Mapeo de datos del programa línea por línea

El "mapeo de datos" es describir, línea a línea, qué datos personales aparecen, dónde se guardan y cómo se usan en el programa. Voy recorriendo el código completo y, cuando una línea toca datos personales, lo indico claramente.

Definición de "base de datos" en memoria

```
usuarios = []
```

Crea la lista donde se guardarán datos personales de usuarios.

Tipo de datos (cuando se llene): nombre, email, contraseña.

```
productos = [
    {"id": 1, "nombre": "Camiseta", "precio": 20},
    {"id": 2, "nombre": "Pantalón", "precio": 35},
    {"id": 3, "nombre": "Zapatillas", "precio": 50}
]
```

Solo hay datos de productos (no personales).

No entra en el mapeo RGPD porque no identifica personas.

```
pedidos = []
```

Crea la lista que almacenará datos de pedidos ligados a usuarios.

Más adelante contendrá nombre/email del cliente y su historial de compra.

Función **registrar_usuario()** – recogida y almacenamiento de datos personales

```
def registrar_usuario():
    print("=== Registrar Usuario ===")
```

No trata datos, solo muestra texto.

```
    nombre = input("Nombre: ")
```

Punto de recogida de datos personales: se captura el nombre.

Origen: lo teclea el usuario (interfaz de consola).

Tipo de dato: identificativo.

```
    email = input("Email: ")
```

Recoge el email, otro dato personal que identifica al usuario.

Origen: entrada manual del usuario.

```
    contrasena = input("Contraseña: ")
```

Recoge la contraseña en texto plano.

Dato muy sensible (credencial de acceso), también origen: teclado.

```
    usuario = {"nombre": nombre, "email": email, "contrasena": contrasena}
```

Crea un registro en memoria que agrupa los tres datos personales del usuario.

Estructura de almacenamiento: diccionario dentro de la variable local **usuario**, que luego se pasará a la lista global.

```
usuarios.append(usuario)
```

Almacenamiento: inserta ese diccionario en la lista global **usuarios**.

Localización del dato: memoria RAM del proceso Python, sin cifrado, sin hash, sin persistencia en disco.

```
print(f"Usuario {nombre} registrado con éxito.\n")
```

Usa el nombre solo para mostrarlo por pantalla, no lo persiste en ningún otro sitio.

Función **mostrar_productos()** – sin datos personales

```
def mostrar_productos():
    print("=== Productos Disponibles ===")
    for producto in productos:
        print(f"{producto['id']}: {producto['nombre']} - ${producto['precio']}")
    print()
```

Solo lee y muestra datos de la lista **productos**.

No usa **usuarios** ni **pedidos**, por tanto no trata datos personales.

Función **crear_pedido()** – uso de datos personales para compras

```
def crear_pedido():
    if len(usuarios) == 0:
        print("No hay usuarios registrados. Regístrate primero.\n")
        return
```

Solo comprueba si ya hay datos personales cargados en **usuarios** , no añade ni modifica.

```
    email = input("Introduce tu email para iniciar pedido: ")
```

De nuevo se recoge el email, esta vez para identificar al cliente que hace el pedido.

Origen: usuario teclea su email.

```
    usuario = next((u for u in usuarios if u["email"] == email), None)
```

Uso / consulta de datos personales: busca en la "tabla" **usuarios** un registro cuyo email coincida.

No cambia los datos, pero los usa para vincular el pedido a una persona concreta.

```
    if not usuario:
        print("Usuario no encontrado.\n")
        return
```

Solo lógica de control, sin nuevos datos.

```
    mostrar_productos()
```

No toca datos personales.

```
producto_id = int(input("ID del producto a pedir: "))
```

Pide un ID de producto (no personal).

```
producto = next((p for p in productos if p["id"] == producto_id), None)
```

Consulta de producto, sin datos personales.

```
if not producto:
    print("Producto no encontrado.\n")
    return
```

Control de flujo.

```
cantidad = int(input("Cantidad: ")) total =
producto["precio"] * cantidad
```

cantidad y **total** no son, por sí solos, datos personales, pero formarán parte del historial de compras cuando se asocien al usuario.

```
pedido = {
    "usuario": usuario["nombre"], "email":
    usuario["email"], "producto":
    producto["nombre"], "cantidad":
    cantidad,
    "total": total
}
```

Creación de un nuevo registro de pedido con datos personales:

- **"usuario"** y **"email"** vienen del diccionario **usuario** (ya almacenado).
- **"producto"** , **"cantidad"** , **"total"** describen la compra.

