UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador

Gestión de Datos de Países en Python

Programación

Alumnos - Grupo

Agustín De Armas, Hugo Adrián Isaurralde.

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional

Docente Titular

Sebastián Bruselario

Docente Tutor

Francisco Quarñolo

11 de Noviembre de 2025

Tabla de Contenidos

Introducción

El presente Trabajo Práctico Integrador (TPI) se enfoca en la resolución de una problemática fundamental en el análisis de datos: la gestión eficiente y el procesamiento de información geográfica estructurada. Específicamente, aborda la necesidad de un sistema capaz de manejar un dataset de países, permitiendo la carga, modificación, consulta, filtrado y generación de indicadores estadísticos clave.
La complejidad creciente de los datos requiere herramientas de software que no solo almacenen la información, sino que también apliquen lógica de negocio para transformarla en conocimiento útil. Por ello, se desarrolló una aplicación de consola en el lenguaje Python que sirve como base para demostrar el uso avanzado de estructuras de datos fundamentales y la correcta implementación de la modularización.

El objetivo central de este desarrollo es afianzar los conceptos de Programación 1, creando un producto funcional que garantice la integridad de los datos a través de validaciones y el manejo de persistencia mediante archivos CSV.

Objetivos

Objetivo General

Desarrollar una aplicación de gestión de datos en Python que permita procesar un *dataset* de países, implementando un sistema modular y robusto, con el fin de afianzar el uso de listas, diccionarios, funciones y algoritmos de consulta.

Objetivos Específicos

- **Estructuras de Datos:** Utilizar correctamente una Lista de Diccionarios como estructura primaria para modelar la relación entre el conjunto de datos y cada registro individual (país).

- **Modularización:** Aplicar el principio de separación de responsabilidades, dividiendo el código en módulos (main.py, consultas.py, utilidades.py) para maximizar la legibilidad y el mantenimiento.

- **Persistencia:** Implementar funciones para la lectura y escritura de los datos en formato CSV, simulando un patrón de data access (DAO) y asegurando que las modificaciones realizadas por el usuario se conserven.

- **Lógica de Negocio:** Desarrollar algoritmos de filtrado, búsqueda, ordenamiento y cálculo estadístico para explotar la información contenida en el *dataset*.

Marco Teórico

1. Estructuras de Datos: Lista de Diccionarios

El proyecto modela el dominio de la información geográfica utilizando una estructura híbrida de **Lista de Diccionarios**. La **Lista** es empleada para contener el conjunto total de países, proporcionando un acceso secuencial e iterable. Cada elemento dentro de esta lista es un **Diccionario**, el cual representa a un país con sus atributos y valores asociados.

La estructura clave-valor de cada país es la siguiente:

Pais = {'nombre': str, 'poblacion': int, 'superficie': int, 'continente': str}

Esta elección permite un acceso semántico a los datos (pais['poblacion']) y facilita la manipulación de los registros de manera eficiente para las funciones de consulta y modificación.

2. Modularización y Abstracción

El sistema se implementó siguiendo un esquema de **código modularizado**, donde cada unidad funcional (función) posee una única responsabilidad. La aplicación de este concepto se materializa en la división del código en tres archivos principales:

- **main.py**: Interfaz de usuario. Contiene el menú y la lógica del *loop* principal.
- **consultas.py**: Lógica de Negocio. Concentra las funciones que manipulan la lista de países (filtros, ordenamientos, estadísticas).
- **utilidades.py**: Funciones de Apoyo. Contiene validaciones y herramientas genéricas (normalizar_str, _input_int).

3. Persistencia de Datos (CSV)

La persistencia se maneja mediante archivos **CSV**. El módulo consultas.py contiene las funciones cargar_paises_desde_csv y guardar_paises_en_csv, responsables de:

1. **Lectura:** Traducir cada línea del archivo CSV a un objeto **Diccionario** en Python, realizando las conversiones de tipo necesarias (*string* a *int* para Población y Superficie).
2. **Escritura:** Convertir la Lista de Diccionarios al formato CSV al finalizar la ejecución del programa, garantizando que los datos agregados o modificados se almacenen permanentemente.

