UNIVERSIDAD TECNOLOGICA NACIONAL

Tecnicatura Universitaria en Programación

Universidad Tecnológica Nacional

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

Programación I

Docente Titular

Ariel Enferrel

Docente Tutora

Martina Zabala

Alumno:

Fernando Picco (C10)

01 de diciembre de 2025

Índice

Trabajo Integrador – Programación I

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

1. Introducción

El presente trabajo práctico integrador tiene como objetivo desarrollar una aplicación en Python que permita gestionar información de un conjunto de países a partir de un archivo CSV. A lo largo del desarrollo se busca:

- Aplicar de manera integrada los conceptos vistos en la materia Programación I: listas, diccionarios, funciones, estructuras condicionales y repetitivas.
- Implementar filtros, búsquedas y ordenamientos sobre un dataset realista, practicando la comparación de cadenas y números.
- Calcular estadísticas básicas (totales, promedios, máximos y mínimos) a partir de los datos de población y superficie de cada país.
- Trabajar con archivos CSV utilizando el módulo estándar csv, incorporando validaciones para evitar errores de formato y de ingreso de datos por parte de la persona usuaria.
- Diseñar un programa modular, donde cada funcionalidad esté encapsulada en una función, facilitando la lectura, el mantenimiento y la futura extensión del código.
- Acostumbrarme a una forma de trabajo más cercana a un proyecto real: planificación previa, desarrollo incremental, pruebas frecuentes y registro del avance en un repositorio de control de versiones (GitHub).

2. Marco Teórico

2.1. Concepto de listas

Las listas en Python son una estructura de datos que permite almacenar varios elementos en una sola variable. Estos elementos pueden ser de distintos tipos (números, cadenas, booleanos, otras listas, etc.) y se encuentran ordenados y accesibles mediante índices.

Además, las listas son mutables, lo que significa que sus valores pueden modificarse, agregarse o eliminarse después de ser creadas (El libro de Python, s.f.).

En este trabajo práctico, las listas se utilizan para almacenar todos los registros de países obtenidos desde el archivo CSV. Cada país se guarda como un diccionario, y la lista completa funciona como un "catálogo" de países.

Ejemplo del código (Python):

```
paises = []   # Lista principal donde se guardarán los países

paises.append({

   "nombre": fila["nombre"],

   "poblacion": poblacion,

   "superficie": superficie,

   "continente": fila["continente"]

})
```

Aquí, cada append() agrega un país nuevo a la lista.

2.2. Concepto de Diccionarios

Los diccionarios en Python son una estructura de datos que permite almacenar su contenido en forma de llave y valor.

Un diccionario en Python es una colección de elementos, donde cada uno tiene una llave key y un valor value. Los diccionarios se pueden crear con paréntesis {} separando con una coma cada par key: value. En el siguiente ejemplo tenemos tres keys que son el nombre, la edad y el documento.

En este trabajo práctico, cada país se representa como un diccionario, con la siguiente estructura:

- "nombre"
- "poblacion"
- "superficie"
- "continente"

Esto facilita acceder a los datos durante búsquedas, filtros y ordenamientos.

Ejemplo del código (Python):

```
pais = {
   "nombre": fila["nombre"],
   "poblacion": poblacion,
   "superficie": superficie,
   "continente": fila["continente"]
}
```

Los diccionarios permiten escribir expresiones que permiten dar claridad y flexibilidad al programa.

Ejemplo del código (Python):

```
pais["nombre"]
pais["superficie"]
```

2.3. Concepto de funciones

Una función es un bloque de código que realiza una tarea específica. Al usar funciones es posible dividir un problema en partes pequeñas y manejables. Se compone de:

- Entrada (argumentos): datos que recibe la función.
- Proceso: instrucciones que realiza.
- Salida (retorno): resultado que devuelve.

Imagen 1. Composición y ejemplo de función

```
def nombre_funcion(argumentos):
    código
    return retorno
```

```
def calcular_tabla_del_6():
    for x in range(11):
        print(f"6 * {x} = {6*x}")

calcular_tabla_del_6()
```

Fuente: Elaboración propia (2025)

Son muy útiles ya que permiten reutilizar una porción de código tantas veces como sea necesario (UTN, 2025).

En este trabajo práctico, todo el programa está modularizado mediante funciones:

- obtener_datos_paises()
- buscar_pais_por_nombre()
- filtrar_por_continente()
- ordenar_paises()
- mostrar_estadisticas()
- mostrar_menu()

Esto se alinea con la consigna, que exige un diseño modular y legible.

Ejemplo del código (Python):

```
def mostrar_pais(pais):
  print(
    f"{pais['nombre']:<12} | "
    f"Población: {pais['poblacion']:>12.0f} | "
    f"Superficie: {pais['superficie']:>10.0f} | "
    f"Continente: {pais['continente']}"
  )
```

Cada función cumple un rol específico, sin mezclar responsabilidades.

2.4. Concepto de Estructuras condicionales

Las estructuras condicionales en programación (como if, elif y else en Python) permiten que un programa tome decisiones basadas en una condición que puede ser verdadera o falsa (DataScientest, s.f.).

Sirven para:

- ejecutar cierto bloque de código solo si se cumple una condición,
- evaluar múltiples alternativas, y
- definir un comportamiento diferente según los datos proporcionados por el usuario o el estado del programa.

En otras palabras, permiten que un programa no sea lineal, sino que reaccione a distintas situaciones, igual que en la vida real (“si llueve, llevo paraguas; si no llueve, no lo llevo”).

En este trabajo práctico, en el desarrollo del menú principal de la aplicación se utilizó la estructura match_case, incorporada en Python 3.10 (El libro de Python, s.f.). Esta estructura permite escribir decisiones múltiples de forma más ordenada, clara y legible cuando existen muchas ramas posibles, como sucede con las ocho opciones del menú. El match evalúa el valor ingresado por el usuario y ejecuta el bloque correspondiente según el caso seleccionado:

Ejemplo del código (Python):

```
opcion = input("Ingrese opción: ").strip()
    match opcion:
        case '1':
            buscar_nombre_paises()
        case '2':
            filtrar_por_continente()
        case '3':
            filtrar_por_rango_poblacion()
        case '4':
            filtrar_por_rango_superficie()
        case '5':
            ordenar_paises()
        case '6':
            mostrar_estadisticas()
        case '7':
            ver_todos_los_registros()
```

```
        case '8':
            print("¡Gracias por utilizar nuestra aplicación!¡Hasta pronto!")
            break
        case _:
            print("La opción seleccionada no es válida para nuestra aplicación")
```

La consigna del TPI no exige específicamente el uso de match, por lo que una solución basada en condicionales tradicionales también habría sido totalmente válida.

Un equivalente usando if sería:

Ejemplo del código (Python):

```
opcion = input("Ingrese opción: ").strip()
if opcion == "1":
    buscar_nombre_paises()
elif opcion == "2":
    filtrar_por_continente()
elif opcion == "3":
    filtrar_por_rango_poblacion()
elif opcion == "4":
    filtrar_por_rango_superficie()
elif opcion == "5":
    ordenar_paises()
elif opcion == "6":
    mostrar_estadisticas()
elif opcion == "7":
```

```
    ver_todos_los_registros()

elif opcion == "8":

    print("¡Gracias por utilizar nuestra aplicación! ¡Hasta pronto!")

    break

else:

    print("La opción seleccionada no es válida para nuestra aplicación")
```

2.5. Concepto de Ordenamientos

El ordenamiento de datos consiste en reorganizar los elementos de una colección —una lista, arreglo o conjunto de registros— según un criterio definido (alfabético, numérico, por atributo, etc.). Los algoritmos de ordenamiento son fundamentales en programación y estructura de datos porque permiten presentar los datos de manera ordenada, mejorar la legibilidad, optimizar búsquedas y facilitar análisis posteriores (GeeKsforGeeks, 2025).

Según la naturaleza de los datos y los requerimientos (cantidad, tipo de búsqueda, frecuencia de consultas), pueden utilizarse distintos métodos: algunos comparan los elementos directamente (comparison-based), otros trabajan por claves o por distribución, pueden ser “in-place” (modificando la colección original) o requerir espacio adicional (GeeKsforGeeks, 2025).

En este trabajo práctico, la opción “Ordenar países” implementa un mecanismo de ordenamiento que permite al usuario reordenar la lista de países por nombre, población o superficie, en orden ascendente o descendente. Esto aplica directamente los conceptos de “sorting” definidos en la teoría, procurando presentar los datos de forma más clara y manejable.

El uso de ordenamientos tiene estas ventajas en este contexto:

- Mejora la visualización de los datos (tabla ordenada).
- Facilita la comparación entre países.
- Permite al usuario explorar distintos criterios sin necesidad de recargar los datos.
- Facilita la generación de estadísticas o reportes sobre los datos ordenados.

En Python, para implementar este ordenamiento se utilizó la función integrada sorted(), acompañada de una clave (key) que indica la variable por la que se ordenará (nombre — casefold para evitar problemas de mayúsculas/minúsculas —, población o superficie). Para nombre, se define un orden lexicográfico; para población y superficie, un orden numérico. Esta implementación responde a las definiciones teóricas de ordenamiento por clave, ascendiendo o descendiendo según elección del usuario.

