UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador de Programación

Gestión de Datos de Países en Python

Alumnos

Agustin Tomás Sánchez Almonacid (agtoez@gmail.com)- Comision 12

Brian Silvero (briandavidsilvero@gmail.com) - Comisión 13

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Programación I

Docente Titular

Ariel Enferrel

Docente Tutor

Franco Gonzalez - Comision 12

Miguel Barrera - Comision 13

Entrega: 11 de Noviembre de 2025

UNIVERSIDAD TECNOLOGICA NACIONAL

Tabla de contenido

Introducción

El presente trabajo tiene como objetivo desarrollar un sistema en Python que permita gestionar información sobre países utilizando estructuras de datos básicas. El programa realiza operaciones como agregar nuevos registros, modificar datos existentes, buscar países, aplicar filtros, ordenar resultados y obtener estadísticas generales a partir del conjunto de datos.

La información de los países se almacena en un archivo CSV y se carga en memoria dentro de una lista de diccionarios. La aplicación se ejecuta en consola mediante un menú interactivo, donde cada opción se encuentra encapsulada en funciones específicas, manteniendo una estructura clara y modular.

Este proyecto permite integrar y aplicar los conceptos principales aprendidos en la materia Programación 1, especialmente el manejo de listas, diccionarios, funciones, condicionales, repetición y lectura/escritura de archivos.

Objetivos

Desarrollar una aplicación en Python para gestionar, organizar y analizar información de países a partir de un archivo CSV, aplicando estructuras de datos adecuadas, modularización del código y validaciones preventivas que aseguren la integridad y coherencia de los datos. El sistema debe ofrecer búsquedas, filtros, ordenamientos y estadísticas, priorizando la claridad, la legibilidad y la estabilidad ante distintos escenarios de uso.

Objetivos específicos:

- **Aplicar programación estructurada**, utilizando listas, diccionarios, funciones y estructuras de control para resolver un problema real de gestión de datos.
- **Diseñar un programa modular**, donde cada función cumpla una responsabilidad específica, favoreciendo la reutilización, la comprensión del código y su mantenimiento.
- **Implementar un menú interactivo en consola** que ofrezca navegación simple y ordenada entre las operaciones disponibles.
- **Incorporar validaciones preventivas** para asegurar la correcta lectura del CSV y el tratamiento adecuado de las entradas del usuario, evitando campos vacíos, formatos inválidos y valores fuera de rango.
- **Desarrollar filtros y ordenamientos** que permitan visualizar los países por nombre, continente, población y superficie, en orden ascendente o descendente.
- **Calcular estadísticas descriptivas**: país con mayor y menor población, promedios de población y superficie, y cantidad de países por continente.
- **Garantizar calidad de código**, con comentarios breves, mensajes claros al usuario y convenciones consistentes de nombres.
- **Fortalecer el razonamiento lógico**, integrando teoría y práctica en una solución funcional, robusta y alineada con los objetivos de la asignatura.

Marco Teórico

El proyecto integra estructuras de datos (listas y diccionarios), funciones con responsabilidad única, estructuras condicionales y repetitivas para el control del flujo, y operaciones de ordenamiento y estadísticas. La lectura desde CSV completa el circuito de “ingreso–procesamiento–salida” de información. A continuación se describen los fundamentos y cómo se aplican en el programa:

Listas

Una lista es una estructura de datos que permite almacenar múltiples valores en una sola variable, manteniendo un orden específico.

Se utilizan para agrupar elementos y luego recorrerlos mediante ciclos.

```
paises = ["Argentina", "Japón", "Brasil"]
```

En este proyecto se utiliza una lista de diccionarios, donde cada elemento representa un país.

Diccionarios

Un diccionario almacena datos en forma de clave : valor.

Esto permite acceder a cada dato identificándolo por su clave.

Ejemplo:

```
pais = {"nombre": "Argentina", "poblacion": 45376763, "superficie": 2780400, "continente": "América"}
```

