UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador

Programación 1

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

Profesor: Virginia Cimino

Tutor: Alberto Cortez

Agustin Miranda - agusmiranda1611@gmail.com

Tobías Correa - tobiasbcorrea@gmail.com

Objetivo del trabajo

El objetivo principal de este Trabajo Práctico Integrador es afianzar y aplicar los conocimientos fundamentales adquiridos en la materia de Programación 1.

Esto se logrará mediante la implementación de un sistema de Gestión de Datos de Países que:

- Utilice estructuras de datos (listas y diccionarios) para representar eficientemente la información de cada país.
- Demuestra la modularización del código a través del uso de funciones (manteniendo responsabilidades separadas) y estructuras de control como condicionales y repetitivas.
- Implemente la persistencia de datos mediante la lectura desde un archivo CSV.
- Desarrolle funcionalidades clave como el filtrado, ordenamiento y cálculo de estadísticas básicas sobre el conjunto de datos, cumpliendo con los requerimientos técnicos del proyecto.

En resumen, buscamos desarrollar un sistema funcional que permita gestionar, consultar y generar indicadores clave a partir del dataset de países, validando la entrada de datos y aplicando las buenas prácticas de programación.

Marco teórico

Listas

Las listas son estructuras de datos que permiten almacenar una colección ordenada y mutable de elementos y permiten agrupar valores diversos bajo una única variable, facilitando operaciones como acceso, modificación, filtrado y ordenamiento.
A diferencia de otras estructuras, las listas admiten tipos de datos heterogéneos, lo que significa que en una misma lista pueden coexistir números, cadenas de texto e incluso otras listas (listas anidadas).
El hecho de que una lista puede contener otras listas permite representar estructuras de tipo matriz, donde cada sublista actúa como una fila. Este enfoque resulta especialmente útil para organizar datos tabulares o realizar operaciones que requieran estructuras de dos dimensiones.

Usos
Las listas se emplean para manejar secuencias de datos, procesar colecciones dinámicas, implementar pilas y colas, y aplicar estadísticas o análisis sobre conjuntos de datos. En proyectos prácticos, son especialmente útiles para almacenar y recorrer registros leídos desde archivos, como sucede en este trabajo integrador con el dataset de países.

Métodos principales

Las listas en Python cuentan con diversos métodos integrados que facilitan la manipulación de sus elementos. Entre los más utilizados se destacan los siguientes:

`append(x)`: Agrega un elemento al final de la lista.

Ejemplo: *`nombres.append("Juan")`*

`extend(iterable)`: Agrega varios elementos al final de la lista desde otro iterable (por ejemplo, otra lista).

Ejemplo: *`nombres.extend(["Ana", "Luis"])`*

`insert(i, x)`: Inserta un elemento en una posición específica, desplazando los demás hacia la derecha.

Ejemplo: *`nombres.insert(0, "Carlos")`*

`remove(x)`: Elimina la primera aparición del elemento indicado. Si no existe, genera un error.

Ejemplo: *`nombres.remove("Luis")`*

`pop([i])`: Elimina y devuelve el elemento de la posición indicada. Si no se especifica un índice, elimina el último.

Ejemplo: *`nombres.pop() o nombres.pop(1)`*

`clear()`: Vacía completamente la lista, eliminando todos sus elementos.

Ejemplo: *`nombres.clear()`*

Ejemplo completo en Python:

```
# Gestión de una lista de productos en un supermercado
productos = ["Leche", "Pan", "Huevos", "Arroz"] # Definir la lista

productos.append("Fideos")      # Agregar un nuevo producto
productos.remove("Pan")         # Eliminar un producto
```

```
print("Productos disponibles:", productos)
```

Salida:

```
>>  ["Leche", "Huevos", "Arroz", "Fideos"]
```

Referencias

Python Software Foundation. (s.f.). *Estructuras de datos*. Documentación de Python. https://docs.python.org/es/3/tutorial/datastructures.html

Báez, R. (s.f.). *Listas en Python*. El Libro de Python. https://ellibrodepython.com/listas-en-python#insertindex-obj

Diccionarios

Los diccionarios son estructuras de datos que permiten almacenar información en forma de pares clave–valor. Cada elemento del diccionario se compone de una clave y un valor asociado.

A diferencia de las listas, donde los elementos se acceden mediante un índice numérico, en los diccionarios el acceso se realiza a través de las claves, lo que permite una búsqueda y manipulación más eficiente cuando se necesita acceder a datos específicos.

Las claves deben ser de un tipo de dato inmutable (como cadenas, números o tuplas que no contengan elementos mutables), y cada una de ellas debe ser única dentro del diccionario. En cambio, los valores pueden ser de cualquier tipo, incluso estructuras más complejas como listas u otros diccionarios, y no es necesario que sean únicos.

Usos

Los diccionarios son especialmente útiles cuando se necesita organizar y gestionar datos que requieren acceder a la información mediante una clave identificadora, lo que permite realizar búsquedas rápidas y modificaciones dinámicas.

En el programa desarrollado para este TPI, los utilizamos constantemente para representar los países, ya que nos permiten asociar fácilmente cada característica (como el nombre, la población, la superficie o el continente) a su valor correspondiente, haciendo el manejo de los datos más ordenado y eficiente.