Diseño y Arquitectura del Proyecto

1. Arquitectura de Módulos

La arquitectura del proyecto se rige por la separación de capas lógicas, como se detalla a continuación:

Modulo main.py

Responsabilidad Principal: Control de Flujo, Menú principal, Inicio y Cierre del sistema.

Interacción: Llama a las funciones de consultas.py en respuesta a la elección del usuario.

Modulo consultas.py

Responsabilidad Principal: Gestión de datos: CRUD, Filtrado, Ordenamiento y Estadísticas.

Interacción: Depende de utilidades.py para la normalización de strings y validaciones.

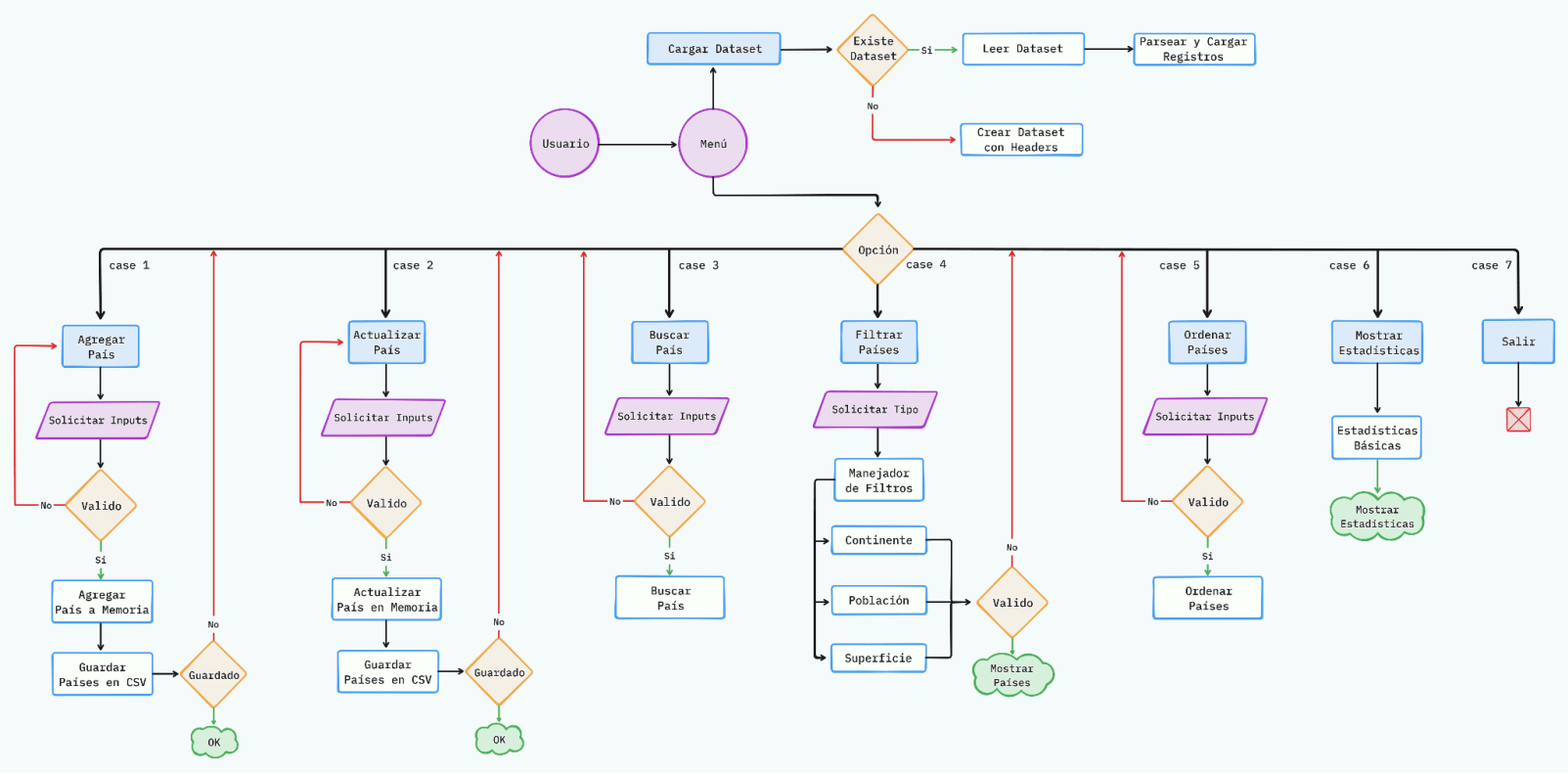
Modulo utilidades.py

Responsabilidad Principal: Funciones de Helper genéricas: validación de enteros, normalización de textos, gestión de rutas.

Interacción: Es importado por main.py y consultas.py.

2. Diagrama de Arquitectura

URL DE DIAGRAMA



Implementación de Funcionalidades

Las funcionalidades del programa están alineadas con las opciones presentadas en el menú principal. A continuación, se describen las implementaciones más relevantes, contenidas en el módulo consultas.py.

1. Gestión de Registros (Opciones 1 y 2)

Agregar País (agregar_pais**):** Implementa un proceso de ingreso guiado, solicitando los cuatro atributos del país. Se utiliza la función **_input_int** del módulo de utilidades para asegurar la correcta conversión de Población y Superficie a números enteros positivos antes de añadir el diccionario a la lista principal.

Actualizar País (actualizar_pais**):** Requiere que el usuario ingrese el nombre exacto del país. Una vez encontrado, se permite la modificación selectiva de los campos numéricos (población, superficie).

2. Consultas y Filtros (Opciones 3 a 6)

Búsqueda por Nombre (buscar_paises**):** La búsqueda se implementa de manera parcial y no sensible a mayúsculas/minúsculas. Para esto, se aplica la función **normalizar_str** (presente en utilidades.py) tanto al término de búsqueda ingresado por el usuario como a los nombres de los países en el *dataset* antes de realizar la comparación.

