

Trabajo Práctico Integrador: Programación I

Sistema de Gestión de Información sobre Países

01

Alumnos

Cristian Alejandro Tapia (c13)

Daniela Velazquez (c12)

02

Coordinador

Alberto Cortez

03

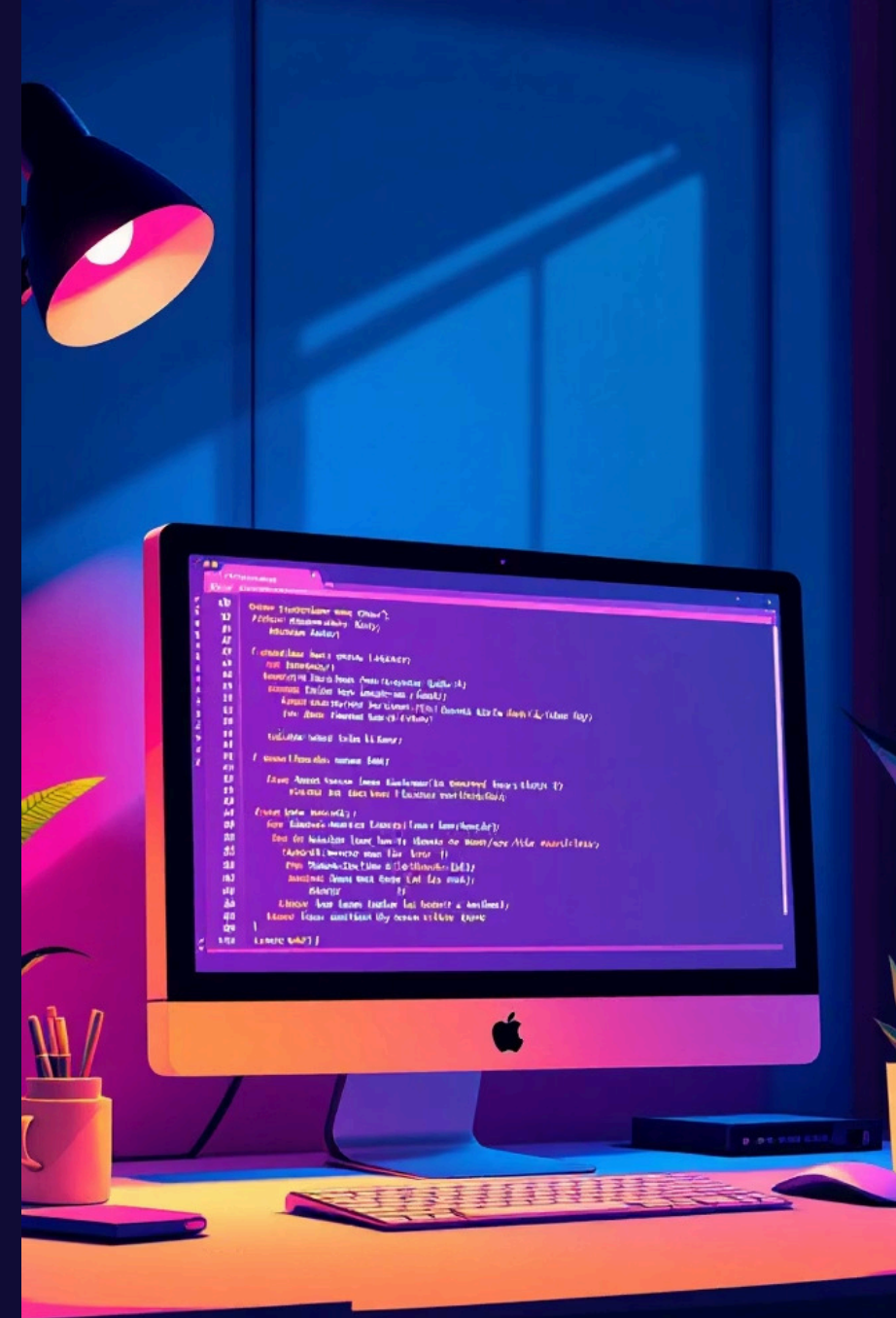
Profesores

Cinthia Rigoni, Sebastian Bruselario, Ariel Enferrel

04

Tutores

Franco Gonzalez, Miguel Barrera



Índice del Proyecto

1 Introducción

Objetivos y alcance del proyecto

2 Marco Teórico

Conceptos fundamentales de programación

3 Caso Práctico

Estructura de diseño modular

4 Pruebas de Concepto

Validación del sistema

1 Diagrama de Flujo

Visualización del proceso

2 Resultados Obtenidos

Funcionalidades implementadas

3 Conclusión

Aprendizajes y reflexiones

4 Bibliografía

Referencias y repositorio



Introducción: Objetivos del Proyecto

El objetivo principal de este trabajo fue desarrollar una aplicación de consola en Python que permita gestionar información sobre países de manera eficiente y estructurada.

