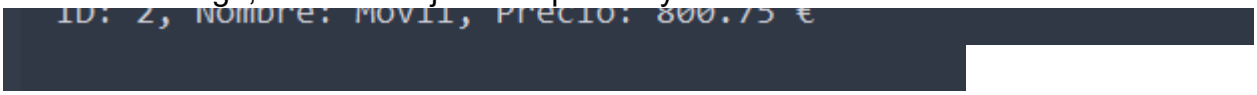
Unidad 5. POO y bases de datos

OPT5 – Tarea individual

- **RA5_a) Se han definido clases y creando objetos utilizando el paradigma de programación orientada a objetos.**
  - **Ejercicio 1.** Crea una clase llamada **Producto** que represente un artículo de una tienda. La clase debe contener:
    - Un constructor que reciba: `id`, nombre, precio.
    - Un método que muestre por pantalla la información del producto.
    - Luego, crea dos objetos de prueba y muestra sus datos.



- **RA5_b) Se han implementado atributos y métodos en clases, aplicando encapsulación.**
  - **Ejercicio 2.** Modifica la clase **Producto** para que el `atributo` precio sea privado. Añade métodos *getters* y *setters* para acceder o modificar el precio del producto de forma controlada (por ejemplo, no permitir precios negativos).

```python
##RA05_B:
#Ejer2:
class ProductoPrivado(Producto):  2 usages  ManuFer15 *
    def __init__(self, id, nombre, precio):  ManuFer15 *
        super().__init__(id, nombre, precio)
        self.__precio = precio

    def get_precio(self):  1 usage (1 dynamic)  ManuFer15
        return self.__precio

    def set_precio(self, newPrecio):  ManuFer15
        if newPrecio >= 0:
            self.__precio = newPrecio
        else:
            print("El precio no puede ser negativo.")

def info_producto_privado(producto):  2 usages  ManuFer15
    return f"ID: {producto.id}, Nombre: {producto.nombre}, Precio: {producto.get_precio()} €"

prod3 = ProductoPrivado( id: 3,  nombre: "Tablet",  precio: 450.00)
prod4 = ProductoPrivado( id: 4,  nombre: "Play Station",  precio: 500.00)

print("\nRA05_B: Ejer2")
print(info_producto_privado(prod3))
print(info_producto_privado(prod4))
```

```
RA05_B: Ejer2
ID: 3, Nombre: Tablet, Precio: 450.0 €
ID: 4, Nombre: Play Station, Precio: 500.0 €
```

- **RA5_c) Se ha utilizado herencia para optimizar la reutilización del código.**
  - **Ejercicio 3.** Crea una nueva clase llamada **ProductoAlimenticio**, que herede de **Producto**. Esta clase debe añadir:
    - Un atributo adicional `fecha_caducidad`.
    - Un método para comprobar si está caducado en función de la fecha actual.
    - Crea un objeto de esta nueva clase y prueba su funcionamiento.

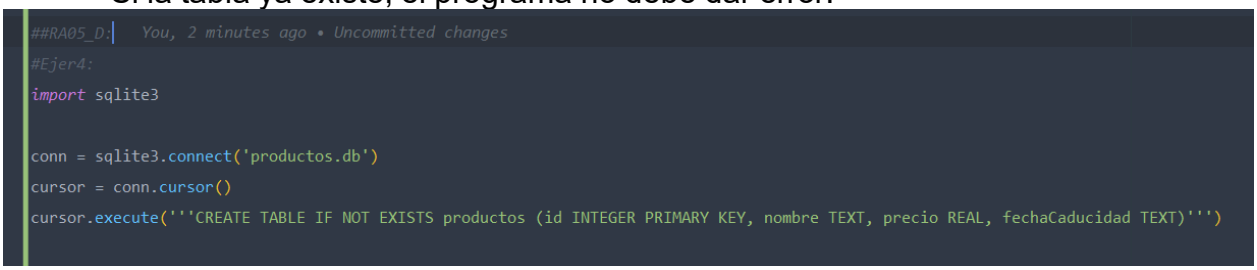
```
Producto: Yogur, Estado: Caducado
```

