



---

**Materia:**  
Computo en la Nube

**Actividad:**  
Actividad 6.2 Ejercicio de programacion 3 y pruebas de  
unidad

**Repositorio GitHub**

Disponible en:

[https://github.com/iracheta09/A00322423\\_A6.2.git](https://github.com/iracheta09/A00322423_A6.2.git)

**Alumno:**  
Cesar Cecilio Iracheta Mata  
A00322423

**Fecha:**  
20 de febrero de 2026

## **1. Introducción**

En esta actividad se desarrolló un sistema en Python para la gestión de clientes, hoteles y reservaciones, aplicando buenas prácticas de ingeniería de software, pruebas unitarias, control de versiones y herramientas de análisis estático.

El objetivo principal fue implementar funcionalidades mediante código limpio y verificable, asegurando su correcta ejecución a través de pruebas automatizadas, cobertura de código y herramientas de calidad.

## 2. Ejecución de pruebas unitarias

Se implementaron pruebas unitarias utilizando el módulo unittest de Python para validar el comportamiento de los módulos desarrollados:

- customer
- hotel
- reservation
- storage

Estas pruebas verifican tanto escenarios positivos como negativos, incluyendo validaciones, errores controlados y persistencia de información.

### Evidencia:

```
python -m unittest discover -s tests -p "test_*.py" -v
```

```
● @iracheta09 → /workspaces/A00322423_A6.2 (main) $ python -m unittest discover -s tests -p "test_*.py" -v
  test_create_and_get_customer (test_customer.TestCustomerRepository.test_create_and_get_customer) ... ok
  test_delete_customer (test_customer.TestCustomerRepository.test_delete_customer) ... ok
  test_delete_nonexistent_raises (test_customer.TestCustomerRepository.test_delete_nonexistent_raises) ... ok
  test_duplicate_customer_id_raises (test_customer.TestCustomerRepository.test_duplicate_customer_id_raises) ... ok
  test_empty_name_raises (test_customer.TestCustomerRepository.test_empty_name_raises) ... ok
  test_list_customers (test_customer.TestCustomerRepository.test_list_customers) ... ok
  test_update_customer (test_customer.TestCustomerRepository.test_update_customer) ... ok
  test_update_nonexistent_raises (test_customer.TestCustomerRepository.test_update_nonexistent_raises) ... ok
  test_create_hotel (test_hotel.TestHotelRepository.test_create_hotel) ... ok
  test_duplicate_hotel_raises (test_hotel.TestHotelRepository.test_duplicate_hotel_raises) ... ok
  test_release_room (test_hotel.TestHotelRepository.test_release_room) ... ok
  test_reserve_no_rooms_raises (test_hotel.TestHotelRepository.test_reserve_no_rooms_raises) ... ok
  test_reserve_room (test_hotel.TestHotelRepository.test_reserve_room) ... ok
  test_cancel_nonexistent_raises (test_reservation.TestReservationRepository.test_cancel_nonexistent_raises) ... ok
  test_cancel_reservation_releases_room (test_reservation.TestReservationRepository.test_cancel_reservation_releases_room) ... ok
  test_cancel_twice_raises (test_reservation.TestReservationRepository.test_cancel_twice_raises) ... ok
  test_create_reservation_customer_not_found (test_reservation.TestReservationRepository.test_create_reservation_customer_not_found) ... ok
  test_create_reservation_hotel_not_found (test_reservation.TestReservationRepository.test_create_reservation_hotel_not_found) ... ok
  test_create_reservation_no_rooms_available (test_reservation.TestReservationRepository.test_create_reservation_no_rooms_available) ... ok
  test_create_reservation_success (test_reservation.TestReservationRepository.test_create_reservation_success) ... ok
  test_load_empty_file_returns_default (test_storage.TestStorage.test_load_empty_file_returns_default)
  Returns default when file exists but content is empty. ... ok
  test_load_invalid_json_returns_default (test_storage.TestStorage.test_load_invalid_json_returns_default)
  Returns default when file contains invalid JSON. ... [storage] Error reading JSON file '/tmp/tmp6c7pvecr/bad.json': Expecting property name
  ok
  test_load_missing_file_returns_default (test_storage.TestStorage.test_load_missing_file_returns_default)
  Returns provided default when file does not exist. ... ok
  test_save_and_load_roundtrip (test_storage.TestStorage.test_save_and_load_roundtrip)
  Saves JSON and loads it back without data loss. ... ok
```

---

Ran 24 tests in 0.030s

OK

### 3. Cobertura de código

Se utilizó la herramienta coverage para medir el porcentaje de código ejecutado por las pruebas unitarias, con el fin de garantizar que los módulos implementados fueran probados adecuadamente.