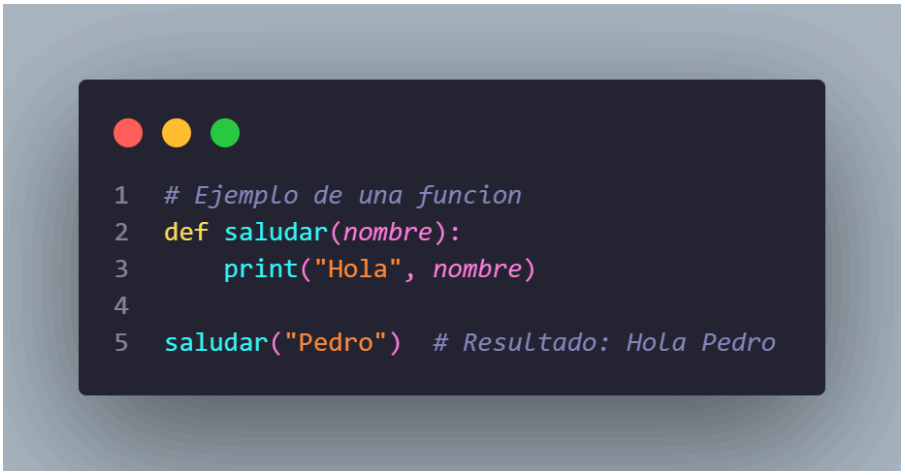
Gracias a esto es posible acceder de manera directa:

```
pais["poblacion"]
```

Funciones

Una función agrupa instrucciones para realizar una tarea específica.
Permiten organizar el código y evitar repeticiones.

Regla utilizada: una función = una responsabilidad**.** Ejemplo:



```python
# Ejemplo de una funcion
def saludar(nombre):
    print("Hola", nombre)

saludar("Pedro")  # Resultado: Hola Pedro
```

Otro ejemplo de una función relacionado con el TPI:

```python
# Ejemplo una funcion relacionada al TPI y como usarla
def contar_paises_por_continente(lista, continente_buscado):
    contador = 0
    indice = 0

    while indice < len(lista):
        pais = lista[indice]
        if pais["continente"].lower() == continente_buscado.lower():
            contador = contador + 1
        indice = indice + 1

    return contador

resultado = contar_paises_por_continente(paises, "América")
print("Cantidad de países en América:", resultado)
```

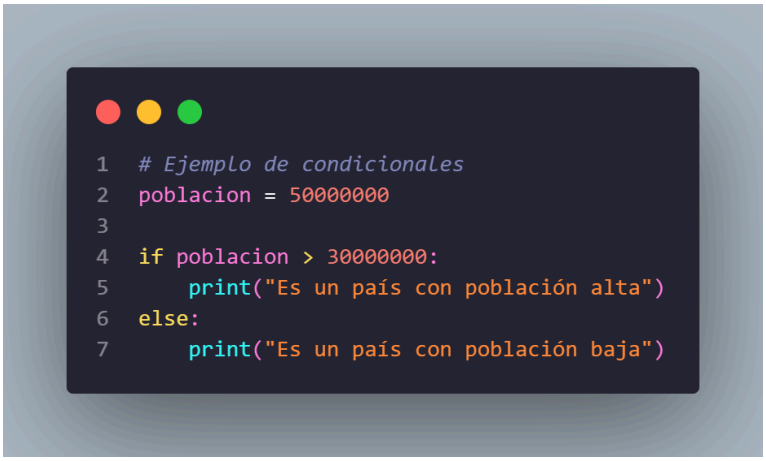
Qué está demostrando este ejemplo:

- Uso de función (def).
- Trabajo con una lista de diccionarios.
- Uso de ciclo while.
- Uso de comparación (if).
- Retorno de un valor.

Condicionales

Los condicionales permiten tomar decisiones según una situación.

Ejemplo:



```python
# Ejemplo de condicionales
poblacion = 50000000

if poblacion > 30000000:
    print("Es un país con población alta")
else:
    print("Es un país con población baja")
```

Estructuras Repetitivas

Sirven para recorrer listas o repetir procesos. Se utilizó while, evitando funciones compactas como map o sorted, tal como exige la consigna.

Ejemplo:

```python
# Ejemplo Estructuras Repetitivas
indice = 0
while indice < len(paises):
    print(paises[indice]["nombre"])
    indice += 1
```

Ordenamientos

El ordenamiento permite reorganizar la lista según un criterio (por ejemplo, de menor a mayor).Se implementó manualmente mediante intercambios de elementos.

Ejemplo básico:

```python
# Ejemplo de Ordenamiento
i = 0
while i < len(paises)-1:
    j = i + 1
    while j < len(paises):
        if paises[i]["poblacion"] > paises[j]["poblacion"]:
            temp = paises[i]
            paises[i] = paises[j]
            paises[j] = temp
        j += 1
    i += 1
```

Este código está haciendo un ordenamiento básico. Se revisa la lista comparando pares de países y, cuando un país tiene más población que otro que viene después, se intercambian. Así poco a poco los países menos poblados van quedando al principio.

