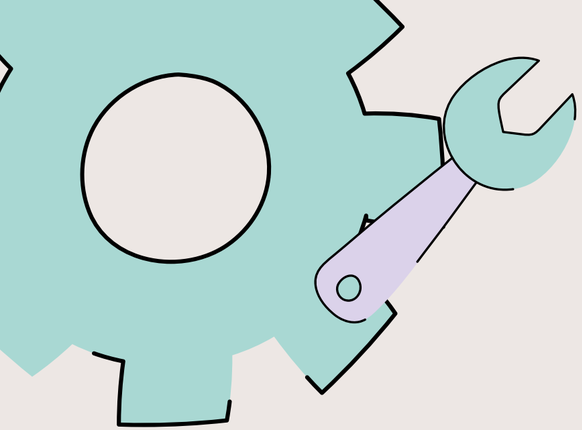
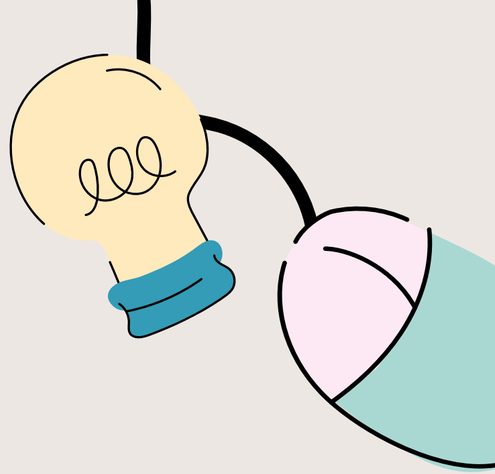
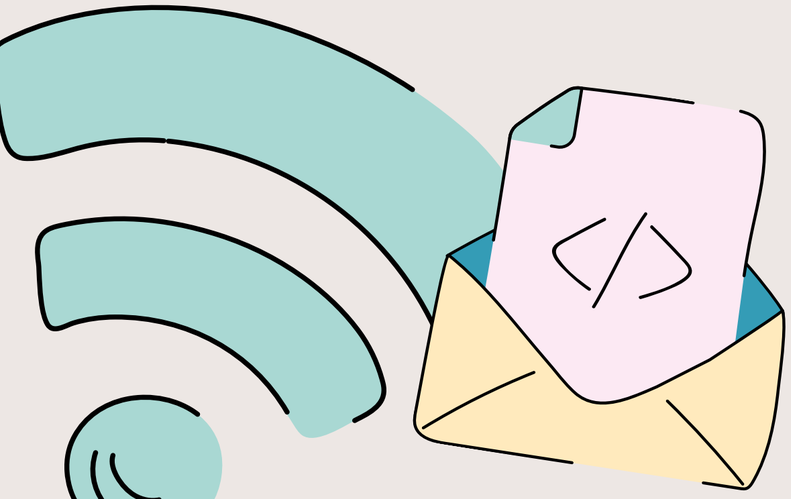


