

Trabajo Práctico Integrador

Unidades (1-9)

Gestión de Datos de Países en Python: filtros, ordenamientos y estadísticas

Alumnos - Grupo 13

Escalante, Juan Facundo. Comisión 5.

Hernández, María Aldana. Comisión 6.

Tecnicatura Universitaria en Programación

Universidad Tecnológica Nacional

Programación 1

Coordinador

Alberto Cortez

Docente Titular

Cinthia Rigoni. Comisión 4.

Sebastián Bruselario. Comisión 6.

Docente Tutor

Ana Mutti. Comisión 4.

Flor Gubiotti. Comisión 6.

11 de noviembre de 2025

Tabla de contenido

Introducción	3
Objetivos	3
Consignas	4
Requerimientos técnicos	5
Entregables	6
Criterios de evaluación	7
Desarrollo	7
Diseño del programa	8
Diagrama de flujo	15
Explicación del funcionamiento	16
Conclusión	19
Referencias bibliográficas	20

Introducción

El presente documento corresponde al **Trabajo Práctico Integrador (TPI)** de la asignatura Programación 1, cuyo objetivo central fue el desarrollo de un sistema interactivo en lenguaje Python para la gestión de datos geográficos y demográficos de países.

El proyecto, titulado "**Gestión de Datos de Países en Python: filtros, ordenamientos y estadísticas**", se concibió como una aplicación modular orientada a menú, capaz de interactuar con una estructura de datos persistente. Para ello, se implementó un sistema que lee y guarda información en un archivo de formato **CSV (Comma Separated Values)**.

El informe teórico a continuación detalla las decisiones de diseño del programa, la implementación de los conceptos mencionados y la estructura lógica del sistema a través de diagramas y explicaciones funcionales, cumpliendo con los requerimientos técnicos y de documentación establecidos en las consignas del trabajo.

Objetivos

Que los participantes logren desarrollar una aplicación en Python que permita gestionar información sobre países, aplicando listas, diccionarios, funciones, estructuras condicionales y repetitivas, ordenamientos y estadísticas.

El sistema debe ser capaz de leer datos desde un archivo CSV, realizar consultas y generar indicadores clave a partir de la data set.

El objetivo principal es afianzar el uso de estructuras de datos, modularización con funciones y técnicas de filtrado/ordenamiento, aplicando los conceptos aprendidos en Programación 1.

Consignas generales

- Lenguaje: Python 3.x
- Estructuras: listas, diccionarios, funciones.
- Archivos: lectura desde CSV.
- Código claro, comentado y modularizado (una función = una responsabilidad).
- Validaciones de entradas y manejo básico de errores.
- Trabajo en equipos de 2 personas.

Dominio (data set de países)

Cada país estará representado con los siguientes datos:

- Nombre (string)
- Población (int)
- Superficie en km² (int)
- Continente (string)

Ejemplo de registro CSV:

nombre,poblacion,superficie,continente

Argentina,45376763,2780400,América

Japón,125800000,377975,Asia

Brasil,213993437,8515767,América

Alemania,83149300,357022,Europa

Requerimientos técnicos**1) Diseño (previo al código)**

- Explicar en un informe teórico los conceptos aplicados:
 - Listas
 - Diccionarios
 - Funciones
 - Condicionales
 - Ordenamientos
 - Estadísticas básicas
 - Archivos CSV
- Definir el flujo de operaciones principales en un diagrama o esquema.

2) Funcionalidades mínimas del sistema

El programa debe ofrecer un menú de opciones en consola que permita:

- Buscar un país por nombre (coincidencia parcial o exacta).
- Filtrar países por:
 - Continente
 - Rango de población
 - Rango de superficie
- Ordenar países por:
 - Nombre
 - Población
 - Superficie (ascendente o descendente)
- Mostrar estadísticas:
 - País con mayor y menor población
 - Promedio de población
 - Promedio de superficie
 - Cantidad de países por continente

3) Validaciones

- Controlar errores de formato en el CSV.
- Evitar fallos al ingresar filtros inválidos o búsquedas sin resultados.
- Mensajes claros de éxito/error.

Entregables (obligatorios)

1. Carpeta digital

- Marco teórico con fuentes bibliográficas.
- Código Python funcional, modular y comentado.
- Capturas de pantalla de ejecución de ejemplos.
- Conclusiones grupales sobre los aprendizajes.