Ejemplo del código (Python):

```
# Ordenar según criterio elegido
  if campo == "nombre":
      paises_ordenados = sorted(paises, key=lambda p:p["nombre"].casefold(), reverse=descendente)
  else: # población y suuperficie
      paises_ordenados = sorted(paises, key=lambda p:p[campo], reverse=descendente)
```

2.6. Estadísticas Básicas

La estadística es la ciencia encargada de recopilar, organizar, analizar e interpretar datos para obtener conclusiones válidas sobre un conjunto de observaciones. En otras palabras, transformar datos “crudos” en conocimiento útil (DATARMONY, 2024).

Dentro de sus ramas, la llamada estadística descriptiva se centra en resumir los datos mediante medidas de tendencia central (por ejemplo, media, mediana, moda), medidas de dispersión (rango, varianza, desviación estándar), y representaciones tabulares o gráficas (DATARMONY, 2024).

Estas técnicas permiten simplificar conjuntos de datos (potencialmente voluminosos) en información clara, comprensible y utilizable, facilitando comparaciones, análisis de patrones, y toma de decisiones basadas en datos.

En el contexto de este trabajo práctico, aunque el script actual se focaliza en lectura, filtrado y ordenamiento de datos de países, la estructura implementada (una lista de diccionarios con atributos como población y superficie) permite con relativa facilidad

extender el programa para realizar análisis estadísticos. Por ejemplo, se podría calcular la población promedio de todos los países, identificar cuáles superan ese promedio, determinar el rango o mediana de superficie, o generar reportes resumidos de los datos.

Así, el programa no solo cumple con los requisitos de manejo de estructuras, funciones y ordenamientos, sino que también constituye una base funcional desde donde incorporar estadísticas básicas.

2.7. Archivos CSV

Un archivo CSV (Valores Separados por Comas) es un archivo de texto plano que almacena datos tabulares: cada línea representa una fila, y los valores de cada fila están separados por un delimitador, habitualmente una coma (,), aunque en algunos casos puede usarse punto y coma u otro carácter .

En la mayoría de los CSV la primera línea actúa como cabecera, definiendo los nombres de las columnas, lo que permite identificar los valores de cada campo de forma más clara.

Debido a su simplicidad, portabilidad y compatibilidad con muchas aplicaciones (hojas de cálculo, bases de datos, scripts, etc.), los archivos CSV son ampliamente utilizados para almacenar, intercambiar o procesar datos tabulares de forma sencilla.

Uso del módulo csv de Paython en la aplicación

Para trabajar con archivos CSV desde Python, el módulo estándar csv ofrece funcionalidades de lectura y escritura que facilitan su manejo sin necesidad de parsear manualmente los textos.

En el trabajo práctico se utilizó la función csv.DictReader() de este módulo para:

- abrir el archivo dataset_paises.csv,
- leer cada línea como un diccionario (donde las claves son los nombres de las columnas — definidas en la cabecera del CSV),
- convertir los valores de las columnas "poblacion" y "superficie" de str a float,
- construir una lista de diccionarios que representa todos los países.

Este enfoque permite trabajar en memoria con estructuras de datos propias de Python (listas y diccionarios), lo que facilita la implementación de búsquedas, filtrados, ordenamientos y estadísticas, sin depender del formato crudo del archivo.

3. Diseño de Caso Práctico

3.1. Analizar el enunciado

A partir del enunciado del TPI, lo primero fue identificar:

- Dominio del problema: un dataset de países con nombre, población, superficie y continente.
- Funcionalidades mínimas:
  - Búsqueda de un país por nombre (coincidencia parcial o exacta).
  - Filtros por continente, rango de población y rango de superficie.
  - Ordenamientos por nombre, población y superficie (ascendente o descendente).
  - Cálculo de estadísticas básicas del conjunto de países.
- Requerimientos técnicos: uso de listas y diccionarios, modularización mediante funciones, lectura desde un archivo CSV y validaciones de entrada.

A partir de esta lectura, cada requisito del enunciado se fue asociando a una función concreta del programa (buscar_nombre_paises(), filtrar_por_continente(), filtrar_por_rango_población(), filtrar_por_rango_superficie(), ordenar_paises(), mostrar_estadisticas(), y ver_todos_los_registros()), lo que ayudó a estructurar el diseño antes de comenzar a escribir el código.

3.2. Diseñar lógica general – Flujo de operaciones

El flujo general de la aplicación se diseñó en torno a un menú principal en consola, controlado por un ciclo while True dentro de la función mostrar_menu().

El flujo de operaciones principales es el siguiente:

1. Inicio del programa.
2. Mostrar menú principal con las 8 (ocho) opciones (búsquedas, filtros, ordenamientos, estadísticas, ver todos los registros, y salir).
3. Leer la opción ingresada por el usuario de la aplicación.

4. Evaluar la opción con una estructura condicional *match* y llamar a la función correspondiente".
    - Opción 1: buscar_nombre_paises().
    - Opción 2: filtrar_por_continente().
    - Opción 3: filtrar_por_rango_población().
    - Opción 4: filtrar_por_rango_superficie().
    - Opción 5: ordenar_paises().
    - Opción 6: mostrar_estadisticas().
    - Opción 7: ver_todos_los_registros().
    - Opción 8: finalizar el programa.
5. Cada función realiza su lógica específica (búsqueda, filtro, ordenamiento o cálculo de estadísticas), muestra los resultados y retorna al menú.
6. El ciclo se repite hasta que el usuario decide abandonar el programa (salir).

Este flujo se complementa en el diagrama de flujo de operaciones, donde se representan las decisiones principales (elección de opción, validaciones de entrada y retornos al menú).

3.3. Construir la base del proyecto

La base del proyecto se construyó creando una carpeta específica para el Trabajo Práctico Integrador (con los siguientes archivos principales: dataset_paises.csv; tpi_gestion_datos_paises.py; Capturas_Apuntes_TPI_P1; Trabajo Práctico Integrador_Programación_I; Presentación_TPI_P1_Gest_Datos_Paises y un README). En el código Python se incluyó un encabezado con los datos de la materia, el tema del trabajo y de quien suscribe.

A nivel de código, el primer paso fue definir el menú principal y una implementación mínima de mostrar_menu(), que permitiera probar el ciclo de interacción con el usuario incluso antes de tener todas las funcionalidades terminadas. De esta forma se trabajó con un enfoque incremental: primero la estructura general, luego el detalle de cada opción.

3.4. Organizar el proyecto

El proyecto se organizó siguiendo la sugerencia de modularización de la cátedra:

- Bloque da datos y persistencia: funciones relacionadas con el CSV (obtener_datos_paises()), normalización de texto (quitar_acentos(), normalizar()) y validaciones generales.
- Bloque de búsquedas y filtros: buscar_nombre_paises(), filtrar_por_continente(), filtrar_por_rango_poblacion(), filtrar_por_rango_superficie().
- Bloque de ordenamientos: ordenar_paises(), que centraliza la lógica para ordenar por nombre, población o superficie.
- Bloque de estadísticas: mostrar_estadisticas(), donde se calculan totales, promedios y máximos/mínimos, además del conteo de países por continente.
- Bloque de presentación / main: mostrar_pais() (para formatear la salida de cada país) y mostrar_menu() como punto de entrada al programa.

Esta organización ayudó a mantener el archivo ordenado y a tener claro qué parte del código correspondía a cada requerimiento.

3.5. Definir estructuras de datos adecuadas

La estructura principal elegida para representar el dataset fue una lista de diccionarios:

- Cada país se guarda como un diccionario con las claves “nombre”, “poblacion”, “superficie” y “continente2.
- Todos los diccionarios se almacenan en la lista paises, que se retorna desde obtener_datos_paises().

Ejemplo de estructura en memoria:

{

"nombre": "Argentina",

"poblacion": 45376763,

"superficie": 2780400,

"continente": "América"

}

Para las estadísticas se utilizaron:

- Listas auxiliares de población y superficie (poblaciones, superficies) para calcular sumas y promedios utilizando sum() y la longitud de la lista.
- Un diccionario de conteo (conteo_continenntes) donde la clave es el continente y el valor es la cantidad de países de ese continente.

3.6. Trabajar en el módulo de persistencia

El módulo de persistencia se basa en la función obtener_datos_paises(), que utiliza csv.DictReader para leer el archivo dataset_paises.csv y convertir cada fila en un diccionario.

Se incorporaron varias validaciones para controlar errores de formato en el CSV:

- Verificación que existan todas las columnas obligatorias (nombre, poblacion, superficie, continente) y que no vengan vacías.
- Conversión de población y superficie a float dentro de un bloque try/except; si algún valor no es numérico, se ignora esa fila y se muestra un mensaje de aviso indicando el número de línea.
- Manejo de errores globales como FileNotFoundError (archivo ausente) y UnicodeDecodeError (problemas de codificación), mostrando mensajes claros a la persona usuaria.