Métodos principales

Los diccionarios en Python disponen de varios métodos integrados, los cuales son muy útiles para gestionar, consultar y modificar sus elementos.
Entre los más empleados, se incluyen los siguientes:

`clear()`: elimina todos los elementos del diccionario.

```
d.clear() # {}
```

`get(clave[, valor_defecto])`: devuelve el valor de una clave o el valor por defecto si no existe.

```
d.get('z', 'No encontrado')
```

`items()`: retorna una vista con los pares (clave, valor).

```
list(d.items()) # [('a', 1), ('b', 2)]
```

`keys()`: devuelve todas las claves del diccionario.

```
list(d.keys()) # ['a', 'b']
```

`values()`: devuelve todos los valores del diccionario.

```
list(d.values()) # [1, 2]
```

Ejemplo completo en python

```python
# Gestión de una lista de productos
# Definir el diccionario
producto = {"nombre": "Notebook", "precio": 850000, "stock": 12}


print(producto["nombre"])
producto["stock"] -= 1
print(producto)
```

Salida:

```
>> Notebook
>> {'nombre': 'Notebook', 'precio': 850000, 'stock': 11}
```

Referencias

Python Software Foundation. (s.f.). *Estructuras de datos*. Documentación de Python. https://docs.python.org/es/3/tutorial/datastructures.html

Báez, R. (s.f.). *Diccionarios en Python*. El Libro de Python.
https://ellibrodepython.com/diccionarios-en-python

Funciones

Las funciones son bloques de código reutilizables que permiten realizar una tarea específica dentro de un programa. Su principal objetivo es dividir el código en partes más pequeñas, ordenadas y comprensibles, facilitando tanto su lectura como su mantenimiento.
Mediante las funciones, se puede ejecutar un mismo conjunto de instrucciones múltiples veces sin necesidad de repetir el código, lo que reduce errores, mejora la eficiencia y promueve la reutilización. Además, permiten organizar los programas en módulos lógicos, haciendo que cada parte del sistema tenga una responsabilidad bien definida.
Cada función puede recibir parámetros o argumentos, los cuales representan los datos de entrada que necesita para realizar su tarea, y puede devolver un valor de salida mediante la instrucción `return`.
Por ejemplo, en el programa desarrollado para este TPI, una función se encarga de calcular el país con mayor población, mientras que otra tiene la responsabilidad de ordenar la lista de países.
Esta separación de tareas permite mantener las responsabilidades bien definidas, favoreciendo una mejor organización, legibilidad y mantenimiento del código.

Funciones Nativas vs. Funciones Propias
Python incluye una gran cantidad de **funciones nativas** que facilitan el trabajo diario del programador, permitiendo realizar tareas comunes sin necesidad de escribir código adicional.
Algunos ejemplos son:

`len(`*`object`*`)`: Retorna el tamaño (el número de elementos) de un objeto.

`isinstance(`*`object, classinfo`*`)`: Retorna True si el argumento object es una instancia del argumento classinfo, o de una subclase (directa, indirecta o virtual) del mismo, de lo contrario, retorna falso.

`input(`*`prompt`*`)`: Si el argumento prompt está presente, se escribe a la salida estándar sin una nueva línea a continuación. La función lee entonces una línea de la entrada, la convierte en una cadena (eliminando la nueva línea), y retorna eso.

Estas son solo algunas de las muchas funciones que Python ofrece por defecto, las cuales pueden consultarse en su documentación oficial para aprovecharlas en distintos contextos.

Por otro lado, también es posible definir funciones propias, es decir, creadas por el programador para resolver necesidades específicas del programa.
Su estructura básica es la siguiente:

```
def nombre_funcion(parametros)
      # cuerpo de la función
      return valor_de_retorno
```

De esta forma, podemos crear bloques de código reutilizables y personalizados, adaptados al funcionamiento que requiere nuestro programa.

Referencias

Báez, R. (n.d.). *Funciones en Python*. El Libro de Python.
https://ellibrodepython.com/funciones-en-python
Python Software Foundation. (n.d.). *Funciones integradas: input*. Documentación de Python.
https://docs.python.org/es/3/library/functions.html#input
freeCodeCamp. (2022, marzo 18). *Guía de funciones de Python con ejemplos*.
https://www.freecodecamp.org/espanol/news/guia-de-funciones-de-python-con-ejemplos/

Condicionales

Las condicionales en Python permiten controlar el flujo de ejecución de un programa, ejecutando distintos bloques de código según se cumpla o no una determinada condición lógica. Son fundamentales para que los programas puedan tomar decisiones y adaptarse a diferentes situaciones o valores de entrada.
Gracias a las estructuras condicionales (como if, elif y else) es posible evaluar expresiones lógicas y ejecutar acciones específicas sólo cuando se cumplen ciertos criterios. Esto permite crear programas dinámicos, con comportamientos que varían según las circunstancias o los datos que se procesan.
Por ejemplo, en el programa desarrollado para el presente TPI, las condicionales nos permiten verificar si la entrada del usuario es válida: si lo es, el programa continúa su ejecución normalmente, en caso contrario, se muestra un mensaje de error o se solicita que el usuario ingrese nuevamente la información.

Estructuras de código

`if`: ejecuta un bloque de código si la condición es verdadera.

```
if edad >= 18:
    print("Es mayor de edad")
```

`elif`: evalúa una nueva condición si la anterior no se cumple.

```
elif edad > 12:
    print("Es adolescente")
```

`else:` se ejecuta cuando ninguna de las condiciones anteriores es verdadera.

```
else:
    print("Es menor de edad")
```

Es importante mencionar que las condiciones se expresan mediante operadores relacionales, como ==, !=, <, >, <= y >=, los cuales permiten comparar valores o establecer relaciones entre variables y condiciones.
Además, existen los operadores lógicos (and, or, not), que permiten combinar o negar expresiones, posibilitando la creación de condiciones más complejas y precisas dentro del flujo del programa.

