Pruebas del software: uso de un framework de testing

*Trabajo Práctico 1 - 29 de Agosto del 2025 Ingeniería de Software 1 (PU) - Ingeniería de Software 2 (LI / II)*

Este trabajo práctico sirve como guía básica sobre las pruebas unitarias y de integración hasta escenarios de prueba más complejos, aumentando gradualmente la complejidad. Puedes usar un lenguaje de programación y framework a elección.

**Problema:** “Sistema de Gestión de Productos en una Tienda”

Vas a desarrollar y probar un sistema básico de gestión de productos en una tienda, utilizando dos clases principales: Producto y Tienda dadas a continuación. La clase Tienda gestionará un inventario de productos, permitiendo agregar, eliminar, buscar productos, aplicar descuentos, y calcular el precio total de un carrito de compras.

# Organización del Proyecto (ejemplo python):

* **Producto** (producto.py): Clase que representa un producto en la tienda.
* **Tienda** (tienda.py): Clase que maneja el inventario y las operaciones de la tienda.
* **Tests** (test\_tienda.py test\_producto.py): Archivos donde se implementarán las pruebas utilizando el framework.

# Código base en python:

class Producto:

def init (self, nombre, precio, categoria): self.nombre = nombre

self.precio = precio self.categoria = categoria

from producto import Producto

class Tienda:

def init (self): self.inventario = []

def agregar\_producto(self, producto): self.inventario.append(producto)

def buscar\_producto(self, nombre): for producto in self.inventario:

if producto.nombre == nombre: return producto

return None

def eliminar\_producto(self, nombre): for producto in self.inventario:

if producto.nombre == nombre: self.inventario.remove(producto) return True

return False

1. **Configuración Inicial y Pruebas Básicas**

**Objetivo**: Aprender a configurar un entorno de pruebas con el framework elegido y a realizar pruebas básicas.

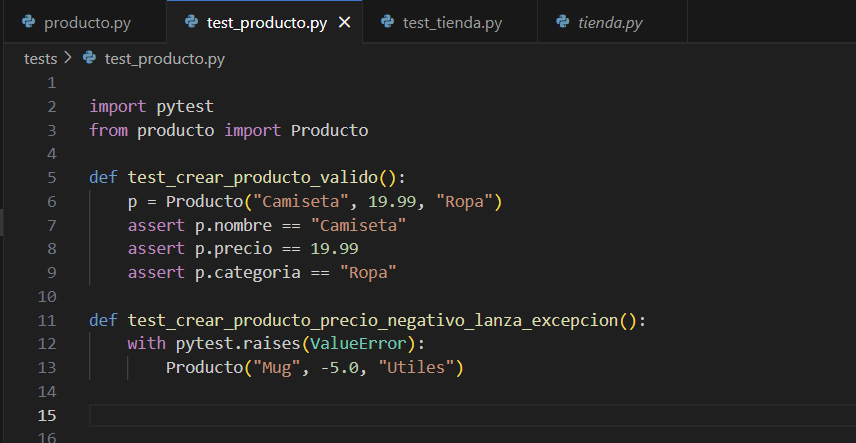
* 1. Usa la implementación base de la clase Producto con atributos como nombre, precio, y

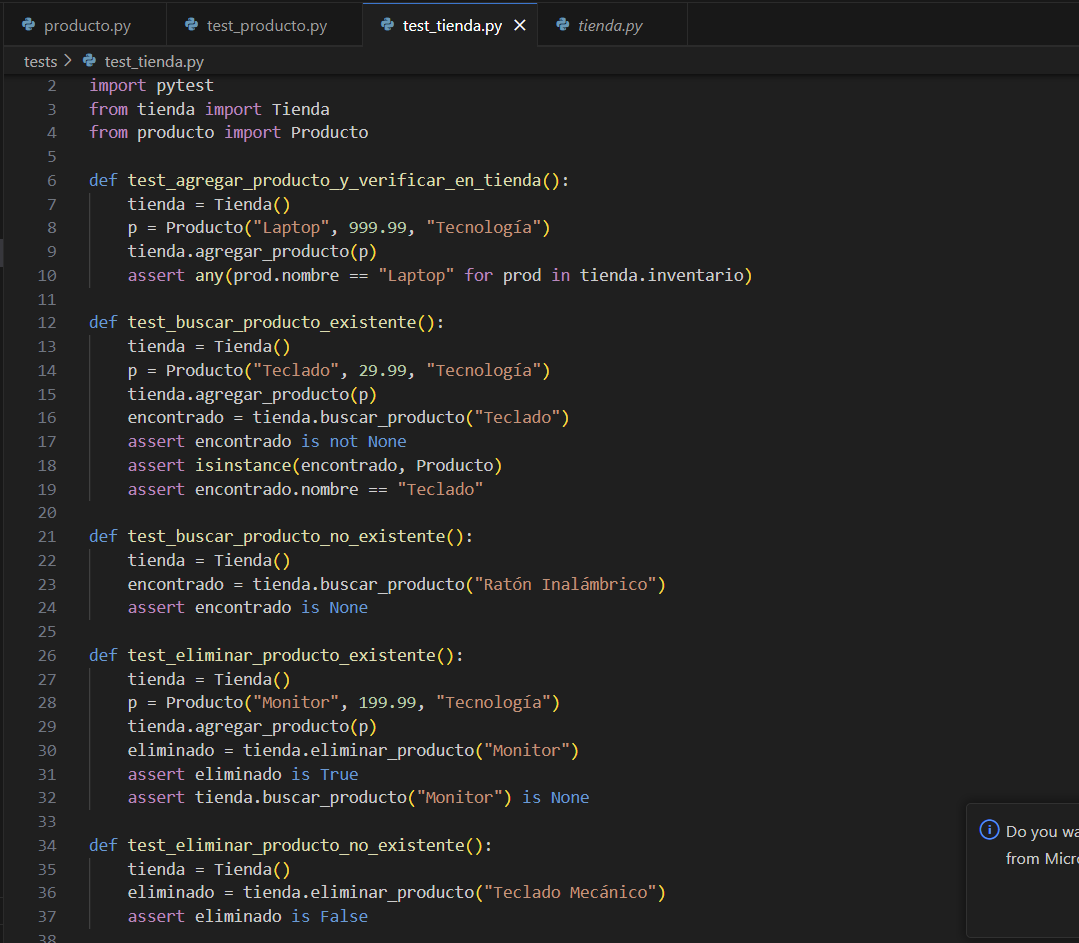
categoria.