Gracias a estas validaciones, el resto del programa puede trabajar con una lista de países consistente, reduciendo la probabilidad de errores en las funciones de filtros y estadísticas.

3.7. Preparar entrega final

En la etapa final se revisó el código para:

- Verificar que cada opción del menú llame a la función correcta y que todas las rutas vuelvan al menú principal.
- Limpiar comentarios en desuso y dejar solo aquellos que explican decisiones importantes.

- Probar manualmente distintos casos: búsquedas sin resultados, filtros con rangos invertidos, ingreso de letras donde se espera un número, selección de ordenamiento inválido, etc.
- Generar capturas de pantalla de la ejecución para los anexos.
- Subir la versión final al repositorio de GitHub e incorporar el informe teórico y la presentación de apoyo.

4. Metodología Utilizada

La metodología de trabajo fue incremental e iterativa. En lugar de intentar escribir todo el programa de una vez, se avanzó por etapas:

- Lectura detallada del enunciado y planificación de las funciones necesarias.
- Implementación del menú principal y de una primera versión de lectura del CSV.
- Desarrollo de las funciones de búsqueda y filtros, probándolas una por una con distintos datos de prueba.
- Incorporación de los ordenamientos y de las estadísticas, verificando que las comparaciones numéricas y de cadenas fueran correctas.
- Agregado de validaciones adicionales (control de entradas, manejo de errores de archivo, normalización de mayúsculas, minúsculas y acentos).

Durante todo el proceso se utilizó Visual Studio Code como entorno de desarrollo, la terminal integrada para ejecutar el script y GitHub Desktop para registrar commits, permitiendo volver atrás en caso de errores. Las pruebas se realizaron de forma manual, ejecutando cada opción del menú con distintos ejemplos hasta lograr un comportamiento estable.

5. Resultados Obtenidos

5.1. Resultados alcanzados

Los resultados alcanzados se pueden resumir en los siguientes puntos:

- El programa implementa todas las funcionalidades mínimas solicitadas: búsquedas por nombre, filtros por continente, rango de población y de superficie, ordenamientos por nombre/población/superficie y cálculo de estadísticas básicas.

- La aplicación tolera errores comunes de ingreso: si se ingresa texto donde se espera un número, si el rango está invertido o si se elige un criterio de orden inexistente, se muestran mensajes de error y se vuelve a pedir el dato sin que el programa se interrumpa.
- Se logró una salida en consola ordenada y legible: la función mostrar_pais() formatea de forma uniforme nombre, población, superficie y continente, lo que facilita la comparación visual entre países.
- Las estadísticas generadas permiten tener una visión global del dataset: cantidad total de países, superficies y poblaciones totales y promedio, países extremos (mayor y menor población/superficie) y distribución por continente.

5.2. Dificultades presentadas y cómo fueron resueltas

Durante el desarrollo surgieron varias dificultades técnicas que requirieron ajustes y correcciones:

- Manejo de mayúsculas, minúsculas y acentos. Al principio algunas búsquedas no encontraban resultados cuando la persona usuaria escribía “america” en lugar de “América” o “argentina” en lugar de “Argentina”. Esto se resolvió implementando las funciones quitar_acentos() y normalizar(), que convierten todo a minúsculas sin acentos antes de comparar.
- Errores al llamar funciones. En algunas opciones se olvidó pasar el parámetro pais a la función mostrar_pais(), lo que generó errores del tipo “missing 1 required positional argument”. Se corrigió revisando las firmas de las funciones y asegurando que se llamaran siempre con los argumentos correctos.
- Ordenamientos con campos distintos. Inicialmente la función de ordenamiento intentaba usar la misma lógica para nombre, población y superficie, lo que provocó errores al aplicar casefold() sobre valores numéricos. Se reorganizó el código para distinguir entre el caso de cadenas (nombre) y el de números (población y superficie).
- Validación de archivo CSV. No controlar el contenido del CSV podía provocar fallos silenciosos o resultados incorrectos. La incorporación de verificaciones de columnas obligatorias y de conversiones numéricas con try/except permitió

detectar y reportar filas inválidas en lugar de dejar que el error aparezca más adelante en el programa.

Estas dificultades, aunque retrasaron el desarrollo, ayudaron a entender mejor la importancia de las validaciones, la lectura atenta de los mensajes de error y el uso de pruebas incrementales para localizar problemas.

6. Conclusiones

El desarrollo del Trabajo Práctico Integrador permitió aplicar de manera concreta los contenidos abordados en Programación I: estructuras de control, funciones, manejo de listas y diccionarios, validaciones, ordenamientos, estadísticas básicas y lectura de archivos CSV. A través del diseño y construcción de una aplicación funcional, se logró integrar estos conceptos en un proyecto único, lo que resultó fundamental para comprender el propósito práctico de cada herramienta del lenguaje.

Además del aspecto técnico, este trabajo implicó un proceso personal y progresivo de adaptación al entorno de desarrollo. Al inicio del cuatrimestre, la utilización de Visual Studio Code, GitHub y el manejo de archivos y módulos representaba *“mucho más”* que un desafío significativo. Sin embargo, a medida que avanzó el proyecto, fue posible adquirir mayor familiaridad con estas herramientas y perder parte del miedo inicial a su uso. Hoy, si bien aún no siento que haya consolidado completamente todos los conceptos fundamentales de programación, sí reconozco que este TPI significó un avance real en términos de autonomía y principalmente confianza.

A través de sucesivas iteraciones, errores corregidos, ajustes en el código, validaciones y mejoras en la lógica, se fortaleció la capacidad para analizar problemas, dividirlos en partes manejables y resolverlos mediante funciones y estructuras adecuadas. También se desarrolló una comprensión más clara de la importancia del diseño previo, la planificación del flujo de operaciones y el control de errores para garantizar un programa robusto.

Finalmente, si bien reconozco que aún necesito más práctica y tiempo para afianzar completamente los conocimientos de la materia, este proyecto constituyó un paso clave para avanzar en la comprensión del lenguaje Python y en el uso de herramientas de

programación modernas. El trabajo realizado permitió adquirir experiencia concreta, superar dificultades y sentar una base más sólida para seguir aprendiendo en las asignaturas siguientes.

7. Bibliografía

Apuntes de cátedra UTN. (2025). *Programación I. Unidad I, II, III y IV*.

Apuntes de cátedra UTN. (2025). *Programación I. Unidad VI.*

DATARMONY. (29 de julio de 2024). *Data Science I: población, muestra, experimentos y tipos de variables*. Obtenido de https://datarmony.com/estadistica-basica-para-ciencia-de-datos-con-python/

DataScientest. (s.f.). *Python If, Else: todo sobre las sentencias condicionales*. Obtenido de https://datascientest.com/es/python-if-else

El libro de Python. (s.f.). *Listas en Python*. Obtenido de https://ellibrodepython.com/listas-en-python

GeeKsforGeeks. (11 de octubre de 2025). *Sorting Algorithms*. Obtenido de https://www.geeksforgeeks.org/dsa/sorting-algorithms/

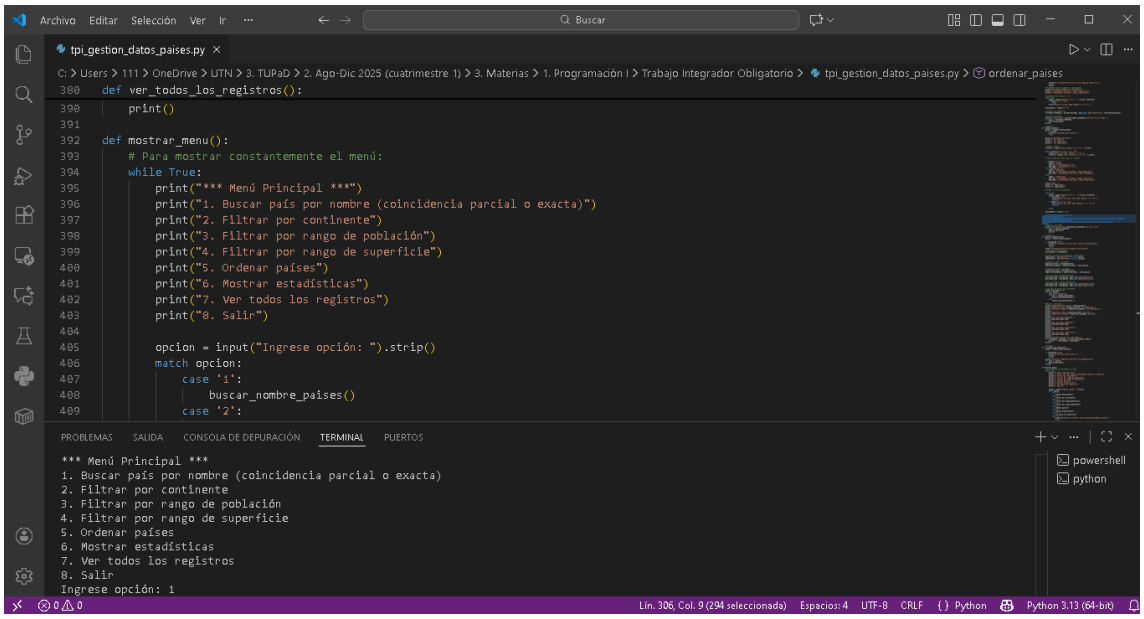
Python. (s.f.). *CSV File Reading and Writing*. Obtenido de https://docs.python.org/es/3/library/csv.html#