Estadísticas Básicas

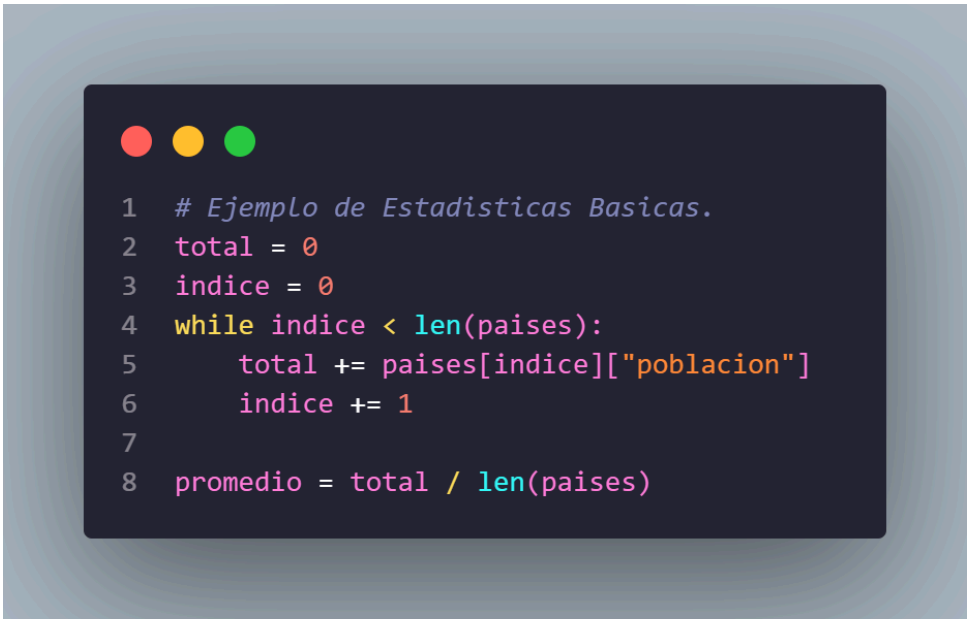
Las estadísticas permiten obtener información general sobre el conjunto de datos.

Ejemplos aplicados:

- Encontrar un país con mayor población.
- Encontrar un país con menor población.
- Calcular promedios.
- Contar países por continente.

Esto se resolvió recorriendo la lista y acumulando valores.

Ejemplo simple de promedio:



```python
# Ejemplo de Estadisticas Basicas.
total = 0
indice = 0
while indice < len(paises):
    total += paises[indice]["poblacion"]
    indice += 1

promedio = total / len(paises)
```

Archivos CSV

Un archivo CSV almacena datos separados por comas. Su estructura es texto plano, lo que permite leerlo y escribirlo fácilmente.

Ejemplo de contenido:

nombre,poblacion,superficie,continente

Argentina,45376763,2780400,América

En este proyecto:

- Se lee línea por línea.
- Se separan los campos usando .split(",").
- Se escriben los cambios al finalizar las operaciones.

Diseño y Flujo del Sistema

El sistema está diseñado a partir de una estructura modular. Esto significa que cada acción del programa se encuentra separada en funciones, lo que facilita la lectura, el mantenimiento y la reutilización del código. La información inicial se obtiene de un archivo CSV, el cual contiene los datos de los países. Ese archivo se carga en memoria y se representa mediante una lista de diccionarios, donde cada diccionario corresponde a un país.

Una vez cargados los datos, el programa presenta un menú principal en consola. El usuario puede seleccionar diferentes operaciones como agregar un país, actualizar los datos de uno existente, buscar por nombre, filtrar según criterios, ordenar la lista o mostrar estadísticas generales.
Cada opción del menú ejecuta una función específica, lo que mantiene el código organizado y evita repeticiones.

Cuando el usuario decide finalizar la ejecución, los datos almacenados en la lista se guardan nuevamente en el archivo CSV, asegurando la persistencia de la información. De esta forma, los cambios realizados no se pierden al cerrar el programa.

