UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador

“Gestión de Datos de Países en Python”

Alumnos:

Bufe, Nancy Yalila – yalilabufe@gmail.com – Comisión 2

Zamudio, Luis Alberto – luiszamudio150115@gmail.com – Comisión 14

Materia: Arquitectura y Sistemas Operativos

Profesores:

- – Comisión 2
- – Comisión 14

Fecha de entrega: 11 de noviembre de 2025

ÍNDICE

Marco Teórico

Listas

Una lista es una secuencia ordenada y mutable de elementos. Ordenada, debido a que cada elemento tiene una posición definida, y es mutable porque su contenido puede modificarse después de su creación. En ellas se pueden almacenar tantos elementos como se desee, inclusive se pueden crear listas vacías, es decir sin elementos. Se define encerrando los valores entre corchetes `[]`, separados por comas, y puede contener datos de distintos tipos (números, cadenas, booleanos, incluso otras listas). Por ejemplo:

```
frutas = ["manzana", "banana", "pera"] # Lista de cadenas
valores = [1, 2, 3, 4]                 # Lista de números
mixta = [10, "Python", True]           # Lista mixta
```

Métodos con listas:

Range: Genera una secuencia de números que van desde 0 por defecto hasta el número que se pasa como parámetro menos 1. En realidad, se pueden pasar hasta tres parámetros separados por coma, donde el primer es el inicio de la secuencia, el segundo el final y el tercero el salto que se desea entre números. Por defecto se empieza en 0 y el salto es de 1. Ejemplo:

```
#range(inicio, fin, salto)
print(list(range(5, 20, 2)))
```

Salida: `[5, 7, 9, 11, 13, 15, 17, 19]`

Slicing: Es posible acceder a los distintos elementos de una lista utilizando el índice del mismo, de manera análoga a como se realiza con los strings.

```
numeros = [10, 20, 30, 40, 50, 60]
segmento = numeros[1:4]
print(segmento)

Salida: [20, 30, 40]
```

Split: Permite crear una lista a partir de un string, utilizando un carácter específico como separador. Por defecto, utiliza los espacios para separar el string en distintos elementos de la lista.

```
frase = "Python es un lenguaje poderoso"
palabras = frase.split()      (El separador por defecto es el espacio)
print(palabras)
```

Salida: `['Python', 'es', 'un', 'lenguaje', 'poderoso']`

Las listas son mutables: ofrecen métodos como append() para añadir elementos al final, remove() para eliminar por valor, y sort() para ordenarlas. A modo de ilustración:

```
numeros = [3, 1, 4]
numeros.append(2)        # Agrega el 2 al final → [3, 1, 4, 2]
numeros.remove(1)        # Elimina el valor 1      → [3, 4, 2]
numeros.sort()           # Ordena la lista         → [2, 3, 4]
```

En resumen, las listas en Python proporcionan una estructura versátil para trabajar con conjuntos de datos ordenados, ofreciendo múltiples operaciones para manipular su contenido de forma sencilla y eficiente.

Diccionarios

Un **diccionario** es una estructura de datos complejas que almacena pares *clave–valor*, sin un orden fijo por posición. En Python, se define con llaves `{}`, donde cada clave, seguida de dos puntos, apunta a su valor asociado. Las claves deben ser únicas e inmutables (por ejemplo, cadenas o números). Por ejemplo:

```
colores = {'azul': 'blue', 'rojo': 'red', 'verde': 'green'}
print(colores['rojo'])  # Imprime 'red'
```

En este ejemplo, `'rojo'` es la clave y su valor asociado es `'red'`. Para **agregar o modificar** un par clave–valor, basta con asignarlo: `diccionario['nueva_clave'] = nuevo_valor`. Por ejemplo:

```
notas = {'Ana': 7.5, 'Luis': 8.0}
notas['Pedro'] = 9.0     # Agrega a Pedro con nota 9.0
notas['Ana'] = 8.0       # Actualiza la nota de Ana a 8.0
```

Para **eliminar** un elemento se puede usar `del diccionario[clave]` o `diccionario.pop(clave)`. También existen métodos como `keys()` y `values()` para obtener las claves o los valores respectivamente. Los diccionarios son mutables y permiten almacenar todo tipo de valores, incluso listas u otros diccionarios.

Los diccionarios resultan muy útiles cuando se requiere una asociación directa entre pares de datos (por ejemplo, palabras y sus definiciones, nombres y números de teléfono, etc.), ya que proporcionan una búsqueda muy eficiente por clave.

Funciones

Una **función** es un bloque de código reutilizable que realiza una tarea específica. En Python se define con la palabra reservada `def`, seguida del nombre de la función y paréntesis (que pueden incluir parámetros). Las funciones pueden recibir argumentos y retornar valores usando `return`. Por ejemplo:

```
def sumar(a, b):
```

```
    return a + b

print(sumar(5, 8))  # Imprime 13
```

Aquí, la función `sumar` toma dos argumentos `a` y `b`, y devuelve su suma. Python también incluye funciones integradas (*built-in*) como `print()`, `len()`, `input()`, etc., que facilitan tareas comunes. Las funciones permiten dividir un problema en partes más pequeñas, mejorando la claridad del código. Además, los parámetros pueden tener valores por defecto: por ejemplo, `def saludar(nombre="Usuario")`: imprimiría "Hola, Usuario" si no se proporciona argumento.