8. Anexos

8.1. Anexo 1 – Capturas del programa en ejecución

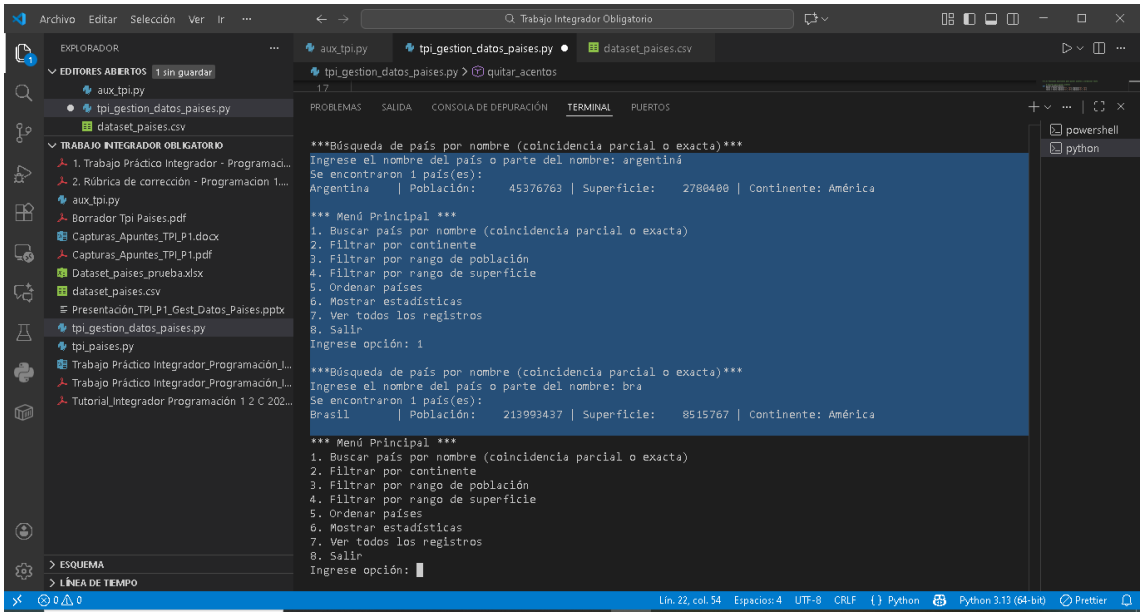
En este anexo se presentan las capturas correspondientes a cada funcionalidad del TPI, con el fin de demostrar el correcto funcionamiento del sistema.

Imagen 2. Menú principal



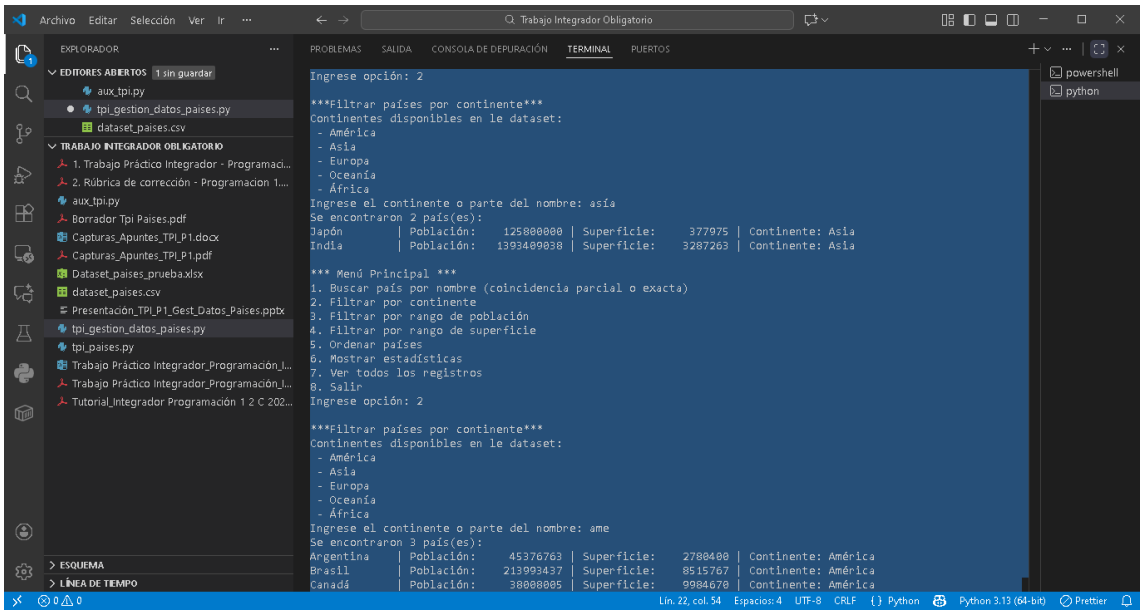
Fuente: Elaboración propia (2025)

Imagen 3. Búsqueda por nombre



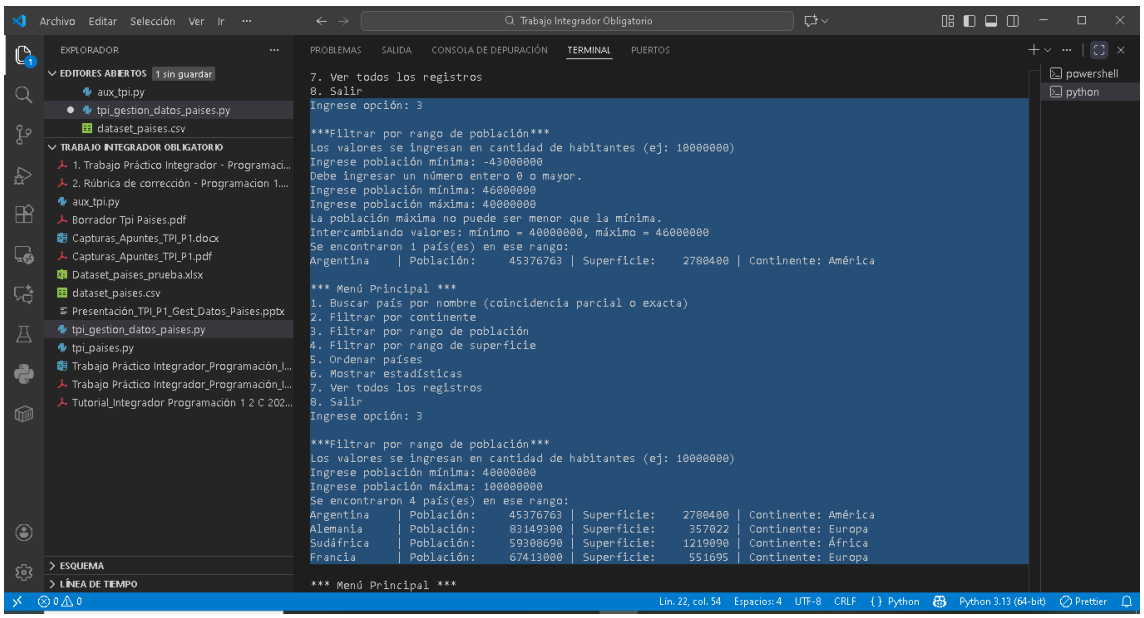
Fuente: Elaboración propia (2025)

Imagen 4. Filtro por continente



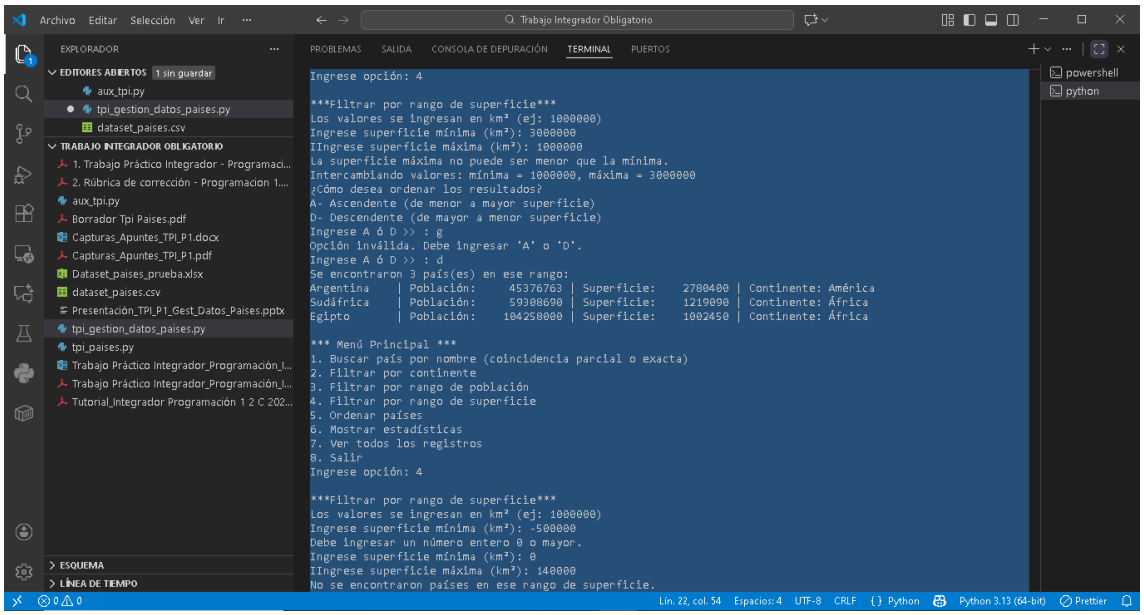
Fuente: Elaboración propia (2025)