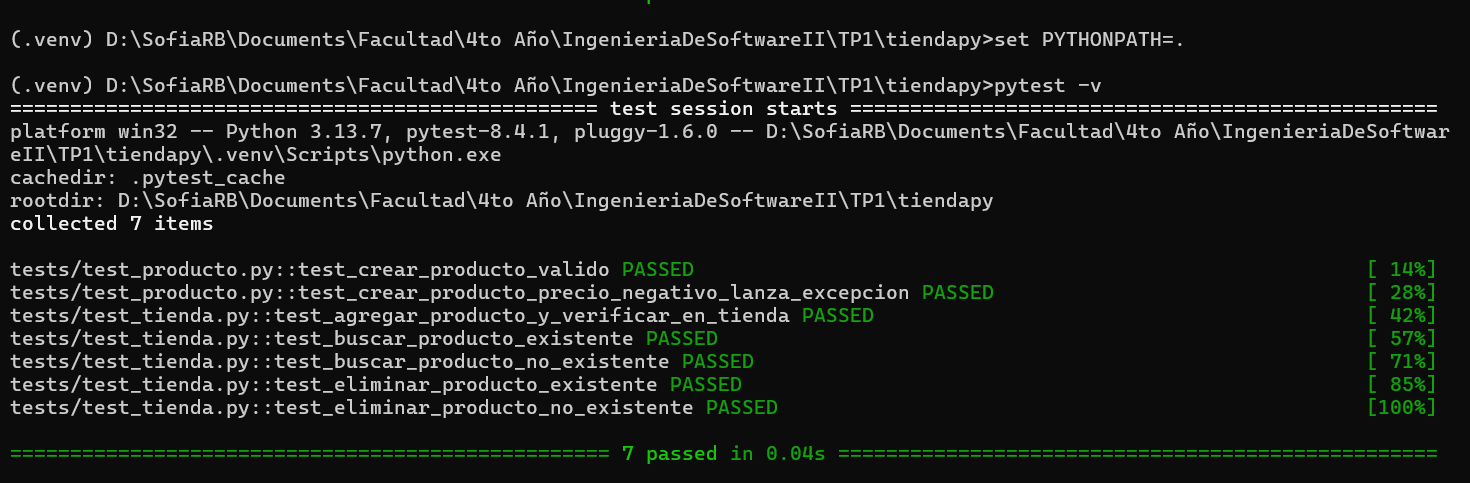
* 1. Usa la implementación base la clase Tienda con métodos para:
     1. Agregar un producto al inventario.
     2. Buscar un producto por su nombre.
     3. Eliminar un producto del inventario.

# Actividad práctica:

* Escribe pruebas básicas para estas clases y métodos utilizando el framework elegido.
* Asegúrate de cubrir casos simples, como agregar un producto y verificar que esté en la tienda, buscar un producto existente y no existente, y eliminar un producto.







# Preguntas conceptuales:

* ¿Puedes identificar pruebas de unidad y de integración en la práctica que se realizó?

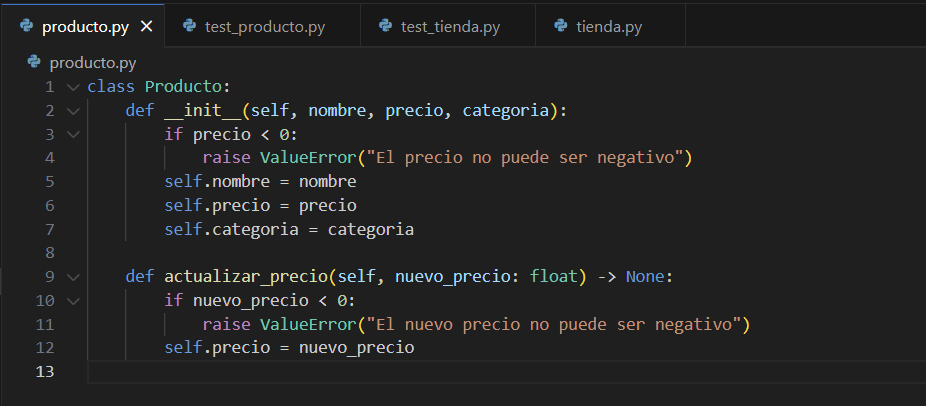
En el archivo test\_producto.py se implementaron únicamente pruebas de unidad. En ambos casos, se prueba el constructor de la clase producto el cual no depende de otros módulos del proyecto. En el primer caso, se verifica que el constructor funcione correctamente y en el segundo caso, que lance la excepción correspondiente cuando el precio es negativo.