2. Repositorio en GitHub

Debe incluir:

- Proyecto completo en Python.
- README.md con:
 - Descripción del programa.
 - Instrucciones de uso.
 - Ejemplos de entradas y salidas.
 - Participación de los integrantes.
- Archivo CSV con el data set base.

3. Video tutorial (10–15 minutos)

- Explicación del problema planteado.
- Presentación de la estructura de datos utilizada.
- Demostración del programa funcionando.
- Reflexión final sobre el desarrollo del proyecto.

Criterios de evaluación

- Correcta funcionalidad (búsquedas, filtros, ordenamientos, estadísticas).
- Uso correcto de estructuras de datos (listas y diccionarios).
- Calidad del código (modularización, legibilidad, comentarios).
- Documentación (README claro, informe teórico coherente).
- Presentación en video (tiempo adecuado, explicación técnica, participación equitativa).
- Entrega completa en GitHub con código, informe y CSV.

Diseño del programa

El sistema fue diseñado bajo el paradigma de Programación Estructurada y Modular, haciendo uso intensivo de las siguientes estructuras de datos y conceptos aprendidos en la asignatura Programación 1.

1. Listas (Estructuras de Datos)

Las listas son la estructura de datos fundamental utilizada para almacenar la colección completa de países. Una lista en Python es una colección ordenada y mutable de elementos. Permite agrupar diferentes tipos de datos.

La variable países es una **lista principal** donde cada elemento es un **diccionario** que representa un país. Este formato es ideal para aplicar iteraciones (bucles), filtros y ordenamientos sobre el conjunto de datos completo.

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > cargar_paises
1 # Importación de bibliotecas (manejar archivos .csv y normalizar strings)
2 import csv
3 import unicodedata
4
5 (constant) CONTINENTES_VALIDOS: list[str]
6 CONTINENTES_VALIDOS = ["África", "América", "Asia", "Europa", "Oceanía", "Antártida"]
7
```

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > cargar_paises
35
36 > def obtener_superficie(pais):...
38
39
40 # Función que carga la lista de países creando un diccionario con los datos que encuentra en el .csv
41 # Cada fila del .csv crea un elemento del diccionario con sus respectivos key=values
42 def (variable) paises: list[ivo]:
43 ✨ paises = [] # Lista vacía donde se cargan los países que están en el .csv
44 archivo = open(nombre_archivo, newline="", encoding="utf-8")
45 lector = csv.DictReader(archivo) # Leemos el archivo .csv
46 # Recorremos el archivo y validamos la existencia y los tipos de datos
```

2. Diccionarios (Estructuras de Datos)

Los diccionarios se utilizan como el contenedor individual para los datos de cada país.

Un diccionario es una colección no ordenada y mutable de datos que almacena información en pares **clave-valor**. Permiten acceder a los datos por su nombre descriptivo (clave) en lugar de por un índice numérico.

Cada país (nombre, población, superficie, continente) se almacena en un diccionario. Esto proporciona un acceso claro y semántico a los atributos. Por ejemplo, para obtener la población, se accede con `pais["poblacion"]`.

