FACULTAD REGIONAL MENDOZA
UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Práctico Integrador

Tecnicatura Universitaria en Programación, comisión 1

Programación

Docente Gerardo Magni

Alumnos

Julieta Herrera, Gabriel D'anna, Tatiana Peralta, Jesús Ramirez y María de los Angeles Zalazar

Fecha

6 de noviembre del 2025

OBJETIVOS.

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 del dataset.

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.

MARCO TEÓRICO.

En el desarrollo del proyecto "Gestión de Datos de Países en Python", fue necesario comprender y aplicar diversos conceptos fundamentales del lenguaje Python. Entre ellos se destacan las estructuras de datos (listas y diccionarios), el uso de funciones, condicionales, ordenamientos, estadísticas básicas, manejo de archivos CSV y principios de programación orientada a objetos (POO). El siguiente marco teórico tiene como objetivo explicar cada uno de estos elementos y su importancia dentro del sistema implementado.

Listas.

Las listas en Python son un tipo de dato que permite almacenar datos de cualquier tipo. En Python debemos escribir estos elementos separados por comas y dentro de corchetes.

Creación y acceso.

Para crear una lista, simplemente encierra los elementos entre corchetes:

```
frutas = ["manzana", "banana", "naranja"]
```

Algunas propiedades de las listas:

- Son ordenadas, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos arbitrarios
- Pueden ser indexadas con [i].
- Se pueden anidar, es decir, meter una dentro de la otra.
- Son mutables, ya que sus elementos pueden ser modificados.
- Son dinámicas, ya que se pueden añadir o eliminar elementos.

https://ellibrodepython.com/listas-en-python#listas-en-python

Diccionario.

Un diccionario es una colección de pares clave-valor que permite acceder rápidamente a los valores utilizando sus claves. Los diccionarios se pueden crear con corchetes {} separando con una coma cada par key: value. En el siguiente ejemplo tenemos tres keys que son el nombre, la edad y el documento:

```
persona = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}

print(persona.keys())    # Imprime dict_keys(["nombre", "edad", "ciudad"])
print(persona.values())  # Imprime dict_values(["Juan", 25, "Madrid"])
print(persona.items())   # Imprime dict_items([("nombre", "Juan"), ("edad",
25), ("ciudad", "Madrid")])

persona.update({"profesion": "Ingeniero"})
print(persona)  # Imprime {"nombre": "Juan", "edad": 25, "ciudad": "Madrid",
"profesion": "Ingeniero"}
```

- Los diccionarios en Python tienen las siguientes características:
- Mantienen el orden en el que se insertan las claves.
- Son mutables, con lo que permiten añadir, borrar y modificar sus elementos.
- Las claves deben ser únicas. A menudo se utilizan las cadenas de texto como claves, pero en realidad podría ser cualquier tipo de datos inmutable: enteros, flotantes, tuplas (entre otros).
- Tienen un acceso muy rápido a sus elementos, debido a la forma en la que están implementados internamente.

  https://ellibrodepython.com/diccionarios-en-python.

  https://pythones.net/diccionarios-en-python-3-guia-ejemplos/.

  https://aprendepython.es/core/datastructures/dicts/.

Funciones.

Las funciones son bloques de código reutilizables que realizan una tarea específica. Son uno de los conceptos fundamentales de la programación y te permiten organizar tu código de manera más clara, evitar repetición y facilitar el mantenimiento.

Componentes:

- **`def:`** Palabra clave que inicia la definición.
- **Nombre:** Debe ser descriptivo y seguir snake_case.
- **Parámetros:** Variables que recibe la función (opcionales).
- **Docstring:** Documentación de la función (opcional pero recomendado).
- **Cuerpo:** Código indentado que se ejecuta.
- **`return:`** Devuelve un valor (opcional).

Para llamar a una función, simplemente escribimos el nombre de la función seguido de paréntesis:

```
def saludo():
    print("¡Hola, mundo!")

saludo()  # Imprime "¡Hola, mundo!"
```

https://elpythonista.com/funciones-en-python-guia-completa-2025-sintaxis-parametros-y-ejemplos

Estructuras condicionales.

