

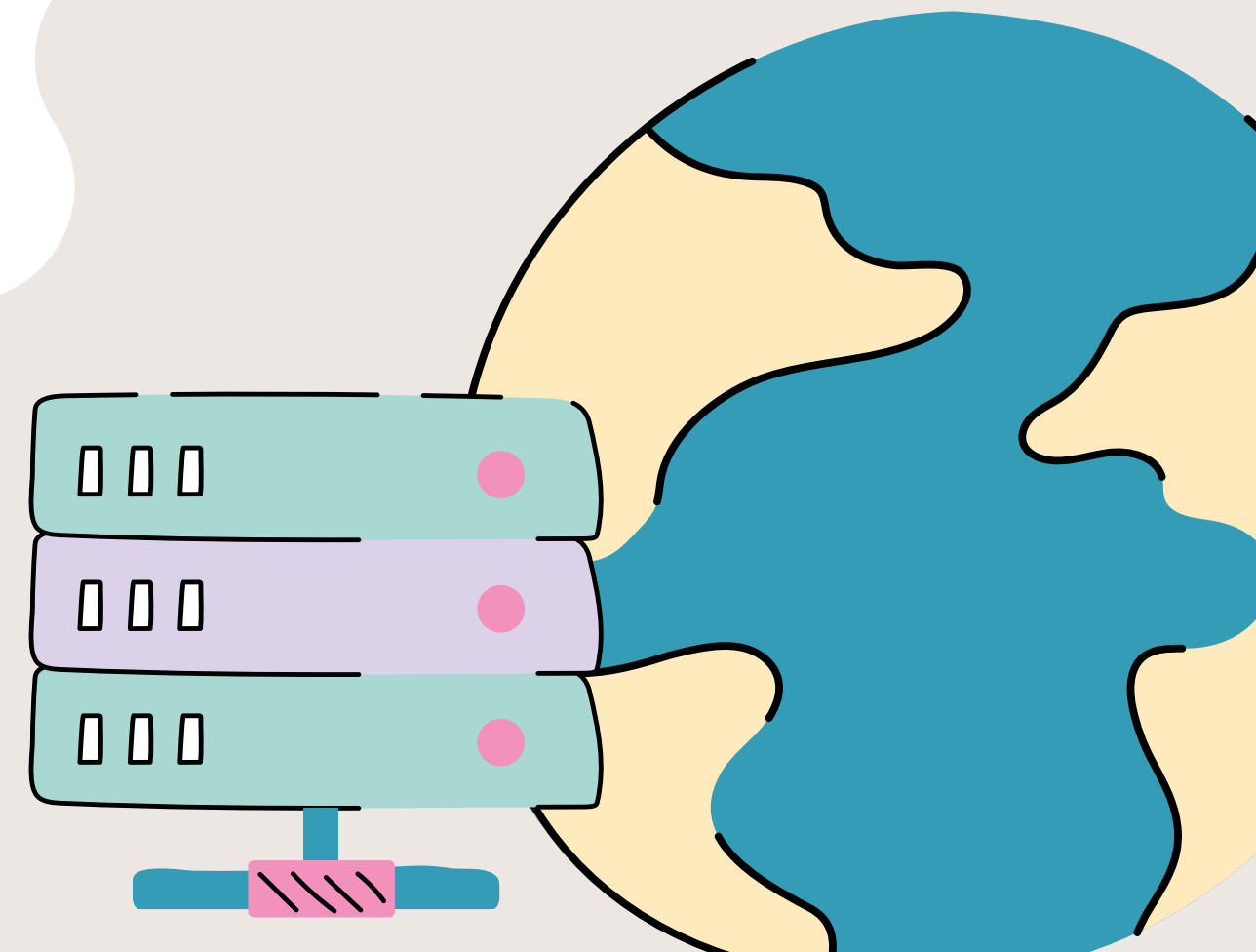


# Testing en Python

La mejor inversión que puedes hacer en tus  
proyectos



Por Juan Duran



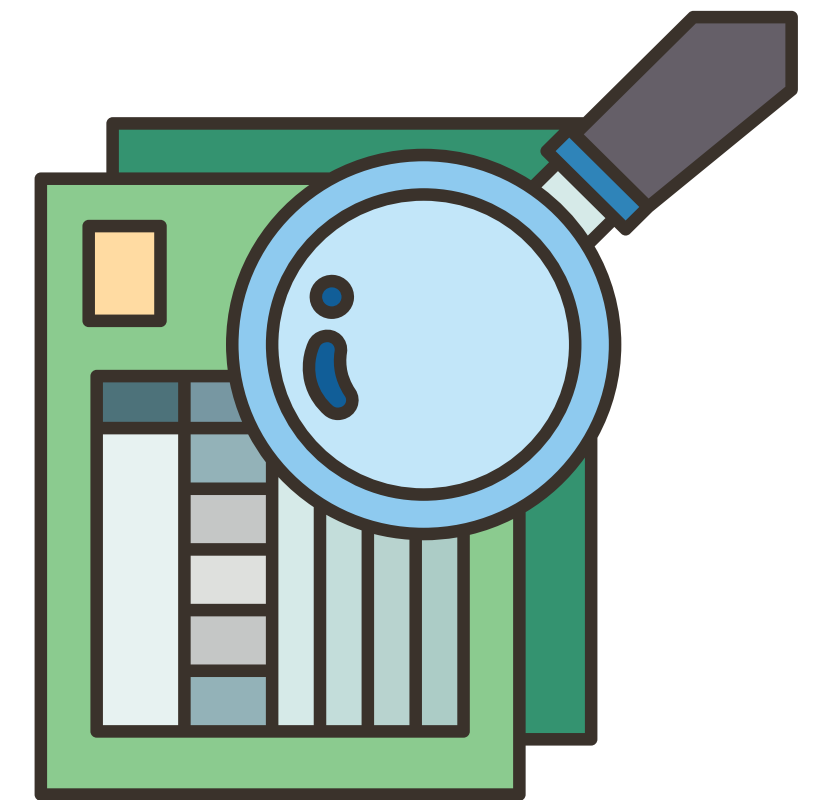
# ¿Por qué importa el testing?

**Testear** no es una tarea secundaria. Es una **parte esencial del desarrollo** de software. Y no, no es solo para grandes corporaciones: si tú escribes código, el testing es para ti.

🔍 **Razones** por las que deberías testear tu código:

- **Evita errores** inesperados, incluso cuando cambias algo que parecía no tener relación.
- **Aumenta tu confianza** a la hora de añadir nuevas funcionalidades.
- **Reduce el tiempo** que dedicas a buscar y corregir errores.
- Te permite **trabajar en equipo** sin miedo a romper lo que otros han hecho.