Por ejemplo

```
if edad >= 18 and nacionalidad == "Argentina":
    print("Puede votar")
```

Referencias

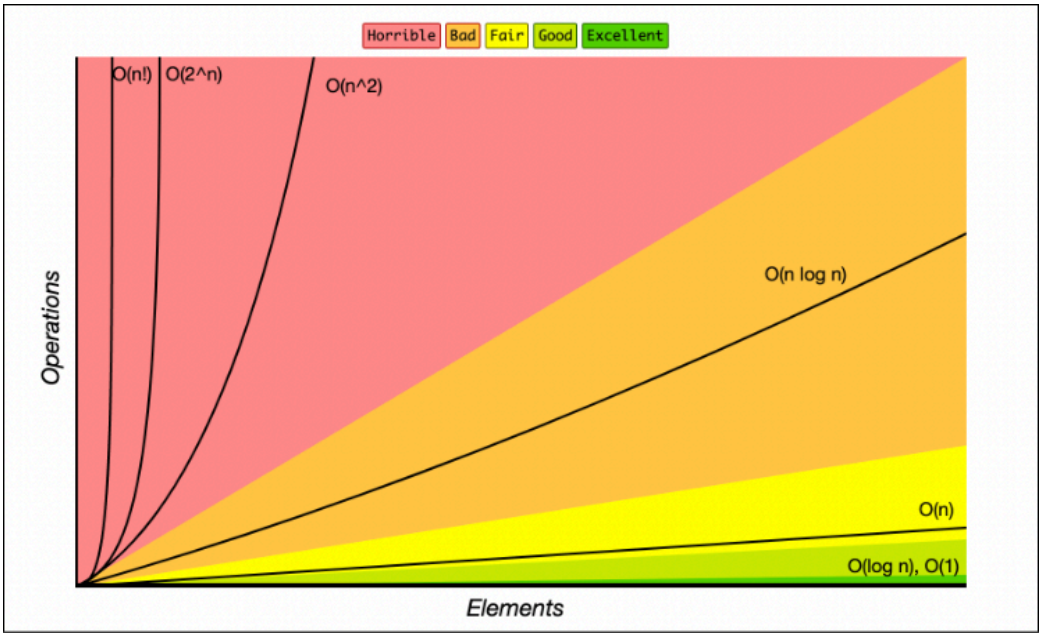
Báez, R. (n.d.). *Sentencias if en Python*. El Libro de Python. https://ellibrodepython.com/if-python

Programa en Python. (n.d.). *Sentencias condicionales en Python*. https://www.programaenpython.com/fundamentos/sentencias-condicionales-en-python/

Tokio School. (n.d.). *Condicionales en Python: if, else y elif*. https://www.tokioschool.com/noticias/condicionales-python/

Ordenamiento Burbuja

El método de ordenamiento que implementamos en este TPI para organizar la lista de países es el método de burbuja.
Recibe este nombre porque, conceptualmente, los elementos más "pesados" (o "livianos", según el criterio) van "burbujeando" o desplazándose hacia su posición final en el arreglo, de forma similar a como una burbuja sube en el agua.
Este método consiste en revisar la lista comparando pares de elementos adyacentes. Se empieza comparando el primer elemento con el segundo; si están en el orden incorrecto (por ejemplo, para un orden ascendente, si el primero es mayor que el segundo), sus posiciones se intercambian. Luego, se compara el segundo con el tercero, el tercero con el cuarto, y así sucesivamente hasta el final.
Este proceso completo se considera una "pasada". El método repite estas pasadas hasta que se pueda recorrer la lista entera sin necesidad de realizar ningún intercambio.
Para optimizarlo, se puede añadir una "bandera" (una variable de control). Esta bandera detecta si se hicieron cambios en la última pasada; si no se realizó ningún intercambio, significa que la lista ya está ordenada y el bucle principal puede finalizar, ahorrándonos repeticiones innecesarias.
El ordenamiento burbuja se caracteriza por ser el método más simple de entender y, por ende, el más didáctico para comenzar a estudiar algoritmos. No obstante, esta simplicidad tiene un costo alto en rendimiento: requiere de dos bucles anidados (uno para las pasadas y otro para las comparaciones), lo que provoca que tenga una complejidad cuadrática de O(n^2) en la notación Big O (que mide cómo crece el tiempo de ejecución a medida que aumentan los datos), siendo muy poco eficiente para casos de uso reales con grandes volúmenes de datos.



Como se observa en el gráfico comparativo de complejidad Big O, un algoritmo que tiene una complejidad O(n^2) es considerado "Horrible" por la rapidez con la que aumenta su tiempo de ejecución a medida que crecen los datos.