Resultados obtenidos:

- Cobertura total del proyecto: **93%**
- customer.py: 92%
- hotel.py: 85%
- reservation.py: 88%
- storage.py: 100%

Esto demuestra que la mayor parte del código desarrollado fue validado mediante pruebas automatizadas.

#### Evidencia:

coverage report -m

Name	Stmts	Miss	Cover	Missing
src/__init__.py	0	0	100%	
src/customer.py	66	5	92%	28-29, 33, 52, 74
src/hotel.py	74	11	85%	30-31, 35, 66, 68, 70, 96, 110, 115, 121-122
src/reservation.py	77	9	88%	45-46, 50, 86, 88, 90, 94, 144-145
src/storage.py	21	0	100%	
tests/test_customer.py	45	1	98%	59
tests/test_hotel.py	36	1	97%	47
tests/test_reservation.py	52	1	98%	75
tests/test_storage.py	27	1	96%	43
TOTAL	398	29	93%	

## 4. Análisis estático del código

Para asegurar la calidad del código se utilizaron herramientas de análisis estático.

### 4.1 flake8

Se ejecutó flake8 para verificar el cumplimiento de estándares de codificación (PEP-8).

Resultado:

- No se detectaron errores de estilo.

Evidencia:

```
flake8 src tests
```

```
› (.venv) @iracheta09 →/workspaces/A00322423_A6.2 (main) $ flake8 src tests
› (.venv) @iracheta09 →/workspaces/A00322423_A6.2 (main) $ █
```

### 4.2 pylint

Se utilizó pylint para evaluar calidad, estructura y buenas prácticas del código.

Resultado:

- Calificación obtenida: **10.00/10**

Esto confirma que el código cumple estándares profesionales de desarrollo.

Evidencia:

```
pylint src
```

```
● (.venv) @iracheta09 →/workspaces/A00322423_A6.2 (main) $ pylint src
-----
Your code has been rated at 10.00/10 (previous run: 10.00/10, +0.00)
○ (.venv) @iracheta09 →/workspaces/A00322423_A6.2 (main) $
```

## 5. Control de versiones con Git

El proyecto fue gestionado mediante Git y GitHub, registrando cada avance mediante commits estructurados, permitiendo trazabilidad del desarrollo.

**Se realizaron commits para:**

- Estructura inicial del proyecto
- Implementación de almacenamiento
- CRUD de clientes
- Gestión de hoteles
- Flujo de reservaciones
- Configuración de herramientas

Esto permitió documentar la evolución del proyecto y asegurar buenas prácticas de gestión de versiones.

The screenshot shows a GitHub commit history for a repository. The main dropdown menu is set to 'main'. The search bar is set to 'All users' and the time filter is set to 'All time'. The commits are listed in chronological order from top to bottom:

- o- Commits on Feb 19, 2026
  - feat: implement reservation flow with unit tests  
iracheta09 committed yesterday c0ada73
  - feat: implement reservation flow with unit tests  
iracheta09 committed yesterday e98547d
  - feat: implement hotel management with room reservation logic  
iracheta09 committed yesterday fc9b1d0
  - chore: add .gitignore for python artifacts  
iracheta09 committed yesterday ccf8a0
  - refactor: improve storage tests and JSON handling  
iracheta09 committed yesterday a03021b
  - feat: implement customer CRUD with persistence and tests  
iracheta09 committed yesterday 94efb99
  - chore: enable unittest discovery  
iracheta09 committed yesterday 5ef573e
  - chore: add dev tools (coverage, flake8, pylint)  
iracheta09 committed yesterday ed880dc
  - chore: init project structure for A6.2  
iracheta09 committed yesterday 62b4ddc
  - Initial commit  
iracheta09 authored yesterday Verified 65b1451

## 6. Conclusiones

La actividad permitió aplicar de manera práctica conceptos fundamentales de ingeniería de software:

- Desarrollo modular en Python
- Pruebas unitarias automatizadas
- Medición de cobertura
- Análisis estático del código
- Uso profesional de Git y GitHub

El sistema implementado demuestra un funcionamiento correcto y robusto, con validación de escenarios positivos y negativos, garantizando confiabilidad y calidad del software desarrollado.

Asimismo, el uso de herramientas de calidad y control de versiones permitió validar el proceso de desarrollo y fortalecer la confiabilidad del software implementado.

**El proyecto desarrollado integra prácticas modernas de desarrollo de software, evidenciando la importancia de las pruebas, la automatización y el control de versiones en entornos profesionales.**