Programación I

Informe Teórico Integrador

Integrantes:
- Benjamin Cajales
- Lautaro Monroy

Introducción

El proyecto es una aplicación de escritorio completa que, aunque tiene una funcionalidad compleja, se construye a partir de una combinación de conceptos de programación fundamentales. Este informe detalla cómo se utilizan estos conceptos teóricos para crear una aplicación eficiente, robusta y modular.

1. Arquitectura de la Aplicación

El proyecto utiliza una arquitectura de dos fases:

- **Fase 1: ETL (Extraer, Transformar y Cargar):** Gestionada por `main.py` y ejecutada por `generarPaises.py`. Esta fase es responsable de:
    - **Extraer:** Consumir datos de la API `restcountries.com`.

```
url = f"https://restcountries.com/v3.1/region/{continente}"
```

    - **Transformar:** Aplanar la estructura JSON, seleccionar campos relevantes (`nombre_comun_es`, `poblacion`, etc.) y, crucialmente, **enriquecer los datos** (añadiendo la columna `continente` a partir del nombre del archivo de origen).

```
# Define los nombres de las columnas que queremos en nuestro CSV
campos = ['nombre_comun_es', 'nombre_oficial_es', 'capital', 'region', 'poblacion', 'area']
```

- **Cargar:** Guardar los datos procesados en un único archivo CSV (`Continentes/Todos.csv`).

```
# Itera sobre cada fila de datos
for fila in lector:
    # Añade el nombre del continente al final de la fila
    fila.append(continente)
    # Escribe la fila completa en el archivo de salida
    escritor.writerow(fila)
```

- **Fase 2: GUI (Interfaz Gráfica de Usuario):** Gestionada por `interfaz.py`, esta fase lee el CSV local y se encarga de toda la lógica de presentación, filtrado y visualización.

Este desacoplamiento es fundamental: la interfaz gráfica no depende de una conexión a internet activa, solo del archivo local.

2. Lógica de Caché y Persistencia

El módulo `main.py` implementa una **estrategia de caché** simple pero efectiva. Utiliza `os.path.exists(ruta_archivo_final)` para comprobar si el archivo `Todos.csv` ya existe.

- Si el archivo **no existe** (primera ejecución), se ejecutan los procesos de descarga y unión (`procesar_todos_los_continentes` y `unir_csvs_en_uno`).

```
# Si no existe, ejecuta el proceso de descarga y unión
print(f"No se encontró el archivo '{ruta_archivo_final}'.")

# PASO 1: Llama a la función para descargar todos los CSV individuales
procesar_todos_los_continentes()

# PASO 2: Llama a la función para unirlos en 'Todos.csv'
unir_csvs_en_uno(carpeta_continentes, ruta_archivo_final)

print("\n--- ¡Archivos de datos generados! ---")
```

- Si el archivo **ya existe**, el programa se salta toda la fase de ETL y lanza la interfaz gráfica inmediatamente.

```
# Lanza la aplicación gráfica llamando a la función principal de 'interfaz.py'
print("Iniciando interfaz gráfica...")
interfaz.iniciar_interfaz()
```

Esto optimiza drásticamente el tiempo de inicio en usos subsecuentes, basándose en el principio de que los datos de los países no cambian con tanta frecuencia.

3. **Consumo de APIs**

La adquisición de datos se basa en la interacción con una **API RESTFUL** externa.

- **Cliente HTTP:** Se utiliza la biblioteca `requests` (definida en `requirements.txt`) para actuar como cliente HTTP. Específicamente, `requests.get(url)` realiza una petición GET para obtener los datos.

```python
import requests  # Para realizar peticiones HTTP a la API
```

```python
# Realiza la petición GET a la URL
response = requests.get(url)
```

- **Manejo de JSON:** La respuesta de la API viene en formato JSON. El método `response.json()` de `requests` se utiliza para deserializar este texto JSON y convertirlo en una estructura de datos nativa de Python (una lista de diccionarios).

```python
paises = response.json()
```

- **Manejo de Errores de Red:** El código demuestra un manejo robusto de errores al usar un bloque `try...except` `requests.exceptions.RequestException` para capturar fallos de conexión, DNS o errores del servidor (ej. 404, 503).

```python
# Maneja errores específicos de la librería 'requests' (ej. no hay internet)
except requests.exceptions.RequestException as e:
    print(f"Error al conectar con la API: {e}")
```

4. **Procesamiento y Enriquecimiento de Datos**

Este módulo no solo descarga datos, sino que los transforma.

- **Aplanamiento de Datos:** La API devuelve datos anidados (ej. `pais.get('translations', {}).get('spa', {}).get('common', 'N/A')`). El script los "aplana", seleccionando solo los campos de interés y manejando claves faltantes de forma segura con `.get()`.

```
# pais.get('translations', {}) -> Obtiene 'translations' o un dict vacío {} si no existe.
# .get('spa', {}) -> Obtiene 'spa' o un dict vacío {} si no existe.
# .get('common', 'N/A') -> Obtiene 'common' o 'N/A' si no existe.
'nombre_comun_es': pais.get('translations', {}).get('spa', {}).get('common', 'N/A'),
'nombre_oficial_es': pais.get('translations', {}).get('spa', {}).get('official', 'N/A'),
```

- **Persistencia en CSV:** Se utiliza el módulo `csv` de la biblioteca estándar, concretamente `csv.DictWriter`, para escribir los datos aplanados en un formato tabular, escribiendo primero una cabecera (`writeheader()`) y luego cada fila.

```
# Crea un "escritor" que entiende de diccionarios
escritor = csv.DictWriter(archivo_csv, fieldnames=campos)

# Escribe la primera fila (la cabecera) con los nombres de los campos
escritor.writeheader()
```

- **Enriquecimiento de Datos:** El concepto más avanzado aquí es el de `unir_csvs_en_uno`. Esta función lee el nombre del archivo (ej. `Europe.csv`), extrae el nombre del continente (`"Europe"`), y lo añade como una **nueva columna** (`continente`) a cada fila de ese archivo antes de guardarlo en el CSV maestro.

5. **Gestión de Estado en la GUI**

La eficiencia de la interfaz se basa en un patrón fundamental de gestión de estado en memoria: la separación entre la "lista maestra" y la "lista de vista".