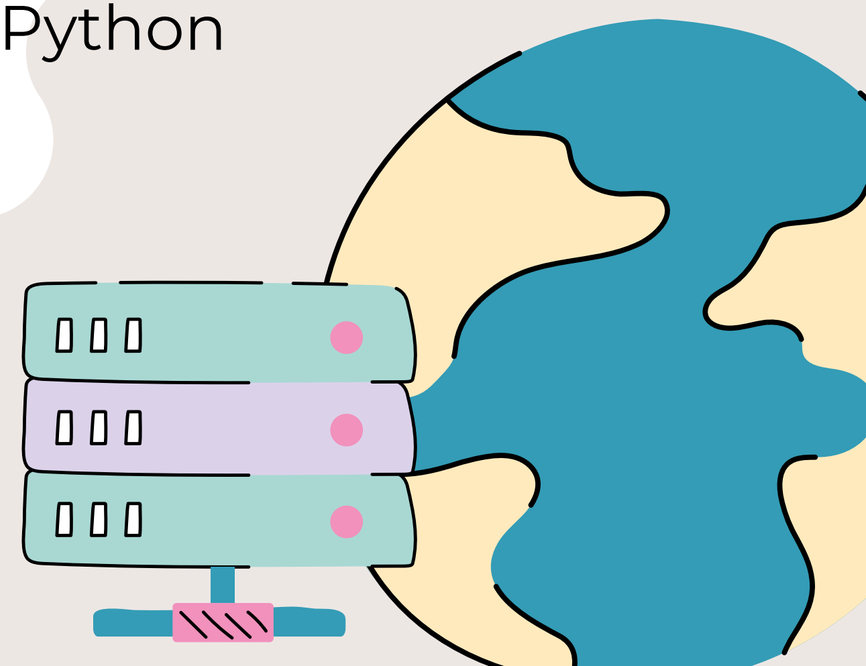


Descubriendo Pytest

Mi librería favorita para testear en Python



Por Juan Duran



¿Por qué empecé a usar Pytest?

🧩 Una **solución simple** a un problema común

Al principio, testear me parecía un paso más que me quitaba tiempo. Pensaba: “Ya sé que el código funciona, ¿para qué escribir tests?”
Pero cuando el proyecto crece, cuando compartes código con otras personas o simplemente cuando vuelves a abrir un archivo después de semanas, o meses... los errores aparecen, si, créeme.
Y ahí entendí que **testear no es opcional**, es autoprotección!

¿**Por qué Pytest**?

- ✅ Es muy **fácil** de aprender.
- ✅ **No necesitas** escribir clases ni estructuras complicadas.
- ✅ Te **muestra** los errores de forma clara.
- ✅ Te **permite** testear muchos escenarios sin repetir todo.

¿Qué es exactamente Pytest?

🔍 Una **librería** de **testing** que se siente natural.

Pytest es una librería de Python para escribir **pruebas automatizadas**. Pero más allá de lo técnico, lo que me gusta es que se **adapta** a cada uno: puedes empezar con lo básico y escalar cuando lo necesites.

🧠 Me gusta pensar que **Pytest** es como ese amigo que:

- Te **avisa** cuando algo no está funcionando.
- Te **da** contexto del error.
- Y **no** te **complica** la vida con reglas innecesarias.

Lo usas con funciones comunes, con nombres simples, y te ayuda a mantener tu proyecto bajo control.

¿Qué me convenció de usarlo?

❤️ Lo **bueno** se nota enseguida

Después de probar otros enfoques, **Pytest** me conquistó por cosas que valoro mucho como dev:

- 🔹 Puedo **escribir** los tests como si fueran cualquier función normal.
- 🔹 Los **nombres** de los tests son **descriptivos** y fáciles de leer.
- 🔹 Me **muestra** qué falló exactamente, con detalle.
- 🔹 Me deja **reutilizar código** para no escribir lo mismo mil veces.
- 🔹 Se **integra** súper bien con otros **entornos** y **herramientas**.

💡 Y lo mejor: no tuve que cambiar nada en mi código para empezar a testear!

Así organizo mis tests con Pytest

🗂️ **Orden** y **claridad**, incluso en proyectos grandes

Para que el testing no se vuelva un lío, tengo una estructura simple que intento seguir siempre:

📁 Creo una **carpeta** llamada tests/.
📄 Cada **archivo** empieza con test_ y agrupa pruebas similares.
✍️ Cada test tiene un **nombre claro**, tipo: test_usuario_invalido.
🚫 **No mezclo lógica del programa con los tests**. Cada cosa en su lugar. Hay que ser ordenado!

Esto me ayuda a que todo sea más legible, fácil de mantener... y mucho más profesional.