- **RA5_d) Se ha conectado Python con bases de datos SQL mediante SQLite3 u otro gestor.**
  - **Ejercicio 4.** Conecta tu programa a una base de datos SQLite llamada

tienda.db. Crea una tabla llamada **productos** con las columnas:

`id INTEGER PRIMARY KEY, nombre TEXT, precio REAL, tipo TEXT, fecha_caducidad TEXT`

- Si la tabla ya existe, el programa no debe dar error.



```
##RA05_D:   You, 2 minutes ago • Uncommitted changes
#Ejer4:
import sqlite3

conn = sqlite3.connect('productos.db')
cursor = conn.cursor()
cursor.execute('''CREATE TABLE IF NOT EXISTS productos (id INTEGER PRIMARY KEY, nombre TEXT, precio REAL, fechaCaducidad TEXT)''')
```

- **RA5_e) Se han implementado consultas básicas (CRUD) en bases de datos desde un programa en Python.**
  - **Ejercicio 5.** Añade a tu programa funciones (o métodos dentro de una clase gestora) que permitan:
    - Insertar un objeto **Producto** o **ProductoAlimenticio** en la tabla.
    - Mostrar todos los registros de la tabla.
  - Inserta al menos dos productos desde tu código y muestra la lista completa al ejecutar el programa.

```
##RA05_E:
#Ejer5:
cursor.execute( sql: "INSERT INTO productos (id, nombre, precio, fechaCaducidad) VALUES (?, ?, ?, ?)",  parameters: (id, "pan", "0.50", "2026-06-10"))
conn.commit()
cursor.execute("SELECT * FROM productos")
cursor.fetchall()
cursor.close()
conn.close()
```

- **RA5_f) Se han aplicado principios de diseño orientado a objetos para mantener la modularidad y escalabilidad del código, asegurando una fácil integración con bases de datos y otros sistemas.**
  - **Ejercicio 6.** Crea una clase llamada **GestorBD** que se encargue de toda la lógica relacionada con la base de datos: conexión, inserción y consulta.
  - Modifica el código existente para que la interacción con SQLite se realice exclusivamente desde esta clase. Asegúrate de que tu programa principal queda organizado, modular y fácil de ampliar.

NOTA: para este apartado he comentado el Ejer 4 y 5 para que no de problemas.