- `dataset_paises` (Lista Maestra): Está lista global se carga **una sola vez** desde el CSV al iniciar. Actúa como la "fuente única de verdad" y **nunca se modifica**. Se usa como base inmutable para todos los filtros y para calcular las estadísticas globales.

```
dataset_paises = []     # La "Lista Maestra". Se carga 1 vez desde el CSV y NUNCA se modifica.
```

- `dataset_mostrado` (Lista de Vista): Esta lista global contiene sólo los datos que el usuario ve en la tabla.
    - Al **filtrar** (`actualizar_vista`), esta lista se *reemplaza* por completo con un nuevo subconjunto de `dataset_paises`.
    - Al **ordenar** (`ordenar_columna`), esta lista se modifica *in-situ* (`dataset_local.sort()`).

```
dataset_mostrado = [] # La "Lista de Vista". Es la que se muestra en la tabla y se modifica (filtra, ordena).
```

Este patrón evita errores comunes, como intentar filtrar sobre un conjunto de datos ya filtrado.

6. **Optimización de Datos para Rendimiento**

Para evitar conversiones de tipo repetitivas durante los filtros y ordenamientos, el proyecto aplica una **optimización de pre-conversión**.

- Durante `cargar_datos_en_memoria`, los campos de población y área (que se leen como texto del CSV) se convierten a enteros **una sola vez** y se guardan en nuevas claves: `poblacion_num` y `area_num`.

```
for pais in dataset:
    try:
        pais['poblacion_num'] = int(pais.get('poblacion', 0))
    except:
        pais['poblacion_num'] = 0 # Valor por defecto si falla
    try:
        pais['area_num'] = int(pais.get('area', 0))
    except:
        pais['area_num'] = 0 # Valor por defecto si falla
```

- Posteriormente, todas las operaciones que requieren lógica numérica (ordenar, filtrar por rango, calcular estadísticas) utilizan estos campos `_num` pre-calculados. Esto es significativamente más rápido que hacer `int(pais['poblacion'])` en cada iteración de un bucle.

7. Diseño de Interfaz Gráfica

La interfaz se construye con la biblioteca estándar `tkinter`.

- **Widgets Modernos (`ttk`):** El proyecto prioriza el uso de `tkinter.ttk` (themed widgets) sobre los widgets clásicos de `tk`. Se utilizan `ttk.Frame`, `ttk.Button`, `ttk.Entry`, `ttk.Combobox` y `ttk.Treeview`, lo que proporciona una apariencia más moderna.

- **Gestión de Layout:** Se utiliza `pack()` para la división principal de la ventana (un panel de control a la izquierda y un panel de datos a la derecha).

- **Visualización de Datos (`Treeview`):** El widget `ttk.Treeview` es el componente central para mostrar los datos tabulares. El script lo configura en modo `show="headings"` y lo puebla dinámicamente con `tree.insert()`.

- **Ventanas Secundarias (`Toplevel`):** La ventana de "Estadísticas" es un widget `Toplevel`, que representa una ventana secundaria que depende de la ventana principal.

- **Validación de Entrada:** Se realiza una validación activa de la entrada del usuario en `_obtener_valor_numerico`. Esta función comprueba que los filtros de rango no contengan texto o números negativos, y utiliza `messagebox.showwarning` para notificar al usuario, previniendo errores en tiempo de ejecución.

```
try:
    valor_str = entrada_widget.get().strip().replace('.', '').replace(',', '')
    if not valor_str:
        return default_val # Campo vacío, usa el default (0 o infinito)

    valor_int = int(valor_str)

    if valor_int < 0:
        messagebox.showwarning("Valor Inválido", "No se permiten números negativos en los filtros de rango.")
        return None

    return valor_int # Número válido

except ValueError:
    messagebox.showwarning("Entrada Inválida", "Por favor, introduce solo números ENTEROS válidos para los filtros.")
    return None
except AttributeError:
    return None
```

8. Programación Dirigida por Eventos

El paradigma fundamental sobre el que opera `tkinter`. A diferencia de un simple script que corre de arriba a abajo y termina, la aplicación utiliza un **modelo dirigido por eventos**.

- **Bucle Principal (`mainloop`):** La línea `ventana.mainloop()` es el corazón de la aplicación. Esta función inicia un bucle infinito que "escucha" eventos del sistema operativo (clics del ratón, pulsaciones de teclas, etc.).

```
# --- 8. Iniciar el bucle de la aplicación ---
# Esta línea mantiene la ventana abierta y escuchando eventos (clics, etc.)
ventana.mainloop()
```

- **Devoluciones de Llamada (`Callbacks`):** La aplicación no hace nada hasta que el usuario genera un evento. La forma en que conecta esos eventos con la lógica es a través de *callbacks*.
    - Cuando definimos `ttk.Button(..., command=actualizar_vista)`, no estás *ejecutando* la función `actualizar_vista`. Le estás diciendo a Tkinter: "Cuando ocurra el evento 'clic' en este botón, *tú* (el `mainloop`) debes *llamar de vuelta* (call back) a mi función `actualizar_vista`".

- **Aplicación Práctica:** Todo el panel de control (`frame_izquierda`) es una demostración de este paradigma: el botón "Aplicar Filtros", el botón "Ordenar", y el botón "Estadísticas" simplemente "registran" los *callbacks* y esperan a que el usuario interactúe.

Estructuras que se usó en el proyecto

Diccionarios

Teoría: Un diccionario es una estructura de datos que almacena información en pares de **clave-valor** (ej. `{'nombre': 'Argentina'}`). Son increíblemente eficientes para buscar, añadir o modificar datos, ya que se accede a los valores directamente a través de su clave única, en lugar de por su posición.

Aplicación en el Proyecto: Los diccionarios son la **unidad fundamental** de los datos.

- **Estructura de Datos Principal:** Las variables `dataset_paises` y `dataset_mostrado` no son simples listas; son **listas de diccionarios**. Cada elemento en esas listas es un diccionario que representa a un país (ej. `{'nombre_comun_es': '...', 'poblacion_num': ...}`).

- **Lectura de Datos:** En `cargar_datos_en_memoria`, usamos `csv.DictReader`. Esta herramienta lee cada fila del CSV y la convierte automáticamente en un diccionario, usando la cabecera del CSV como las claves.