Imagen 5. Filtro por población



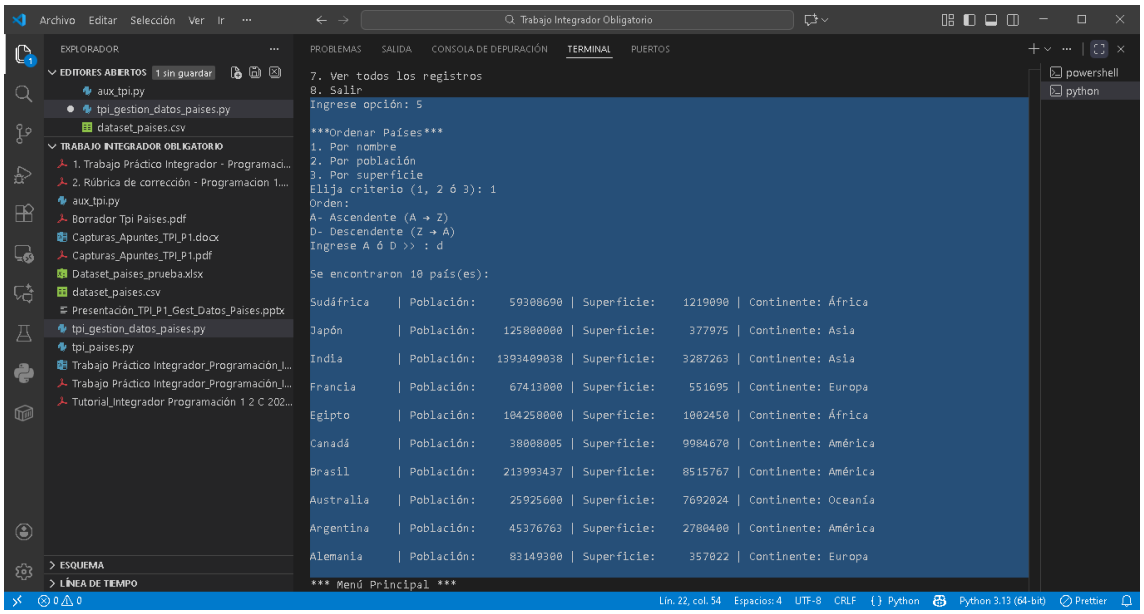
Fuente: Elaboración propia (2025)

Imagen 6. Filtro por superficie



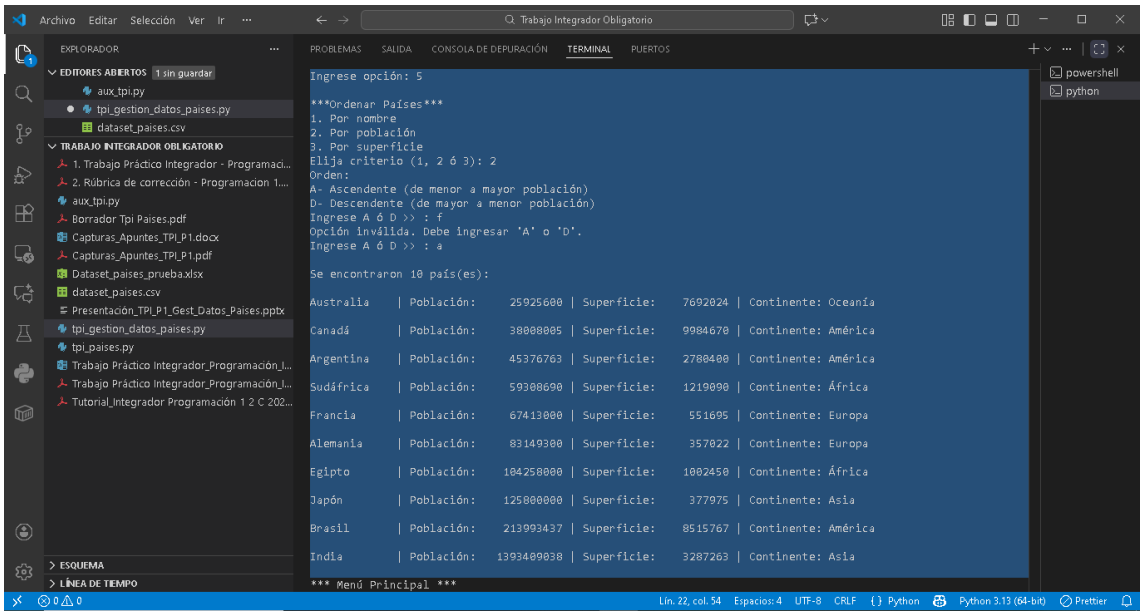
Fuente: Elaboración propia (2025)

Imagen 7. Ordenamientos (nombre)



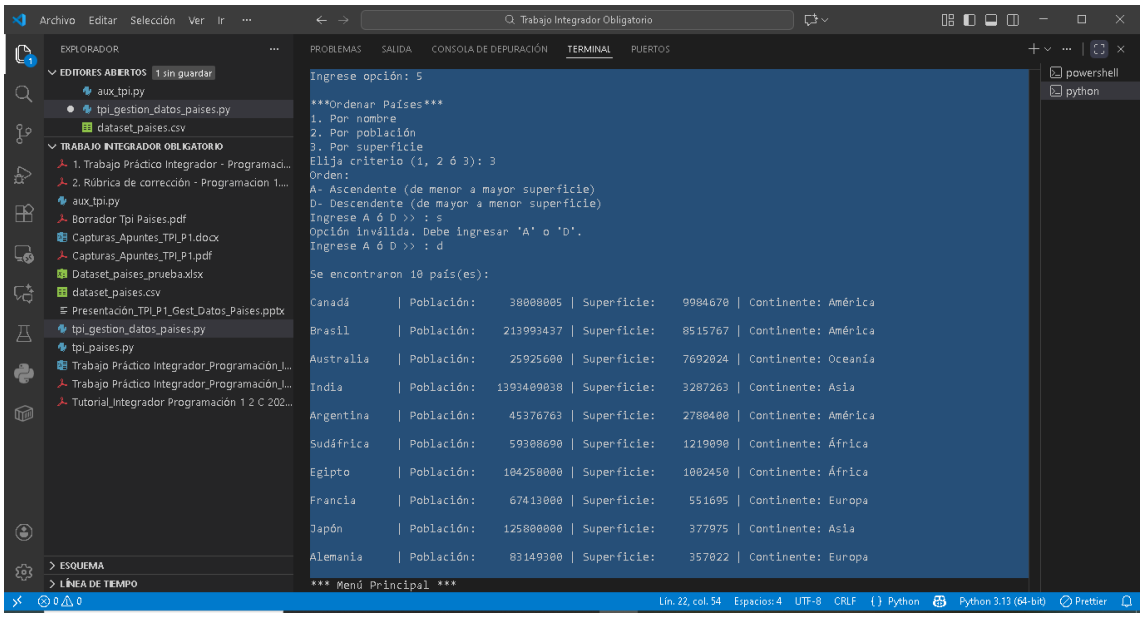
Fuente: Elaboración propia (2025)

Imagen 8. Ordenamientos (población)