- **Mejora la documentación** del sistema, ya que los tests muestran cómo debería comportarse el código.

Un código sin tests es como un coche sin frenos: puede parecer que va bien... hasta que deja de ir.



# Tipos de pruebas en testing

## ◆ Tests unitarios

Son los más básicos. Comprueban que funciones o métodos individuales hagan lo que se espera. Por ejemplo, si tienes una función que suma dos números, el test verifica que devuelve el resultado correcto.

## ◆ Tests de integración

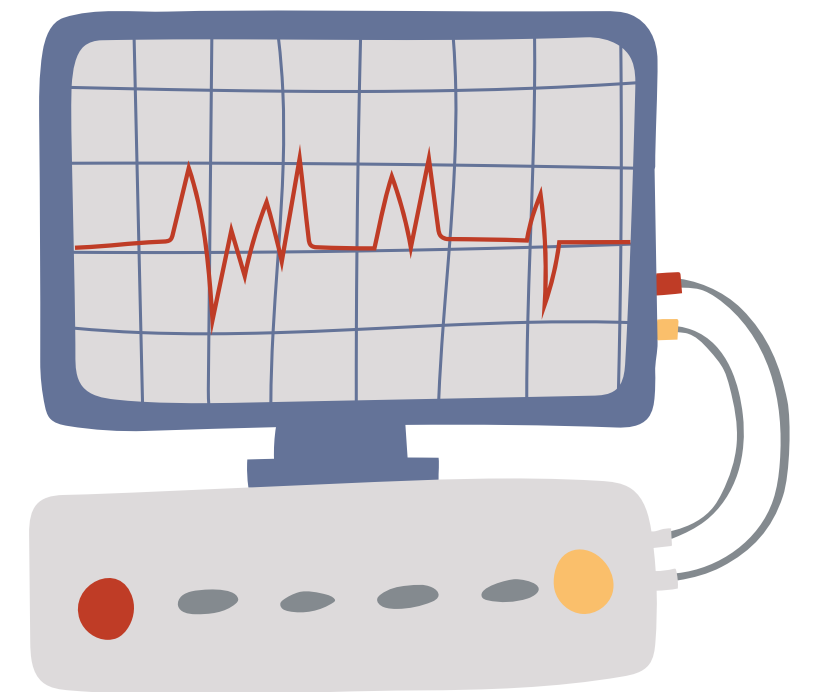
Aseguran que varios componentes del sistema funcionen bien juntos. Por ejemplo, comprobar que una función que accede a una base de datos devuelva los datos correctos.