Ejemplo en Python

```
my_array = [64, 34, 25, 12, 22, 11, 90, 5]

n = len(my_array)
for i in range(n-1):
    for j in range(n-i-1):
        if my_array[j] > my_array[j+1]:
            my_array[j], my_array[j+1] = my_array[j+1], my_array[j]

print("Sorted array:", my_array)
```

Salida

```
>> Sorted array: [5, 11, 12, 22, 25, 34, 64, 90]
```

Referencias

Codecademy. (s.f.). *Bubble Sort Cheatsheet*. Recuperado el 6 de noviembre de 2025, de https://www.codecademy.com/learn/sorting-algorithms-java/modules/bubble-sort-java/cheatsheet

Smith Morales, H. (s.f.). *Métodos de Búsqueda y Ordenamiento* [Diapositivas]. Facultad de Ciencias de la Computación, Benemérita Universidad Autónoma de Puebla. https://www.cs.buap.mx/~hilario_sm/slide/p1/sort-find2A-ok.pdf

W3Schools. (n.d.). *Bubble Sort Algorithm*. https://www.w3schools.com/dsa/dsa_algo_bubblesort.php

Estadística Básica

En nuestro Trabajo Práctico Integrador, utilizamos procedimientos de la estadística básica (o para recolectar, organizar y resumir los datos crudos de nuestro archivo CSV. El objetivo no es solo mostrar los datos, sino extraer información de valor y obtener conclusiones clave sobre el conjunto de países, tal como se solicita en los requerimientos del sistema.

Para lograr este objetivo, aplicamos los siguientes procedimientos:

1. **Cálculo de máximos y mínimos**

   Para encontrar el país con mayor y menor población, fue necesario implementar un algoritmo para encontrar estos valores extremos en nuestra lista de países.

   El procedimiento consiste en:

1. Inicializamos una variable (ej. mayor_pais o menor_pais) que almacena el diccionario completo del primer país de la lista. Esta variable actuará como nuestro máximo (o mínimo) temporal.
2. Recorremos completa de países, iterando sobre cada país.

3. En cada iteración, comparamos el valor de 'poblacion' del pais actual con el valor de 'poblacion' del país almacenado en nuestra variable temporal. (mayor_pais['poblacion']).
4. Actualizamos la variable:
    - Para el mayor, si la población del pais actual es mayor (>), se sobrescribe la variable mayor_pais con el diccionario completo del pais actual.
    - Para el menor, si la población del pais actual es menor (<), se sobrescribe la variable menor_pais.
5. Al finalizar el bucle, la variable (mayor_pais o menor_pais) contiene toda la información del país al que pertenece ese valor.

```
def pais_mayor_poblacion(paises):
    mayor_pais = paises[0]
    for pais in paises:
        if pais['poblacion'] > mayor_pais['poblacion']:
            mayor_pais = pais

    return mayor_pais

def pais_menor_poblacion(paises):
    menor_pais = paises[0]
    for pais in paises:
        if pais['poblacion'] < menor_pais['poblacion']:
            menor_pais = pais

    return menor_pais
```

2. Cálculo de Media Aritmética (Promedio)

La media aritmética, o promedio, se obtiene al sumar todos los datos de un conjunto y dividir este resultado entre el número total de datos. En nuestro programa, aplicamos este concepto para obtener el "Promedio de población" y el "Promedio de superficie".

El procedimiento que utilizamos fue el siguiente:

1. Inicializamos una variable acumulador_total en 0.

2. Iteramos sobre toda la lista de países, sumando el valor (población o superficie) de cada país al acumulador_total.
3. Finalmente, dividimos el acumulador_total por la cantidad total de países (obtenida, por ejemplo, con len(lista_paises)).
4. El resultado de esa división es el valor promedio del dato para todo el conjunto.

```
def promedio(paises, campo):
    total = 0
    for pais in paises:
        total += pais[campo]

    promedio = total / len(paises)
    return promedio
```

3. Cálculo de frecuencias

Por último, para obtener la "Cantidad de países por continente", analizamos la frecuencia de los datos. La frecuencia es el número de veces que se presenta un valor en un conjunto; en este caso, contamos cuántas veces aparece cada nombre de continente.

Lo logramos de la siguiente forma

1. Creamos un diccionario vacío llamado conteo_continentes.
2. Recorremos la lista de países.
3. Por cada país, obtenemos su valor de “contionente”.
4. Si ese continente no existe como clave en nuestro diccionario, se añade como una nueva clave y se le asigna el valor 0, posteriormente se le suma 1. Si ya existe, simplemente se incrementa en 1 el valor asociado a esa clave.
5. Al finalizar el recorrido, el diccionario conteo_continentes almacena el recuento exacto de países por cada continente.

```
def paises_por_continente(paises):

    contador = {}

    for pais in paises:
```

```
        continente = normalizar_string(pais['continente'])
        contador[continente] = contador.get(continente, 0) + 1

    return contador
```

Referencias

Gómez García, A. F. (2023). *Estadística 10: Mínimo, máximo, moda, mediana, media aritmética, cuartiles* [Material de estudio]. Colegio Técnico Profesional Valle la Estrella, Departamento de Matemática. https://coned.ac.cr/images/Antologias/Academicas/10/Estad%C3%ADstica_10-M%C3%ADnimo_m%C3%A1ximo_moda_mediana_media_aritm%C3%A9tica_cuartiles.pdf

Universidad Nacional Autónoma de México. (n.d.). *Frecuencia*. B@UNAM. https://uapas1.bunam.unam.mx/matematicas/frecuencia/

Archivos CSV

Un archivo CSV (del inglés *Comma Separated Values*), como su nombre lo indica, es un archivo de texto utilizado para intercambiar y almacenar datos estructurados de forma sencilla.
Cada línea del archivo representa una fila de datos. Dentro de esa línea, se utiliza un delimitador (generalmente una coma) para separar los valores que corresponden a cada columna. A pesar de ser un archivo de texto, esta estructura permite representar datos tabulares de manera eficiente.