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > cargar_paises
42 def cargar_paises(nombre_archivo):
53     and "superficie" in fila
54     and "continente" in fila
55     and fila["poblacion"].isdigit()
56     and fila["superficie"].isdigit():
57         (variable) pais: dict[str, Any]
58     pais = {
59         "nombre": fila["nombre"],
60         "poblacion": int(fila["poblacion"]),
61         "superficie": int(fila["superficie"]),
62         "continente": fila["continente"],
63     }
64     paises.append(pais) # Agregamos el elemento pais a paises[]
65     archivo.close()
66     return paises # La función cargar_paises devuelve paises[]
67
```

3. Funciones (Modularización)

El código está completamente modularizado mediante el uso de funciones.

Una función es un bloque de código reutilizable que realiza una tarea específica.

Se aplica el principio de **"una función = una responsabilidad"**. Las funciones principales son llamadas desde el menú (`ejecutar_programa()`) para realizar una única tarea, como `agregar_pais()`, `buscar_pais()`, o `ordenar_paises()`. Esto mejora la legibilidad y el mantenimiento del código.

Se utilizan funciones pequeñas para tareas específicas, como `normalizar()` para eliminar acentos y las funciones `obtener_nombre()`, `obtener_poblacion()`, etc., para facilitar los ordenamientos.

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > ejecutar_programa
350     print(f"    {cont}: {cant}")
351
352
353 # Función def ejecutar_programa() -> Any
354 def ejecutar_programa():
355     paises = cargar_paises(
356         "paises.csv"
357     ) # Llama a la función cargar_paises que almacena los paises en paises[] desde paises.csv
358     while True:
359         mostrar_menu() # Llama a la función que muestra el menú de opciones
360         opcion = input("Seleccione una opción: ").strip() # Solicita una opción
361         if opcion == "1":
362             agregar_pais(paises) # Llama a la función para agregar pais
363         elif opcion == "2":
364             actualizar_pais(paises) # Llama a la función para actualizar población y superficie de un pais existente
365         elif opcion == "3":
366             buscar_pais(paises) # Llama a la función que busca un pais
367         elif opcion == "4":
368             filtrar_paises(paises) # Llama a la función que filtra países por continente, rango de poblacion o superficie
369         elif opcion == "5":
370             ordenar_paises(paises) # Llama a la función que ordena por nombre, poblacion o superficie
371         elif opcion == "6":
372             mostrar_estadisticas(paises) # Llama a la función que muestra las estadísticas de los paises (mayor población, etc).
373         elif opcion == "7":
374             print("🔥 ¡Hasta luego!")
375             break # Termina el programa
376         else:
377             print("⚠️ Opción no válida.")
378
379
380 # Ejecutar
381 if __name__ == "__main__":
382     ejecutar_programa()
383
```

4. Condicionales (Estructuras de Control)

Las estructuras condicionales son esenciales para la lógica de validación y la toma de decisiones del menú. Permiten ejecutar bloques de código específicos basándose en si una condición es verdadera (True) o falsa (False).

Se utilizan if/elif/else para validar que los datos ingresados por el usuario (población, superficie, continente) cumplan con los criterios de formato y rango (ej. `if not poblacion.isdigit() or int(poblacion) <= 0:` en `agregar_pais`).

El flujo del programa se controla mediante condicionales (`if opcion == "1"; elif opcion == "2";` etc.) en la función `ejecutar_programa()`.

Se usan condiciones dentro de comprensiones de lista para generar subconjuntos de datos (ej. `[p for p in paises if min_pob <= p["poblacion"] <= max_pob]`).

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py •
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > guardar_paises
98
99 # Función para agregar país
100 def agregar_pais(paises):
101     nombre = input("Nombre del país: ").strip() # Solicita ingresar nombre sin espacios
102     poblacion = input("Población: ").strip() # Solicita ingresar población sin espacios
103     superficie = input(
104         "Superficie (km2): "
105     ).strip() # Solicita ingresar superficie sin espacios
106     continente_input = input(
107         "Continente: "
108     ).strip() # Solicita ingresar continente sin espacios
109
110     continente = encontrar_continente(
111         continente_input
112     ) # Convierte el continente ingresado en uno valido de CONTINENTES_VALIDOS[]
113
114     if not nombre.replace(
115         " ", ""
116     ).isalpha(): # Si no es una cadena de caracteres solo alfabéticos
117         print(
118             "❌ El nombre del país debe contener solo letras."
119         ) # Muestra mensaje de error
120         return # Vuelve al menú
121
122     if (
123         not poblacion.isdigit() or int(poblacion) <= 0
124     ): # Si no es un valor de solo dígitos numéricos positivo
125         print(
126             "❌ La población debe ser un número entero positivo."
127         ) # Muestra mensaje de error
128         return # Vuelve al menú
129
130     if not superficie.isdigit() or int(superficie) <= 0:
131         print(
132             "❌ La superficie debe ser un número entero positivo."
133         ) # Muestra mensaje de error
134         return # Vuelve al menú
135
136     if continente is None: # Si es un valor vacío
137         print("❌ Continente inválido.") # Muestra mensaje de error
138         print(
139             f"🌐 Continentes válidos: {' '.join(CONTINENTES_VALIDOS)}"
140         ) # Muestra los continentes válidos
141         return # Vuelve al menú
```


5. Ordenamientos (Procesamiento de Datos)

La función de ordenamiento utiliza el método `sorted()` de Python, aprovechando las funciones auxiliares para definir la clave de ordenamiento.

El ordenamiento permite organizar los elementos de una colección (la lista de países) según un criterio específico (nombre, población o superficie).

En la función `ordenar_paises()`, el usuario elige el campo. Se emplea el parámetro `key` con las funciones auxiliares (ej. `key=obtener_poblacion`) y el parámetro `reverse=reverse` para alternar entre orden **Ascendente (A)** y **Descendente (D)**.