Estructuras de Datos

Afianzar el uso de listas y diccionarios para organizar y manipular información compleja de manera eficiente.

Modularización

Implementar funciones con responsabilidades únicas que faciliten el mantenimiento y la escalabilidad del código.

Técnicas Avanzadas

Aplicar filtrado, ordenamiento y cálculo de estadísticas básicas sobre el dataset de países.

El sistema debe ser capaz de leer datos desde un archivo CSV, realizar consultas complejas y generar indicadores clave a partir del dataset, proporcionando una interfaz intuitiva para la gestión de información geográfica y demográfica.

Marco Teórico: Estructuras de Datos

Diccionarios en Python

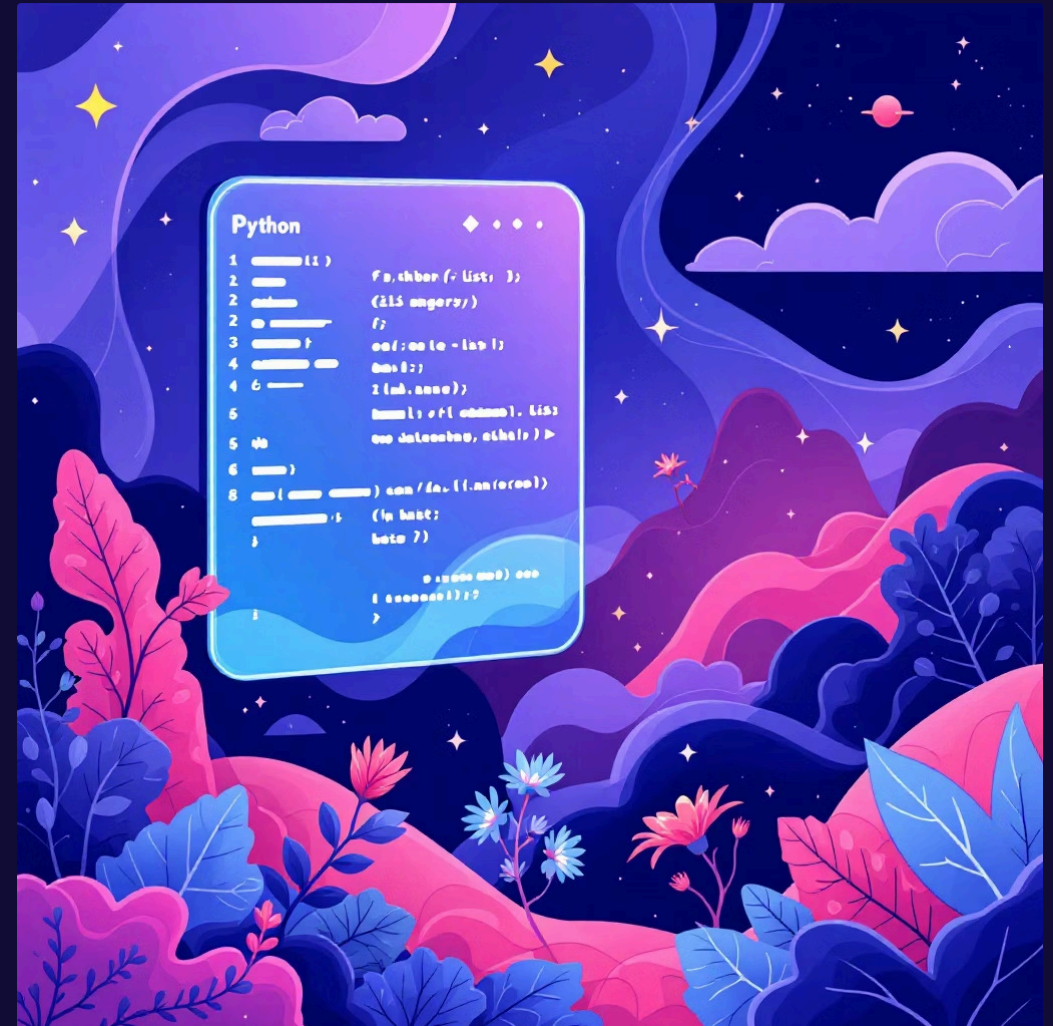
Un **diccionario** en Python es una colección desordenada, mutable e indexada por claves, que se utiliza para almacenar datos en pares de clave: valor.

- ❏ **Aplicación en el TPI:** Cada país se modeló como un diccionario. Esto permitió acceder a los atributos del país de forma clara y semántica (ej. `pais["POBLACION"]`), reflejando la estructura del CSV (nombre, poblacion, superficie, continente).