- **Mapeo de UI:** En `ordenar_desde_controles`, utilizamos un diccionario (`mapa_columnas`) para "traducir" la opción visible en el menú (ej. "Población") a la clave interna real del diccionario (ej. "poblacion").

- **Gestión de Estado:** `estado_orden` es un diccionario que recuerda la dirección de ordenamiento (`{col: True}`) para cada columna

Listas

Teoría: Una lista es una colección **ordenada** y **mutable** de elementos. Los elementos se acceden por su índice (posición). Son la estructura ideal para almacenar colecciones de elementos sobre los que se necesita iterar (recorrer uno por uno).

Aplicación en el Proyecto: Si los diccionarios son el "país", las listas son el "mundo" que los contiene.

- **Almacenamiento de Datos:** `dataset_paises` y `dataset_mostrado` son las listas principales que almacenan todos los diccionarios de países.

- **Filtrado:** La lógica central de `actualizar_vista` se basa en la **comprensión de listas** (ej. `lista_temporal = [pais for pais in lista_temporal if ...]`). Esta es una forma pitónica y eficiente de crear una *nueva* lista basada en los elementos de otra que cumplen una condición.

- **Iteración:** Usas bucles `for` (ej. `for pais in dataset:`) para recorrer estas listas y realizar acciones, como convertir números en `cargar_datos_en_memoria` o aplicar filtros.

Funciones

Teoría: Una función es un bloque de código **reutilizable** que realiza una tarea específica. Las funciones son la base de la **modularidad**: permiten encapsular la lógica (ocultar la complejidad) y darle un nombre descriptivo. Esto hace que el código sea más fácil de leer, depurar y mantener.

Aplicación en el Proyecto: Nuestro proyecto está perfectamente modularizado gracias a las funciones.

- **Separación de Conceptos:** Tiene archivos separados (`generarPaises.py`, `interfaz.py`) que contienen funciones para tareas distintas (obtener datos vs. mostrarlos).

- **Lógica de la GUI:** Cada botón de la interfaz llama a una función (ej. `resetear_vista`, `ordenar_desde_controles`, `mostrar_ventana_estadisticas`).

- **Funciones Unificadas:** `actualizar_vista` es un gran ejemplo. Centraliza toda la lógica de filtrado en un solo lugar. Si necesitas cambiar cómo se filtra, solo modificas esa función.

- **Funciones Auxiliares:** `_obtener_valor_numerico` es una función auxiliar perfecta. Es una tarea compleja (validar entrada de usuario) que se necesita varias veces. Al aislarla, `actualizar_vista` se mantiene limpia y legible.

Condicionales (if / else)

Teoría: Los condicionales (instrucciones `if`, `elif`, `else`) son la herramienta principal para el **control de flujo**. Permiten que el programa tome decisiones y ejecute diferentes bloques de código basados en si una condición es `True` o `False`.

Aplicación en el Proyecto: Los condicionales son los que dan "inteligencia" a la aplicación.

- **Lógica de Caché:** En `main.py`, el `if not os.path.exists(...)` es el condicional más importante del proyecto. Decide si descargar todos los datos (lento) o iniciar la app inmediatamente (rápido).

- **Validación de Errores:** En `_obtener_valor_numerico`, usamos `if valor_int < 0:` para capturar errores del usuario. En `actualizar_vista`, el `if min_pob is None:` decide si detener el filtro o continuar.

- **Lógica de Filtrado:** En `actualizar_vista`, los `if filtro_continente != "Todos":` y `if termino_busqueda:` deciden si un filtro debe aplicarse o no.

Ordenamientos

Teoría: El ordenamiento es el proceso de organizar una colección (como una lista) en un orden específico (ascendente o descendente). Python ofrece métodos potentes como `list.sort()`, que puede personalizarse con un parámetro `key`.

Aplicación en el Proyecto: Implementar un ordenamiento personalizado y eficiente.

- **Función `ordenar_columna`:** Esta función utiliza `dataset_local.sort(key=clave_orden, reverse=es_descendente)`.

- **Parámetro `key`:** Este es el concepto clave. En lugar de ordenar por defecto, le dices a `.sort()` que llame a tu función `clave_orden` para *cada* elemento.

- **Función `clave_orden`:** Esta función es inteligente. Usa un condicional (`if col == 'poblacion'`) para decidir *cómo* ordenar. Si es "poblacion", devuelve el valor numérico (`poblacion_num`) para un ordenamiento matemático. Si es "Nombre", devuelve el valor en minúsculas (`str(valor).lower()`) para un ordenamiento alfabético que no distinga mayúsculas.

Estadísticas Básicas

Teoría: La estadística descriptiva básica consiste en resumir un conjunto de datos mediante cálculos simples, como el máximo, el mínimo, el promedio (media) y el conteo (frecuencia).

Aplicación en el Proyecto: La función `mostrar_ventana_estadisticas` es una aplicación directa de estos conceptos.

- **Máximo y Mínimo:** Se utiliza `max(dataset_paises, key=...)` y `min(...)` para encontrar los países con mayor y menor población. De nuevo, el parámetro `key` es crucial.

- **Promedio (Media):** Calculamos el promedio con la fórmula clásica: `sum(p['poblacion_num'] ... ) / len(dataset_paises)`.

- **Conteo (Frecuencia):** Usamos la herramienta perfecta para esto: `collections.Counter`. Esta clase especializada de diccionario toma una lista (como los continentes) y devuelve un diccionario con cada continente como clave y su número de apariciones como valor.

Archivos CSV (Valores Separados por Comas)

Teoría: CSV es un formato de archivo de texto plano universal para almacenar datos tabulares (como una hoja de cálculo). Cada línea es una fila de datos, y cada valor en esa fila está separado por una coma.

Aplicación en el Proyecto: El CSV es la **capa de persistencia** de tu proyecto; es tu "base de datos".

- **Escritura (`generarPaises.py`):**

    - `csv.DictWriter`: Lo usamos para escribir los datos de la API en los CSV de continentes. Es ideal porque podemos pasarle directamente un diccionario (un país) y se encarga de mapear los valores a las columnas correctas.

    - `csv.writer`: Lo usamos en `unir_csvs_en_uno` para un trabajo más manual: leer filas (como listas) y escribir filas modificadas (añadiendo el continente).