Usos

La principal fortaleza de los archivos CSV es su interoperabilidad. Pueden ser exportados e importados fácilmente por una gran variedad de programas, desde hojas de cálculo hasta sistemas gestores de bases de datos.
Son muy utilizados en el sector financiero, siendo un formato común para exportar transacciones. En el ámbito de la programación, también son muy útiles y se consideran un estándar para importar y exportar conjuntos de datos para su análisis.
Esta simplicidad y compatibilidad fue lo que nos permitió utilizar un archivo CSV como el núcleo de almacenamiento de nuestro Trabajo Práctico Integrador. En nuestro proyecto, el

archivo CSV es donde persisten los países con todos sus datos, actuando como la fuente de información que nuestro programa lee y gestiona.

A continuación, un ejemplo de cómo reescribimos completamente el archivo CSV utilizando la lista de países en memoria:

```
def actualizar_csv(paises):
    with open("paises.csv", "w") as archivo:

        archivo.write("nombre,poblacion,superficie,continente\n")

        for pais in paises:
            nombre = pais["nombre"]
            poblacion = pais["poblacion"]
            superficie = pais["superficie"]
            continente = pais["continente"]


archivo.write(f"{nombre},{poblacion},{superficie},{continente}\n")
```

Referencias

Adobe. (s.f.). *¿Qué es un archivo CSV y cómo se utiliza?* Adobe Acrobat. https://www.adobe.com/es/acrobat/resources/document-files/text-files/csv-file.html

Python Software Foundation. (s.f.). *csv — Lectura y escritura de archivos CSV*. Documentación de Python 3. https://docs.python.org/es/3/library/csv.html

Diseño del caso práctico

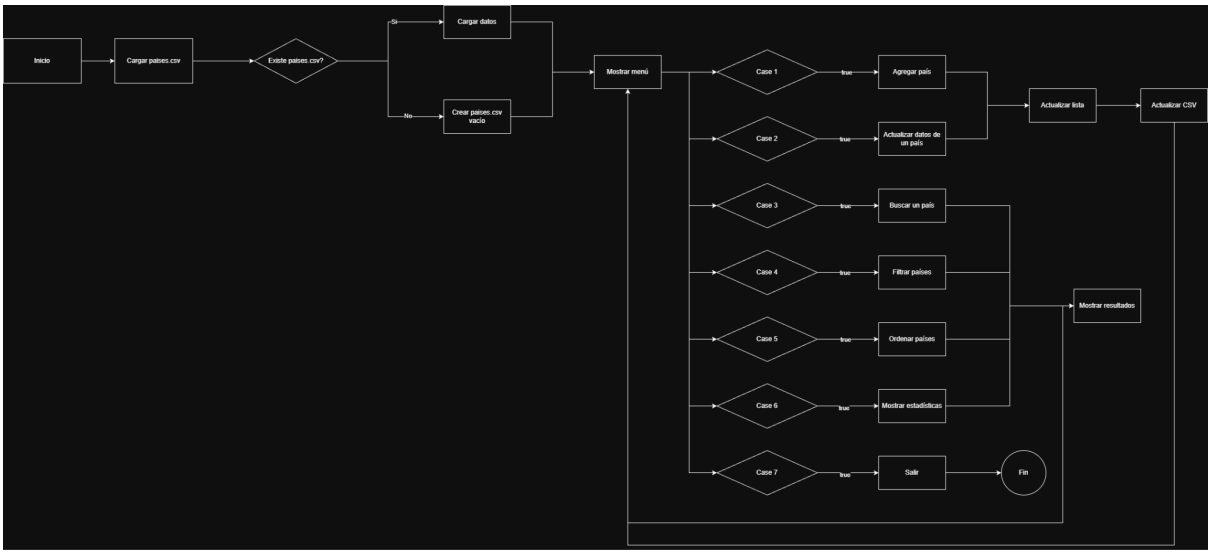
Para cumplir con los requerimientos planteados para este Trabajo Práctico Integrador, se realizó una etapa de diseño previa al código, con el fin de estructurar la solución de manera modular, facilitar la codificación y evitar errores futuros. Nos basamos en tres pilares: el modelo de datos, la modularización del código y el flujo principal del programa.

1. **Modelo de Datos**
   Para una gestión eficiente y clara de la información de los países, optamos por una estructura de lista de diccionarios.
   La lista principal funciona como el contenedor en memoria de todos los registros cargados desde el archivo CSV. Cada registro, a su vez, es un diccionario dentro de esta lista. Esta estructura facilita un acceso directo y semántico, lo que la hace ideal para las operaciones propuestas en este trabajo, como búsqueda, actualización, filtrado y ordenamiento.

2. **Modularización del código**
   Para asegurar la modularización del código y mantener una única responsabilidad por función, decidimos dividir el sistema en varios bloques, separados lógicamente por comentarios, que a su vez contienen funciones modulares.

   - Funciones de utilidad: Bloque destinado a funciones de validación de entrada del usuario y de normalización de datos, asegurando la consistencia de la información en todo el programa.
   - Bloque de gestión de archivos: Funciones encargadas de la interacción con el archivo CSV, se encargan tanto de la lectura de los registros, como de la escritura y persistencia de los cambios.
   - Bloque CRUD: En este bloque se encuentran las funciones que realizan modificaciones para actualizar la lista en memoria, o para consultar datos de la misma, por ejemplo, `agregar_pais()` o `buscar_pais()`
   - Bloque de filtrado: Contiene todas las funciones relacionadas con el filtrado según distintos criterios.
   - Bloque de ordenamiento: Funciones dedicadas a implementar lógica de ordenamiento (ascendente o descendente) sin alterar la lista original.
   - Bloque de estadísticas: Tiene como fin implementar funciones que se encarguen de analizar los datos con el fin de extraer métricas.
   - Bloque principal (main): Orquesta el flujo del programa, presentando el menú principal y derivando la ejecución a los módulos correspondientes.