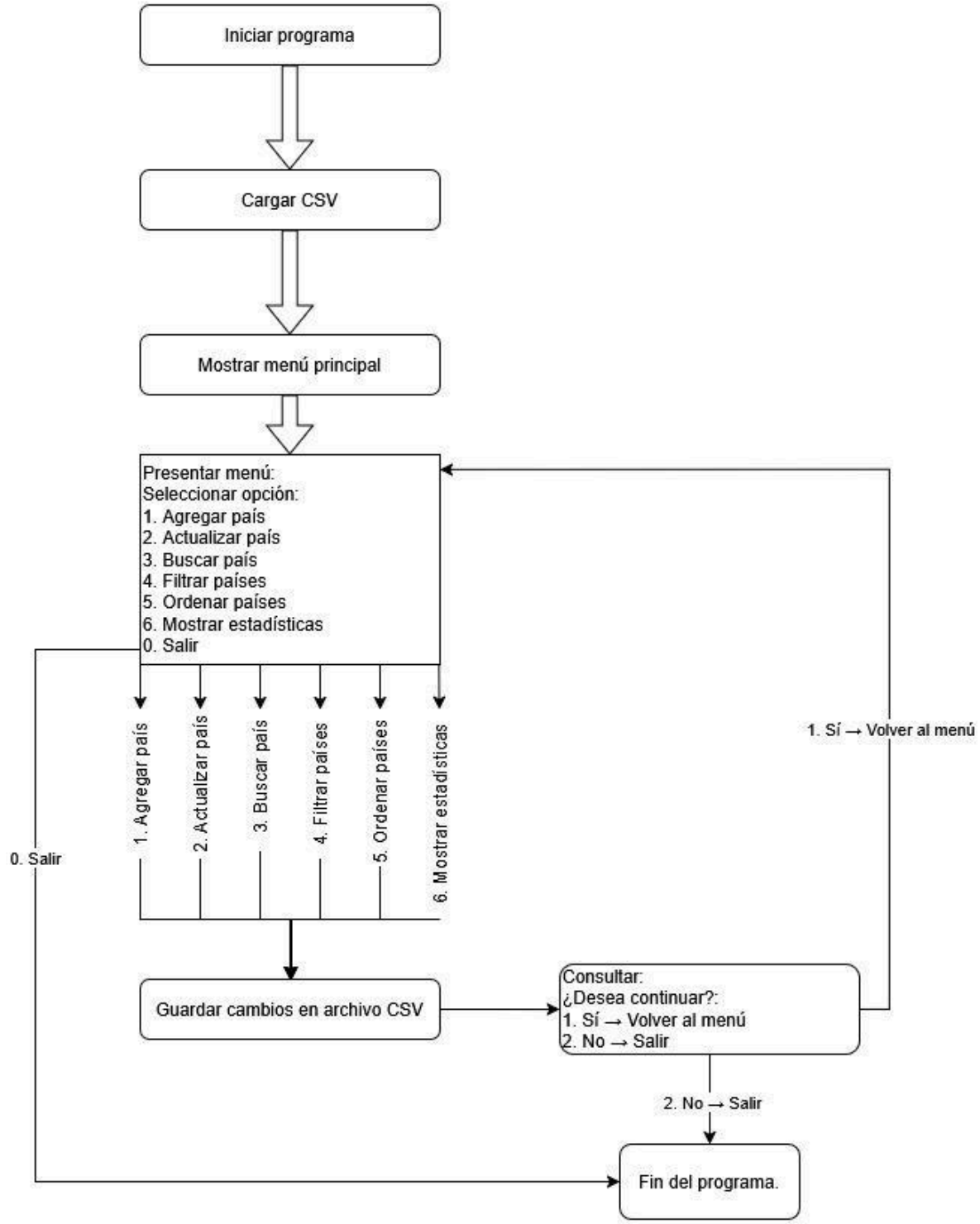
Flujo de Operación del Programa

1. **Lectura del CSV** → Se abre el archivo y se construye la lista de países.
2. **Menú Principal** → Se muestran las opciones disponibles.
3. **Interacción del Usuario** → Se elige una operación del menú.
4. **Ejecución de Funciones Específicas** → El programa realiza la acción seleccionada.
5. **Visualización de Resultados** → Se muestran datos, listados o mensajes informativos.
6. **Guardado Final** → Antes de salir, se actualiza el archivo CSV.

Diagrama de flujo

El sistema inicia con la carga y validación del CSV, construye la lista países y luego entra al menú principal (bucle). Cada opción del menú invoca un subproceso: alta, actualización, búsqueda, filtros, ordenamientos o estadísticas. Tras ejecutar la acción, el flujo retorna al menú hasta que el usuario elija salir. Todas las entradas se validan antes de operar los datos.