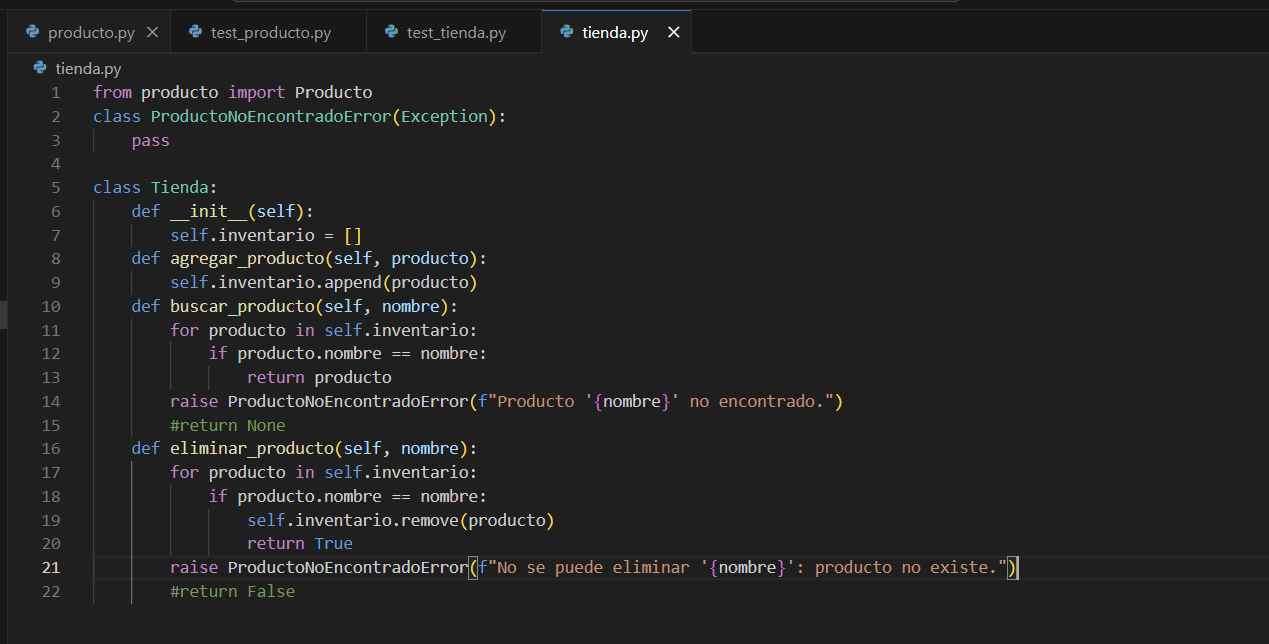
En el archivo test\_tienda.py, test\_agregar\_producto\_y\_verificar\_en\_tienda, test\_buscar\_producto\_existente y test\_eliminar\_producto\_existente hacen uso de los constructores de la clase producto por lo que implica una prueba de integración. En los métodos restantes, se prueban los métodos de la clase tienda sin dependencia de otros módulos.