Filtro por Criterio Simple (filtrar_por_continente**):** Recorre la lista de países y construye una nueva lista de resultados que cumple estrictamente con el continente ingresado.

Filtro por Rango (filtrar_por_rango_poblacion**,** filtrar_por_rango_superficie**):** Estas funciones solicitan un valor mínimo y un valor máximo. Aplican la condición:
Mínimo < Valor Atributo < Máximo para devolver la sub-lista filtrada. Se utiliza el input validado (_input_int) para garantizar que los límites del rango sean numéricos y válidos.

3. Procesamiento de Datos (Opciones 7 y 8)

Ordenamiento (ordenar_paises**):** Es una de las funciones más relevantes. Permite ordenar la lista de países por tres criterios (nombre, población, superficie) y en dos direcciones (ascendente o descendente). La implementación utiliza la función nativa sorted() de Python, aprovechando el parámetro key para indicar el campo de ordenamiento. Para la clave 'nombre', se utiliza una función helper que convierte el nombre a minúsculas, garantizando un ordenamiento alfabético uniforme.

Estadísticas Básicas (estadisticas_basicas**):** Esta función devuelve un diccionario con las cinco métricas solicitadas:

1. Promedio de Población.
2. Promedio de Superficie.
3. País con Mayor Población.
4. País con Menor Población.
5. Cantidad de Países por Continente (calculado mediante acumulación en un diccionario).

Validaciones y Manejo de Errores

Se implementaron medidas de control de calidad, tal como lo exige la consigna, para asegurar la estabilidad del sistema y ofrecer una experiencia de usuario clara.

Validación de Entradas de Usuario: La función _input_int en utilidades.py obliga al usuario a ingresar un valor numérico entero positivo en el prompt. Se utiliza en todas las opciones que requieren datos numéricos.