A continuación se presenta el diagrama de flujo de la aplicación:


Iniciar programa
Cargar CSV
Mostrar menú principal
Presentar menú:
Seleccionar opción:
1. Agregar país
2. Actualizar país
3. Buscar país
4. Filtrar países
5. Ordenar países
6. Mostrar estadísticas
0. Salir
1. Agregar país
2. Actualizar país
3. Buscar país
4. Filtrar países
5. Ordenar países
6. Mostrar estadísticas
0. Salir
1. Sí → Volver al menú
Guardar cambios en archivo CSV
Consultar:
¿Desea continuar?:
1. Sí → Volver al menú
2. No → Salir
2. No → Salir
Fin del programa.

Desarrollo del Programa

El proyecto se organiza en módulos, lo que permite mantener el código claro y fácil de mantener. La lógica principal se encuentra distribuida entre los archivos main.py y funciones.py, mientras que el archivo paises.csv almacena la información de los países de forma persistente.

Archivo main.py

Este archivo actúa como punto de entrada del programa. Su rol principal es cargar los datos desde el archivo CSV, mostrar el menú de opciones y coordinar la ejecución de las funciones correspondientes, dependiendo de la acción seleccionada por el usuario.

Luego de cada operación, los cambios quedan guardados en la estructura en memoria. Finalmente, cuando el usuario elige salir, se llama a la función de guardado para actualizar el archivo CSV.
De esta manera, el programa conserva la información para futuras ejecuciones.

Archivo funciones.py

Este módulo contiene todas las funciones que implementan la lógica del sistema. Cada función cumple una responsabilidad concreta, lo que permite dividir el problema en partes pequeñas y manejables.

- **A) Manejo del Archivo CSV:**
    - cargar_csv(ruta): Lee el archivo línea por línea y construye una lista de diccionarios, donde cada diccionario representa un país.
      Se utiliza un ciclo while para recorrer el archivo y separar cada campo mediante split(","). Esto permite convertir texto plano en una estructura de datos útil dentro del programa.

- guardar_csv(ruta, lista): Escribe nuevamente todo el contenido de la lista en el archivo CSV. Cada país se convierte en una línea formateada. Este guardado asegura la persistencia de los cambios realizados durante la ejecución.

- **B) Validaciones de Entrada:**
  - leer_texto_validado(mensaje): Solicita un texto y verifica que contenga únicamente letras y espacios, además de normalizarlo para mantener consistencia (primera letra en mayúscula).
    Evita que ingresen nombres inválidos o mal escritos.
  - leer_entero_positivo(mensaje): Garantiza que los valores ingresados para población y superficie sean números válidos y no negativos.
  - seleccionar_continente(): Presenta una lista de continentes disponibles, evita errores de tipeo y devuelve el valor correctamente escrito.

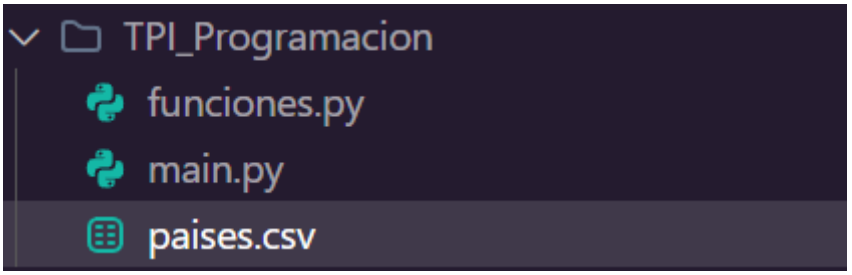
- **C) Gestión de Países:**
  - agregar_pais(lista, ruta_csv): Permite incorporar un nuevo país. Verifica primero que no exista otro con el mismo nombre, luego solicita sus datos y finalmente actualiza la lista y el archivo CSV.
  - actualizar_pais(lista, ruta_csv: Modifica la población y la superficie de un país existente. Su uso es directo: se ingresa el nombre del país y se actualizan los valores.
  - buscar_por_nombre(lista, texto: Realiza una búsqueda parcial dentro de los nombres de los países. Esto permite encontrar coincidencias aunque no se escriba el nombre completo.

- **D) Filtros:**
    - filtrar_por_continente(lista): Muestra solo los países que pertenecen a un continente seleccionado. Recorre la lista con un ciclo while y compara el campo correspondiente, respetando mayúsculas y minúsculas.