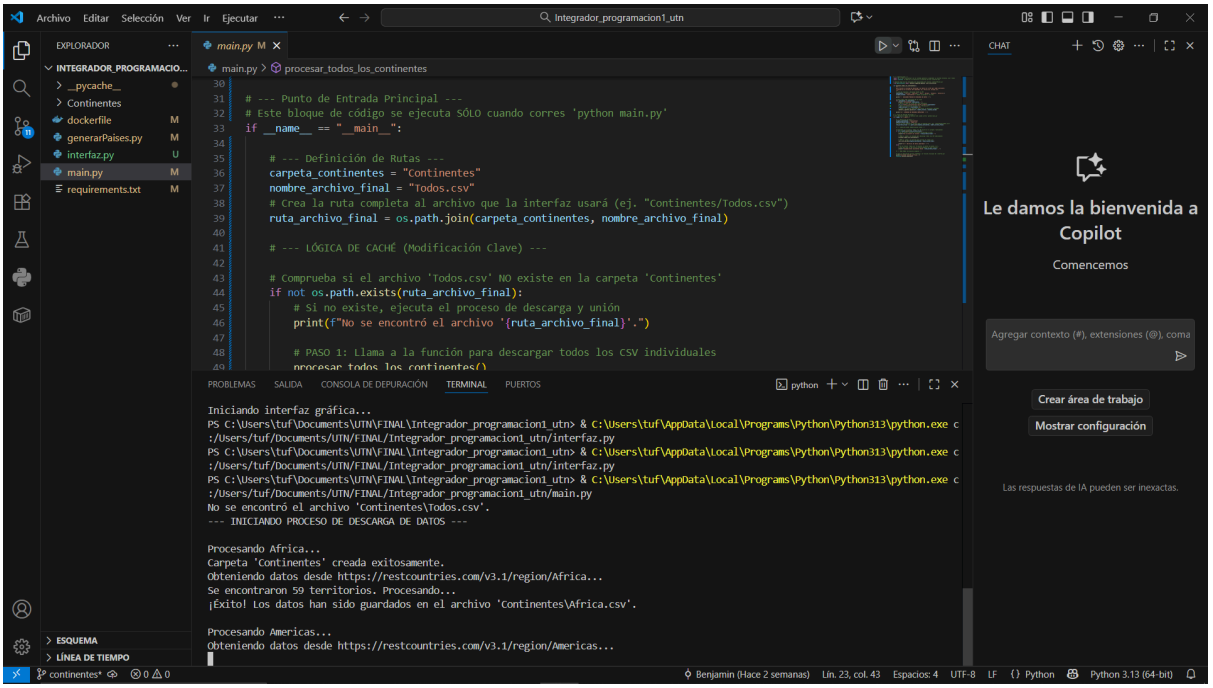
- **Lectura (`interfaz.py`):**

    - `csv.DictReader`: Lo usamos en `cargar_datos_en_memoria` para leer el archivo `Todos.csv`. Es la contraparte de `DictWriter` y convierte cada fila del CSV en un diccionario, que es exactamente la estructura que necesita la aplicación.

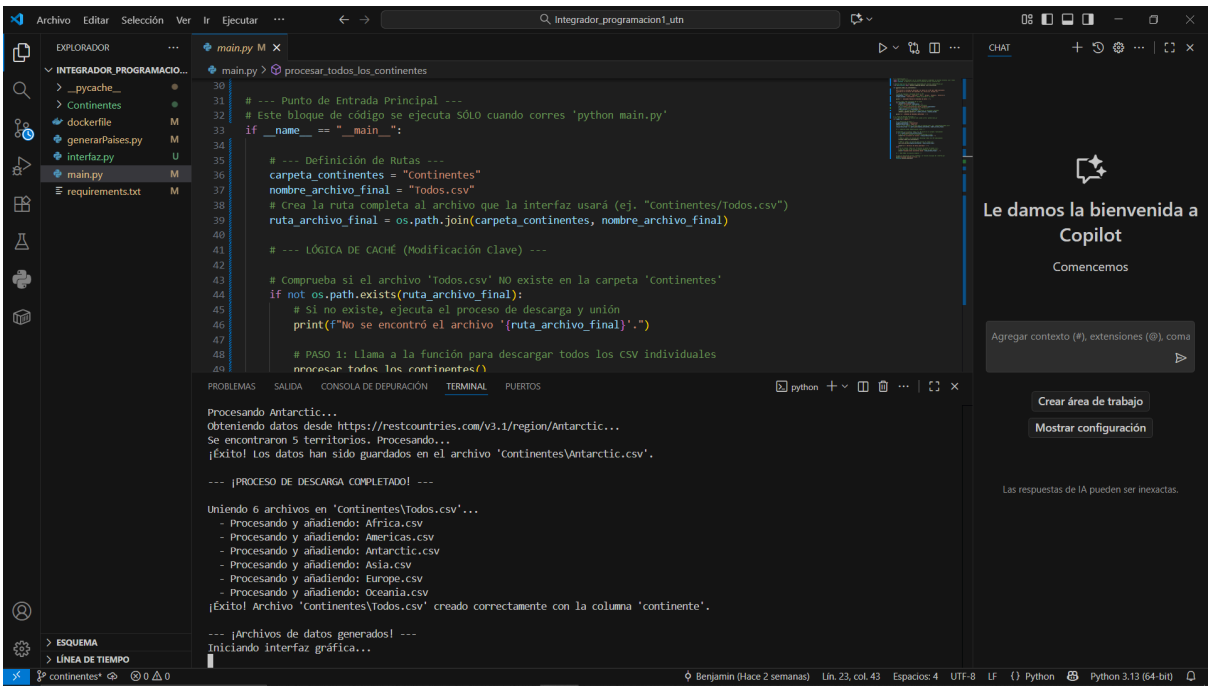
FUNCIONALIDAD DE LA APLICACIÓN

(capturas de pantalla)