Las estructuras condicionales son el mecanismo que permite a tus programas tomar decisiones. Sin ellas, tu código solo podría ejecutarse de forma lineal, una instrucción tras otra, sin capacidad de adaptarse a diferentes situaciones.

Python ofrece tres palabras clave principales para construir condicionales:

- **`if`:** ejecuta código si una condición es verdadera
- **`elif`:** (else if) evalúa una condición alternativa si la anterior fue falsa
- **`else`:** ejecuta código si ninguna condición anterior fue verdadera

Sentencia if.

La sentencia condicional más básica en Python es la sentencia if, la cual se expresa de la siguiente forma:

```
if condición:
      #Ejecutar código.
```

En la expresión previa:

- La condición es una expresión booleana que se evalúa como verdadera (True) o falsa (False).
- Se requiere el uso de dos puntos (:) al final de la condición.
- Todas las líneas de código a ejecutar si se cumple la condición tienen que estar indentadas respecto la sentencia if.

Sentencia else.

A la sentencia if se le puede añadir opcionalmente una sentencia else. Esta sentencia contiene el código a ejecutar en caso de que no se cumpla la condición de la sentencia if. Esta estructura se expresa del siguiente modo:

```
if condición:
      #Ejecutar código.
else:
      # Ejecutar código distinto.
```

Sentencia elif.

A una sentencia if else se pueden añadir un número indefinido de condiciones adicionales a verificar. Estas condiciones se definen mediante la sentencia elif, la cual es un abreviación de else if. Ésta se define así:

```
if condición:
      # Ejecutar código.
Elif otra_condición:
      # Ejecutar otro código.
else:
      # Ejecutar código distinto.
```

https://elpythonista.com/if-elif-y-else-en-python-guia-de-condicionales-2025-ejemplos
https://www.programaenpython.com/fundamentos/sentencias-condicionales-en-python/

Métodos de Ordenamiento.

Las listas de Python tienen un método incorporado list.sort() que modifica la lista en el sitio. También hay una función incorporada sorted() que crea una nueva lista ordenada a partir de un iterable.

Conceptos básicos de ordenación.

Una clasificación ascendente simple es muy fácil: simplemente llame a la función sorted(). Retorna una nueva lista ordenada:

```
sorted([5, 2, 3, 1, 4])
# Devuelve [1, 2, 3, 4, 5]
```

También puede usar el método list.sort(). Modifica la lista in situ (y retorna None para evitar confusiones). Por lo general, es menos conveniente que sorted(), pero si no necesita la lista original, es un poco más eficiente.

```
a = [5, 2, 3, 1, 4]
a.sort()
a
#Devuelve [1, 2, 3, 4, 5]
```

Otra diferencia es que el método list.sort() solo aplica para las listas. En contraste, la función sorted() acepta cualquier iterable.

```
sorted({1: 'D', 2: 'B', 3: 'B', 4: 'E', 5: 'A'})
#Devuelve [1, 2, 3, 4, 5]
```

https://docs.python.org/es/3/howto/sorting.html.

Estadísticas básicas.

Las estadísticas básicas permiten obtener información general sobre los datos, como máximos, mínimos y promedios. En nuestra aplicación, implementamos cálculos como el país con mayor y menor población, el promedio de población y de superficie, y la cantidad de países por continente. Ejemplo directo del integrador:

```
def prom_poblacion(paises):
    suma=0
    for pais in paises:
        suma+=int(pais.poblacion) # Suma la poblacion de cada pais

    return suma/(len(paises)) # Retorna el promedio de la poblacion
```

```
def prom_superficie(paises):
    suma1=0
    for pais in paises:
        suma1+=int(pais.superficie) # Suma la superficie de cada pais

    return suma1/(len(paises)) # Retorna el promedio de superficie
```

Archivo CSV.

Un archivo CSV (valores separados por comas) permite que los datos se guarden en una estructura tabular con una extensión .csv. Los archivos CSV se han usado de manera extensiva en aplicaciones de comercio electrónico porque se consideran muy fáciles de procesar. Algunas de las áreas en donde han sido usadas incluyen:

- Importar y exportar datos de clientes.
- Importar y exportar productos.
- Exportar órdenes.
- Exportar informes analíticos de comercio electrónico.