```

202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > ...
279
280 # Función para ordenar países
281 def ordenar_paises(paises):
282     print("\n--- ORDENAR POR ---")
283     print("1. Nombre")
284     print("2. Población")
285     print("3. Superficie")
286     campo = input("Seleccione campo: ").strip()
287     orden = (
288         input("Ascendente (A) o Descendente (D): ").strip().upper()
289     ) # Solicita entrada para orden sin espacios y en mayúsculas
290
291     if campo not in ["1", "2", "3"] or orden not in [
292         "A",
293         "D",
294     ]: # Si las entradas son incorrectas
295         print("❌ Opción inválida.")
296         return # Vuelve al menú principal
297
298     reverse = (
299         orden == "D"
300     ) # Si el orden en D reverse toma el valor de true y el orden sera descendiente, si el "A" = false ascendiente
301     if campo == "1":
302         ordenados = sorted(
303             paises, key=obtener_nombre, reverse=reverse
304         ) # Ordena por nombre usando la función auxiliar obtener_nombre
305     elif campo == "2":
306         ordenados = sorted(
307             paises, key=obtener_poblacion, reverse=reverse
308         ) # Ordena por población usando la función auxiliar obtener_poblacion
309     elif campo == "3":
310         ordenados = sorted(
311             paises, key=obtener_superficie, reverse=reverse
312         ) # Ordena por superficie usando la función auxiliar obtener_superficie
313
314     print("\n 📋 Países ordenados:") # Muestra el resultado con el orden solicitado
315     for p in ordenados:
316         print(
317             f"{p['nombre']} - Población: {p['poblacion']} - Superficie: {p['superficie']} km²"
318         ) # Imprime cada país con su información formateada
319

```

6. Estadísticas Básicas (Análisis de Datos)

El sistema calcula estadísticas clave para resumir la información del conjunto de datos.

Son indicadores que resumen las propiedades de un conjunto de datos.

Se usan las funciones `max()` y `min()` de Python con el argumento `key` para encontrar el país con mayor y menor población.

El promedio de población y superficie se calcula usando la función `sum()` sobre los datos, dividido por el número total de países (`len(países)`).

Se utiliza un diccionario (`continentes`) para contar la cantidad de países por cada continente, demostrando el uso de diccionarios como contadores.