- **E) Ordenamientos:**
    - Los ordenamientos se implementaron manualmente mediante intercambios de elementos en la lista, utilizando ciclos while anidados.
        - Ejemplos utilizados:
            - Por **nombre** (A-Z).
            - Por **población** (menor a mayor o mayor a menor).
            - Por **superficie** (menor a mayor o mayor a menor)

- **F) Estadísticas:**
    - mostrar_estadisticas(lista): Recorre la lista para:
        - Determinar el país con mayor y menor población.
        - Calcular el promedio de población y superficie.
        - Contar cuántos países pertenecen a cada continente.

Estructura del registro.



- main.py: carga inicial, menú, submenús y coordinación.
- funciones.py: validadores, E/S CSV, altas, actualización, búsqueda, filtros, ordenamientos, estadísticas y salida formateada.
- paises.csv: **encabezados exactos**: nombre,poblacion,superficie,continente.

Interfaz por consola (menú y submenús)

El programa desarrollado implementa una interfaz basada en consola, organizada mediante un menú principal y diversos submenús que permiten acceder a las distintas funcionalidades del sistema (agregar, buscar, actualizar, ordenar, etc.).
Esta estructura facilita la navegación del usuario y promueve una interacción clara y ordenada.

Cada vez que el programa espera una acción, se muestra el menú antes de solicitar la opción, garantizando que el usuario tenga siempre visible la lista completa de alternativas disponibles.
Además, los mensajes del sistema son breves y consistentes, lo que favorece la comprensión y evita la sobrecarga de información.
Tras finalizar cada operación, el programa retorna automáticamente al menú principal, salvo cuando el usuario selecciona la opción “0”, que indica la finalización del programa y el guardado de los cambios.

Capturas de Ejecución

Requisitos previos

- Python 3.10+ (por el uso de match).
- Consola/terminal para ejecutar: python main.py.

Incorporación de colores ANSI y emojis en la interfaz

Para mejorar la experiencia visual del usuario, el menú y los mensajes de salida incorporan el uso de códigos de color ANSI y emojis Unicode, los cuales permiten enriquecer la presentación de texto en la consola.

Los códigos ANSI (American National Standards Institute) son secuencias de escape que permiten modificar el formato del texto mostrado en la terminal, aplicando colores, negrita o subrayado.
Su utilización mejora la legibilidad y facilita la interpretación de los mensajes.

Captura 1: Menú Principal.

