UNIVERSIDAD NACIONAL TECNOLÓGICA, FACULTAD REGIONAL DE MENDOZA

PROGRAMACIÓN I

Trabajo Práctico Integrador: Gestión de Datos de Países en Python: filtros, ordenamientos y estadísticas.

PROFESORES

Rigioni Cintia, Hualpa Ramiro

ESTUDIANTES

Agüero Lautaro, Limina Matias

29 de octubre del año 2025

Índice

Listas en Python

¿Qué es una lista en Python?

Una **lista** es una colección de elementos almacenados en una sola variable. Es uno de los cuatro tipos de datos integrados en Python para almacenar colecciones:

- **Lista**: ordenada, modificable, permite duplicados
- **Tupla**: ordenada, inmodificable, permite duplicados
- **Conjunto (Set)**: no ordenado, no indexado, sin duplicados
- **Diccionario**: no ordenado (aunque desde Python 3.6+ conserva el orden de inserción), pares clave-valor.

Propiedades clave de las listas

Propiedad	**Descripción**
Ordenada	Los elementos mantienen el orden en que fueron agregados
Indexada	Cada elemento tiene una posición: Mi_lista[0] es "manzana"
Modificable	Puedes modificar, agregar o eliminar elementos
Permite duplicados	Se pueden repetir valores dentro de la lista

Caso de implementación:

Para la manipulación de los datos del CSV, implementamos dos métodos complementarios. Primero, desglosamos el archivo en **listas para gestionar cada columna por separado** cuando la lógica del

programa así lo requería. Segundo, empleamos una **lista de diccionarios** para tener un acceso integral y referenciado por clave a la totalidad del contenido del CSV.

Diccionarios en Python

Un **diccionario** se utiliza para almacenar valores de datos en pares **clave:valor**.

Es una colección que es **ordenada** (a partir de Python 3.7), **modificable** y **no permite duplicados**.

Los diccionarios se escriben con **llaves** {}, y cada elemento tiene una clave y un valor.

Características de los diccionarios

Propiedad	**Descripción**
Ordenados	Desde Python 3.7, los elementos mantienen el orden de inserción
Modificables	Se pueden cambiar, agregar o eliminar elementos
Sin duplicados	No se pueden tener dos claves iguales; si se repite una clave, se sobrescribe

¿Ordenado o no ordenado?

- **Python 3.7 en adelante**: los diccionarios son **ordenados**.
- **Python 3.6 y versiones anteriores**: los diccionarios son **no ordenados**.

Ordenado significa que los elementos tienen un orden definido que no cambia.

No ordenado significa que no se puede acceder a los elementos por índice.

Modificable

Puedes cambiar, agregar o eliminar elementos después de crear el diccionario.

No se permiten claves duplicadas

Si se repite una clave, el valor anterior se sobrescribe:

```
Diccionario = {
  "marca": "Ford",
  "modelo": "Mustang",
  "año": 1964,
  "año": 2020
}
print(Diccionario)  # El valor de "año" será 2020
```

Caso de implementación:

En nuestra arquitectura, la estructura de diccionario es un componente vital. Nos permite tratar cada fila del CSV como un **registro de datos mapeado (clave-valor)**, donde las claves son los encabezados de las columnas. Esto nos otorga un acceso directo y semántico a cada dato, lo cual es fundamental para el procesamiento y la lógica de negocio durante la ejecución del programa.

Funciones en Python

Una **función** es un bloque de código que solo se ejecuta cuando se llama.

Puedes pasarle datos a una función, conocidos como **parámetros**.

Una función puede **devolver datos** como resultado.

¿Parámetros o argumentos?

Los términos **parámetro** y **argumento** pueden usarse para referirse a lo mismo: la información que se pasa a una función.

Desde la perspectiva de una función:

- Un **parámetro** es la variable que aparece entre paréntesis en la definición de la función.
- Un **argumento** es el valor que se envía a la función cuando se llama.