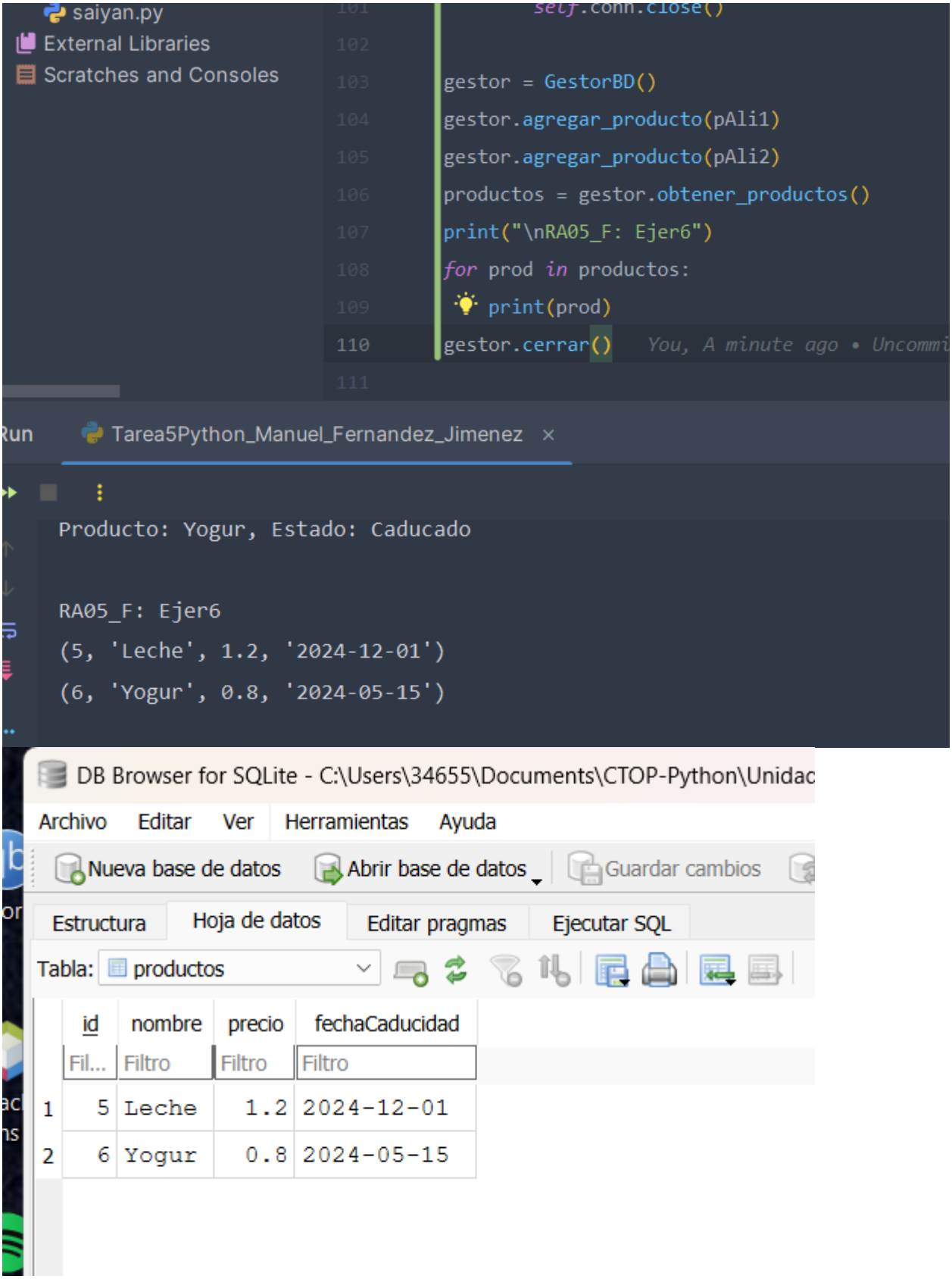
```
##RA05_F:
#Ejer6:
import sqlite3
class GestorBD:  1 usage   new *
    def __init__(self, db_name='productos.db'):  new *
        self.conn = sqlite3.connect(db_name)

        self.cursor = self.conn.cursor()
        self.cursor.execute('''CREATE TABLE IF NOT EXISTS productos (id INTEGER PRIMARY KEY, nombre TEXT, precio REAL, fechaCaducidad TEXT)''')

    def agregar_producto(self, producto):  2 usages   new *
        self.cursor.execute( sql: "INSERT INTO productos (id, nombre, precio, fechaCaducidad) VALUES (?, ?, ?, ?)",
                             parameters: (producto.id, producto.nombre, producto.precio, getattr(producto, 'fechaCaducidad', None)))
        self.conn.commit()

    def obtener_productos(self):  1 usage   new *
        self.cursor.execute("SELECT * FROM productos")
        return self.cursor.fetchall()

    def cerrar(self):  1 usage   new *
        self.cursor.close()
        self.conn.close()
```


saiyan.py
External Libraries
Scratches and Consoles
self.conn.close()
gestor = GestorBD()
gestor.agregar_producto(pAli1)
gestor.agregar_producto(pAli2)
productos = gestor.obtener_productos()
print("\nRA05_F: Ejer6")
for prod in productos:
print(prod)
gestor.cerrar()
You, A minute ago • Uncommi
Run
Tarea5Python_Manuel_Fernandez_Jimenez
Producto: Yogur, Estado: Caducado
RA05_F: Ejer6
(5, 'Leche', 1.2, '2024-12-01')
(6, 'Yogur', 0.8, '2024-05-15')
DB Browser for SQLite - C:\Users\34655\Documents\CTOP-Python\Unidac
Archivo
Editar
Ver
Herramientas
Ayuda
Nueva base de datos
Abrir base de datos
Guardar cambios
Estructura
Hoja de datos
Editar pragmas
Ejecutar SQL
Tabla:
productos
id
nombre
precio
fechaCaducidad
Fil...
Filtro
Filtro
Filtro
1
5
Leche
1.2
2024-12-01
2
6
Yogur
0.8
2024-05-15