# Pruebas con Excepciones

**Objetivo**: Ampliar las pruebas para manejar y verificar el uso adecuado de excepciones.

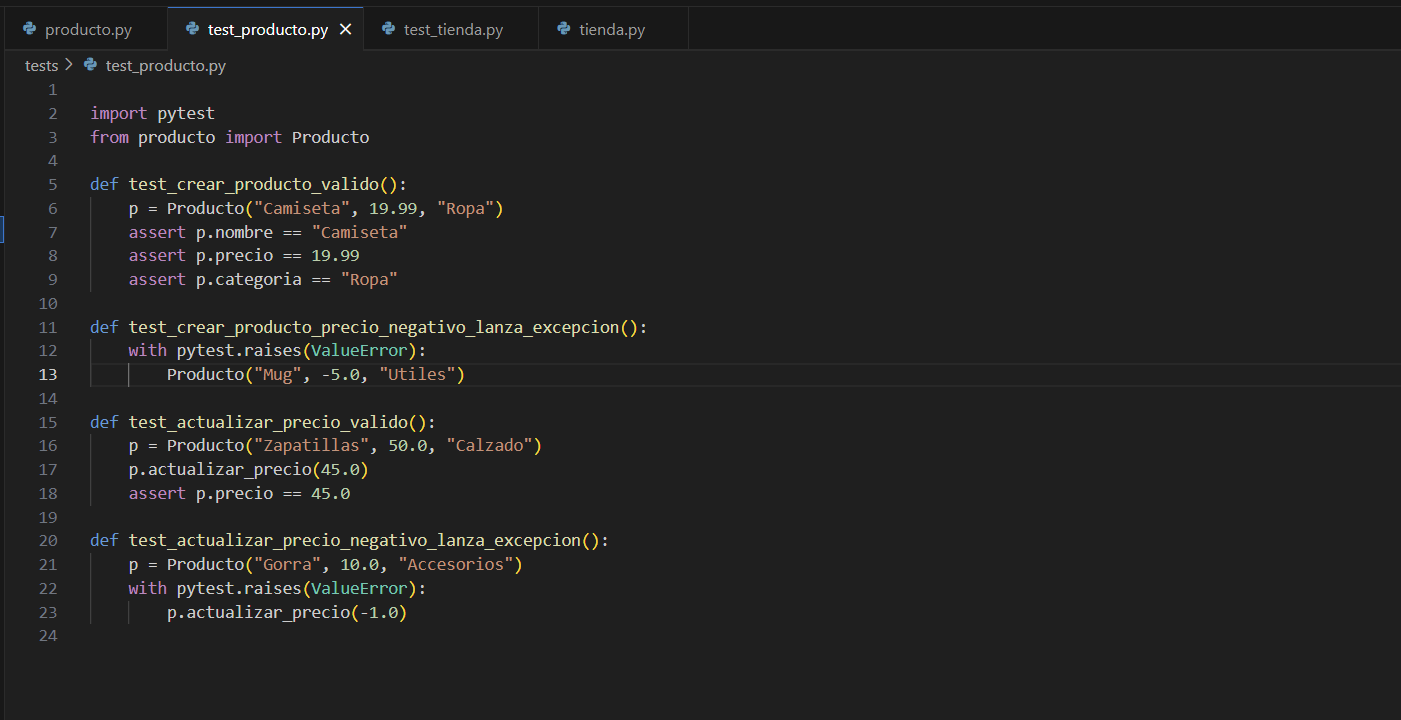
* 1. Modifica el método buscar\_producto para que lance una excepción si no se encuentra un producto.
  2. Modifica el método eliminar\_producto para que lance una excepción si se intenta eliminar un producto que no existe.
  3. Implementa un método actualizar\_precio en la clase Producto que permita cambiar el precio de un producto y lance una excepción si el nuevo precio es negativo.

