Informe del Trabajo Integrador Final: Sistema de Gestión de Países

1. Introducción al Proyecto

Este trabajo integrador final consiste en el desarrollo de una aplicación en Python que permite gestionar información sobre países del mundo. El objetivo principal fue crear un programa que, a partir de una fuente de datos externa (una API pública), descargue información real, la almacene de forma estructurada y ofrezca al usuario herramientas para consultarla, filtrarla, ordenarla y analizarla de manera interactiva.

El proyecto no solo cumple con los requisitos técnicos, sino que también está diseñado pensando en la claridad, la modularidad y la facilidad de uso. Se utilizó Python como lenguaje principal porque es ideal para principiantes en programación, pero también es muy potente para trabajar con datos. Además, se incluyó el uso de Docker para que cualquier persona pueda ejecutar el programa sin preocuparse por instalar Python o las librerías necesarias.

A lo largo de este informe, se explicará paso a paso cómo está organizado el proyecto, qué hace cada parte del código, cómo funciona el programa y por qué se tomaron ciertas decisiones técnicas. Todo está pensado para que un estudiante de primer año de programación pueda entenderlo sin problemas, aunque también incluye detalles más avanzados para quienes quieran profundizar.

2. Contexto y Motivación

En la vida real, los programadores trabajan con datos que provienen de muchas fuentes: bases de datos, archivos, sensores o, como en este caso, APIs (interfaces de programación de aplicaciones). Una API es como una "puerta" que permite a un programa pedir información a otro sistema a través de internet.

En este proyecto, se utilizó la API pública **REST Countries**, que ofrece información actualizada sobre todos los países del mundo: nombre, capital, población, área, moneda, idiomas, región, etc. Esta API es gratuita y no requiere clave de acceso, lo que la hace perfecta para un proyecto educativo.

El programa no descarga los datos cada vez que se ejecuta. En cambio, **los descarga una sola vez**, los guarda en un archivo CSV (un tipo de archivo que se puede abrir con Excel) y luego trabaja con ese archivo. Esto es mucho más rápido y evita depender de la conexión a internet después de la primera ejecución.

3. Estructura General del Proyecto

El proyecto está organizado en carpetas y archivos de forma clara y lógica. Esta organización es muy importante en programación, porque permite encontrar rápidamente lo que se necesita y facilita el trabajo en equipo.

Árbol de directorios completo:

```
Trabajo_integrador_final/
├── data/
│   └── paises.csv              ← Archivo con los datos de los países (se genera automáticamente)
├── src/
│   ├── funciones/              ← Carpeta con todos los módulos de funcionalidades
│   │   ├── buscar.py           ← Búsqueda de un país por nombre
│   │   ├── filtrar.py          ← Filtros por región y población
│   │   ├── ordenar.py          ← Ordenar por población o área
│   │   ├── estadisticas.py     ← Cálculo de estadísticas
│   │   └── menu.py             ← Menú principal del programa
│   ├── getpaises.py            ← Descarga datos de la API y crea el CSV
│   └── main.py                 ← Archivo principal que ejecuta todo
├── Dockerfile                  ← Para ejecutar el programa en un contenedor
├── README.md                   ← Instrucciones del proyecto
└── requirements.txt            ← Librerías necesarias
```

Explicación de cada parte:

- **data/**: Es la carpeta donde se guarda el archivo paises.csv. Este archivo contiene toda la información de los países en filas y columnas, como una tabla de Excel. Al principio está vacío, pero se llena cuando se ejecuta getpaises.py.
- **src/**: Es la carpeta principal del código fuente. El nombre "src" viene de "source" (fuente en inglés). Aquí está todo el código Python.
- **src/funciones/**: Dentro de src, hay una subcarpeta llamada funciones que contiene varios archivos pequeños. Cada archivo tiene una funcionalidad específica. Esto se llama **modularidad**: que es dividir el programa en partes pequeñas y claras.
- **getpaises.py**: Este archivo es el encargado de hablar con la API, descargar los datos y guardarlos en el CSV. Solo se ejecuta una vez.
- **main.py**: Es el "cerebro" del programa. Desde aquí se importa todo lo necesario y se controla el flujo de ejecución. **Siempre se ejecuta este archivo para usar el programa**.
- **Dockerfile**: Un archivo de configuración que permite empaquetar todo el proyecto en un contenedor Docker. Es como una "caja" que contiene Python, las librerías y el código, lista para ejecutarse en cualquier computadora.
- **README.md**: Es un archivo de texto con formato que explica cómo instalar y usar el proyecto. Es como la "portada" del repositorio en GitHub.
- **requirements.txt**: Contiene la lista de librerías externas que necesita el programa. En este caso, solo dos: rich y requests. El archivo requirements.txt es una lista de las librerías y sus versiones específicas que requiere un proyecto de Python. Permite que otros desarrolladores instalen todas las dependencias necesarias con un solo comando, como pip install -r requirements.txt, asegurando que el proyecto funcione correctamente en diferentes entornos. Esto simplifica la colaboración y la implementación al evitar problemas de incompatibilidad de versiones.