```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py X
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > ...
321 # Función para mostrar estadísticas
322 def mostrar_estadisticas(países):
323     if not países: # Sí no hay países
324         print("⚠ No hay países cargados.") # Informamos
325         return # Vuelve al menú
326
327     mayor = max(países, key=obtener_poblacion) # Obtiene el país con mayor población
328     menor = min(países, key=obtener_poblacion) # Obtiene el país con menor población
329     promedio_pob = sum(obtener_poblacion(p) for p in países) / len(
330         países
331     ) # Promedio de población
332     promedio_sup = sum(obtener_superficie(p) for p in países) / len(
333         países
334     ) # Promedio de superficie
335
336     continentes = {} # Diccionario de continentes
337     for p in países: # Recorremos países[]
338         cont = p["continente"] # Extrae el nombre del continente del .csv
339         continentes[cont] = (
340             continentes.get(cont, 0) + 1
341         ) # Crea el continente como elemento e inicializa en 1 pero si ya existe incrementa en 1
342
343     print("\n📊 Estadísticas:") # Muestra los resultados
344     print(f"País con mayor población: {mayor['nombre']} ({mayor['poblacion']})")
345     print(f"País con menor población: {menor['nombre']} ({menor['poblacion']})")
346     print(f"Promedio de población: {int(promedio_pob)}")
347     print(f"Promedio de superficie: {int(promedio_sup)} km²")
348     print("Cantidad de países por continente:")
349     for cont, cant in continentes.items():
350         print(f"    {cont}: {cant}")
351
```

7. Archivos CSV (Persistencia de Datos)

Se utiliza el módulo estándar csv para garantizar la persistencia de los datos.

CSV (Comma Separated Values) es un formato de archivo de texto plano que utiliza comas para separar valores y líneas nuevas para separar registros.

La función `cargar_paises()` utiliza `csv.DictReader` para leer el archivo `paises.csv` y convertir cada fila automáticamente en un diccionario.

La función `guardar_paises()` utiliza `csv.DictWriter` para escribir la lista de diccionarios de vuelta al archivo, asegurando que los datos agregados o modificados se conserven.

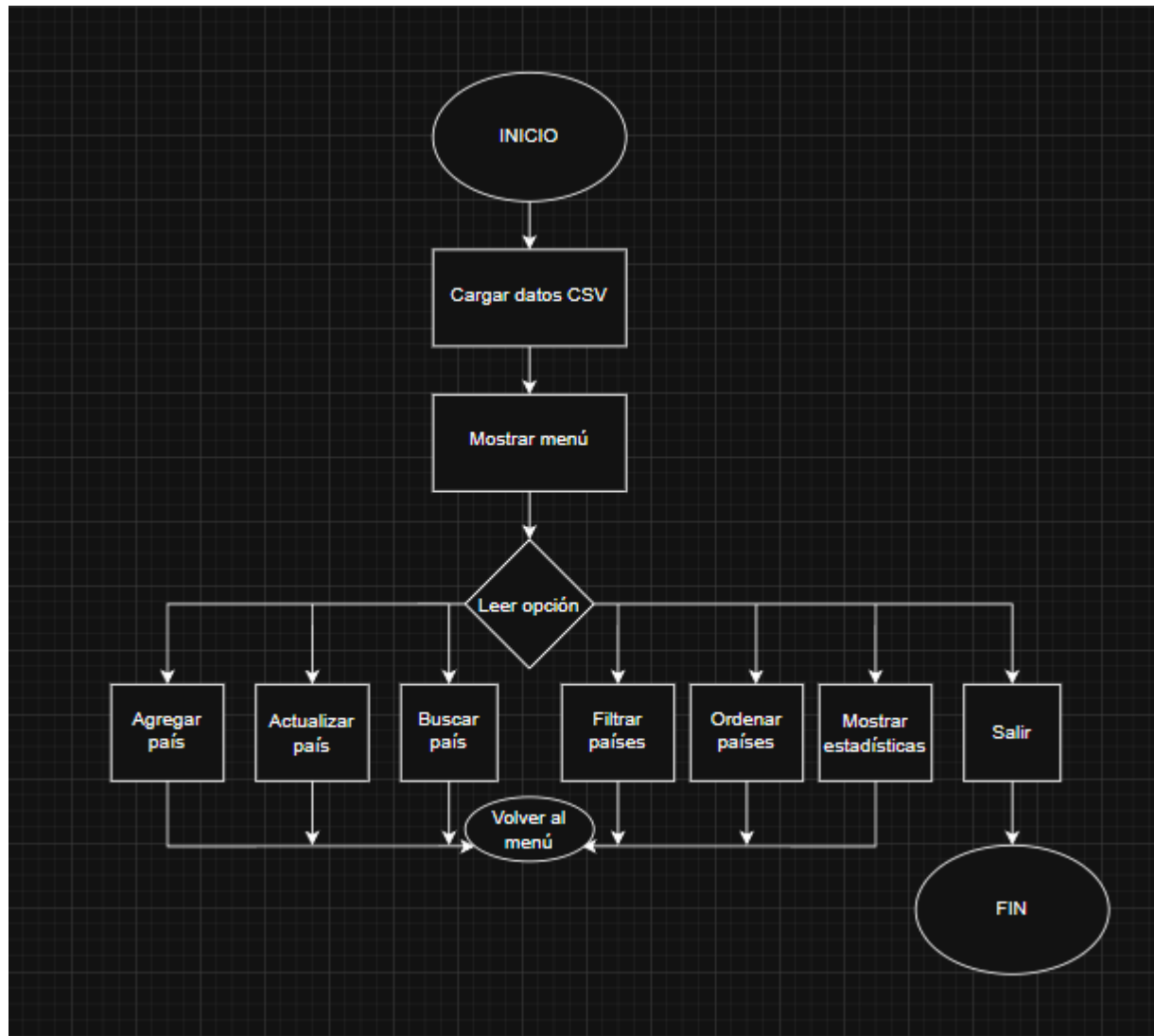
```
202509_P1_TPI_ESCALANTE_HERNANDEZ.py
202509_P1_TPI_ESCALANTE_HERNANDEZ.py > guardar_paises