Abrir Archivo CSV y Leer Datos con Python.

Para abrir y leer un archivo CSV en Python, puedes usar el módulo integrado csv. Aunque no es necesario.

```
import csv

with open('example.csv', 'r') as file:
    reader = csv.reader(file)
    for row in reader:
        print(row)
```

Cómo Guardar en un Archivo CSV en Python.

Guardar datos en un archivo CSV es una tarea común en Python. Los archivos CSV son fáciles de leer y se pueden abrir fácilmente en cualquier software de hoja de cálculo. En Python, podemos usar el módulo csv para escribir en un archivo CSV. Aquí hay algunos ejemplos de cómo guardar en un archivo CSV en Python.

Este ejemplo demuestra cómo escribir una simple lista de valores en un archivo CSV.

```
import csv

# Example data
data = [['Name', 'Age', 'Gender'], ['Alice', '25', 'Female'], ['Bob', '30', 'Male'

# Open csv file in write mode
with open('example.csv', mode='w') as file:
    writer = csv.writer(file)
    # Write data to csv file
    writer.writerows(data)
```

Agregar contenido.

```
with open("salida.txt", "a") as archivo:
      archivo.write ("Nueva línea\n") archivo.write("Otra línea mas\n”)
```

- https://diveintopython.org/es/learn/file-handling/csv.
- Teoria de la materia programacion.

Programación Orientada a Objetos.

Este modo o paradigma de programación nos permite organizar el código de una manera que se asemeja bastante a como pensamos en la vida real, utilizando las clases.

La programación orientada a objetos está basada en 6 principios o pilares básicos:

- **Herencia.**
- **Cohesión.**
- **Abstracción.**
- **Polimorfismo.**
- **Acoplamiento.**
- **Encapsulamiento.**

Definición de clases.

Se muestra cómo en Python se define una clase simple:

```
class Perro:
      pass
```

Esto define la clase “Perro”, aunque sin contenido.

Luego se puede instanciar un objeto:

```
mi_perro = Perro()
```

Atributos (atributos de instancia vs atributos de clase).

- **Atributos de instancia:** son propios de cada objeto/instancia. Ejemplo: nombre, raza de un perro.
- **Atributos de clase:** pertenecen a la clase en general, compartidos por todos los objetos de esa clase. Ejemplo: `especie = 'mamífero'` para todos los perros. En Python se define el constructor __init__ para inicializar atributos de instancia:

```
class Perro:
      especie = 'mamífero'

      def __init__(self, nombre, raza):
            self.nombre = nombre
            self.raza = raza
```

Luego se puede acceder así:

```
print(mi_perro.nombre)
print(mi_perro.especie)
```

Métodos

Los métodos son funciones definidas dentro de una clase que operan sobre los objetos (normalmente usando self para referirse a la instancia.

```
def ladra(self):
      print("Guau")

def camina(self, pasos):
      print(f"Caminando {pasos} pasos")
```

Y luego se invoca algo como:

```
mi_perro.ladra()
mi_perro.camina(10)
```

https://ellibrodepython.com/programacion-orientada-a-objetos-python.

DISEÑO DEL CASO PRÁCTICO.

Se nos ha pedido un programa de Python capaz de leer y operar conforme a un archivo .csv llamado “paises.csv”, donde se hallen datos sobre países. Más específicamente, sus nombres, números de habitantes, superficies territoriales y respectivos continentes. Con esta información debemos poder:

- Realizar búsquedas parciales o completas de sus nombres para el retorno de la información asociada a cada país.
- Filtrar la lista de países por continente, rango de población y rango de superficie.
- Ordenar los países por nombre, población y superficie.
- Mostrar estadísticas sobre cuál país cuenta con menor población y cuál con mayor población, el promedio general de población, el promedio de superficie general, y la cantidad de países por continente.

Para esto se necesita un empleo general de listas, diccionarios, funciones, estructuras condicionales, ordenamientos, estadísticas básicas y archivos .csv.