Caso de implementación:

Las funciones cumplen un rol clave a la hora de poder modularizar el código , dividiendo cada una de sus funcionalidades para su reuso dentro del programa y para facilitar la lectura del código

Condiciones y sentencias if en Python

Python admite las condiciones lógicas habituales de las matemáticas:

- **Igual a**: a == b
- **Distinto de**: a != b
- **Menor que**: a < b
- **Menor o igual que**: a <= b

- **Mayor que**: a > b
- **Mayor o igual que**: a >= b

Estas condiciones se pueden usar de varias formas, pero lo más común es en **sentencias if** y **bucles**.

Sentencia if

Una sentencia if se escribe usando la palabra clave if:

```
a = 33

b = 200

if b > a:

  print("b es mayor que a")
```

En este ejemplo usamos dos variables, a y b, para comprobar si b es mayor que a. Como b es 200 y a es 33, se imprime en pantalla que “b es mayor que a”.

Identación

Python **depende de la indentación** (espacios en blanco al comienzo de una línea) para definir el alcance del código. Otros lenguajes usan llaves {} para esto.

Ejemplo incorrecto (sin indentación):

```
a = 33

b = 200

if b > a:
```

```
print("b es mayor que a")  # Esto generará un error
```

elif

La palabra clave elif es la forma de Python de decir: “si las condiciones anteriores no fueron verdaderas, prueba esta”.

```
a = 33

b = 33

if b > a:

  print("b es mayor que a")

elif a == b:

  print("a y b son iguales")
```

Como a es igual a b, la primera condición no se cumple, pero la condición elif sí, por lo que se imprime “a y b son iguales”.

else

La palabra clave else captura cualquier caso que no haya sido cubierto por las condiciones anteriores.

```
a = 200

b = 33

if b > a:
```

```
  print("b es mayor que a")

elif a == b:

  print("a y b son iguales")

else:

  print("a es mayor que b")
```

Aquí, como a es mayor que b, ni if ni elif se cumplen, así que se ejecuta el bloque else.

También puedes usar else sin elif:

```
a = 200

b = 33

if b > a:

  print("b es mayor que a")

else:

  print("b no es mayor que a")
```

Caso de implementación:

La estructura if es un pilar fundamental en nuestro código para la **gestión del flujo de control**. La empleamos para definir el comportamiento del programa ante situaciones específicas, permitiéndonos ejecutar bloques de código de manera condicional. Su omnipresencia se debe a que ofrece un método

claro y directo para **evaluar distintas condiciones** y asegurar que la ejecución se alinee con los requisitos deseados.

Funciones de python: sorted()

La función sorted() devuelve una lista ordenada del objeto iterable especificado. Puedes especificar un orden ascendente o descendente. Las cadenas de texto se ordenan alfabéticamente y los números se ordenan numéricamente.

Caso de implementación:

En nuestro programa, integramos funciones lambda para gestionar el ordenamiento de los datos. Esta capacidad nativa de Python nos permite **definir dinámicamente la clave (el campo del CSV) por la cual se debe ordenar** el conjunto de datos, basándonos en los criterios que el usuario solicita.

Estadísticas

Caso de Implementación: Módulo de Estadísticas

El módulo de estadísticas (Estadisticas.py) utiliza los conceptos de listas de diccionarios, funciones, bucles, y la función nativa max() y min() con funciones lambda para realizar análisis básicos sobre la colección de datos de países.

Hallar País con Mayor y Menor Población

Función: estadistica_mayor_menor_poblacion(paises)

Principio Clave: Se utiliza la función max() y min() de Python sobre la lista de diccionarios (paises).

Uso de lambda: Para especificar cómo comparar los elementos (diccionarios), se usa el argumento key con una función lambda x: x["poblacion"]. Esto le indica a max/min que debe basar su comparación únicamente en el valor asociado a la clave "poblacion" de cada diccionario.