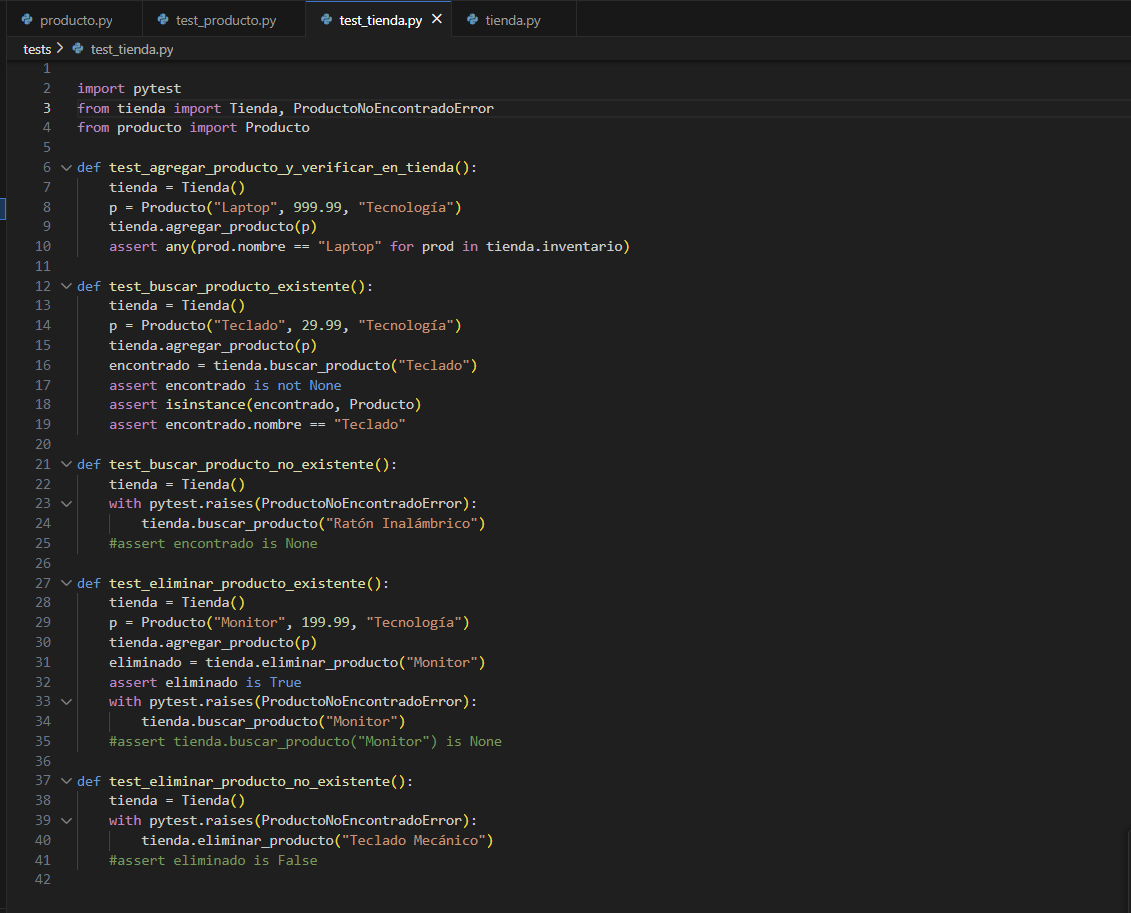


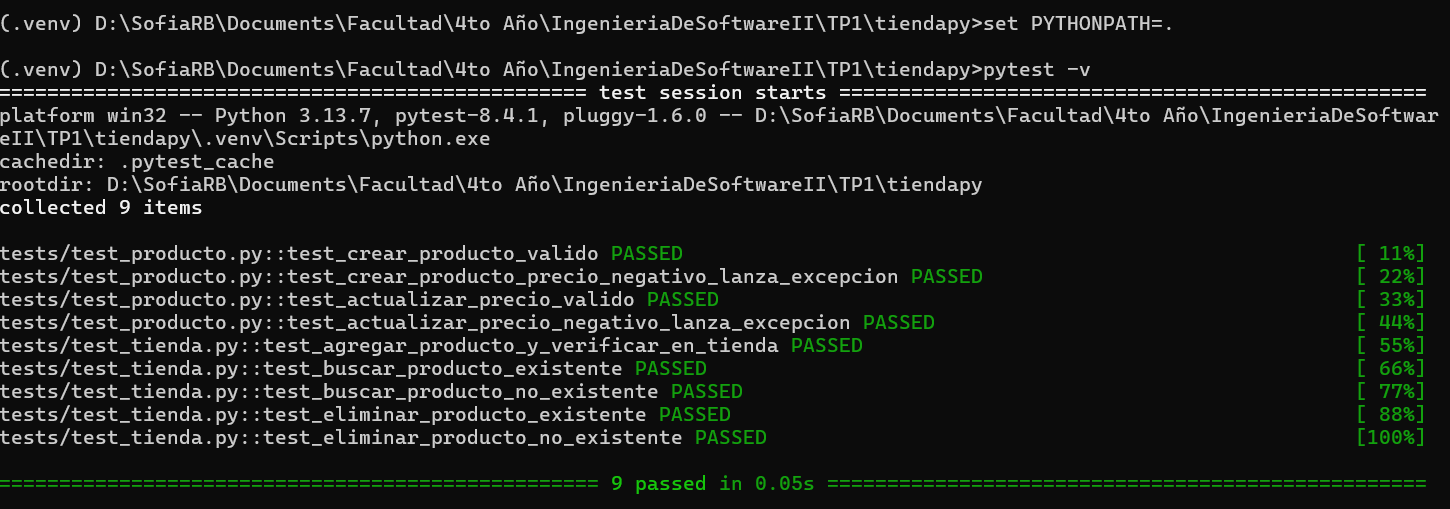


# Actividad práctica:

* Escribe pruebas que verifiquen que se lanzan las excepciones correctas al intentar eliminar un producto inexistente o actualizar un precio a un valor negativo.
* Utiliza el framework para comprobar que las excepciones se lanzan correctamente.







# Preguntas conceptuales:

- Podría haber escrito las pruebas primero antes de modificar el código de la aplicación?

¿Cómo sería el proceso de escribir primero los tests? Describe el proceso con tus palabras.

Si, podrían escribirse los tests antes del código lo que se conoce como Test Driven Development (TDD). En ese caso, debería analizar el comportamiento o requisitos esperados del sistema. Por ejemplo, si los productos deben contener un nombre y precio, y éste último no debe ser negativo, entonces debería primero crear un test que construya un objeto producto con un precio negativo y verificar que el código lance una excepción y otro en el que se cree un producto exitosamente. Una vez creado el test, se debe escribir el código de manera que pase el test. Siguiendo el ejemplo, debería implementar el constructor que realice el control del precio y que en caso de que sea negativo, lance una excepción del tipo especificado en el test. Caso contrario, se creará el objeto. Por último, si al ejecutar el test éste no falla el código cumple con el comportamiento especificado. En caso de que no lo pase satisfactoriamente, se deberá reformular el código.

# Uso de Mocks para aislar unidades bajo prueba

**Objetivo:** Introducir el uso de “test doubles” para aislar unidades de prueba y evitar dependencias innecesarias durante la prueba. En las pruebas de software, los dobles de prueba son objetos o componentes que reemplazan componentes reales para aislar la unidad bajo prueba.

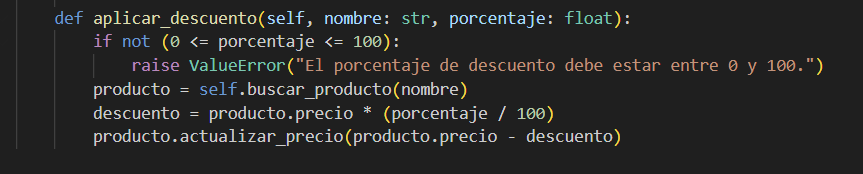
* 1. Implementa un método aplicar\_descuento en la clase Tienda que acepte un nombre de producto y un porcentaje de descuento y que modifique el precio del producto.
  2. Utiliza un “test double” para simular el comportamiento del producto.