38
39
40 # Función que carga la lista de paises creando un diccionario con los datos que encuentra en el .csv
41 # Cada fila del .csv sera un elemento del diccionario con sus respectivos key-values
42 def cargar_paises(nombre_archivo):
43     paises = [] # Lista vacía donde se cargan los paises que están en el .csv
44     archivo = open(nombre_archivo, newline="", encoding="utf-8")
45     lector = csv.DictReader(archivo) # Leemos el archivo .csv
46     # Recorremos el archivo y validamos la existencia y los tipos de datos
47     # Si los datos son correctos creamos un elemento para el pais con las keys nombre, población, superficie y continente y asignamos los valores del .csv
48     # Cada fila se convertirá en un elemento país dentro de paises[]
49     for fila in lector:
50         if (
51             "nombre" in fila
52             and "poblacion" in fila
53             and "superficie" in fila
54             and "continente" in fila
55             and fila["poblacion"].isdigit()
56             and fila["superficie"].isdigit()
57         ):
58             pais = {
59                 "nombre": fila["nombre"],
60                 "poblacion": int(fila["poblacion"]),
61                 "superficie": int(fila["superficie"]),
62                 "continente": fila["continente"],
63             }
64             paises.append(pais) # Agregamos el elemento pais a paises[]
65     archivo.close()
66     return paises # La función cargar_paises devuelve paises[]
67
68
69 # Función que recibe como parámetros el nombre del archivo y la lista, en este caso se usara para paises[]
70 def guardar_paises(nombre_archivo, paises):
71     archivo = open(nombre_archivo, mode="w", newline="", encoding="utf-8") # Abrimos el archivo en modo sobrescribir
72     campos = [
73         "nombre",
74         "poblacion",
75         "superficie",
76         "continente",
77     ] # Definimos los headings en una lista
78     escritor = csv.DictWriter(
79         archivo, fieldnames=campos
80     ) # creamos un escritor para paises.csv con los headings
81     escritor.writeheader() # Escribimos el encabezado
82     for pais in paises: # Por cada elemento pais en paises[]
83         escritor.writerow(pais) # Escribimos una fila
84     archivo.close() # Cerramos archivo
85
```

ibc SonarQube focus: overall code Ln 71, Col 19 (4 selected) Spaces: 4 UTF-8 CRLF Python

Diagrama de flujo

A continuación, se presenta el diagrama que define el flujo de operaciones principales del sistema. Este esquema visualiza cómo el programa se inicia, presenta el menú de opciones y gestiona la interacción con el usuario hasta su finalización.



Explicación del funcionamiento

El programa opera mediante un bucle principal gestionado por la función `ejecutar_programa()`, que se encarga de iniciar el sistema y mantener la interacción constante con el usuario a través del menú.

A continuación, se detalla el flujo de cada funcionalidad:

1. Inicio y Persistencia de Datos

El flujo comienza con dos acciones cruciales para la gestión de datos:

- **Carga Inicial (`cargar_paises`):** Al iniciar `ejecutar_programa()`, se llama a `cargar_paises("paises.csv")`. Esta función lee el archivo CSV, utilizando `csv.DictReader` para mapear automáticamente las cabeceras a las claves del diccionario. Cada fila se convierte en un diccionario, y todos se añaden a la lista `países`, la estructura de datos principal del sistema.
- **Persistencia (`guardar_paises`):** Después de cualquier operación que modifique la lista (como agregar o actualizar un país), se llama a `guardar_paises()`. Esta función utiliza `csv.DictWriter` para sobrescribir el archivo `paises.csv` con el estado actual de la lista, garantizando que los cambios sean permanentes.

2. Gestión de Datos (Opciones 1 y 2)

Opción 1: Agregar País (`agregar_pais`)

1. **Solicitud de Datos:** Se pide el nombre, población, superficie y continente del nuevo país.
2. **Validación Robusta:** Se implementan validaciones estrictas:
 - a. **Población y Superficie:** Se verifica que los valores sean numéricos y positivos (if not `poblacion.isdigit()` or `int(poblacion) <= 0`).
 - b. **Continente:** Se usa la función auxiliar `encontrar_continente()` que, tras normalizar la entrada (quitar acentos y minúsculas con `normalizar()`), compara con una lista de `CONTINENTES_VALIDOS`, asegurando la coherencia de datos.
3. **Registro y Guardado:** Si todos los datos son válidos, se crea un nuevo diccionario y se añade a la lista `países` con `.append()`. Finalmente, se llama a `guardar_paises()` para actualizar el CSV.