Control de Formato del CSV: Se incluye lógica para verificar el formato de los distintos campos, previniendo fallos al trabajar con el *dataset*.

Manejo de Resultados Vacíos: Todas las funciones de búsqueda y filtrado (buscar_paises, filtrar_por_continente, etc.) retornan una lista vacía si no encuentran coincidencias acompañado de un error en formato *string*. El módulo main.py detecta esta condición y muestra un mensaje informativo al usuario: "*No se encontraron países que cumplan con el criterio solicitado.*"

Mensajes de Éxito y Error: Al realizar operaciones de mutación (Agregar/Actualizar), se utilizan feedback explícitos como "*País agregado exitosamente*" o "*Error: El país no se encontró en el dataset.*"

Conclusiones

El desarrollo del Trabajo Práctico Integrador nos permitió aplicar y consolidar los conceptos clave de la asignatura. El proyecto demostró con éxito que una Lista de Diccionarios es una estructura de datos idónea para la representación de registros heterogéneos y ha servido como base sólida para la aplicación de lógica de negocio compleja.

Los principales aprendizajes se centraron en:

- El dominio de la modularización, permitiendo un código desacoplado y fácil de mantener.
- El uso de la función sorted() con el parámetro key para la implementación eficiente de algoritmos de ordenamiento con múltiples criterios.
- La importancia del manejo de validaciones para crear una aplicación robusta que maneje correctamente la interacción con archivos CSV y las entradas del usuario.

Las principales dificultades se presentaron en la implementación de la lógica de ordenamiento con criterios múltiples (campo y dirección) y en la garantía de la integridad de los datos al momento de leer y escribir el CSV. Estos desafíos fueron superados mediante el uso de funciones de apoyo (utilidades.py) y el diseño de funciones claras con responsabilidad única.

En resumen, el sistema cumple con todos los requisitos funcionales de la consigna, sirviendo como una prueba concreta de la capacidad para diseñar y desarrollar una aplicación completa utilizando los paradigmas de programación procedural y estructurada.

Anexo

- **Google Drive:** TPI-Programacion Drive
- **GitHub:** TPI-Programacion GitHub
- **Youtube:** TPI-Programacion Youtube

```
--------------------------
> Ingrese una opcion (1-7): 1
=== AGREGAR PAIS ===
> Ingrese el nombre del pais: colombia
> Ingrese la poblacion: 123123
> Ingrese la superficie: 1231231
> Ingrese el continente: america
CSV guardado correctamente.

> Ingrese una opcion (1-7): 2
=== ACTUALIZAR PAIS ===
> Ingrese el nombre del pais: colombia
datos actuales -> poblacion: 123123 | superficie: 1231231
> Nueva poblacion: 124124
> Nueva superficie: 1241241
CSV guardado correctamente.

> Ingrese una opcion (1-7):
```

```
> Ingrese una opcion (1-7): 4
=== ELEGIR TIPO DE FILTRO ===
1. Continente
2. Rango de Población
3. Rango de Superficie
> Ingrese la opción de filtro (1-3): 2
=== FILTRAR POR RANGO DE POBLACION ===
> Ingrese el minimo de poblacion: 124124
> Ingrese el maximo de poblacion: 83150300

Se encontraron 3 pais(es) para el filtro de población:
- Argentina | Población: 45376763 | Superficie: 2780400 | Continente: America
- Alemania | Población: 83149300 | Superficie: 357022 | Continente: Europa
- Colombia | Población: 124124 | Superficie: 1241241 | Continente: America
> Ingrese una opcion (1-7):
```

```
> Ingrese una opcion (1-7): 6
=== ESTADISTICAS BASICAS ===
Total de paises: 5
Promedio de poblacion: 93688724.8
Promedio de superficie: 2654481.0
País con mayor población: Brasil - 213993437
País con menor población: Colombia - 124124
Cantidad de paises por continente:
  America : 3
  Asia : 1
  Europa : 1
```