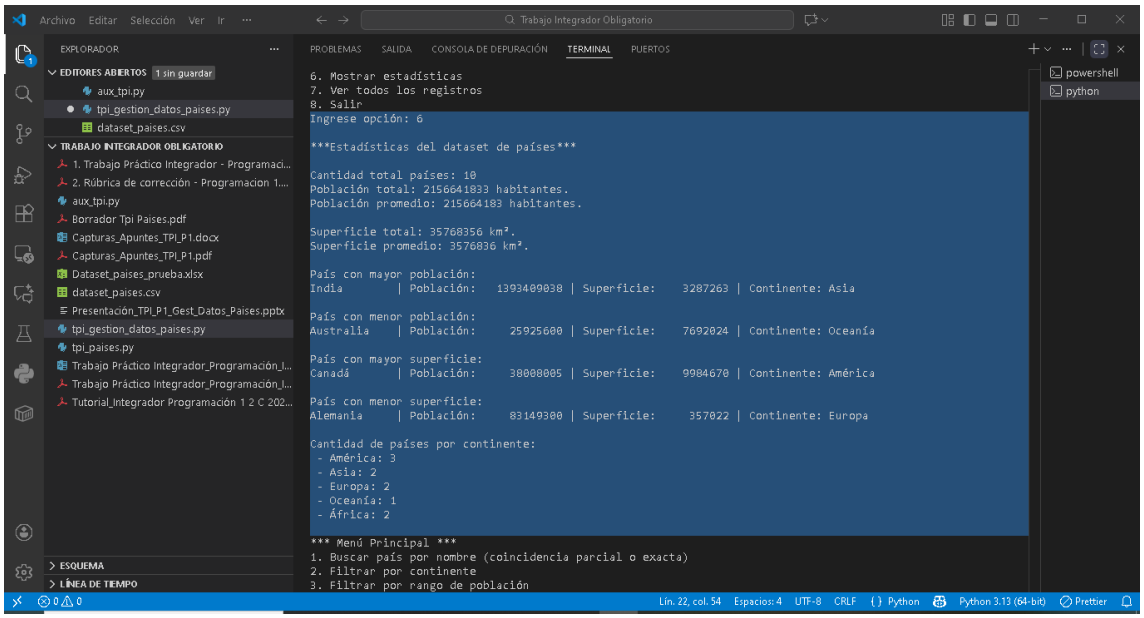
Fuente: Elaboración propia (2025)

Imagen 9. Ordenamientos (superficie)



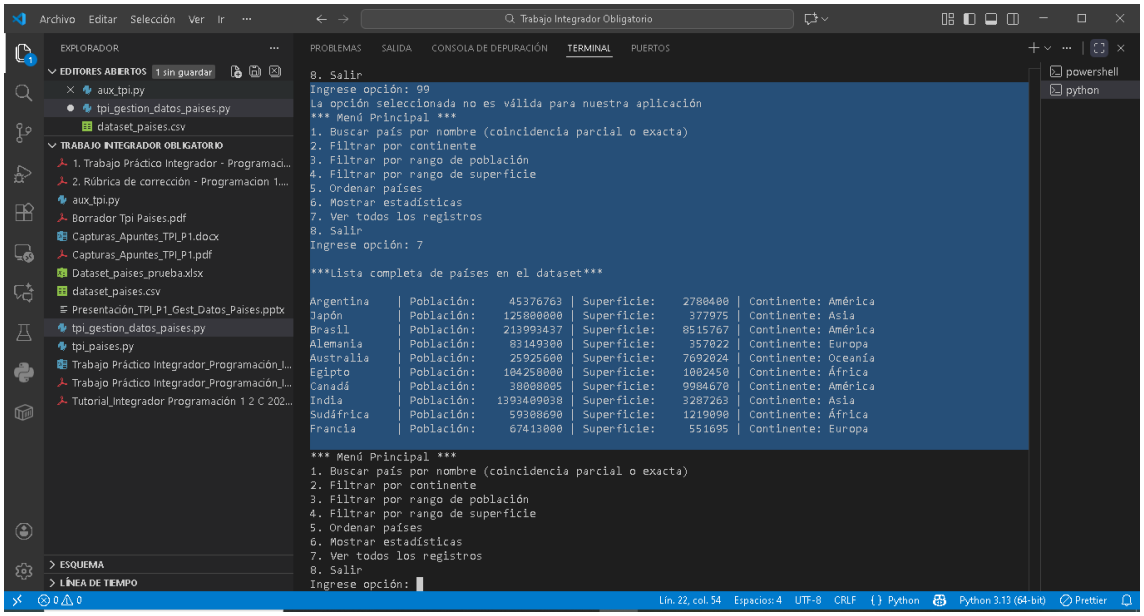
Fuente: Elaboración propia (2025)

Imagen 10. Estadísticas



Fuente: Elaboración propia (2025)

Imagen 11. Ver registros

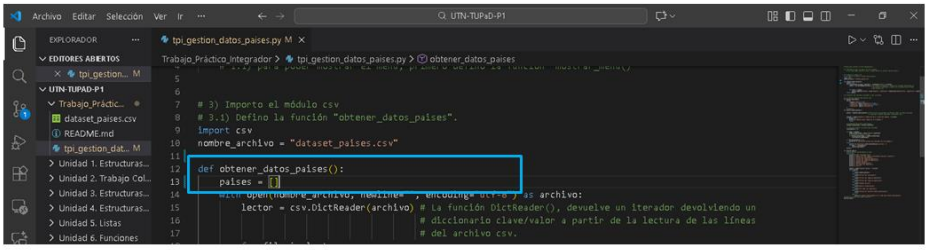


Fuente: Elaboración propia (2025)

Imagen 12. Control de errores (carga de los países por duplicado)

Error x

Solución



20

Fuente: Elaboración propia (2025)

Error:

- Usaba la misma lista global / cada llamada volvía a agregar países.

Solución:

- La función creo una lista nueva cada vez

Imagen 13. Control de errores (opción 2 no funciona)

Error x

Solución

20

Fuente: Elaboración propia (2025)

Error:

- Olvidé de agregar en match (opción 2) la función filtrar_por_continente()

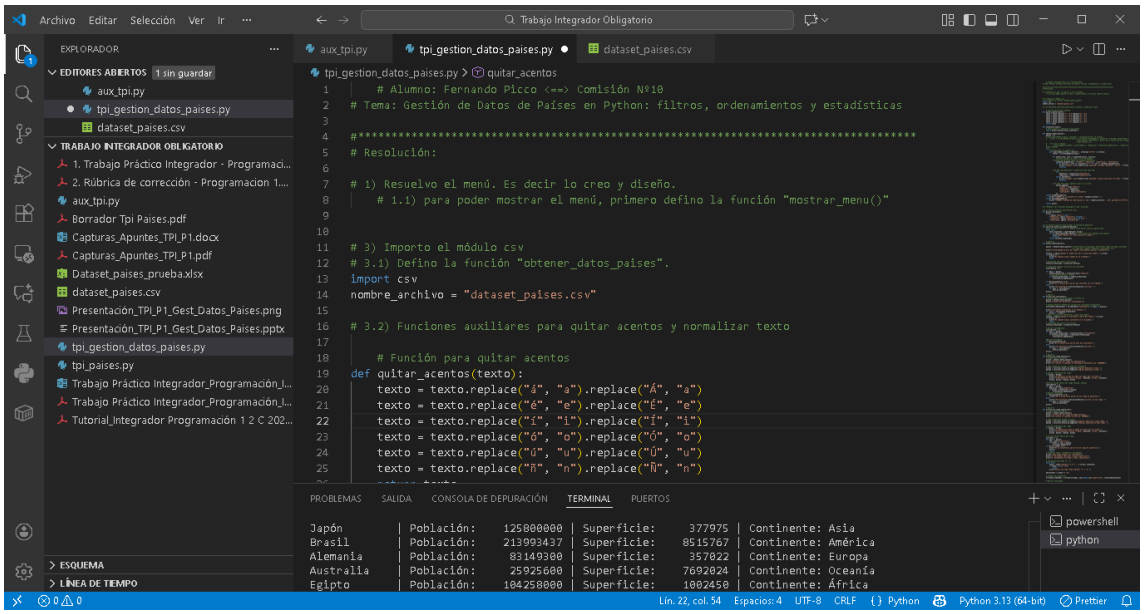
Solución:

- Agregué la función en cuestión.

8.2. Anexo 2 – Estructura del proyecto

Se presenta la estructura de carpetas y archivos dentro del entorno Visual Studio Code, detallando la organización interna del proyecto, el archivo CSV de entrada y el script principal del TPI.

Imagen 14. Organización del TPI

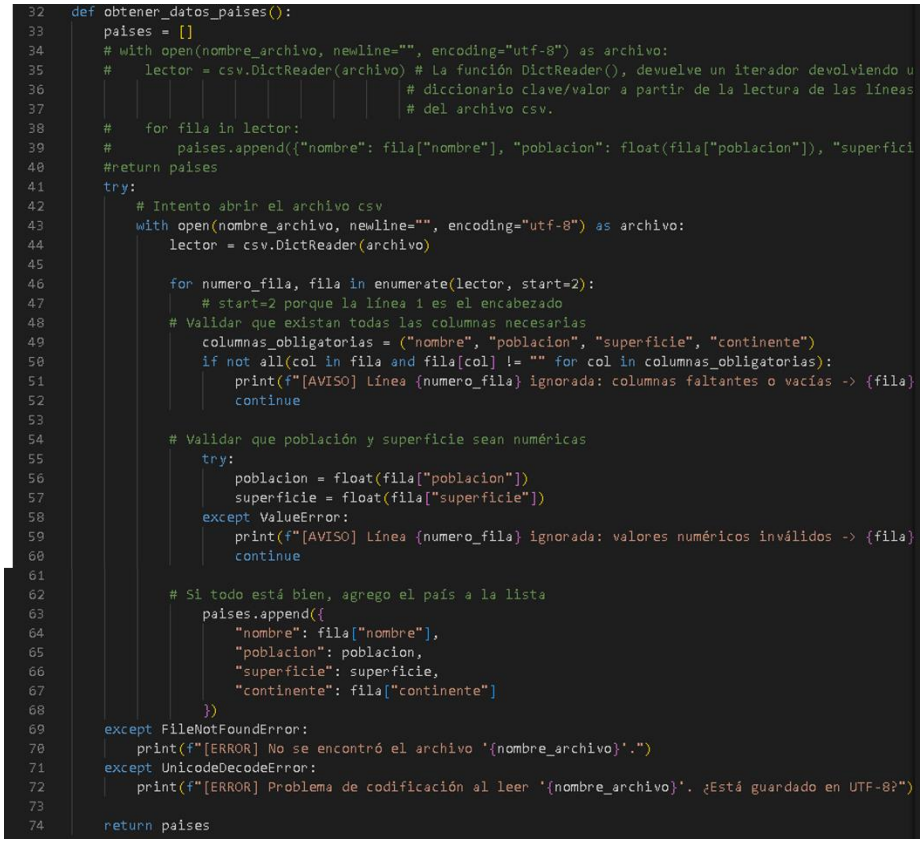


Fuente: Elaboración propia (2025)