Al ligar email/nombre con sus pedidos, se construye un perfil de consumo de la persona.

```
        pedidos.append(pedido)
```

Almacenamiento: añade el diccionario **pedido** a la lista global **pedidos** .

Localización del dato: también en memoria RAM, en la variable global **pedidos** , sin cifrado ni persistencia.

```
        print(f"Pedido de {cantidad} x {producto['nombre']} realizado con éxito. Total:
$${total}\n")
```

Usa datos del pedido solo para mostrar un resumen por pantalla, no nuevos almacenamientos.

Función **mostrar_pedidos()** – exposición de datos personales + compras

```
def mostrar_pedidos():
    if len(pedidos) == 0:
        print("No hay pedidos realizados.\n")
        return
```

Solo comprueba si hay datos en **pedidos** .

```
    print("=== Pedidos Realizados ===")
    for p in pedidos:
        print(f"{p['usuario']} - {p['producto']} x {p['cantidad']} = ${p['total']}")
    print()
```

Uso y visualización de datos:

- Lee cada elemento **p** de la lista **pedidos**, que contiene nombre del usuario y su compra.
- Muestra: nombre de la persona (**p['usuario']**), producto y cantidades/compras.

A nivel de flujo de datos, aquí se produce una salida de datos personales (por consola), aunque no se vuelven a guardar.

Función **menu()** – no introduce datos nuevos, solo orquesta

```
def menu():
    while True:
        print("=== Tienda Online ===")
        print("1. Registrar Usuario") print("2.
        Mostrar Productos") print("3.
        Realizar Pedido") print("4. Ver
        Pedidos") print("5. Salir")
        opcion = input("Elige una opción: ")
        print()
```

input("Elige una opción: ") recoge una elección de menú, pero no es dato personal.

```
        if opcion == "1": registrar_usuario()
        elif opcion == "2": mostrar_productos()
        elif opcion == "3": crear_pedido()
        elif opcion == "4": mostrar_pedidos()
        elif opcion == "5":
            print("¡Gracias por usar la tienda online!")
            break else:
            print("Opción no válida.\n")
```

El menú decide qué función llamar, pero no trata directamente datos personales. El tratamiento real está en las funciones que ya hemos analizado.

Bloque principal

```
if __name___ == "__main__":
    menu()
```

Control para que, al ejecutar el archivo, se lance el menú. No introduce datos personales, solo inicia el flujo.

Resumen del mapeo de datos (visión RGPD)

Siguiendo el enfoque de data mapping RGPD (identificar dónde se recogen, almacenan, usan y muestran datos personales):

Puntos de recogida (input)

- **nombre** , **email** , **contrasena** en **registrar_usuario** .
- **email** en **crear_pedido** (para localizar al usuario).

Almacenamiento

- Lista **usuarios** : diccionarios con nombre, email, contraseña.
- Lista **pedidos** : diccionarios con nombre/email + detalle de compras.
- Todo en RAM, sin persistencia ni cifrado.

Uso interno

- Búsqueda de usuarios por email (**crear_pedido**).
- Cálculo de importes y asociación de compras a un usuario (**pedido**).

Salidas / visualización

- Consola:

- Confirmaciones de registro y pedidos.
- Listado de pedidos con nombre y compras (**mostrar_pedidos**).

Con este mapeo línea a línea ya se ve claramente qué datos personales hay, dónde viven en el código y cómo fluyen, que es justo lo que se busca en un mapa de datos para luego hacer la evaluación de riesgos RGPD