Ejercicio de módulos en Python – Paso a paso completo

Este documento explica **qué archivo va dónde**, **qué escribir en cada uno** y **qué hace cada línea**, sin asumir conocimientos previos.

---

1. Estructura del proyecto

Tu carpeta del proyecto debe verse así:

```
modules_exercise/
│
├── .venv/                  # Entorno virtual (no se toca)
├── create_files.py         # Script que genera datos falsos
├── main.py                 # Script principal que procesa los datos
├── datefile1.txt           # Se crea automáticamente
├── datefile2.txt           # Se crea automáticamente
```

---

2. Archivo: `create_files.py`

Este archivo **solo sirve para generar datos**. No procesa nada, no muestra tablas, no calcula medianas.

Código completo comentado

```python
from faker import Faker      # Importamos Faker para generar datos falsos

fake = Faker()               # Creamos un objeto Faker


def create_file(num):        # Función que crea un archivo de fechas
    # Abrimos (o creamos) un archivo en modo escritura
    # El nombre será datefile1.txt, datefile2.txt, etc.
    with open(f"datefile{num}.txt", "w") as f:
        # Repetimos 100 veces
        for _ in range(100):
            # Escribimos una fecha falsa en cada línea
            f.write(fake.date() + "\n")


# Llamamos a la función dos veces para crear dos archivos
create_file(1)
create_file(2)
```

Cómo se ejecuta

Desde la terminal (con el `.venv` activado):

```
python create_files.py
```

Resultado esperado: - Se crean `datefile1.txt` y `datefile2.txt` - No se imprime nada (esto es correcto)

---

3. Archivo: `main.py`

Este es el **programa principal**. Lee los archivos, muestra los datos y calcula la fecha mediana.

Código completo comentado

```
import sys                              # Para leer argumentos desde la
terminal
from tabulate import tabulate           # Para mostrar datos en forma de tabla


def extract_file_contents(file_path):
    """
    Lee un archivo de texto y devuelve una lista de líneas limpias.
    """
    with open(file_path, "r") as f:
        # Leemos cada línea, quitamos espacios y descartamos líneas vacías
        return [line.strip() for line in f if line.strip()]


def display_files(file1_contents, file2_contents):
    """
    Muestra el contenido de los dos archivos en formato de tabla.
    """
    table = {
        "file1": file1_contents,
        "file2": file2_contents
    }
    print(tabulate(table))


def main(file1, file2):
    # Leemos el contenido de cada archivo
    file1_data = extract_file_contents(file1)
    file2_data = extract_file_contents(file2)

    # Mostramos los datos en tabla
    display_files(file1_data, file2_data)
```

```
    # Unimos las fechas de ambos archivos, eliminamos duplicados y ordenamos
    dates = sorted(set(file1_data).union(file2_data))

    # Calculamos la fecha mediana
    print("\nFecha mediana:")
    print(dates[len(dates) // 2])


# Punto de entrada del programa
if __name__ == "__main__":
    # Verificamos que el usuario haya pasado exactamente 2 archivos
    if len(sys.argv) != 3:
        print("Usá: python main.py datefile1.txt datefile2.txt")
        sys.exit(1)

    # Ejecutamos la función principal con los archivos recibidos
    main(sys.argv[1], sys.argv[2])
```

---

4. Cómo se ejecuta `main.py`

Siempre desde la terminal:

```
python main.py datefile1.txt datefile2.txt
```

Qué hace este comando

- `python main.py` → ejecuta el programa
- `datefile1.txt` → primer archivo de entrada
- `datefile2.txt` → segundo archivo de entrada

---

5. Ideas clave para estudiar (resumen realista)

- Cada archivo cumple **una sola función**
- `create_files.py` genera datos
- `main.py` procesa datos
- `sys.argv` permite pasar información desde la terminal
- El `.venv` vive en la raíz del proyecto y no se toca

---

6. Regla final

> Cuando no se entiende Python, casi siempre falta **orden**, no inteligencia.

Este ejercicio, bien estructurado, ya es Python real.