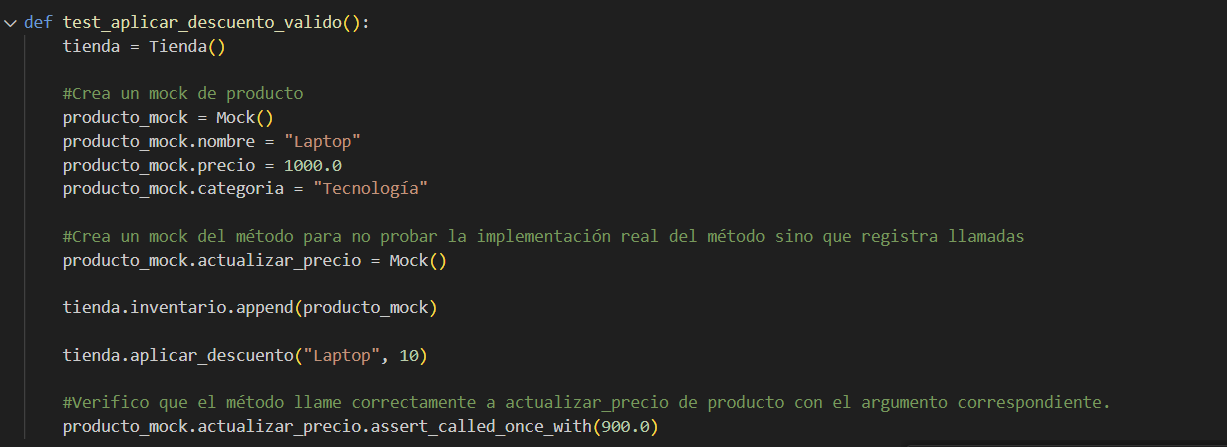
# Actividad práctica:

* Escriba pruebas utilizando mocks para verificar que el método aplicar\_descuento calcula correctamente el nuevo precio sin usar un objeto real.
* Asegúrate de verificar que la función llama correctamente al método actualizar\_precio

del producto.

* [Opcional] Utilizar dobles de prueba para los tests de agregar, buscar y eliminar productos.





# Preguntas conceptuales:

* En lo que va del trabajo práctico, ¿puedes identificar 'Controladores' y 'Resguardos'?

Los mocks para simular un producto y sus métodos funcionan como resguardos para poder llevar a cabo la prueba de unidad de tienda ya que nos permite hacer una prueba de unidad de la clase tienda sin tener que implementar la dependencia del módulo Producto.

Los métodos como test\_buscar\_producto\_existente o test\_eliminar\_producto\_existente funcionan como controladores para los métodos buscar\_producto y eliminar\_producto de la clase tienda respectivamente ya que genera las condiciones y objetos necesarios para podes utilizar dichas funcionalidades de la clase tienda y luego las invoca para podes testearlas.

* ¿Qué es un “test double”? ¿Hay otros nombres para los objetos/funciones simulados?

Un test double es cualquier objeto u componente utilizado para reemplazar otro objeto o componente real de nuestro sistema para poder llevar a cabo una prueba, permitiendo aislar una unidad y centrarse en su lógica. Dependiendo el propósito con el que se lo utiliza, un test double puede tomar una de las siguientes formas:

* Dummy object: es un contenedor de un objeto que se le pasa al sistema que se está testeando (SUT, System Under Testing) como argumento pero que nunca se utiliza realmente. solo existen para completar el espacio en los argumentos de un método o constructor.
* Test stub: es un objeto que reemplaza a un componente real del cual depende el sistema bajo prueba de manera que el test pueda controlar las entradas indirectas del mismo. Proporciona entradas indirectas predeterminadas que permiten controlar la ruta de ejecución del SUT para que el código pueda seguir un flujo de ejecución que de otra manera sería difícil de alcanzar.
* Test spy: es una versión mas avanzada de un test stub, que no solo proporciona entradas indirectas al SUT sino que registra llamadas que se le hacen, incluyendo los argumentos que se pasaron. Esto permite controlar al final de la ejecución que las salidas indirectas del SUT son las correctas.
* Mock Objects: un mock funciona como un punto de observación para las salidas indirectas del sistema como un spy. Puede necesitar devolver información de las llamadas a sus métodos como un stub y también, brinda información sobre como fueron invocados como un spy. Lo que diferencia a un mock object es que compara las llamadas reales recibidas con las expectativas predefinidas usando aserciones y hace que la prueba falle en nombre del método de prueba.
* Fake Objects: un fake es un objeto que reemplaza una dependencia real con una implementación simplificada y funcional de la misma. Suelen utilizarse cuando la dependencia real es demasiado lenta o no está disponible, como una base de datos en memoria que se usa en lugar de una base de datos real.

FUENTE: Meszaros, G. (2007). xUnit Test Patterns: Refactoring Test Code. Addison-Wesley Professional.