8.3. Anexo 3 – Fragmentos de códigos relevantes

A continuación, se incluyen las funciones más significativas del sistema, seleccionadas por su aporte directo al cumplimiento del enunciado y por representar las características más importantes del programa (validaciones, filtrado, ordenamiento y lectura del CSV).

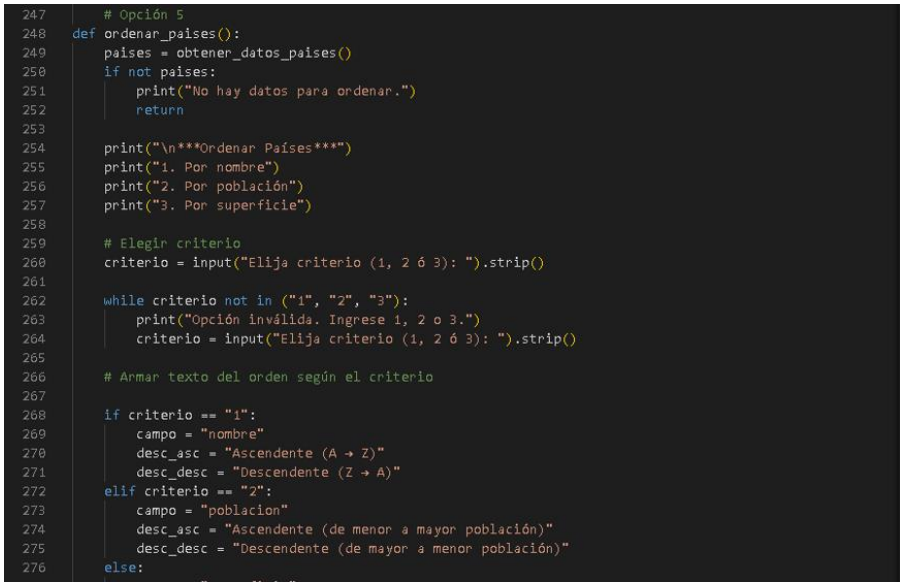
Imagen 15. Función obtener_datos_países()



```python
def obtener_datos_paises():
    paises = []
    # with open(nombre_archivo, newline="", encoding="utf-8") as archivo:
    #     lector = csv.DictReader(archivo) # La función DictReader(), devuelve un iterador devolviendo u
                                           # diccionario clave/valor a partir de la lectura de las líneas
                                           # del archivo csv.
    #     for fila in lector:
    #         paises.append({"nombre": fila["nombre"], "poblacion": float(fila["poblacion"]), "superfici
    #return paises
    try:
        # Intento abrir el archivo csv
        with open(nombre_archivo, newline="", encoding="utf-8") as archivo:
            lector = csv.DictReader(archivo)

            for numero_fila, fila in enumerate(lector, start=2):
                # start=2 porque la línea 1 es el encabezado
                # Validar que existan todas las columnas necesarias
                columnas_obligatorias = ("nombre", "poblacion", "superficie", "continente")
                if not all(col in fila and fila[col] != "" for col in columnas_obligatorias):
                    print(f"[AVISO] Línea {numero_fila} ignorada: columnas faltantes o vacías -> {fila}
                    continue

                # Validar que población y superficie sean numéricas
                try:
                    poblacion = float(fila["poblacion"])
                    superficie = float(fila["superficie"])
                except ValueError:
                    print(f"[AVISO] Línea {numero_fila} ignorada: valores numéricos inválidos -> {fila}
                    continue

                # Si todo está bien, agrego el país a la lista
                paises.append({
                    "nombre": fila["nombre"],
                    "poblacion": poblacion,
                    "superficie": superficie,
                    "continente": fila["continente"]
                })
    except FileNotFoundError:
        print(f"[ERROR] No se encontró el archivo '{nombre_archivo}'.")
    except UnicodeDecodeError:
        print(f"[ERROR] Problema de codificación al leer '{nombre_archivo}'. ¿Está guardado en UTF-8?")

    return paises
```

Fuente: Elaboración propia (2025)

Imagen 16. Función ordenar_paises() – Parte 1



```python
    # Opción 5
def ordenar_paises():
    paises = obtener_datos_paises()
    if not paises:
        print("No hay datos para ordenar.")
        return

    print("\n***Ordenar Países***")
    print("1. Por nombre")
    print("2. Por población")
    print("3. Por superficie")

    # Elegir criterio
    criterio = input("Elija criterio (1, 2 ó 3): ").strip()

    while criterio not in ("1", "2", "3"):
        print("Opción inválida. Ingrese 1, 2 o 3.")
        criterio = input("Elija criterio (1, 2 ó 3): ").strip()

    # Armar texto del orden según el criterio

    if criterio == "1":
        campo = "nombre"
        desc_asc = "Ascendente (A → Z)"
        desc_desc = "Descendente (Z → A)"
    elif criterio == "2":
        campo = "poblacion"
        desc_asc = "Ascendente (de menor a mayor población)"
        desc_desc = "Descendente (de mayor a menor población)"
    else:
```

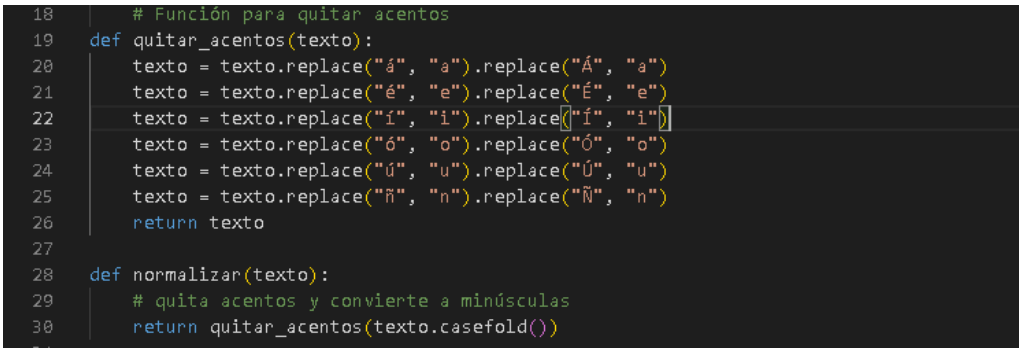
Fuente: Elaboración propia (2025)

Imagen 17. Función ordenar_paises() – Parte 2

```
        campo = "superficie"
        desc_asc = "Ascendente (de menor a mayor superficie)"
        desc_desc = "Descendente (de mayor a menor superficie)"

    print("Orden: ")
    print(f"A- {desc_asc}")
    print(f"D- {desc_desc}")

    # Elegir A / D con validación

    while True:
        orden = input("Ingrese A ó D >> : ").strip().casefold()
        if len(orden) != 1:
            print("Opción inválida. Solo debe ingresar 'A' o 'D'.")
            continue
        if orden not in ("a", "d"):
            print("Opción inválida. Debe ingresar 'A' o 'D'.")
            continue

        break

    descendente = (orden == "d")

    # Ordenar según criterio elegido
    if campo == "nombre":
        paises_ordenados = sorted(paises, key=lambda p:p["nombre"].casefold(), reverse=descendente)
    else: # población y suuperficie
        paises_ordenados = sorted(paises, key=lambda p:p[campo], reverse=descendente)

    # Mostrar resultados
    print(f"\nSe encontraron {len(paises_ordenados)} país(es): \n")
    for pais in paises_ordenados:
        mostrar_pais(pais)
        print()
```

Fuente: Elaboración propia (2025)

Imagen 18. Funciones: quitar_acentos() y normalizar()



```
# Función para quitar acentos
def quitar_acentos(texto):
    texto = texto.replace("á", "a").replace("Á", "a")
    texto = texto.replace("é", "e").replace("É", "e")
    texto = texto.replace("í", "i").replace("Í", "i")
    texto = texto.replace("ó", "o").replace("Ó", "o")
    texto = texto.replace("ú", "u").replace("Ú", "u")
    texto = texto.replace("ñ", "n").replace("Ñ", "n")
    return texto

def normalizar(texto):
    # quita acentos y convierte a minúsculas
    return quitar_acentos(texto.casefold())
```