Ejemplo:

```
pais_mayor = max(paises, key=lambda x: x["poblacion"])
pais_menor = min(paises, key=lambda x: x["poblacion"])
```

Calcular Promedio de Población y Superficie

Funciones: estadistica_promedio_poblacion(paises) y estadistica_promedio_superficie(paises)

Principio Clave: El promedio se calcula como la Suma Total de la propiedad (población o superficie) dividida por la Cantidad Total de países.

Uso de sum() y Comprensión de Listas/Generadores: Se aprovecha la función sum() de Python junto con una expresión generadora para sumar eficientemente los valores de todos los diccionarios.

Ejemplo (Población):

```
total_poblacion = sum(p.get("poblacion", 0) for p in paises)
cantidad_paises = len(paises)
promedio = total_poblacion / cantidad_paises
```

Uso de len(): La cantidad de elementos en la lista (paises) se obtiene con len(paises).

Contar Países por Continente

Función: estadistica_cant_paises_por_continente(paises)

Principio Clave: Se utiliza un diccionario (continentes) para llevar la cuenta, donde la clave es el nombre del continente y el valor es la cantidad de países en ese continente.

Uso de Bucles for e if:

Se itera sobre la lista de países mediante un bucle for.

Dentro del bucle, se usa el método get() del diccionario con un valor por defecto para incrementar el contador de un continente existente o inicializarlo en 1 si es la primera vez que se encuentra.

Ejemplo:

```
continentes = {}
```

```
for p in paises:
    cont = p.get("continente", "Desconocido").capitalize()
    continentes[cont] = continentes.get(cont, 0) + 1
```

Uso de capitalize(): Se utiliza para estandarizar el nombre del continente antes de usarlo como clave.

Impresión: Un segundo bucle for itera sobre los ítems del diccionario continentes para mostrar el resultado final de la cuenta.

Archivo CSV

Un archivo **CSV** (del inglés *Comma-Separated Values*, o **Valores Separados por Comas**) es un formato de archivo de texto plano muy simple que se utiliza para almacenar datos en una estructura de tabla, similar a una hoja de cálculo o una tabla de base de datos.

Es uno de los formatos más comunes para intercambiar datos entre diferentes aplicaciones.

Características Principales

- **Texto Plano:** Puedes abrir, leer e incluso editar un archivo CSV con cualquier editor de texto básico (como el Bloc de Notas en Windows, TextEdit en Mac o Gedit en Linux).
- **Estructura:**
  - Cada **línea** del archivo representa una **fila** de la tabla.
  - Los **valores** (columnas) dentro de cada fila están separados por un **delimitador**.
- **Delimitador:** Aunque el nombre "CSV" implica que el separador es una **coma (,)**, en la práctica se usan varios delimitadores:
  - **Coma (,)**: El estándar más común (ej. Juan,Pérez,30).
  - **Punto y coma (;)**: Muy común en regiones donde la coma se usa como separador decimal (ej. Juan;Pérez;30,5). Excel en español a menudo usa este por defecto.

- **Tabulación (\t)**: A estos se les suele llamar archivos **TSV** (Tab-Separated Values).

Caso de implementación:

El archivo CSV es la pieza central de nuestro proyecto, ya que **contiene los datos maestros** que alimentan el programa. La funcionalidad completa de nuestra aplicación depende de la correcta extracción y procesamiento de esta información, tarea que logramos eficientemente mediante la librería csv de Python.

Modulo CSV

Definición y Uso

El módulo csv lee y escribe datos tabulares en formato CSV (Valores Separados por Comas).

Úsalo con dialectos, opciones de entrecomillado y clases de conveniencia como DictReader y DictWriter.

Miembros

Miembro	**Descripción**
DictReader	Lee datos CSV en diccionarios, usando la primera fila como nombres de campo (*fieldnames*).
DictWriter	Escribe diccionarios a CSV, usando las claves (*keys*) como nombres de campo (*fieldnames*).