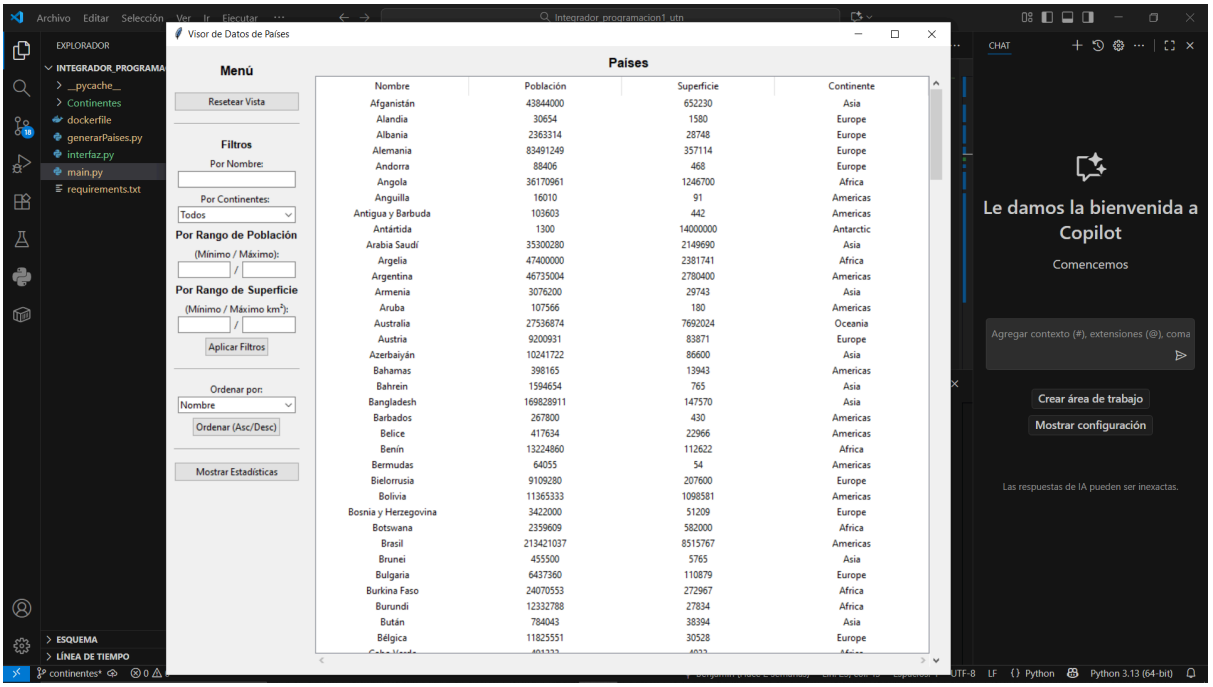
Ejecutamos el archivo “main.py” y comenzamos a descargar los datos de la api a una carpeta con archivos csv. Si ya existe se saltea esta parte.



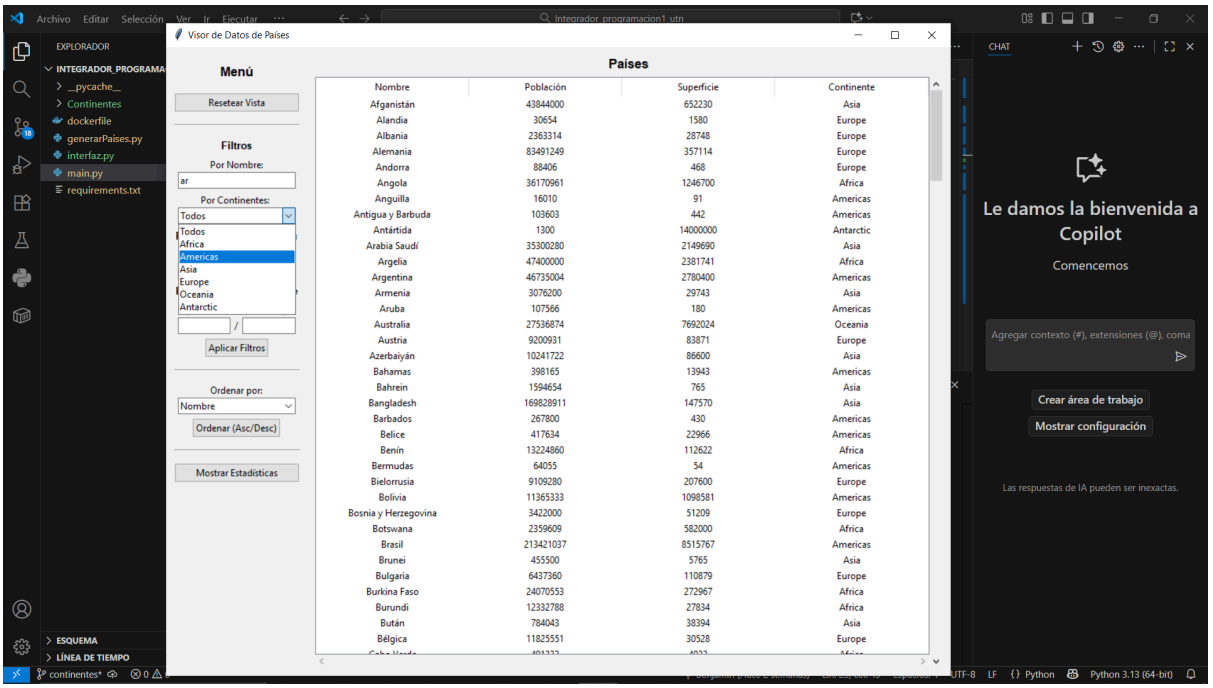
Se ejecuta la interfaz gráfica. Para este proyecto decidimos utilizar tkinter por su diseño y facilidad de uso.



Se abre la ventana principal, que se va a encargar de visualizar los datos de los países (nombre, población, superficie, continente) en base a estas categorías funcionará la aplicación y las opciones del menú.