4. Dependencias del Proyecto

El programa usa dos librerías externas que no vienen con Python por defecto:

1. **requests**
    - Sirve para hacer peticiones HTTP (hablar con sitios web desde código).

- Se usa en getpaises.py para descargar los datos de la API.
- Ejemplo: requests.get("https://...") es como abrir un enlace en el navegador, pero desde Python.

2. **Rich**
   - La librería *rich* es una biblioteca de Python que se utiliza para crear texto enriquecido y de formato atractivo en la consola, haciendo que las aplicaciones de línea de comandos sean más visualmente atractivas y legibles. Permite añadir color, estilo, tablas, barras de progreso, resaltado de sintaxis, etc. y mostrar Markdown o trazas de forma atractiva.

Estas librerías se instalan con un solo comando:

bash

```
pip install -r requirements.txt
```

Esto lee el archivo requirements.txt y descarga automáticamente las versiones correctas.

5. Flujo de Ejecución del Programa

El programa sigue un flujo claro y predecible:

1. **Se ejecuta main.py**.
2. El programa verifica si existe el archivo data/paises.csv.
   - Si **no existe**, ejecuta automáticamente getpaises.py para crearlo.
   - Si **sí existe**, lo carga en memoria como una tabla.
3. Se muestra el **menú principal**.
4. El usuario elige una opción (1 al 7).
5. Según la opción, se llama a una función específica de la carpeta funciones/.
6. La función realiza su tarea y muestra resultados.
7. Se vuelve al menú hasta que el usuario elija "Salir".

Este flujo es **lineal** y **predecible**, lo que lo hace ideal para aprender.

6. Explicación Detallada de Cada Módulo

A continuación, se explica qué hace cada archivo, **sin mostrar grandes bloques de código**, pero describiendo paso a paso cómo funciona y por qué se hizo de esa manera.

6.1 getpaises.py – Descarga de Datos desde la API

Este archivo tiene una sola función importante: obtener_paises().

¿Qué hace?

1. Se conecta a la dirección https://restcountries.com/v3.1/all.
2. Recibe una respuesta en formato JSON (un tipo de texto estructurado).
3. Recorre cada país uno por uno.
4. Extrae información útil: nombre, capital, población, área, moneda, idiomas, región.
5. Guarda todo en una lista de diccionarios.
6. Convierte esa lista en una tabla.
7. Guarda la tabla en data/paises.csv.

¿Por qué es importante?

- Solo se ejecuta **una vez**.
- Evita depender de internet después.
- Crea un archivo que cualquiera puede abrir con Excel o LibreOffice.
- Maneja errores: si no hay internet, si falta un dato, o si la API cambia, el programa no se rompe.

Detalles técnicos para principiantes:

- Se usa try y except para capturar errores (como cuando no hay conexión).
- Se verifica si un país tiene capital o moneda antes de intentar leerla (porque algunos países no tienen).
- Se usa rich para crear la tabla.
- to_csv() guarda el archivo sin números de fila extra.

6.2 main.py – El Archivo Principal

Este es el archivo que **siempre se ejecuta**. Es como el "director" de una orquesta: no hace el trabajo pesado, pero coordina todo.

Funciones clave:

- Verifica si existe el CSV.
- Si no existe, llama a obtener_paises().
- Carga el CSV en una variable llamada df (abreviatura de DataFrame).
- Entra en un bucle infinito que muestra el menú.
- Según la opción elegida, llama a la función correspondiente.

