_reservation(
28             self.customer, self.hotel, "2024-01-01", "2024-01-05")
29         self.assertEqual(reservation.status, "confirmed")
30         self.assertEqual(reservation.hotel, self.hotel)
31         self.assertEqual(reservation.customer, self.customer)
32         self.assertEqual(reservation.start_date, "2024-01-01")
33         self.assertEqual(reservation.end_date, "2024-01-05")
34
35     def test_create_reservation(self):
36         # Intentamos crear una reserva
```

```
PS C:\Users\verick\Documents\MAA\Calidad y Pruebas de Software\Tarea 6> coverage run --pytest test_hotel.py test_customer.py test_reservation.py -v
test session starts
platform win32 -- python 3.12.0, pytest-8.0.1, pluggy-1.4.0 -- C:\Python312\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\verick\Documents\MAA\Calidad y Pruebas de Software\Tarea 6
plugins: anyio-4.0.0
collected 20 items

test_hotel.py::TestHotel::test_cancel_reservation_negative PASSED
test_hotel.py::TestHotel::test_cancel_reservation_positive PASSED
test_hotel.py::TestHotel::test_display_information_negative PASSED
test_hotel.py::TestHotel::test_display_information_positive PASSED
test_hotel.py::TestHotel::test_modify_information_negative PASSED
test_hotel.py::TestHotel::test_modify_information_positive PASSED
test_hotel.py::TestHotel::test_reserve_room_negative PASSED
test_hotel.py::TestHotel::test_reserve_room_positive PASSED
test_customer.py::TestCustomer::test_cancel_reservation_positive PASSED
test_customer.py::TestCustomer::test_create_reservation_positive PASSED
test_customer.py::TestCustomer::test_display_information_negative PASSED
test_customer.py::TestCustomer::test_display_information_positive PASSED
test_customer.py::TestCustomer::test_display_reservations_negative PASSED
test_customer.py::TestCustomer::test_display_reservations_positive PASSED
test_customer.py::TestCustomer::test_modify_information_negative PASSED
test_customer.py::TestCustomer::test_modify_information_positive PASSED
test_reservation.py::TestReservation::test_cancel_reservation_negative PASSED
test_reservation.py::TestReservation::test_cancel_reservation_positive PASSED
test_reservation.py::TestReservation::test_create_reservation_negative PASSED
test_reservation.py::TestReservation::test_create_reservation_positive PASSED

----- 20 passed in 0.14s -----
PS C:\Users\verick\Documents\MAA\Calidad y Pruebas de Software\Tarea 6> coverage report
-----
Name                               Stmts   Miss  Cover
-----
customer.py                         49      8   84%
hotel.py                            72      8   89%
reservation.py                      68     13   79%
test_customer.py                   39      1   97%
test_hotel.py                      62      7   89%
test_reservation.py                37      1   97%
-----
TOTAL                               327     38   88%
PS C:\Users\verick\Documents\MAA\Calidad y Pruebas de Software\Tarea 6>
```

Resultados:

En general se reportó una cobertura del 88% que es mayor al requerido en la práctica, para realizar el análisis de cobertura se empleó la librería *PyTest* junto a *Coverage*.