Ejecutar: python main.py

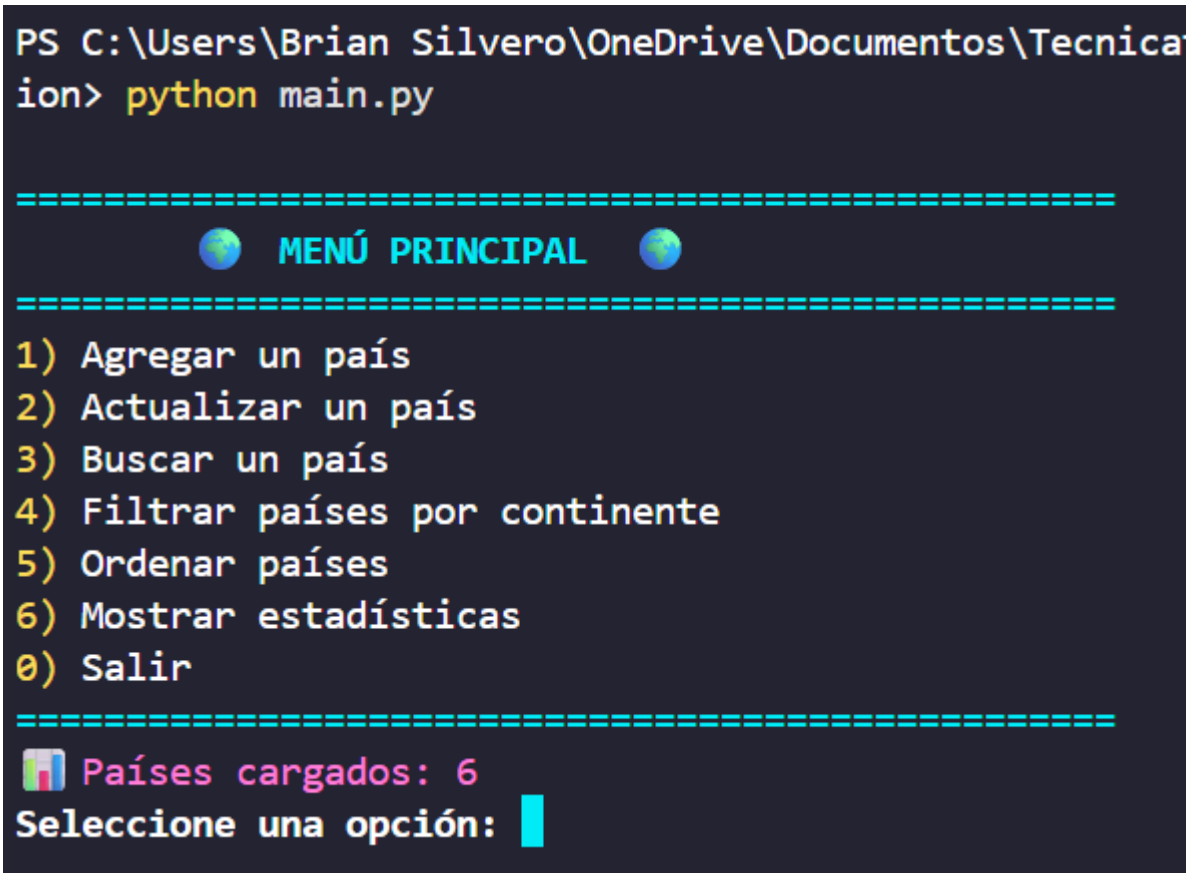


Figura 1. Vista del menú principal del programa, desde donde el usuario puede seleccionar las distintas operaciones.

Captura 2: Agregar un País. Elegimos la 1.

```
==================================================
📊 Países cargados: 6
Seleccione una opción: 1

--- Agregar País ---
Nombre: Chile

Seleccione un continente:
1) América
2) Europa
3) Asia
4) África
5) Oceanía
6) Antártida

Opción: 1
Población: 19000000
Superficie: 756102

País agregado correctamente.

--- Lista de Países ---
Argentina | Población: 45376763 | Superficie: 2780400 | Continente: América
Argelia | Población: 47400000 | Superficie: 2381741 | Continente: África
Colombia | Población: 53110609 | Superficie: 1141748 | Continente: América
Alemania | Población: 83149300 | Superficie: 357022 | Continente: Europa
Japón | Población: 125800000 | Superficie: 377975 | Continente: Asia
Brasil | Población: 213993437 | Superficie: 8515767 | Continente: América
Chile | Población: 19000000 | Superficie: 756102 | Continente: América
```

Figura 2. Proceso de carga de un nuevo país, con validación de datos y actualización de la lista.