# Uso de Fixtures para Reutilización de Datos

**Objetivo:** Aprender a usar “fixtures” para preparar y limpiar el entorno de pruebas.

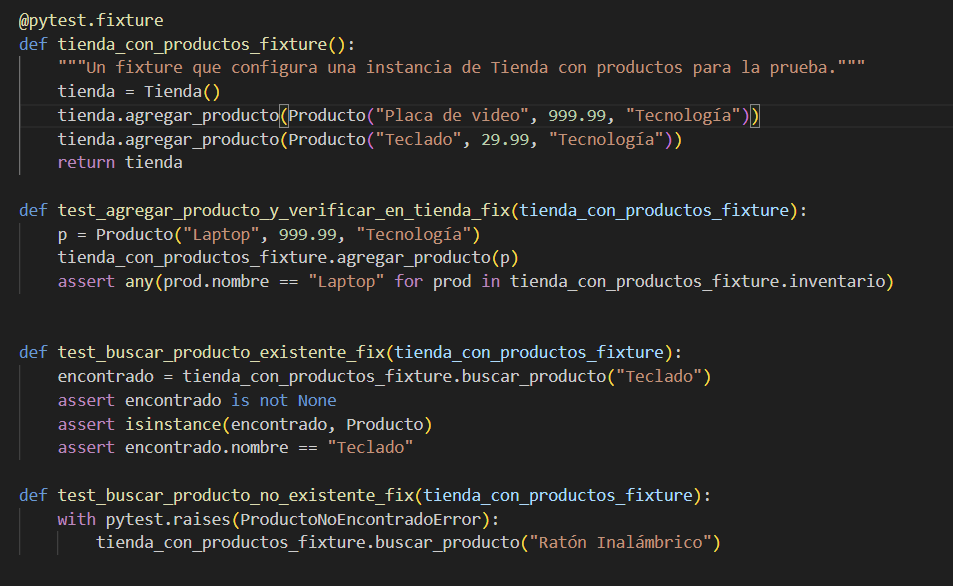
* 1. Crea un fixture que inicialice una tienda con algunos productos de ejemplo antes de cada prueba.
  2. Modifica las pruebas existentes para usar este fixture en lugar de crear los productos manualmente en cada prueba.

# Actividad práctica:

* Implementa un fixture y utilízalo en las pruebas de agregar\_producto y

buscar\_producto.

* Verifica que las pruebas se ejecutan correctamente utilizando los productos predefinidos.



# Preguntas conceptuales:

* ¿Qué ventajas ve en el uso de fixtures? ¿Qué enfoque estaríamos aplicando (caja negra/blanca)?

El uso de fixtures nos permite preparar de una manera más optimizada nuestro entorno de prueba. El fixture nos permite reutilizar código ya que por ejemplo, en este caso, nos permite inicializar una tienda con productos para poder probar sus métodos evitando que en cada uno de los tests debamos crear los objetos necesarios para su realización y repetir código. A su vez, esto nos permite que nuestras pruebas sean más legibles y mantener la consistencia entre ellas.

No considero que el fixture en general tenga un único enfoque. Puede ser utilizado tanto en un enfoque de caja blanca inicializando objetos con determinadas características para probar flujos de control específico. En nuestro caso en particular estamos llevando a cabo este tipo de enfoque. Sin embargo, podría existir otro caso en el que el fixture que establezca un estado externo como un script que carga información que provendría de una base de datos. Esto representaría un enfoque de caja negra.

* Explique los conceptos de Setup y Teardown en testing.

El concepto de Setup en testing implica la creación e inicialización del entorno de prueba de manera que se establezcan todas las condiciones previas necesarias para llevar a cabo las pruebas del código. Esto puede implicar desde crear e inicializar objetos, abrir conexiones de bases de datos, entre otros.

El concepto de Teardown se refiere a una fase posterior a la prueba donde se busca deshacer los efectos del setup como cerrar las conexiones a bases de datos, borrar archivos temporales, etc. El objetivo es que se limpie el entorno de prueba para que los efectos de una prueban no tengan impacto en la siguiente.

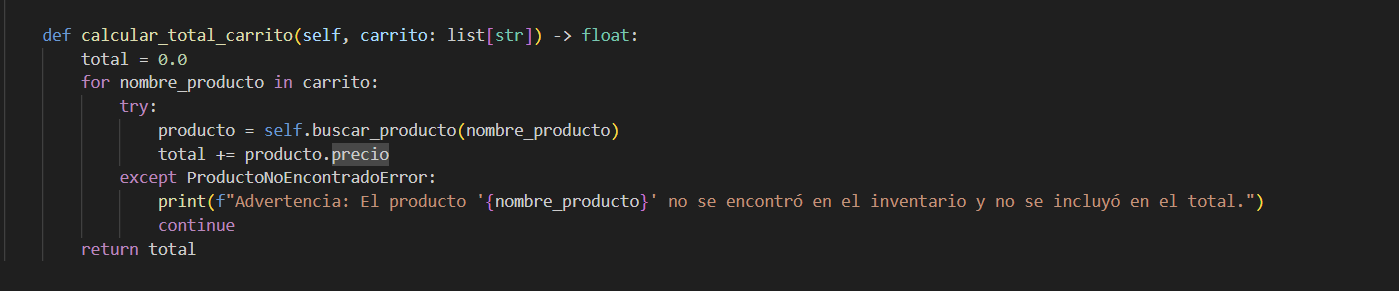
# Pruebas de Integración y Cobertura Completa