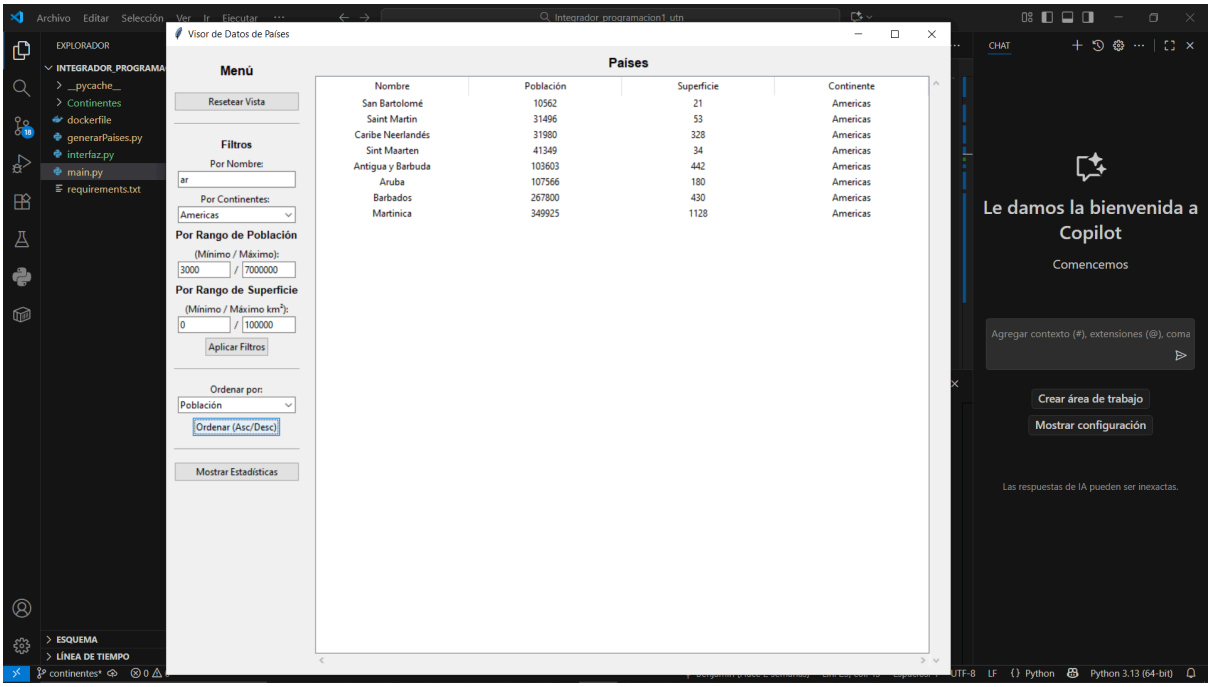


En la sección de Filtro encontramos la opción de búsqueda que puede ser parcial o total, abajo se desplaza un “combobox” que filtra por nombre de continente.

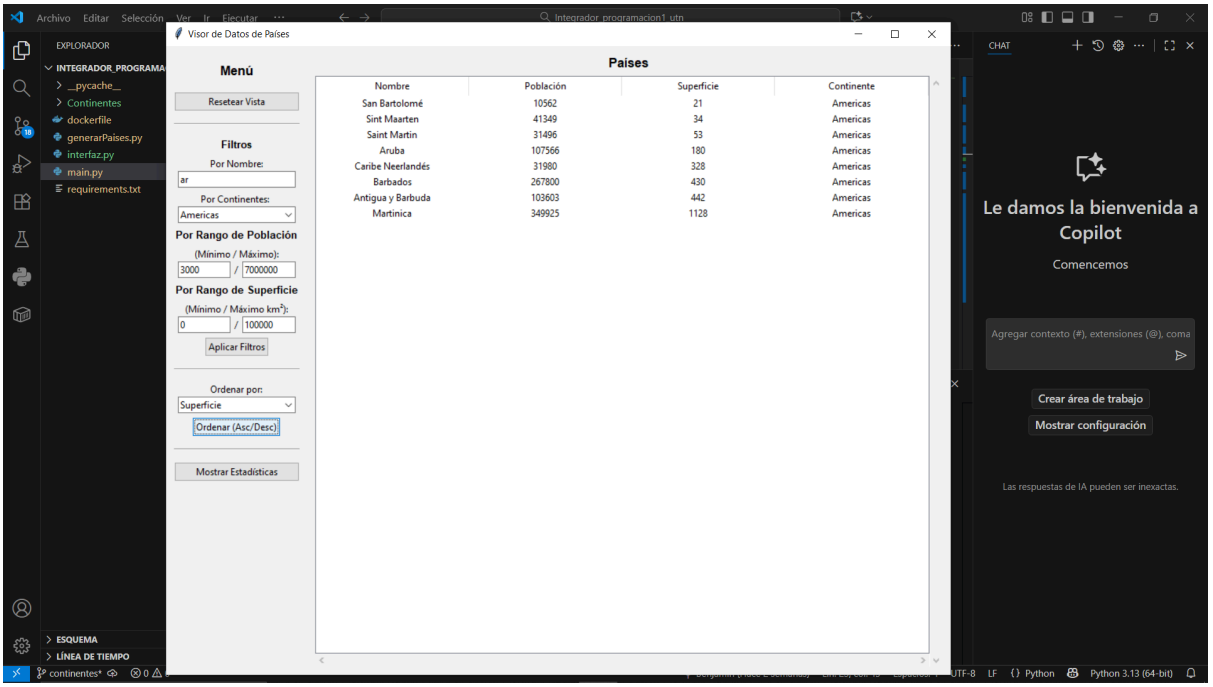


Además tenemos la opción de filtrar por rango de población y/o rango de superficie, que va del mínimo al máximo indicado por el usuario.

Una vez que visualizamos los datos solicitados, estos se pueden ordenar por población de forma Ascendente o Descendente.