Ejemplos donde Pytest me salvó

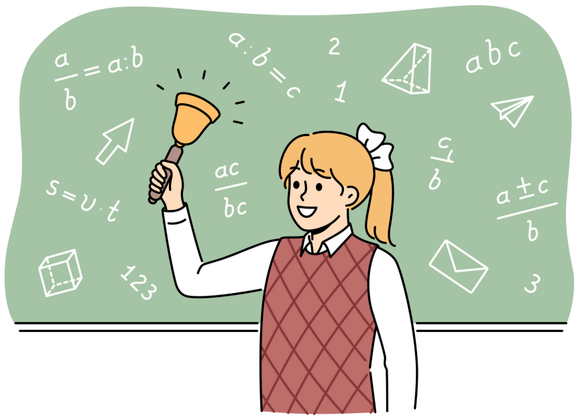
🛟 De errores invisibles a **soluciones rápidas.**

🔥 **Caso 1**: Tenía una función que procesaba datos... pero solo fallaba si le pasabas un valor raro. Un test me lo mostró en segundos.

🔥 **Caso 2**: Hice un pequeño cambio en una línea... y rompí tres funciones (Te suena?). Los tests lo detectaron antes de hacer un commit catastrófico/apocalíptico.

🔥 **Caso 3**: Estaba agregando nuevas funcionalidades, y los tests me recordaron cosas que ni me acordaba que había programado 😅

Con esto aprendí que los **tests** no son solo para **evitar errores**, también sirven como **documentación viva**.



¿Cómo testeo sin repetir todo?

🔁 **Testear muchos casos, sin copiar y pegar**

A veces tengo una función que debería funcionar con diferentes entradas. Antes escribía lo mismo muchas veces con datos distintos. Con **Pytest**, puedo hacer algo mucho más **elegante**: armar una lista de casos y **testearlos** todos de forma **automática**.

✨ Esto me permite:

- **Ver** si todos los escenarios funcionan igual.
- **Detectar** errores escondidos.
- **Tener** un test limpio y legible.

📌 Es como hacer una revisión masiva sin tener que escribir 10 veces lo mismo.

Reutilizar datos en varios tests

🧱 La clave para proyectos grandes y complejos

En algunos proyectos tengo datos que necesito usar en muchos tests. Crear esos datos a mano en cada test es... un infierno.
Con **Pytest**, puedo **definir** esos datos **una sola vez y usarlos donde quiera.**

Esto me da y me permite:

- **Más orden y limpieza**.
- **Menos líneas de código**.
- **Flexibilidad** para cambiar un dato sin tener que tocar mil archivos.

🔄 Me gusta pensar que es como tener piezas de **LEGO**: los montas una vez y después los usas en todas tus construcciones.

Lecciones que aprendí

📚 Mis **consejos personales** para testear mejor

Después de varios proyectos y algún que otro bug doloroso, te dejo mis aprendizajes más valiosos:

✔️ **Ponle nombres** que cuenten lo que el test hace.
✔️ Intenta hacer que cada **test** sea **independiente**. Si uno falla, que no se caigan todos.
✔️ **No** te **saltes** los **tests** “**obvios**”. A veces ahí están los errores.
✔️ No te obsesiones con la cobertura total. **Mejor pocos tests bien hechos, que muchos que no testean nada**.

💬 Y lo más importante: no veas los tests como un castigo... son tu red de seguridad!

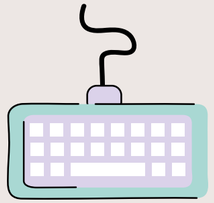
¿Por qué sigo usando Pytest?

🎯 Porque me permite escribir mejor código, sin miedo

Desde que uso **Pytest**, **programo con más confianza**. Sé que si algo se rompe, me voy a enterar rápido. Y siempre se rompe algo.

Y lo mejor es que **no me exige ser un experto** ni aprender un lenguaje nuevo. Solo tengo que pensar en lo que quiero que mi código haga... y verificar que lo haga.

💡 Si estás en el mundo Python, te **recomiendo** que le des una **oportunidad**.
Para mí, fue uno de esos descubrimientos que **cambian** la **forma** en que **trabajas**.



Gracias

Por Juan Duran

“Coding, Gaming and Leveling Up”