Afrontamos esta tarea dividiéndola en tareas más pequeñas:

- Leer y almacenar la información del .csv en alguna lista u objeto.
- Crear un menú principal donde el usuario pueda elegir qué operación realizar.
- Dividir las operaciones en funciones para evitar repetir código.
- Programar una función para buscar países según el input del usuario.
- Crear funciones para organizar los datos extraídos del archivo en los órdenes requeridos.
- Crear funciones que escaneen el archivo, recopilen y retornen información sobre población, superficie y continentes.
- Procurar la mínima cantidad de errores o interrupciones posibles en cuanto a lógica e inputs.
- Procurar la comodidad de ejecución y manejo para el usuario.

Entonces planeamos un programa que siguiese el siguiente flujo superficial:

1. Recorrer el archivo 'paises.csv' y recopilar sus valores en un lista bidimensional donde cada fila sea un objeto cuyos atributos son los datos del .csv (nombre (str), población (int), superficie (float) y continente (str)), de forma que cada línea del archivo tenga un objeto equivalente y fácilmente operable en la lista.
2. Entrar en el menú principal, una función que ofrezca opciones de operación al usuario, y que una de ellas sea la opción de salir, puesto que se pretende repetir eternamente el menú hasta que el usuario decida salir del programa por cuenta propia.
3. Llevar al usuario, mediante inputs, a las múltiples operaciones (encapsuladas en funciones) que él decida, para su ejecución en el orden que vea necesario. Cuando se complete el código de sus funciones, deberán volver al menú.
4. Resolver las operaciones que requieran hacer un escaneo de los países recorriendo la lista de países línea por línea con un bucle for.
5. Programar un submenú específico para la operación de filtrado de países. Éste, como el principal, deberá funcionar con inputs que llevan a distintas funciones y se deberá repetir hasta que el usuario seleccione la opción para volver al menú principal.

Con esto en mente, nos pusimos a trabajar hasta completar el código, en cuyo funcionamiento íntegro pienso elaborar ahora mismo.

Documentación del código.

Para amenizar la explicación, dividiremos esta documentación en:

I. Preparativos y menú principal.

II. Búsqueda de un país.

III. Filtrado de países.

IV. Ordenamiento de países.

V. Muestra de estadísticas.

I. Preparativos y menú principal.

1. El programa importa la clase **Pais** de **Pais.py**, cuyos atributos son:
   - **nombre** (str),
   - **continente** (str),
   - **poblacion** (int),
   - **superficie** (float).

2. El programa entra a la función **lista_paises()** para definir el valor de la lista **paises**, con la que se harán las operaciones de países en el futuro. En la función **lista_paises()**:
   1) Se inicializa la lista **listaPaises**.
   2) Se abre el archivo **paises.csv** en modo lectura con la función **open()**.
   3) Se empieza un bucle **for** en el que cada iteración es una línea de **paises.csv**. En este bucle:
      1. Se crea la lista **datos**, donde se guardan los datos de la línea separados por la coma empleando **split(",")**.
      2. Se crea un objeto **pais** con los datos de **datos** transformados a los tipos de dato acordes a los atributos de la clase **Pais**.
      3. Se añade **pais** a **listaPaises**, que hasta ahora estaba vacío.
   4) Se retorna **listaPaises**, que ahora contiene todos los datos de **paises.csv** en forma de lista; valores que pasan a la lista **paises** del código principal.

3. El programa entra a la función **menu()** recibiendo **paises** como argumento. Esta función actúa como menú principal. Dependiendo de qué ingrese el usuario, se realizará una u otra acción. Eventualmente, el usuario volverá a este menú, y no lo podrá abandonar hasta elegir la opción de salir. Desde esta función se puede:
   1) Buscar un país mediante la función **buscar_pais(paises)**.
   2) Filtrar países, que lleva a la función **filtrar_paises(paises)**.
   3) Ordenar paises por nombre, población y superficie mediante las funciones **ordenar_paises(paises)**, **ordenar_poblacion(paises)** y **ordenar_superficie(paises)**.