## ◆ Tests funcionales (end-to-end)

Simulan el comportamiento del usuario y comprueban que el sistema completo responde adecuadamente. Son más complejos, pero muy valiosos.

## ◆ Tests de regresión

Se usan para garantizar que una funcionalidad que ya funcionaba antes, sigue funcionando después de hacer cambios en el código.



# ¿Qué deberías testear y qué no?

No todo el código necesita ser testeado. Hay que **priorizar**.

## ✓ Deberías testear:

- Toda función que contenga lógica: condicionales, cálculos, procesamiento de datos.
- Casos límite: entradas vacías, valores nulos, errores esperados.
- Reglas de negocio críticas: procesos que no pueden fallar.
- Funcionalidades nuevas, antes de que crezcan o se vuelvan más complejas.

## ✗ Puedes dejar sin testear:

- Funciones extremadamente simples (por ejemplo, retornar un valor constante).
- Código provisional que sabes que vas a eliminar o rehacer.
- Librerías externas (ya están testeadas por sus creadores).

La clave es buscar **equilibrio**: no hace falta tener cientos de tests, sino tener los adecuados.



# Cómo escribir buenos tests

Escribir **tests** no consiste solo en que “funcionen”, sino en que sean **útiles, claros y fáciles de mantener**.

📌 **Consejos** para escribir buenos tests:

- **Nómbrales** con claridad. Un buen nombre describe lo que se está probando.
- Sé **específico**. Cada test debería centrarse en una única cosa.
- **Utiliza ejemplos reales**. Usa datos que se parezcan a lo que los usuarios podrían introducir.
- **Cubre** diferentes **escenarios**. Éxito, fallo, límites, errores comunes.
- **Mantenlos simples**. Un test complicado es difícil de entender y mantener.
- **Evita dependencias externas**. Si tu test falla porque la API de otro servicio no responde, no es fiable.

Los **tests** son como **instrucciones** que le das a tu futuro yo para entender qué esperabas que hiciera el código.



# Cuándo escribir los tests

Hay **diferentes enfoques**, y cada uno tiene sus ventajas. Lo importante es integrar el testing en tu flujo de trabajo.

🕒 **Opciones** más comunes:

- **Después de escribir el código.** Es el método más extendido. Escribes la función y luego pruebas que haga lo que tiene que hacer.
- **Durante el desarrollo.** Vas testeando cada parte mientras avanzas.
- **Test-Driven Development (TDD).** Primero escribes el test, luego el código necesario para que pase. Es una técnica potente, pero requiere disciplina.

🎯 **Consejo:** Empieza escribiendo tests después del código. Cuando te sientas más cómodo, puedes probar TDD.

El mejor momento para testear tu código es... **ahora**. No esperes a que algo se rompa.



# Automatización del testing

Hacer tests manuales puede estar bien al principio, pero en cuanto tu **proyecto crece**, necesitas **automatizar**. Es la única forma de escalar sin perder el control.

⚙️ ¿Qué significa **automatizar**?

- Que los tests se ejecuten solos cada vez que haces un cambio.
- Que puedas detectar errores antes de que lleguen al usuario.
- Que no tengas que comprobar manualmente todo en cada modificación.

💡 Usa **herramientas** como:

- **pytest** para ejecutar tests en Python.
- **GitHub Actions** para integrar tests en tus repositorios.