Opción 2: Actualizar País (actualizar_pais)

1. **Búsqueda Preliminar:** Se pide al usuario el nombre del país a modificar. La función itera sobre la lista de países para encontrar una coincidencia exacta.
2. **Modificación:** Si se encuentra el país, se solicitan los nuevos valores de población y superficie. Estos valores se validan de la misma manera que en la función de agregar.
3. **Actualización y Persistencia:** Se actualizan los valores del diccionario del país encontrado. Si la actualización es exitosa, se llama a `guardar_paises()`.

3. Consultas y Procesamiento (Opciones 3, 4 y 5)**Opción 3: Buscar País (buscar_pais)**

1. **Normalización de Búsqueda:** Se pide el término de búsqueda. Tanto el nombre del país en el diccionario como el término ingresado por el usuario se normalizan usando la función `normalizar()`. Esto garantiza que la búsqueda sea insensible a mayúsculas, minúsculas y acentos, mejorando la experiencia del usuario.
2. **Filtro:** Se recorre la lista `países` y se añade a una nueva lista de resultados (`países_encontrados`) todo aquel diccionario cuyo nombre normalizado contenga (coincidencia parcial) el término de búsqueda normalizado.

Opción 4: Filtrar Países (filtrar_paises)

Esta función ofrece un submenú para elegir el criterio de filtrado:

- **Por Continente:** Se pide el continente. Se utiliza la función `encontrar_continente()` para validar la entrada. Luego, mediante una comprensión de lista, se genera un subconjunto de países que coinciden con el continente validado.
- **Por Rango (Población/Superficie):** Se piden un valor mínimo y máximo. Las entradas se validan para asegurar que sean números y que el mínimo sea menor o igual al máximo. La función utiliza comprensiones de lista para incluir solo los países cuyo valor cae dentro del rango especificado.

Opción 5: Ordenar Países (ordenar_paises)

1. Criterio y Sentido: El usuario selecciona el criterio (Nombre, Población, Superficie) y el sentido (Ascendente/Descendente).
2. Uso de sorted() con Clave: Se aplica la función nativa sorted() de Python sobre la lista paises.
 - a. Se utiliza el parámetro key al que se le asigna una función auxiliar (ej. obtener_nombre, obtener_poblacion) para indicar por qué atributo del diccionario debe ordenarse.
 - b. Se usa el parámetro reverse=True si el usuario eligió orden Descendente.

Opción 6: Mostrar Estadísticas (mostrar_estadisticas)

Esta función realiza un análisis completo de los datos en memoria:

1. Máximos y Mínimos: Las funciones nativas max() y min() se utilizan con el argumento key=obtener_poblacion para encontrar directamente el país con la mayor y menor población.
2. Promedios: El promedio de población y superficie se calcula sumando el valor de todos los países (sum()) y dividiendo por el total de países (len(paises)).
3. Conteo por Categoría: Se inicializa un diccionario vacío (continentes = {}). Se itera sobre la lista paises y se utiliza el nombre del continente como clave, incrementando el valor del contador por cada país, demostrando el uso eficiente de diccionarios para el conteo de frecuencias.

Conclusión

El proyecto demostró la importancia de planificar la estructura de datos desde el inicio, ya que el uso de diccionarios facilitó enormemente la manipulación de los atributos de cada país. El trabajo en equipo permitió abordar las diferentes funcionalidades de forma distribuida, optimizando el tiempo de desarrollo e integrando los módulos de manera eficiente. También permitió fortalecer nuestras habilidades blandas, como la organización, planificación de tareas y mejorar nuestra comunicación para poder lograr el objetivo propuesto.

Como potencial mejora o línea futura, el proyecto es escalable. La sólida base modular permitiría, sin modificar la lógica interna de procesamiento, migrar la interfaz de consola a una interfaz gráfica (GUI) o integrar servicios de visualización de datos (como gráficos o mapas), expandiendo así su utilidad.

En síntesis, este TPI valida la adquisición de las habilidades necesarias para construir una aplicación funcional, robusta y bien diseñada en Python.

Referencias

- **Downey, A.** (2015). *Piensa en Python: Aprende a pensar como un informático* (2.ª ed., v. 2.4.0).
- **Python Software Foundation.** (s.f.). *The Python Standard Library (Howto/Sorting)*. Recuperado de <https://docs.python.org/3/howto/sorting.html>
- **W3Schools.** (s.f.). *Python sorted() Function*. Recuperado de https://www.w3schools.com/python/ref_func_sorted.asp