4) Mostrar estadísticas generales sobre población y superficie mediante las funciones **may_minPoblacion(paises)**, **prom_poblacion(paises)** y **cant_paisesx-Cont(paises)**.
5) Salir del programa mediante un **break**.

II. Búsqueda de un país.

1. El programa le solicita al usuario que opción del menú desea llevar a cabo. Para buscar el país debe ingresar "a".
2. El programa solicita al usuario que ingrese el nombre del país que desea buscar.
3. Se inicializa **encontrado** como False.
4. La función recorre la lista completa de objetos **Pais** cargados desde el archivo.
5. La búsqueda se realiza utilizando la palabra clave **in** de Python, lo que permite la búsqueda parcial o de subcadena.
6. Si se encuentra una coincidencia (o más de una, ya que la lógica actual permite mostrar múltiples países si el termino coincide), el programa imprime los atributos del objeto **País** de la siguiente manera:
   - Nombre del país (con formato original).
   - Población.
   - Superficie (seguida de "km^2").
   - Continente.

   1) La variable **encontrado** pasa a ser True.
7. Si el bucle finaliza sin haber encontrado ninguna coincidencia (**encontrado** sigue siendo False), el programa informa al usuario que el país no se encuentra en el archivo.

III. Filtrado de países.

1. En el menú, si se desea Filtrar países se selecciona la palabra "b".
2. Se despliega un submenú que ofrece tres criterios de filtrado distintos para subseleccionar los países de la lista principal:

1) **Buscar por continente:** permite filtrar países por su ubicación geográfica.
    1. El programa le pide al usuario que ingrese el nombre del continente que desea filtrar.
    2. La entrada del usuario se convierte a minúsculas (.lower()) para garantizar que la búsqueda sea insensible a mayúsculas y minúsculas ('america' es igual a 'America').
    3. Se recorre la lista de países y se verifica si el atributo continente de cada objeto **Pais** contiene el término de búsqueda ingresado.
    4. El programa imprime los datos de todos los países que cumplen con el criterio del continente.
2) **Rango poblacional:** muestra países cuya población se encuentra dentro de un rango específico.
    1. Se recorre la lista de países y se verifica si el atributo **poblacion** de cada país.
    2. Clasifican automáticamente a todos los países de la base de datos en cuatro categorías preestablecidas (Baja, Media, Alta, Muy Alta) y luego muestran los países que caen en cada grupo:

Categoría	**Condición de población.**
Baja	< 20.000.000 habitantes.
Media	< 100.000.000 habitantes.
Alta	< 500.000.000 habitantes.
Muy alta	> 500.000.000 habitantes.

    3. El programa imprime una lista de los países que pertenecen a cada una de estas cuatro categorías (.
3) **Rango de superficie:** muestra países cuya superficie se encuentra dentro de un rango específico.
    1. De manera igual, esta función clasifica a todos los países según su superficie (en km cuadrados) en cuatro grupos fijos:

Categoría	Condición de población.
Baja	< 500.000 km^2.
Media	< 2.000.000 km^2.
Alta	< 5.000.000 km^2.
Muy alta	> 5.000.000 km^2.

2. El programa imprime una lista de los países que pertenecen a cada una de estas cuatro categorías de superficie.

IV. Ordenamiento de países.

1. La opción "c. Ordenar países" permite al usuario reordenar la lista principal de países basándose en tres criterios principales. Utiliza la función sort() de Python con una clave (key=lambda x: x.atributo) para realizar el ordenamiento. Muestra:
2. **Ordenamiento alfabético:** utiliza la función **ordenar_paises(paises).**
3. **Ordenamiento por población:** utiliza la función **ordenar_poblacion(paises)**.
4. **Ordenamiento por superficie:** utiliza la función **ordenar_superficie(paises)**.
5. El programa ejecuta cada función de ordenamiento, imprime el resultado del listado ordenado y luego solicita al usuario presionar una tecla para continuar con el siguiente criterio de ordenamiento.