¿Por qué está todo importado aquí?
Porque main.py es el **único punto de entrada**. Esto es una buena práctica: el usuario solo necesita saber un comando: python src/main.py.

6.3 menu.py – La Interfaz con el Usuario

Este archivo tiene una sola función: mostrar_menu().

¿Qué hace?

- Imprime en pantalla las 7 opciones.
- Pide al usuario que ingrese un número del 1 al 7.
- Devuelve ese número como texto (string).

¿Por qué es un archivo separado?

- Para mantener el código limpio.
- Si en el futuro quiero cambiar el diseño del menú (por ejemplo, agregar colores), solo modifico este archivo.
- Es más fácil de leer y mantener.

6.4 buscar.py – Búsqueda de un País

Funcionalidad:

- Pide al usuario que ingrese el nombre del país a buscar, permitiendo la búsqueda completa o parcial.
- Si el campo de búsqueda está vacío, muestra un mensaje de error y finaliza.
- Carga los datos del archivo data/paises.csv usando el módulo csv.
- Normaliza la entrada del usuario y los nombres de los países a minúsculas para una comparación sin distinción de mayúsculas/minúsculas.
- Prioriza la búsqueda de coincidencia exacta: Recorre el archivo, y si encuentra una coincidencia completa, la guarda y detiene la búsqueda (break).
- Si no hay coincidencia exacta, realiza una búsqueda parcial: Si el texto ingresado por el usuario tiene 3 o más caracteres, agrega a los resultados todos los países que contengan ese texto.
- Si se encuentran resultados, se presenta la información en una tabla elegante usando la librería rich, mostrando el Nombre, Población, Superficie y Continente.
- Si no se encuentran resultados (ni exactos ni parciales válidos), muestra un mensaje de "No se encontró el país".

Conceptos importantes:

- .lower(): Conversión a minúsculas, crucial para búsquedas insensibles a mayúsculas/minúsculas.
- if nombre_pais_normalizado == nombre_normalizado: Lógica para la coincidencia exacta y uso de break para optimizar (sale del bucle al encontrar el país exacto).
- elif nombre_normalizado in nombre_pais_normalizado: Lógica para la búsqueda parcial (el texto de búsqueda está contenido en el nombre del país).
- if len(nombre_normalizado) >= 3: Condición que filtra las búsquedas parciales muy cortas (de 1 o 2 caracteres), mejorando la relevancia de los resultados.
- {int(r.get('poblacion', 0)):,}.replace(",", "."): Formateo de números grandes (Población, Superficie) para incluir separadores de miles (puntos en este caso) para mejor legibilidad.

6.5 filtrar.py – Filtros Avanzados

1 *Funcionalidad general:*

- Pide al usuario el nombre de un continente.
- Lee el archivo paises.csv y filtra la lista de países para incluir solo aquellos cuya columna 'continente' coincida con la entrada del usuario (ignorando mayúsculas/minúsculas).
- Si no encuentra países en ese continente o si el archivo no existe, notifica al usuario con un mensaje de error.
- Si encuentra resultados, los presenta en una tabla formateada con la librería rich.
- Implementa un sistema de paginación para dividir los resultados en páginas de 10 elementos, mostrando una tabla por vez y esperando la entrada del usuario para pasar a la siguiente.

.

2. *filtrar_por_region()*

- Pide una región (ej: "South America").
- Muestra todos los países de esa región.
- Ejemplo: si escribo "Europe", muestra Alemania, Francia, etc.

3. filtrar_por_poblacion()

- Pide un rango: población mínima y máxima.
- Muestra países dentro de ese rango.
- Incluye control de errores: si el usuario escribe letras, avisa.

Conceptos:

- .strip().lower() aseguran que la búsqueda sea robusta e insensible a mayúsculas/minúsculas, eliminando también espacios extra
- fila.get('continente', '') agrega seguridad al manejar posibles filas sin la columna 'continente', asignando una cadena vacía en ese caso para evitar errores.
- Slicing de listas: Se utiliza dentro del bucle de paginación para seleccionar exactamente los 10 (o menos) elementos correspondientes a la página actual.
- console.clear: se usa para limpiar la consola en cada iteración, creando el efecto de navegación página a página.