Captura 3: Búsqueda. Elegimos la opción 3

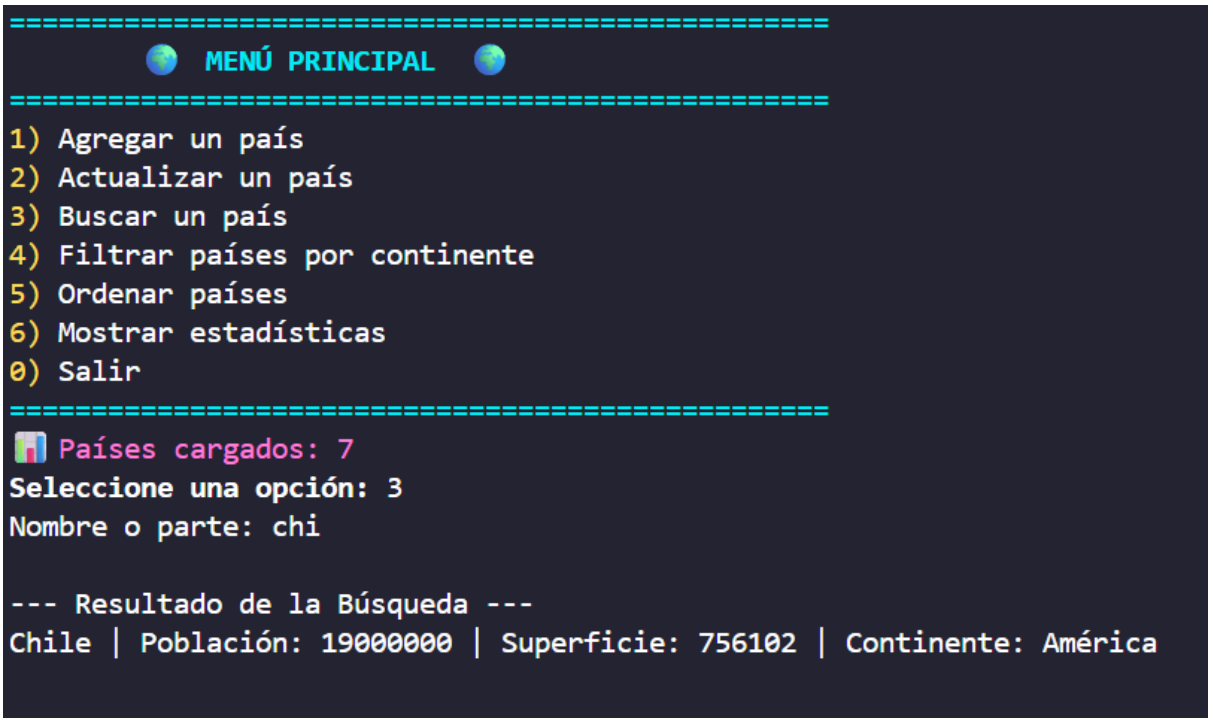


Figura 3. Ejemplo de búsqueda de país por coincidencia parcial de nombre. El sistema muestra sólo los resultados que cumplen con el criterio.

Captura 4: Estadísticas. Elegimos la opción 6

```
==================================================
        🌎 MENÚ PRINCIPAL 🌎
==================================================
1) Agregar un país
2) Actualizar un país
3) Buscar un país
4) Filtrar países por continente
5) Ordenar países
6) Mostrar estadísticas
0) Salir
==================================================
📊 Países cargados: 7
Seleccione una opción: 6

--- Estadísticas Generales ---
País con mayor población: Brasil con 213993437
País con menor población: Chile con 19000000
Promedio de población: 83975729.86
Promedio de superficie: 2330107.86 km²

Cantidad de países por continente:
- América: 4
- África: 1
- Europa: 1
- Asia: 1
```

Figura 4. Estadísticas generales obtenidas a partir del dataset: máximos, mínimos, promedios y cantidad de países por continente.

Conclusiones Grupales

El desarrollo de este trabajo nos permitió integrar los conocimientos fundamentales de Programación 1 en una solución completa y funcional. Representar los países mediante una lista de diccionarios nos permitió organizar la información de manera clara y trabajar de forma eficiente con sus atributos. La modularización en funciones facilitó la lectura del código y reforzó la importancia de dividir un problema en partes lógicas y coherentes.

La implementación manual de búsquedas, filtros y ordenamientos, sin recurrir a funciones predefinidas, nos llevó a profundizar en la lógica interna de estas operaciones. Esto nos permitió comprender no solo el resultado, sino también el proceso necesario para obtenerlo. Además, trabajar con archivos CSV nos introdujo al concepto de persistencia de datos, entendiendo cómo guardar y recuperar información fuera de la memoria.

Desde lo grupal, logramos coordinarnos, distribuir tareas y revisar decisiones en conjunto, lo cual mejoró tanto el resultado final como la comprensión del proceso. Este trabajo no solo reforzó contenidos técnicos, sino también nuestra capacidad para analizar problemas, diseñar soluciones y comunicar nuestro razonamiento de forma clara.

En conclusión, el proyecto nos brindó una experiencia completa: teoría aplicada, práctica guiada y construcción colaborativa. Esto consolidó nuestra base para continuar avanzando en el camino de la programación.

Bibliografía

Se listan a continuación las fuentes utilizadas para el desarrollo del trabajo. Se priorizó documentación oficial de Python y material técnico directamente relacionado con las funcionalidades implementadas.

- Python Software Foundation. (2024). *Python documentation*. https://docs.python.org/3/
- Downey, A. (2015). *Think Python: How to Think Like a Computer Scientist* (2.ª ed.). Green Tea Press. http://greenteapress.com/thinkpython2/thinkpython2.pdf
- Severance, C. (2016). *Python for Everybody: Exploring Data Using Python 3.* Michigan Publishing. https://www.py4e.com/book
- McKinney, W. (2018). *Python for Data Analysis* (2.ª ed.). O'Reilly Media.
- Shafranovich, Y. (2005). *Common Format and MIME Type for Comma-Separated Values (CSV) Files* (RFC 4180). IETF. https://www.rfc-editor.org/rfc/rfc4180
- Cormen, T., Leiserson, C., Rivest, R. & Stein, C. (2009). *Introduction to Algorithms* (3.ª ed.). MIT Press. (Fundamento teórico de comparaciones y ordenamientos.)
- Universidad Tecnológica Nacional (UTN). (2025). *Material de cátedra: Programación 1* (apuntes y clases).