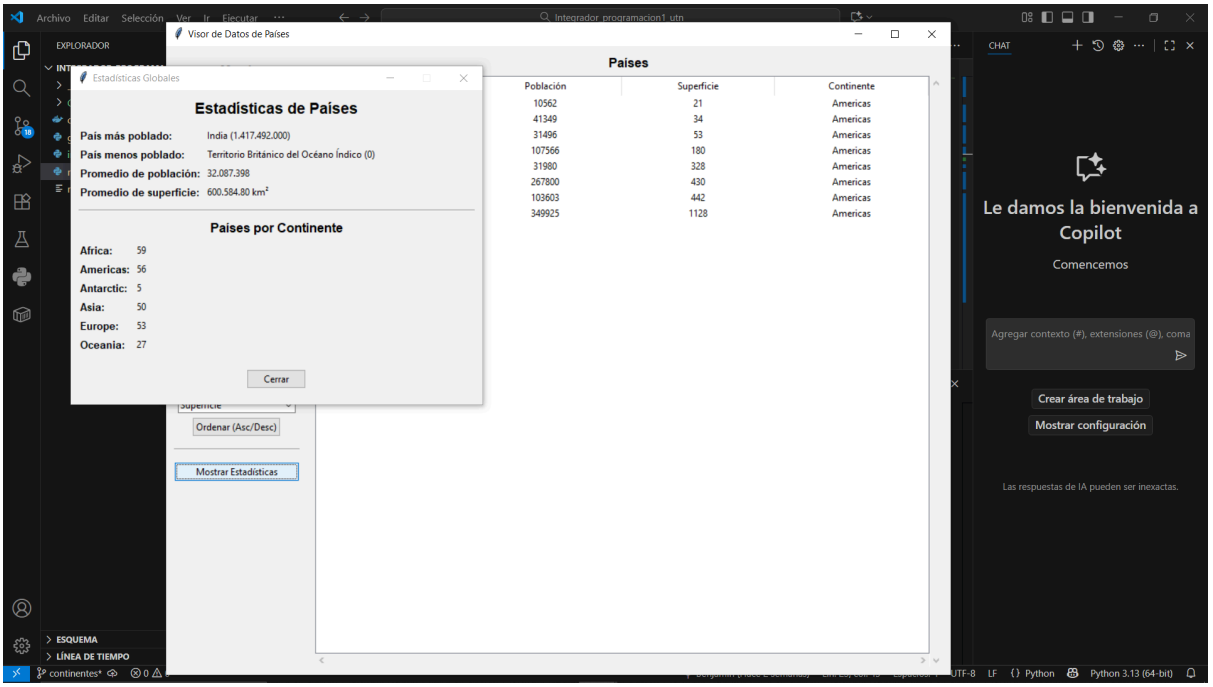


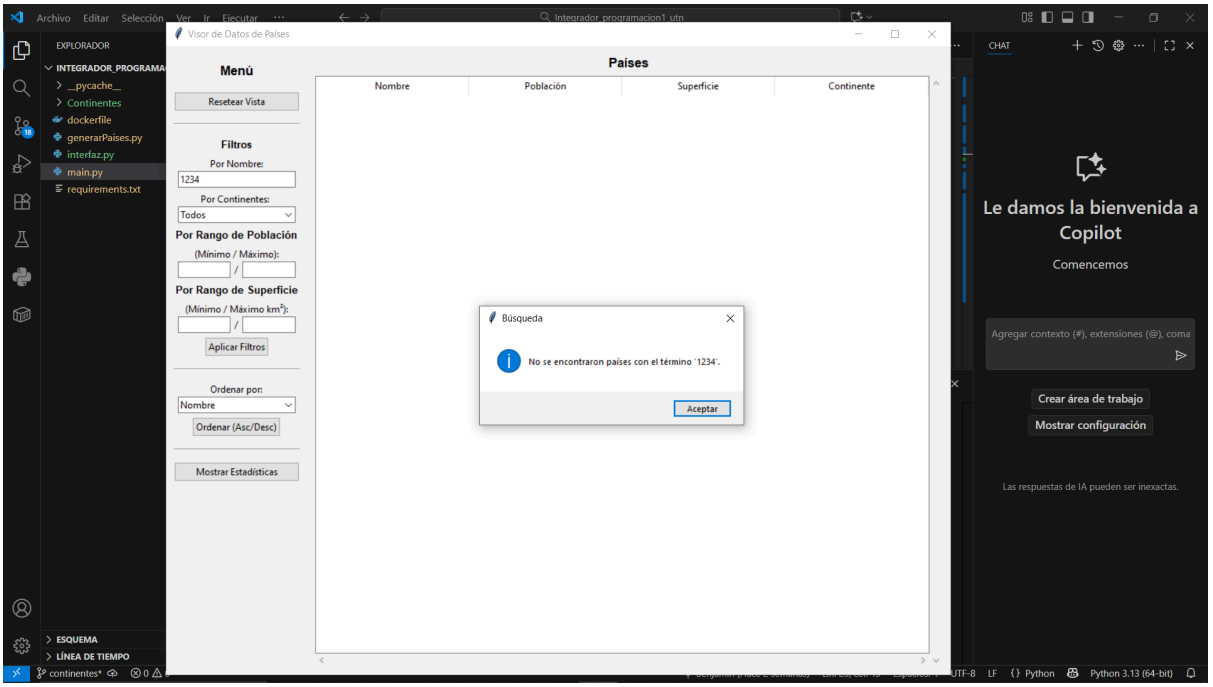
U ordenar por superficie de igual manera.