6.6 ordenar.py – Ordenamiento de Datos

Funcionalidad general:

- Solicitar la Opción de Ordenamiento: Muestra un menú claro al usuario para que elija el tipo de ordenamiento: Ascendente (menor a mayor) o Descendente (mayor a menor) por población, asegurando que solo se acepte una opción válida (1 o 2).
- Ordenar la Lista: Utiliza la función integrada sorted() de Python, combinada con una función lambda y una función auxiliar de manejo de errores (obtener_valor_seguro), para ordenar los datos por población.

Dos funciones:

1. ordenar_por_poblacion()

- Ordena todos los países de mayor a menor población.
- Muestra solo los **10 primeros** (los más poblados).

2. ordenar_por_area()

- Igual, pero por área (los países más grandes).

Conceptos:

- Una lambda es una función pequeña y anónima (sin nombre formal como def) que solo puede tener una expresión.
- Uso en sorted(): Se pasa como argumento a key. Le indica a la función sorted() que, en lugar de ordenar por el diccionario completo, debe usar como clave de ordenamiento el valor devuelto por la lambda, que es el resultado de obtener_valor_seguro(fila, 'poblacion').

6.7 estadisticas.py – Análisis de Datos

Funcionalidad:

- Calcula:

- Promedio de población y área
- Mediana
- Mínimo y máximo
- Cantidad de países por región

Conceptos:

- Se usa el len para contar los países una vez definido el continente
- Utiliza la función `sum()` de Python para sumar todos los elementos generados por continente
- Se utilizan validaciones esenciales que previenen errores en tiempo de ejecución como el if not datos o if cantidad_de_paises>0
- Se utiliza una estructura `if/else` para determinar el **título y el resumen** que se mostrarán. Esto permite que la misma función sirva para calcular y reportar el **promedio global** (si no hay filtro) o el **promedio específico por continente** (si el filtro se aplicó).

7. Ejemplo de Uso Completo

Supongamos que es la primera vez que ejecuto el programa:

bash

```
python src/main.py
```

Salida:

text

```
Archivo CSV no encontrado. Generando datos desde API...
Se cargaron 250 países en data/paises.csv
Datos cargados: 250 países disponibles.

=== MENÚ PRINCIPAL ===
1. Busca por nombre de País
2. Filtrar países
3. Ordenar países
4. Estadísticas
5. Salir
Ingrese la opción deseada: 1
Ingrese  nombre del país( completo/parcial) : argentina
```

Resultados para 'argentina'

Nombre	Población	superficie	Continente
Argentina	46.735.004	2.780.400	South America

```
Presiona [Enter] para continuar...
```

Luego vuelvo al menú y puedo elegir otra opción.

8. Uso con Docker

Docker permite ejecutar el programa **sin instalar nada**. Es como tener una máquina virtual pequeña.

Pasos:

bash

```
# 1. Construir la imagen
docker build -t pais-app .

# 2. Ejecutar el contenedor
docker run -it pais-app
```

Dentro del contenedor, todo funciona igual. Esto es ideal para compartir el proyecto con compañeros o profesores.

9. Buenas Prácticas Aplicadas

Práctica	**Explicación**
Modularidad	Cada funcionalidad en un archivo separado
Nombres claros	buscar_pais, filtrar_por_region, etc.
Comentarios	Explican qué hace cada parte
Manejo de errores	try/except en entradas y conexiones
Separación de datos y código	CSV en data/, código en src/
Documentación	README claro y completo
Reutilización	El CSV se genera una vez y se usa siempre

10. Conclusión

Este proyecto es un ejemplo de cómo puede construirse una aplicación en Python, desde la obtención de datos hasta la interacción con el usuario. Aunque parece simple, el desarrollo es complejo. En este caso incluye conceptos fundamentales de programación:

- Uso de APIs
- Manejo de archivos
- Estructuras de datos como paginación, tablas y paneles con una librería, en este caso Rich
- Modularidad
- Control de flujo
- Entrada/salida
- Manejo de errores
- Contenerización con Docker

Todo está explicado de forma clara, con una estructura organizada y comentarios útiles. El código es fácil de leer, modificar y extender. Este trabajo no solo cumple con los objetivos del integrador, sino que sirve como base para proyectos más avanzados. Es un buen punto de partida para cualquier estudiante que quiera aprender a programar con datos reales.