Finalmente, realizamos un diagrama representando el flujo de ejecución principal del programa, con el fin de tener una guía visual antes de comenzar la codificación:



Metodología utilizada

Para llevar a cabo la codificación de la consigna, implementamos una metodología de trabajo incremental, con división de tareas en equipo y un proceso de validación constante.

Luego de definir el diseño y el diagrama de flujo, pasamos al desarrollo. Lo hicimos siguiendo un orden lógico por bloques:

1. Primero, implementamos el bloque de gestión de archivos. Esto incluyó las funciones para la lectura inicial del archivo CSV y el volcado de los datos en la estructura de lista de diccionarios que habíamos definido.
2. Una vez asegurada la carga de datos, continuamos por el bloque CRUD, donde implementamos las funciones esenciales para agregar , actualizar y buscar países.
3. Mientras desarrollamos estas funciones, notamos que necesitábamos validar y normalizar datos de entrada. Para ahorrarnos retrabajo futuro, creamos un bloque de funciones de utilidad dedicado a cumplir esta tarea.
4. Luego de tener la base funcionando, implementamos los bloques de filtrado , ordenamiento y estadísticas. En esta etapa fue clave mantener la regla de separar todo en funciones con una única responsabilidad.
5. Finalmente, implementamos el bloque *main* para orquestar todo el flujo del programa. Esta parte se encarga de presentar el menú de opciones en consola y llamar a las funciones correspondientes de los otros bloques según lo que elija el usuario.

Pruebas y validación

A lo largo de todo el desarrollo, fuimos realizando pruebas manuales para verificar el correcto funcionamiento de cada función por separado.

Una vez que finalizamos con el desarrollo, llevamos a cabo las pruebas integrales para garantizar que el sistema funcione correctamente en su totalidad. En esta etapa, nos enfocamos en probar los casos de uso y los posibles errores, como ingresar filtros inválidos o realizar búsquedas sin resultados. El objetivo era verificar que el sistema manejara estos casos con mensajes claros de éxito/error y sin interrumpir la ejecución del programa.

Resultados obtenidos

Como resultado, obtuvimos un programa en Python completamente funcional que no solo facilita la consulta y modificación de datos sobre países, sino que también garantiza la persistencia de estos en un archivo y la obtención de información clave del mismo.
Realizar una evaluación de este proceso nos permite analizar en detalle los aciertos, los desafíos y la dinámica de trabajo que tuvimos.

Lo que funcionó bien
Lo más destacable fue la implementación de la modularización. Al separar el código en funciones con responsabilidades claras y organizarlo en bloques comentados, logramos una significativa reutilización de código y una notable mejora en la legibilidad.
Otro aspecto positivo fue el trabajo en equipo, mantuvimos sincronización constante y una división de tareas eficiente que agiliza el proceso de desarrollo.

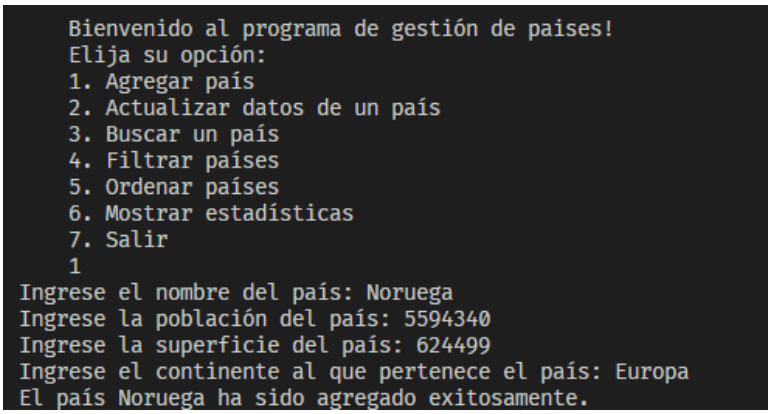
Lo que presentó un desafío
El mayor desafío que encontramos fue el diseño de la función de ordenamiento. Lograr que esta única función fuera capaz de ordenar la lista tanto por orden alfabético (como el nombre) como por valores numéricos (población o superficie), y que además permitiera al usuario elegir la dirección (ascendente o descendente), requirió una lógica más compleja de la que habíamos imaginado inicialmente.

Decisiones que cambiaron durante el desarrollo
La decisión que más provocó cambios durante la codificación fue nuestro enfoque de la modularización. A medida que desarrollamos una nueva funcionalidad, con frecuencia notamos que debíamos utilizar una lógica similar a la de una función que ya habíamos hecho. En lugar de duplicar código, optamos por refactorizar las funciones anteriores: las adaptamos para hacerlas más genéricas, asegurando que fueran compatibles con el nuevo requisito y permitiendo así su reutilización. Este proceso de adaptación y mejora de la modularización fue una constante durante el desarrollo.