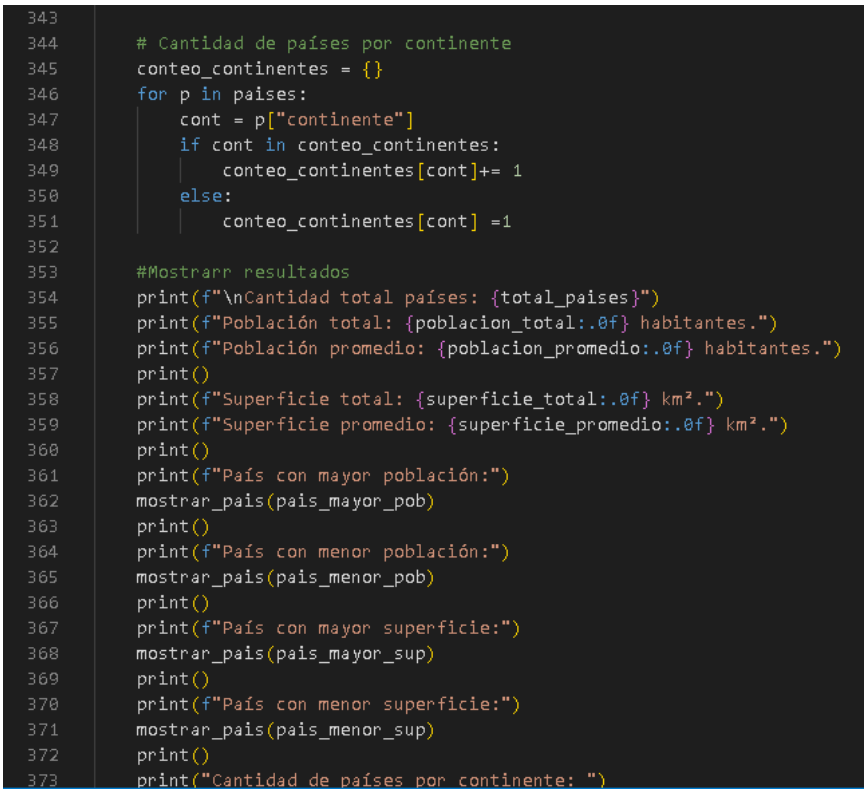
Fuente: Elaboración propia (2025)

Imagen 19. Función mostrar_estadisticas() – Parte 1

```
    # Opción 6
def mostrar_estadisticas():
    paises = obtener_datos_paises()

    if len(paises) == 0:
        print("No hay datos de países para calcular estadísticas.")
        print()
        return
    print("\n***Estadísticas del dataset de países***")
    # Cantidad total de paíese
    total_paises = len(paises)

    # Listas auxiliares de población y superficieç
    poblaciones = [p["poblacion"] for p in paises]
    superficies = [p["superficie"] for p in paises]

    # Población total y promedio
    poblacion_total = sum(poblaciones)
    poblacion_promedio = poblacion_total / total_paises

    # Superficie total y promedio
    superficie_total = sum(superficies)
    superficie_promedio = superficie_total / total_paises

    # País con mayor y menor población
    pais_mayor_pob = max(paises, key=lambda p:p["poblacion"])
    pais_menor_pob = min(paises, key=lambda p:p["poblacion"])

    # País con mayor y menor superficie
    pais_mayor_sup = max(paises, key=lambda p:p["superficie"])
    pais_menor_sup = min(paises, key=lambda p:p["superficie"])
```

Fuente: Elaboración propia (2025)

Imagen 20. Función mostrar_estadisticas() – Parte 2



```

    # Cantidad de países por continente
    conteo_continentes = {}
    for p in paises:
        cont = p["continente"]
        if cont in conteo_continentes:
            conteo_continentes[cont]+= 1
        else:
            conteo_continentes[cont] =1

    #Mostrarr resultados
    print(f"\nCantidad total países: {total_paises}")
    print(f"Población total: {poblacion_total:.0f} habitantes.")
    print(f"Población promedio: {poblacion_promedio:.0f} habitantes.")
    print()
    print(f"Superficie total: {superficie_total:.0f} km².")
    print(f"Superficie promedio: {superficie_promedio:.0f} km².")
    print()
    print(f"País con mayor población:")
    mostrar_pais(pais_mayor_pob)
    print()
    print(f"País con menor población:")
    mostrar_pais(pais_menor_pob)
    print()
    print(f"País con mayor superficie:")
    mostrar_pais(pais_mayor_sup)
    print()
    print(f"País con menor superficie:")
    mostrar_pais(pais_menor_sup)
    print()
    print("Cantidad de países por continente: ")
```

Fuente: Elaboración propia (2025)

Imagen 21. Función mostrar_menu()

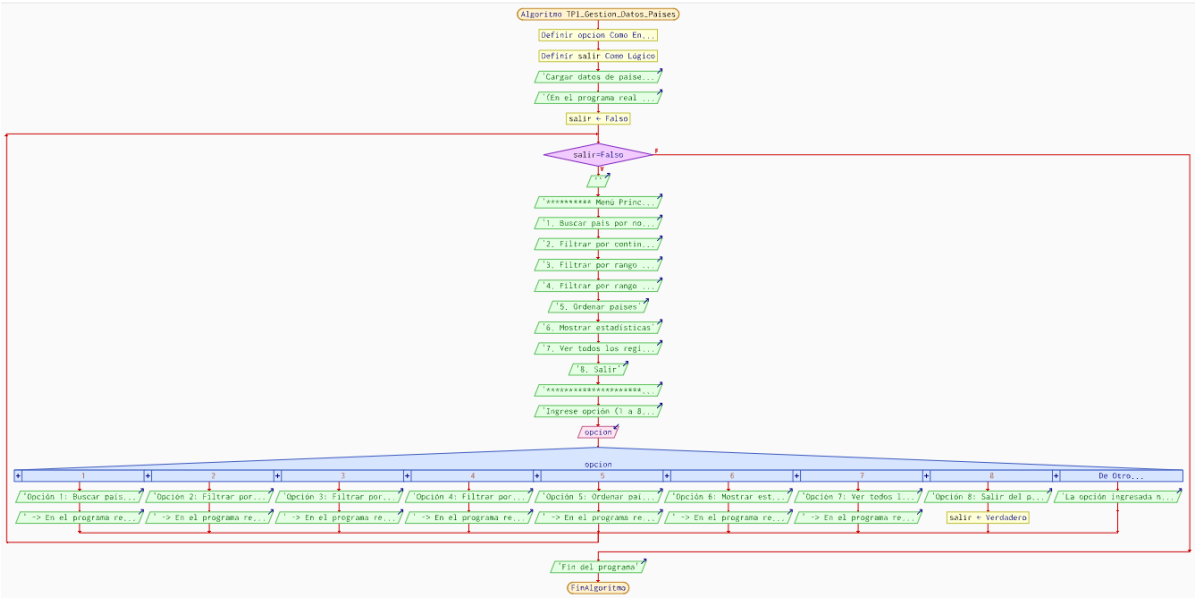
```python
def mostrar_menu():
    # Para mostrar constantemente el menú:
    while True:
        print("*** Menú Principal ***")
        print("1. Buscar país por nombre (coincidencia parcial o exacta)")
        print("2. Filtrar por continente")
        print("3. Filtrar por rango de población")
        print("4. Filtrar por rango de superficie")
        print("5. Ordenar países")
        print("6. Mostrar estadísticas")
        print("7. Ver todos los registros")
        print("8. Salir")

        opcion = input("Ingrese opción: ").strip()
        match opcion:
            case '1':
                buscar_nombre_paises()
            case '2':
                filtrar_por_continente()
            case '3':
                filtrar_por_rango_poblacion()
            case '4':
                filtrar_por_rango_superficie()
            case '5':
                ordenar_paises()
            case '6':
                mostrar_estadisticas()
            case '7':
                ver_todos_los_registros()
            case '8':
                print("¡Gracias por utilizar nuestra aplicación!¡Hasta pronto!")
                break
            case _:
                print("La opción seleccionada no es válida para nuestra aplicación")
# Invoco la función para poder mostrarla
mostrar_menu()
```

Fuente: Elaboración propia (2025)

8.4. Anexo 4 – Diagrama del flujo de operaciones

Se adjunta el diagrama de flujo general del programa, elaborado en base a la lógica implementada en Python y acorde al diseño requerido en la consigna. En este se muestra la carga inicial del archivo CSV, el bucle principal del menú y las acciones asociadas a cada opción, hasta la finalización del sistema.

Imagen 22. Diagrama de flujo general del programa



Fuente: Elaboración propia (2025)

8.5. Anexo 5 – Enlace al repositorio GitHub

Se presenta el enlace público al repositorio donde se encuentra alojado el código fuente del proyecto, junto con el README y archivos complementarios.

Repositorio del proyecto (GitHub)

Enlace: https://github.com/FernandoPicco/UTN-TUPaD-P1.git