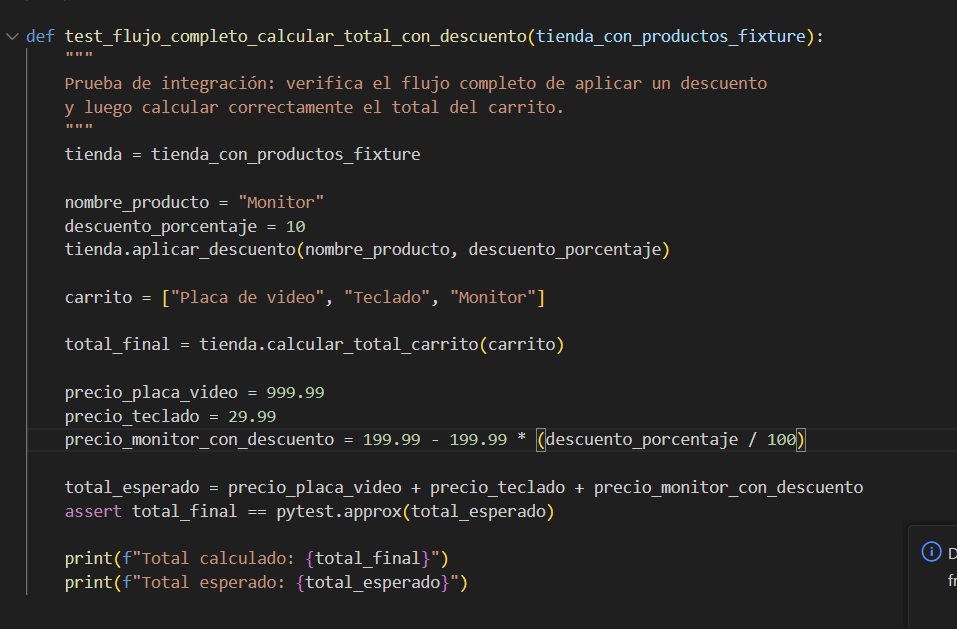
**Objetivo:** Realizar pruebas más complejas que verifiquen la integración de varias funciones y un flujo completo del sistema.

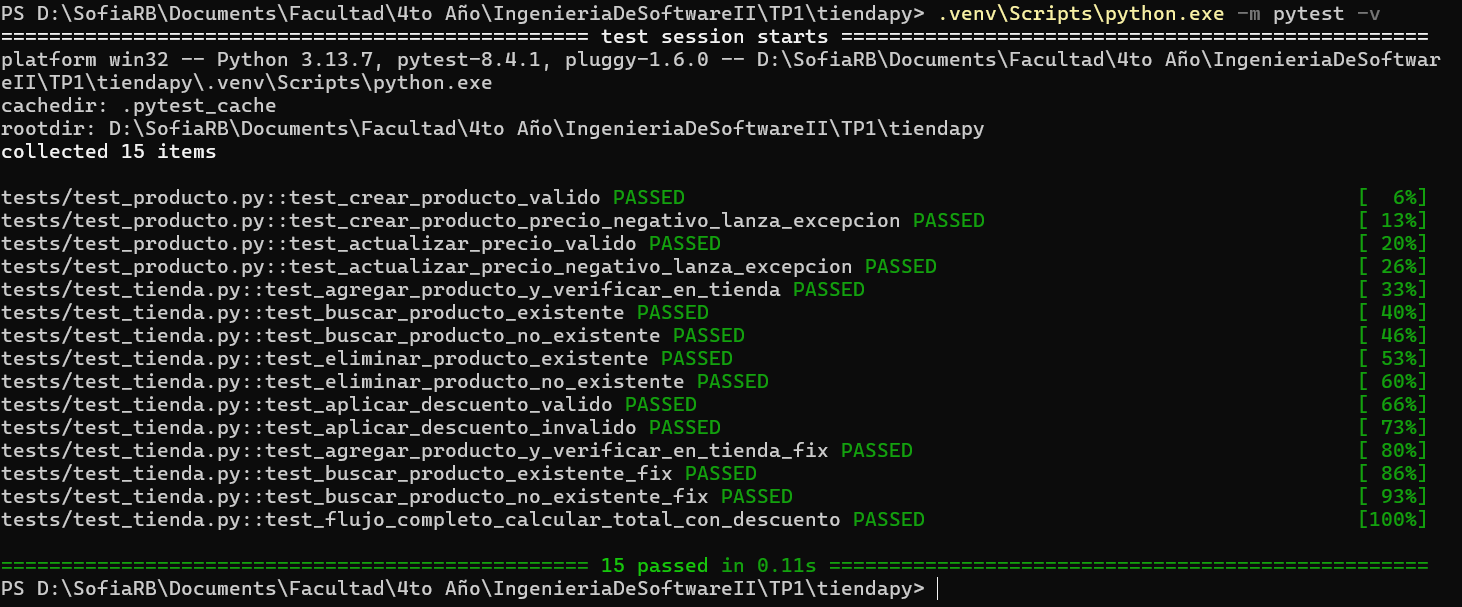
* 1. Implementa un método calcular\_total\_carrito en la clase Tienda que reciba una lista de nombres de productos (el carrito) y devuelva el precio total.
  2. Integra todas las funciones anteriores para permitir agregar productos a un carrito y calcular el total.

# Actividad práctica:

* Escribe pruebas de integración que verifiquen el flujo completo para calcular el total del carrito después de aplicar los descuentos.
* Utiliza un fixture para inicializar una tienda con varios productos.
* Verifica que el precio total calculado sea correcto.







# Preguntas conceptuales:

* ¿Puede describir una situación de desarrollo para este caso en donde se plantee pruebas de integración ascendente? Describa la situación.

En nuestro sistema de tienda se pueden observar pruebas de integración ascendente probando primero la clase producto con todos sus métodos ya que representa una clase que no depende de otra y de la cual si depende nuestra clase tienda.

Una vez probada la clase producto, se prueban en la clase tienda todos los métodos que operan con productos como agregar\_producto, buscar\_producto y eliminar\_producto.

Finalmente, se prueban los métodos que invocan a los métodos base de tienda mencionados anteriormente como el método calcular\_carrito.