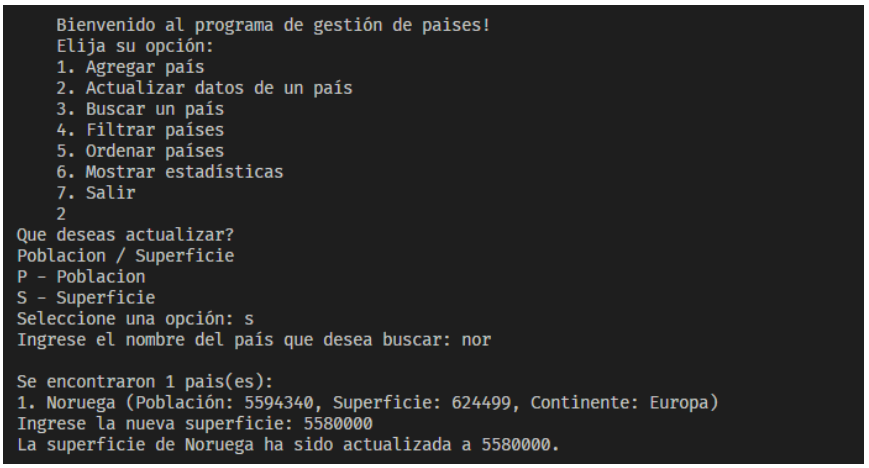
Comunicación del equipo
La comunicación del equipo fue muy próspera y fluida durante todo el proyecto. La gestión del código se realizó mediante Git, donde cada integrante trabajó en su propia rama para evitar conflictos. Para la coordinación diaria, utilizamos WhatsApp y Discord para notificarnos sobre los cambios subidos y discutir detalles puntuales. Complementamos esto reuniéndonos en llamada cada algunos días para charlar sobre lo que habíamos hecho, sincronizar el avance general y resolver de forma conjunta las dudas que tuvimos.

Capturas de pantalla de la ejecución del programa

1. **Agregar país**



2. **Actualizar los datos de un país**



3. Buscar un país

```
    Bienvenido al programa de gestión de paises!
    Elija su opción:
    1. Agregar país
    2. Actualizar datos de un país
    3. Buscar un país
    4. Filtrar países
    5. Ordenar países
    6. Mostrar estadísticas
    7. Salir
    3
Ingrese el nombre del país que desea buscar: a

Se encontraron 2 resultado(s):
- Argentina (Población: 45376763, Superficie: 2780400, Continente: América)
- Angola (Población: 35588900, Superficie: 1246700, Continente: África)
```

4. Filtrar países

```
    Bienvenido al programa de gestión de paises!
    Elija su opción:
    1. Agregar país
    2. Actualizar datos de un país
    3. Buscar un país
    4. Filtrar países
    5. Ordenar países
    6. Mostrar estadísticas
    7. Salir
    4
Ingrese por que criterio desea filtrar (C - Continente / P - Población / S - Superficie): c
Ingrese el continente por el cual desea filtrar: américa
Estos son los países del continente América:
- Argentina | Población: 45376763 | Superficie: 2780400 | Continente: América
- Bolivia | Población: 11673029 | Superficie: 1098581 | Continente: América
- Uruguay | Población: 3426260 | Superficie: 176215 | Continente: América
```

5. Ordenar países

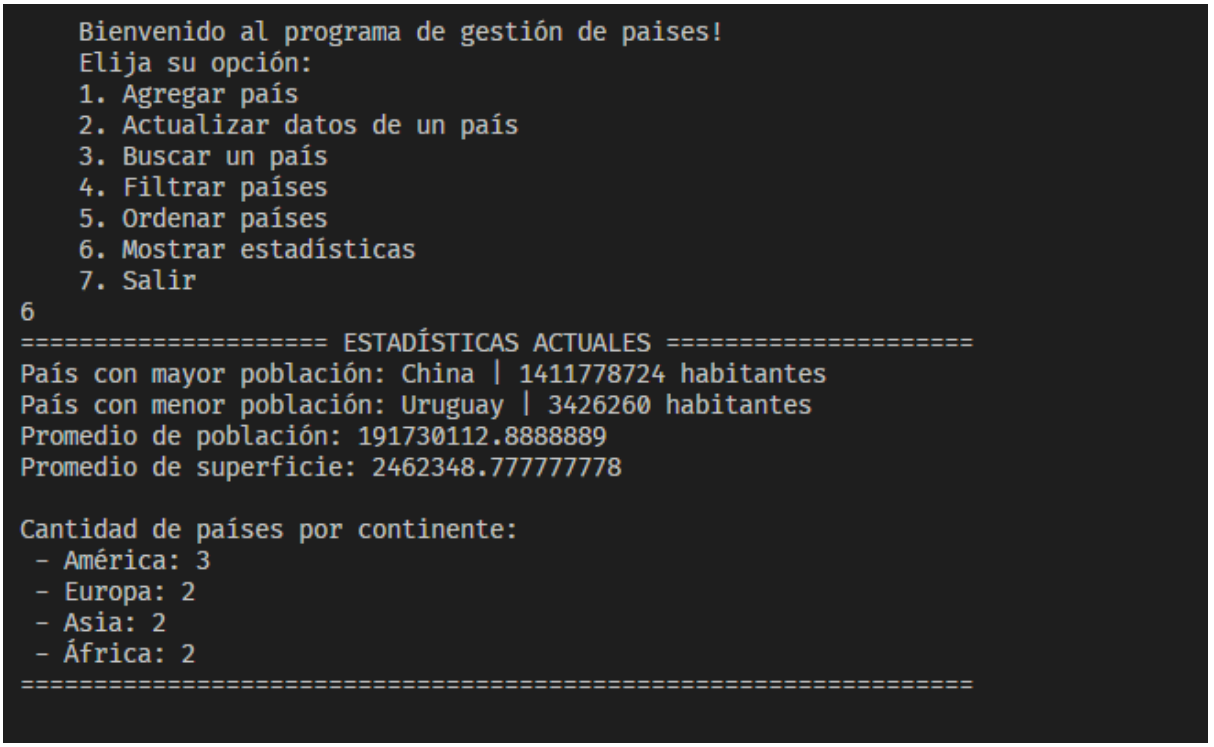
```
    Bienvenido al programa de gestión de paises!
    Elija su opción:
    1. Agregar país
    2. Actualizar datos de un país
    3. Buscar un país
    4. Filtrar países
    5. Ordenar países
    6. Mostrar estadísticas
    7. Salir
    5

    Ordenar por:
    1. Nombre
    2. Población
    3. Superficie

    2

==== LISTA ORDENADA ====
- Uruguay | Población: 3426260 | Superficie: 176215 | Continente: América
- Noruega | Población: 5594340 | Superficie: 5580000 | Continente: Europa
- Bolivia | Población: 11673029 | Superficie: 1098581 | Continente: América
- Zambia | Población: 18920000 | Superficie: 752612 | Continente: África
- Angola | Población: 35588900 | Superficie: 1246700 | Continente: África
- Argentina | Población: 45376763 | Superficie: 2780400 | Continente: América
- Francia | Población: 67413000 | Superficie: 551695 | Continente: Europa
- Japón | Población: 125800000 | Superficie: 377975 | Continente: Asia
- China | Población: 1411778724 | Superficie: 9596961 | Continente: Asia
```