Listas en Python

Las **listas** son colecciones ordenadas y mutables de elementos. En este proyecto, la lista fue la estructura central del dataset.

- ❏ **Aplicación en el TPI:** La lista principal (`países`) almacenó el conjunto total de diccionarios, representando la colección completa de países. Esto facilitó la iteración sobre el dataset para aplicar búsquedas, filtros y ordenamientos.



Modularización y Técnicas de Ordenamiento

Modularización y Funciones

La modularización es la práctica de dividir el código en funciones con responsabilidades únicas.

Aplicación en el TPI: Se cumplió con el principio de "una función = una responsabilidad". Por ejemplo, las funciones de Persistencia (ObtenerPaises, PersistirCsv) solo manejan el archivo, mientras que las de Lógica de Negocio (AgregarPais, ActualizarDatos) sólo aplican las reglas del sistema.

Ordenamiento con sorted()

Para ordenar los países por diversos criterios (Nombre, Población, Superficie) se utilizó la función nativa sorted().

Función sorted() con key: Esta función devuelve una nueva lista ordenada a partir de una iterable. El uso crucial fue el parámetro key, que permite especificar una función para determinar qué valor de cada elemento debe usarse como criterio de ordenación (por ejemplo, el valor de la clave "POBLACION" del diccionario). El argumento reverse=True/False permitió gestionar el orden ascendente o descendente.

Estadísticas y Manejo de Archivos CSV



Funciones min() y max()

Estas funciones son utilizadas para encontrar el elemento más pequeño o más grande de un iterable.

Aplicación en el TPI: Se utilizaron con el parámetro key para encontrar eficientemente el país completo (el diccionario) con la mayor y menor población.



Agrupación con setdefault()

Para calcular indicadores de agrupación, como la cantidad de países por continente, se utilizó la función setdefault() de los diccionarios.

Función setdefault(): Permite obtener el valor de una clave, y si la clave no existe, la inserta con un valor por defecto (en este caso, una nueva lista vacía) antes de devolver dicho valor. Esto es fundamental para construir el diccionario de agrupación: {continente: [país1, país2, ...]}.



Archivos CSV

El acceso a los datos se realiza a través de un archivo en formato **CSV** (Comma Separated Values).

Aplicación en el TPI: El módulo de persistencia (ObtenerPaíses y PersistirCsv) se encarga de: 1) Leer el archivo y convertir cada fila en un diccionario. 2) Reescribir la lista completa de diccionarios al archivo cada vez que se produce una modificación.

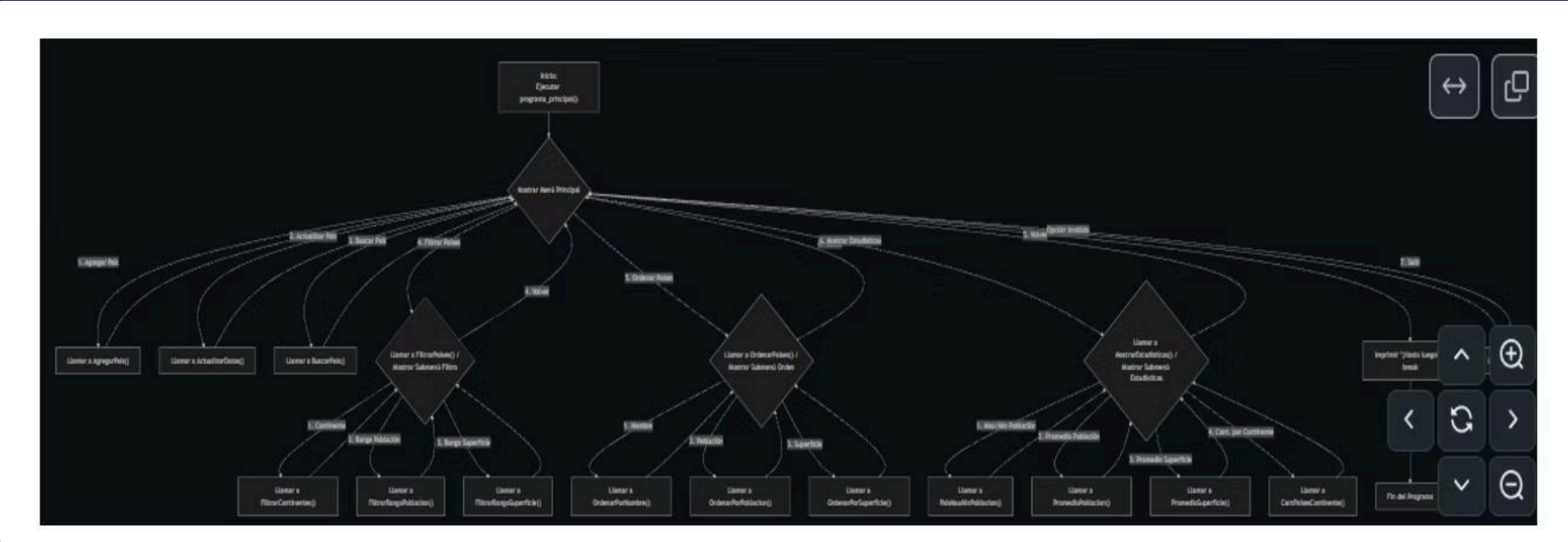
Caso Práctico: Arquitectura de Tres Capas