V. Muestra de estadísticas.

1. La opción 'd. Estadísticas de países' ejecuta una serie de cálculos sobre la población y la superficie de todos los países de la lista, y presenta un conteo por continente.
2. Esta opción ejecuta cuatro cálculos primarios y los imprime en pantalla, separados por líneas horizontales:

1) País con Mayor y Menor Población:

1. La función **may_minPoblacion(paises)** utiliza la función max() y min() de Python con una clave (key=lambda x: x.poblacion) para identificar y mostrar el nombre y la cantidad de habitantes del país más poblado y del país menos poblado.

3. **Promedio Mundial de Población:**
    1) La función **prom_poblacion(paises)** suma la población de todos los países y la divide por la cantidad total de países.
    2) El resultado se redondea a dos decimales y se muestra como el "Promedio de Población Mundial".
4. **Promedio Mundial de Superficie:**
    1) La función **prom_superficie(paises)** suma la superficie (km^2) de todos los países y la divide por la cantidad total de países.
    2) El resultado se redondea a dos decimales y se muestra como el "Promedio de la superficie" en km^2.
    3) Finalmente, la función **cant_paisesxCont(paises)** recorre la lista de países y cuenta cuántos pertenecen a cada uno de los seis continentes (Europa, América, África, Asia, Oceanía y Antártida).
5. Imprime el conteo total para cada continente bajo el título "Cantidad de países por continente".

ESQUEMA GENERAL.