6. **Mostrar estadísticas**



Conclusiones

01. Aprendizajes

Como primer punto, aprendimos acerca del uso del módulo .csv de Python , una herramienta fundamental para la persistencia de datos que nos permitió leer y actualizar el dataset de países. Más allá de esta herramienta específica, este proyecto nos ayudó a comprender cómo modelar datos del mundo real utilizando estructuras de datos de Python, utilizando una lista de diccionarios para gestionar la colección de países de manera eficiente.

Aprendimos a aplicar la modularización , no solo en teoría, sino en la práctica, refactorizando nuestro código para crear funciones genéricas y reutilizables. Este enfoque fue un pilar del proyecto, como se ve en la separación del código en bloques lógicos (gestión de archivos, CRUD, filtrado, etc.), lo cual mejoró enormemente la legibilidad y el mantenimiento. Finalmente, implementamos algoritmos desde cero, como el Ordenamiento Burbuja y los cálculos para estadísticas básicas (máximos, mínimos, promedios y frecuencias),con el fin de conocer cómo manipular y analizar colecciones de datos.

02. La importancia de estas herramientas

Las herramientas que utilizamos demostraron ser cruciales para el desarrollo de software. Las listas y diccionarios son fundamentales en Python porque permiten estructurar información compleja de una manera intuitiva y accesible , facilitando operaciones como búsquedas y actualizaciones.
Las funciones son la base de un código limpio y escalable, estas nos permitieron encapsular la lógica, evitar repetir código y aislar responsabilidades.
El módulo CSV es vital, ya que es un formato universal que permite a nuestro programa interactuar con hojas de cálculo y bases de datos. Por último, herramientas de colaboración como Git fueron indispensables para el trabajo en equipo, permitiéndonos gestionar versiones, trabajar en paralelo en ramas separadas y evitar conflictos.

03. Justificación del uso de las herramientas y ejemplos de uso

Cada herramienta se eligió para resolver un problema específico

- **Lista de Diccionarios**: Se usó para mantener los datos en memoria. La lista principal (paises) contenía un diccionario por cada país. Esta estructura fue ideal, como se ve en la función crear_pais(), que genera estos diccionarios, o en agregar_pais(), que los añade a la lista.
- **Módulo csv**: Se justificó por la necesidad de persistir los datos. Lo usamos en carga_inicial() (con csv.reader) para leer el archivo paises.csv al iniciar, y en actualizar_csv() (con csv.DictWriter) para reescribir el archivo después de cualquier modificación (como agregar o actualizar un país).
- **Funciones**: Se usaron para cumplir con el principio de responsabilidad única. Por ejemplo, las funciones pedir_num() que se encarga solo de validar una entrada numérica, filtrar_continente() se enfoca solo en el filtrado unicamente por continente y promedio() solo calcula una media.
- **Ordenamiento Burbuja**: Aunque no es el más eficiente, su uso se justifica por su valor didáctico. Nos permitió implementar un algoritmo de ordenamiento manualmente en la función ordenar_paises() y entender su lógica interna.

04. Organización del trabajo entre los miembros del equipo

La organización del equipo fue un factor clave para el éxito del proyecto. La comunicación fue constante y fluida, utilizando WhatsApp y Discord para la coordinación diaria y notificar cambios. Complementamos esto con reuniones periódicas por llamada para sincronizar el avance general y resolver dudas de forma conjunta.

Para la gestión del código, adoptamos un flujo de trabajo basado en Git, donde cada integrante trabajó en su propia rama para desarrollar funcionalidades específicas posteriormente enviandolas al main a partir de pull requests. Esto nos permitió evitar conflictos y mantener un control de versiones ordenado.

Repositorio GitHub