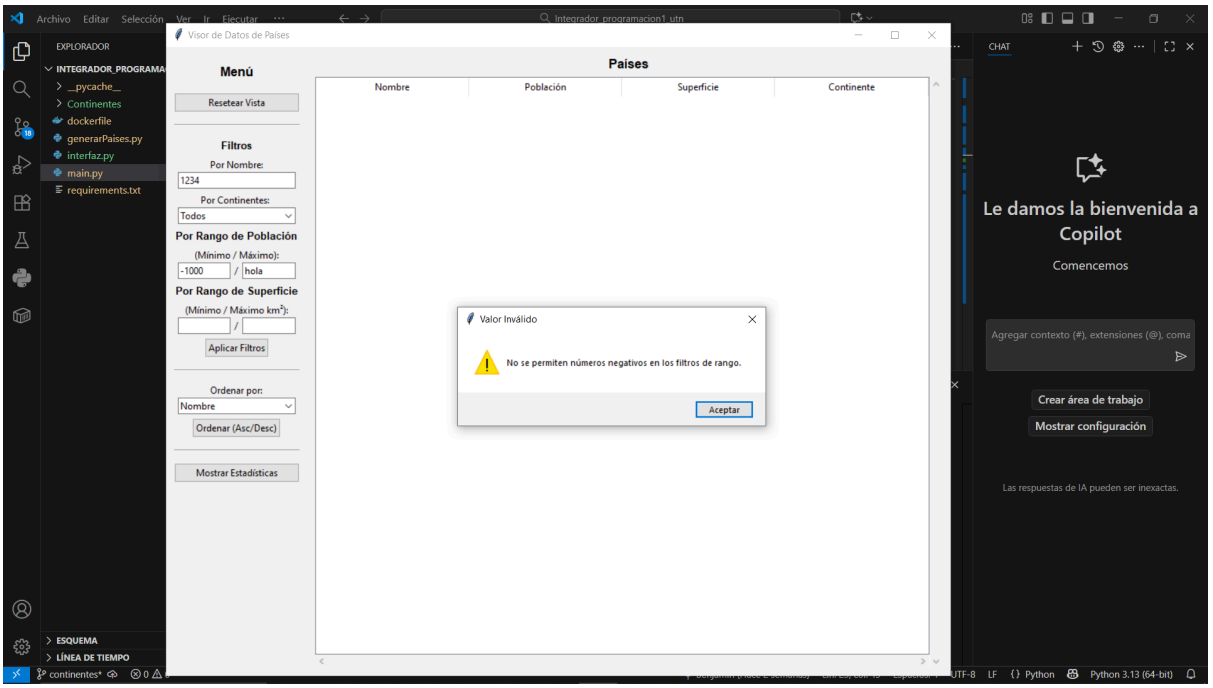


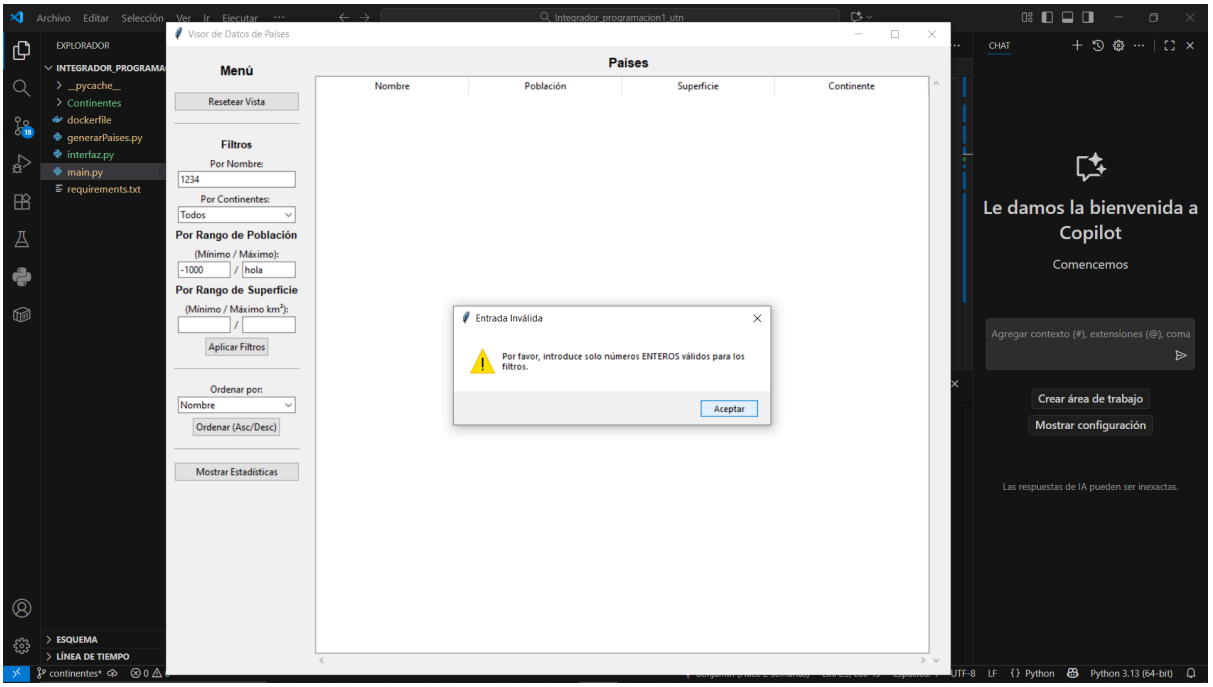
También creamos un botón donde muestra en una ventana aparte las estadísticas generales de todos los países basados en:

País más poblado, País menos poblado, Promedio de población, Promedio de Superficie y Cantidad de países por Continente.




Visor de Datos de Países
Menú
Resetear Vista
Filtros
Por Nombre:
1234
Por Continentes:
Todos
Por Rango de Población
(Mínimo / Máximo):
Por Rango de Superficie
(Mínimo / Máximo km²):
Aplicar Filtros
Ordenar por:
Nombre
Ordenar (Asc/Desc)
Mostrar Estadísticas
Países
Nombre
Población
Superficie
Continente
Búsqueda
No se encontraron países con el término '1234'.
Aceptar
Le damos la bienvenida a Copilot
Comencemos
Crear área de trabajo
Mostrar configuración
Las respuestas de IA pueden ser inexactas.


Visor de Datos de Países
Menú
Resetear Vista
Filtros
Por Nombre:
1234
Por Continentes:
Todos
Por Rango de Población
(Mínimo / Máximo):
-1000
hola
Por Rango de Superficie
(Mínimo / Máximo km²):
Aplicar Filtros
Ordenar por:
Nombre
Ordenar (Asc/Desc)
Mostrar Estadísticas
Países
Nombre
Población
Superficie
Continente
Valor Inválido
No se permiten números negativos en los filtros de rango.
Aceptar
Le damos la bienvenida a Copilot
Comencemos
Crear área de trabajo
Mostrar configuración
Las respuestas de IA pueden ser inexactas.



Fuentes Bibliográficas

- **Python Software Foundation.** *Documentación Oficial de Python 3.* (Específicamente para los módulos `csv`, `os`, `tkinter` y `collections`).

  https://docs.python.org/3/

- **Python Requests.** *Requests: HTTP for Humans™.* (Documentación de la biblioteca `requests`).

  https://requests.readthedocs.io/

- **Tkinter Documentation.** *TkDocs.* (Tutoriales y referencias para `Tkinter` y `ttk`).

  https://tkdocs.com/

- **Lutz, M. (2013).** *Programming Python (5th ed.).* O'Reilly Media. (Referencia exhaustiva sobre desarrollo de aplicaciones y GUI con Tkinter).

- **Documentación de REST Countries API.** (Fuente de los datos y especificación de los *endpoints* utilizados).

  https://restcountries.com/