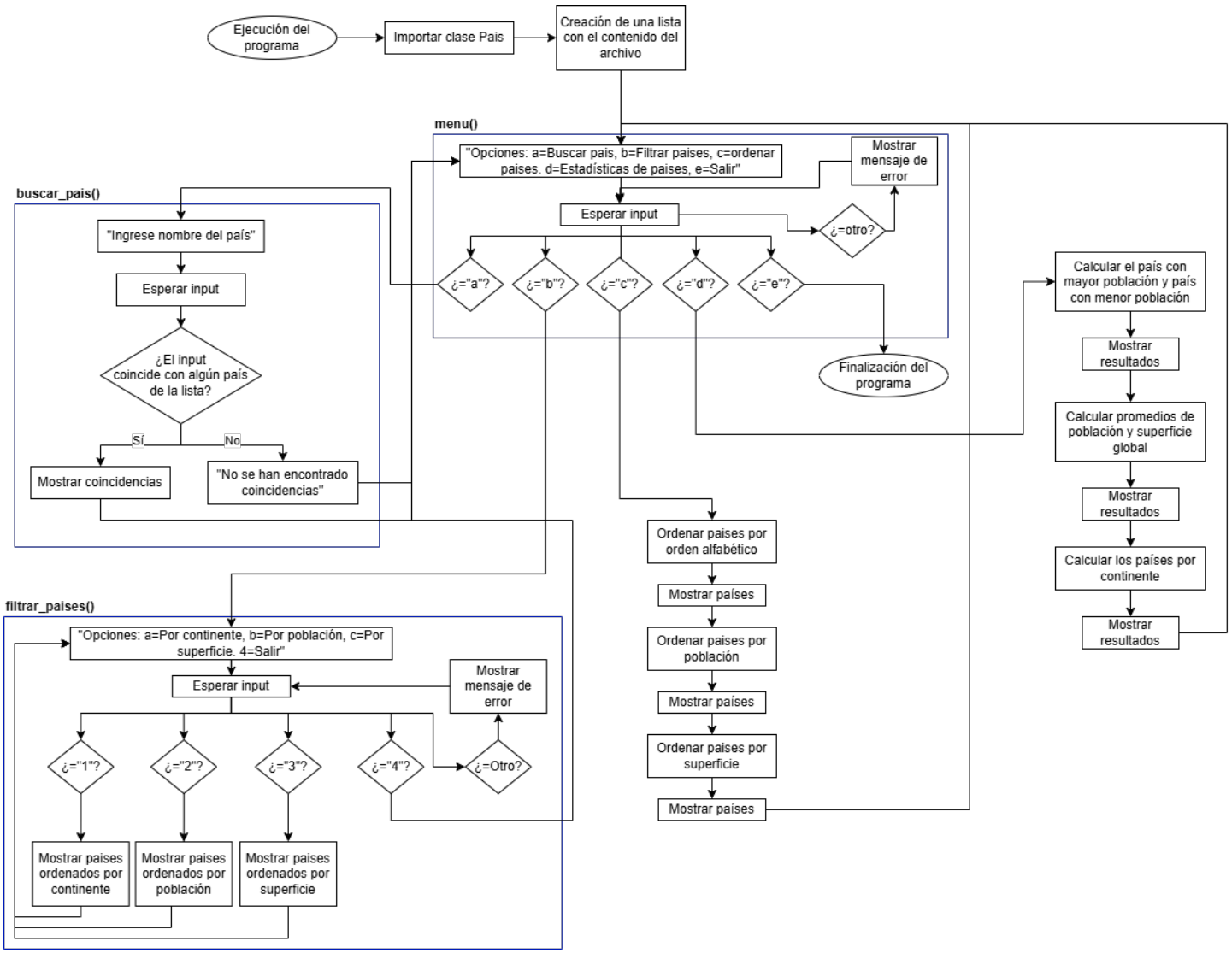

Diagrama general del flujo del programa.

REFELXIONES, RESULTADOS Y CONCLUSIONES.

Este proyecto fue una inmersión en las posibilidades que ofrece Python. Al principio, la idea de gestionar países parecía simple, pero a medida que avanzamos, nos dimos cuenta de lo crucial que resulta el orden y la estructura para un código limpio y funcional. Este trabajo también nos hizo ver lo que es el tener que colaborar con un equipo para realizar programas como este.

Crear la clase **Pais** fue conveniente y combinarla con listas nos ayudó a darnos cuenta de la cantidad de posibilidades que proveen. También ayuda su nivel de abstracción; el cómo se lo trata como un objeto compuesto de partes, no una variable aislada. Más allá de las líneas de código, lo que nos llevamos es la confianza de poder tomar un problema del mundo real, descomponerlo y construir una solución desde cero. Este proyecto nos enseñó que la planificación con POO es vital, y que el trabajo en equipo es esencial par la integridad del programa final.

Nuestro grupo desarrollo el proyecto "Gestión de Datos de Países" en Python, en el cual se implementó un sistema capaz de administrar, procesar y analizar información proveniente de un archivo CSV. Empleamos listas y diccionarios como estructuras principales para el almacenamiento de datos. Las listas permitieron que podamos mantener orden y dinámicas, mientras que los diccionarios nos facilitaron el acceso a información específica. Se desarrollaron funciones para modular el código, logrando operaciones como lectura del archivo CSV, filtrado, búsqueda y cálculo de estadísticas. Tales funciones nos permitió mejorar la organización del programa, evitando repetir ciertas instrucciones. Como complemento aplicamos estructuras condicionales (if, elif, else) para validar datos, controlar el flujo de ejecución y tomar decisiones dentro del mismo programa. Se aplicó métodos de ordenamiento, como sort() o sorted() para organizar países según distintos atributos siguiendo los criterios dados en la teoría. El uso del módulo CSV permitió leer y guardar datos en archivos CSV, permitiendo la continuidad de la información y facilitando la exportación de resultados. Además se utilizaron conceptos básicos de Programación Orientada a Objetos, creando clases para representar entidades relacionadas con el manejo de países. Como resultado obtuvimos un

proyecto funcional que demuestra la correcta aplicación de los conceptos teóricos aprendidos.

Conclusión.

En conclusión el desarrollo del proyecto nos permitió comprobar y analizar que el uso en conjunto de listas, diccionarios, funciones, condicionales y manejo de archivos es fundamental para la creación de un proyecto en Python que puede gestionar datos de forma eficiente. La lectura y escritura de archivos CSV se volvió una operación fluida gracias al módulo csv, que nos permite construir un flujo de datos completo dentro de la aplicación. El uso de la Programación Orientada a Objetos es considerado como el pilar más importante de nuestro proyecto, ya que facilito la reutilización de código y la separación de responsabilidades. Finalizando podemos decir que este proyecto no solo nos permitió aplicar los conceptos aprendidos, sino también comprender su utilidad en el entorno.