El diseño implementado sigue una arquitectura simple de tres capas, que separa el flujo de control, la lógica del negocio y la interacción con los datos.

| Capa | Funciones | Responsabilidad Principal |
|-----------------------------|--|---|
| Programa principal | programa_principal() | Actúa como el motor del sistema, presentando el menú y delegando la ejecución. |
| Lógica de Negocio | AgregarPais(), ActualizarDatos(), BuscarPais(), OrdenarPaises() | Implementa las reglas del sistema, coordinando las Utilidades y la Persistencia. |
| Capas de datos y Utilidades | ObtenerPaises(), PedirNombre(), PedirCantidad() | Maneja la lectura/escritura del CSV y garantiza la calidad (validación) de los datos ingresados por el usuario. |

Diagrama de Flujo del Sistema

El siguiente diagrama visualiza la modularización y el flujo de operaciones principales del sistema, mostrando cómo interactúan las diferentes funciones y capas de la aplicación.



| 1 | 2 |
|--|--|
| Inicio Ejecutar programa_principal() | Menú Principal Mostrar opciones al usuario |
| 3 | 4 |
| Operaciones Agregar, Actualizar, Buscar, Filtrar | Persistencia Guardar cambios en CSV |

Estrategia de Validaciones y Manejo de Errores

El código incluye un robusto sistema de validación para garantizar la integridad de los datos:

- Control de Errores en CSV:** La función `ObtenerPaíses()` implementa bloques `try-except` para capturar posibles errores de formato (ej. valores no numéricos en Población o Superficie), evitando que el programa se detenga al leer datos corruptos.
- Validación de Entradas del Usuario:** Funciones como `PedirCantidad()` utilizan bucles `while True` para asegurar que el usuario ingrese únicamente números enteros positivos. Las funciones de validación también impiden campos vacíos.
- Manejo de Búsquedas:** Se implementaron mensajes claros de error o éxito para evitar fallos al ingresar filtros inválidos o búsquedas sin resultados, mejorando la experiencia del usuario.



Resultados Obtenidos: Funcionalidades Completas

El sistema cumple con la totalidad de las funcionalidades mínimas requeridas, proporcionando una solución integral para la gestión de información sobre países.

1

Gestión de Datos

- Agregar un país con validación de campos vacíos
- Actualizar los datos de Población y Superficie de un país existente

1

Ordenamientos

- Ordenar la lista de países por Nombre
- Ordenar por Población
- Ordenar por Superficie
- Orden ascendente o descendente

2

Consultas

- Buscar un país por nombre (coincidencia parcial o exacta)

2

Estadísticas

- Mostrar país con mayor y menor población
- Promedio de población y superficie
- Cantidad de países por continente

3

Filtros

- Filtrar por Continente
- Rango de Población
- Rango de Superficie

Conclusión y Referencias

Aprendizajes Clave

El desarrollo de este TPI nos permitió afianzar y aplicar conceptos teóricos clave de programación en un proyecto real y funcional.



Modularización

La división del código en funciones de responsabilidad única simplificó tanto el desarrollo como el proceso de prueba y corrección de errores. El módulo de Validaciones permitió aislar y testear la lógica de control de entradas de manera independiente.



Estructuras de Datos

El uso combinado de la Lista de Dicionarios fue clave para modelar un dataset relacional simple, demostrando la versatilidad de estas estructuras para trabajar con datos estructurados en Python.



Trabajo en Equipo

La comunicación constante y la asignación clara de tareas permitió un desarrollo eficiente y organizado, facilitado por el uso de Git para la gestión de versiones y la integración del trabajo.

Referencias Bibliográficas

- Fuente de función sorted(): <https://docs.python.org/3/howto/sorting.html#ascending-and-descending>
- Fuente de funciones Min() Max(): <https://realpython.com/python-min-and-max>
- Fuente de setdefault(): https://www.w3schools.com/python/ref_dictionary_setdefault.asp

Repositorio GitHub

<https://github.com/CTapia10/TPI-Programacion/tree/main>

Muchas gracias por su lectura.