Error	Lanzado (generado) por errores de análisis (*parsing*) o escritura de CSV.
reader	Devuelve un objeto lector (*reader*) que iterará sobre las líneas del archivo CSV.
writer	Devuelve un objeto escritor (*writer*) responsable de convertir los datos al formato CSV.
writerow()	Escribe una fila en el archivo CSV.
writerows()	Escribe múltiples filas en el archivo CSV.

Caso de implementación

En mi trabajo, la implementación del módulo csv resultó ser fundamental. Me permitió optimizar la gestión de los datos contenidos en los archivos, facilitando enormemente tanto la **modificación de registros existentes** como la **adición de nuevas entradas** de manera estructurada.

Módulo os de Python

Python tiene un módulo incorporado os con métodos

para interactuar con el sistema operativo, como crear archivos y directorios,

gestión de archivos y directorios, entrada, salida, variables de entorno, gestión de procesos, etc.

Caso de implementación:

La implementación del módulo os fue esencial para nuestra solución. Nos permitió, por un lado, **corroborar fehacientemente la existencia del archivo** antes de intentar operarlo y, por otro, **obtener dinámicamente el path** absoluto o relativo, facilitando así su localización y acceso desde el programa.

Bucles

Python tiene dos comandos de bucle primitivos:

- bucles while
- bucles for

Usa la palabra clave break para salir de un bucle. Usa la palabra clave continue para finalizar la iteración actual, pero continuar con la siguiente.

Bucles while de Python

El bucle while Con el bucle while podemos ejecutar un conjunto de sentencias siempre que una condición sea verdadera.

Bucles for de python

La palabra clave for se usa para crear un bucle "for" (bucle "para"). Se puede usar para iterar a través de una secuencia, como una lista, tupla, etc.

Caso de implementación

La implementación de bucles fue fundamental para la funcionalidad del programa. Específicamente, utilizamos bucles while para **controlar el ciclo de vida de los menús** interactivos, asegurando que permanecieran en pantalla durante su uso y se cerraran a petición del usuario. Paralelamente, los bucles for fueron indispensables para **procesar las colecciones de datos**, permitiéndonos iterar eficientemente sobre las listas que soportan distintas funciones del código.

Conclusion:

La realización de este proyecto representó un desafío significativo que trascendió la mera implementación de código. Desde la fase inicial, enfrentamos la curva de aprendizaje del sistema de control de versiones Git, una herramienta que identificamos como fundamental para la gestión moderna de proyectos. Esta etapa requirió una investigación dedicada y la realización de ejercicios prácticos para internalizar el flujo de trabajo profesional, incluyendo la gestión de ramas (*branches*), la integración de cambios (*merges*) y la resolución de conflictos.

A pesar del desafío inicial, concluimos que Git es una herramienta de un valor incalculable que optimiza drásticamente el trabajo colaborativo. La capacidad de gestionar contribuciones independientes y fusionarlas de manera ordenada fue clave para la eficiencia de nuestro equipo. Esta experiencia ha cimentado nuestro interés profesional y nos deja expectantes ante la oportunidad de aplicar estas metodologías en equipos de mayor envergadura.

En lo que respecta al desarrollo de la aplicación, el obstáculo principal fue la conceptualización inicial. Definir la arquitectura base del programa, nuestro "esqueleto central", supuso un reto que requirió una planificación cuidadosa antes de la implementación. No obstante, una vez establecida esta estructura fundacional, el flujo de trabajo adquirió una notable fluidez.

La posibilidad de implementar desarrollo en paralelo, habilitada por el control de versiones, fue un factor determinante que resultó en una optimización significativa de nuestros tiempos.

Finalmente, destacamos que la implementación de muchas de las funcionalidades deseadas nos impulsó a un proceso de investigación constante. Este proceso no solo resolvió los problemas técnicos planteados, sino que también nos permitió descubrir nuevas herramientas auxiliares y, más importante, "buenas prácticas" y patrones de diseño que demostraron ser altamente efectivos.

En resumen, este proyecto fue una experiencia formativa integral, donde combinamos la resolución de problemas algorítmicos con el desarrollo de competencias críticas en colaboración, investigación y gestión de proyectos.