**Automatizar** los tests es como tener un escáner que te avisa cuando algo no encaja. Te quita un peso de encima.






# Testing en equipo

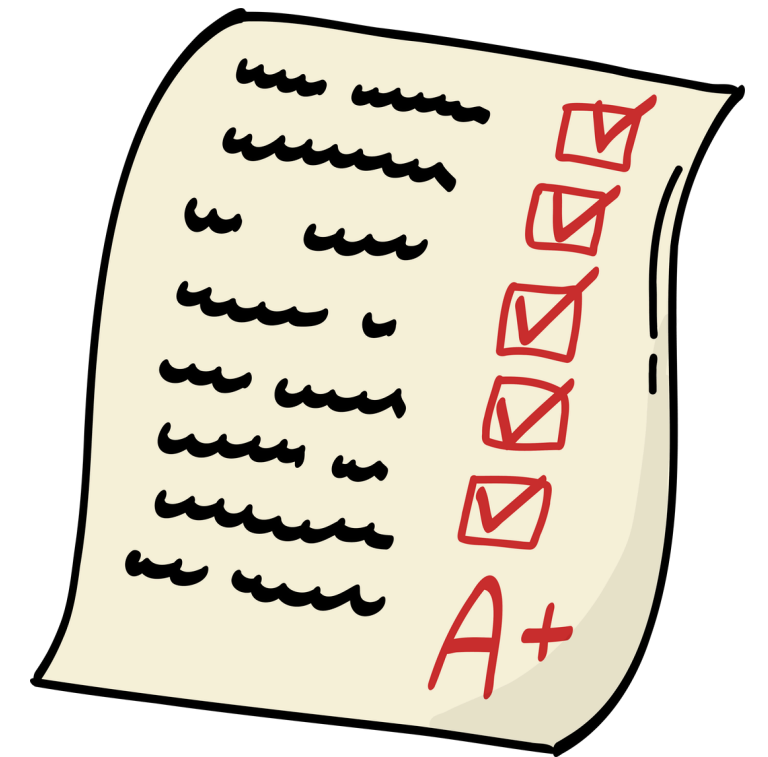
Cuando **trabajas** en **equipo**, los **tests** son aún más **importantes**. No solo protegen tu código, sino el de todos.

## Beneficios del testing en equipos:

- **Facilita la integración** de cambios de múltiples personas.
- **Ayuda** a entender cómo debería funcionar una parte del sistema.
- **Evita errores** al modificar código que tú no escribiste.
- **Reduce el tiempo** que se pierde buscando bugs.

 **Ejemplo real:** si un compañero cambia una función que tú usas, y tus tests fallan, el equipo lo sabrá de inmediato antes de hacer un merge.

Los tests no solo protegen el código, también protegen la comunicación y la confianza entre personas.





# Buenas prácticas que realmente ayudan

Algunas **recomendaciones** que pueden marcar la diferencia en tu forma de trabajar:

- ✅ **Organiza bien tus tests.** Guarda los archivos en una carpeta específica (tests/) y utiliza nombres descriptivos.
- ✅ **Usa mocks cuando toque.** Así puedes simular respuestas de APIs, bases de datos o servicios externos.
- ✅ **Mide la cobertura.** Herramientas como coverage.py te ayudan a saber cuánto de tu código está cubierto por tests.
- ✅ **Revisa y actualiza.** Un test desactualizado puede dar una falsa sensación de seguridad.
- ✅ **No apuntes al 100%.** Es mejor tener un 80% bien cubierto que un 100% lleno de tests innecesarios o mal hechos.

La calidad de tus tests dice mucho sobre tu forma de programar. No se trata de cantidad, sino de calidad.



# El testing no es un extra, es una necesidad

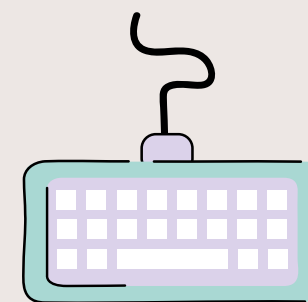
Podemos resumir todo esto con una idea muy clara: **el testing no es un añadido, es una parte clave del desarrollo moderno**. Si quieres escribir buen código, profesionalizar tu forma de trabajar y tener menos dolores de cabeza, el testing es tu aliado.

🔍 Qué te llevas de esta presentación:

- **Testear mejora tu código** y tu tranquilidad.
- **No todo se testea**, pero lo importante sí.
- Es **mejor** tener pocos **tests útiles** que muchos inútiles.
- **Automatizar** es esencial para crecer sin miedo.
- **En equipo, el testing** es una red de **seguridad**.

**No se trata de evitar los errores, sino de detectarlos antes de que hagan daño. Eso es el testing.**





# Gracias



Por Juan Duran

“Coding, Gaming and Leveling Up”