En resumen, las funciones permiten encapsular lógica y reutilizar código de forma eficiente. Al diseñar funciones, es recomendable que cada una realice una sola tarea específica (principio de responsabilidad única), usar nombres descriptivos para mejorar la legibilidad, y documentarlas adecuadamente. Esto mejora la claridad y mantenibilidad del código.

Condicionales

Las **estructuras condicionales** permiten que un programa tome decisiones y ejecute bloques de código solo si se cumplen ciertas condiciones lógicas. Constituyen uno de los pilares fundamentales de la programación, ya que introducen flujo de ejecución no lineal, bifurcando el camino según las evaluaciones booleanas que se realicen. En Python, las condicionales se implementan principalmente con las sentencias `if`, `elif` (else-if) y `else`.

La forma más simple es la estructura **if/else**: si la condición especificada después de `if` es verdadera (`True`), se ejecuta el bloque de código indentado debajo de ella; si es falsa (`False`), se ejecuta (opcionalmente) el bloque bajo `else`. Por ejemplo:

```
 edad = 20
if edad >= 18:
    print("Es mayor de edad")
else:
    print("Es menor de edad")
```

En este caso, el programa imprimiría "Es mayor de edad" solo si la variable `edad` es 18 o más; de lo contrario imprimiría "Es menor de edad". Solo uno de los dos bloques `print` se ejecuta dependiendo del valor de la condición (`edad >= 18`).

Cuando se necesitan evaluar múltiples condiciones encadenadas, se utiliza `elif`. La estructura completa **if/elif/else** permite manejar varios casos mutuamente excluyentes. Solo se ejecutará el primer bloque cuya condición sea verdadera, ignorando el resto. Las estructuras condicionales en Python proporcionan la capacidad de *decidir* qué hacer en función de condiciones lógicas. Bien utilizadas, permiten que los programas reaccionen de forma dinámica a diferentes situaciones o entradas, ejecutando solo aquellas acciones que correspondan según las circunstancias evaluadas.

Ordenamientos

Ordenar una lista significa reorganizar sus elementos según algún criterio (por ejemplo, de menor a mayor). Python proporciona mecanismos incorporados mucho más eficientes para ordenar. En particular, el tipo de dato lista tiene un método .sort() que ordena la lista *in place* (es decir, sin crear una nueva lista) y la función global sorted(lista) que devuelve una nueva lista ordenada a partir de un iterable dado A modo de ejemplo:

```
valores = [5, 2, 9, 1]
valores.sort()
print(valores)  # Salida: [1, 2, 5, 9]
```

Para tareas cotidianas es recomendable utilizar las funciones de ordenamiento nativas de Python por su eficiencia y simplicidad. Solo en casos educativos o muy específicos se implementan manualmente algoritmos como el de burbuja, selección, inserción u otros, principalmente con fines didácticos o para entender su funcionamiento interno.

Estadísticas básicas

Python facilita el cálculo de estadísticas simples sobre listas de datos. Por ejemplo, `min(lista)` y `max(lista)` devuelven el menor y mayor valor, respectivamente. El promedio (media aritmética) se calcula sumando todos los elementos y dividiendo por su cantidad, p.ej. `sum(lista)/len(lista)`. Para cálculos estadísticos más formales existe el módulo estándar `statistics`, con funciones como `mean()`, `median()`, `pstdev()`, etc. Por ejemplo:

```
import statistics
notas = [7, 8, 10, 5]
print(statistics.mean(notas))   # Promedio: 7.5
print(statistics.median(notas)) # Mediana: 7.5
```

Estas herramientas permiten obtener rápidamente indicadores básicos de cualquier conjunto de datos numéricos.

Archivos CSV

Un archivo CSV (Comma-Separated Values) almacena datos tabulares en texto plano, separando columnas mediante comas (u otro delimitador). Python cuenta con el módulo estándar `csv` para leer y escribir este formato. Por ejemplo, para **leer** un CSV se puede hacer:

```
import csv
with open('datos.csv', newline='') as archivo:
    lector = csv.reader(archivo)
    for fila in lector:
        print(fila)
```

Cada `fila` obtenida es una lista de valores (cadenas) correspondientes a las columnas del archivo. De modo inverso, `csv.writer` permite **escribir** filas en un archivo CSV nuevo.

Este formato es muy útil para intercambiar datos con hojas de cálculo u otros programas, y el módulo `csv` de Python simplifica su manejo correcto (gestiona comillas, saltos de línea, etc.) sin necesidad de procesar el texto manualmente.

Los archivos CSV son una forma simple pero poderosa de almacenar datos estructurados. Python, mediante su módulo csv, ofrece herramientas muy convenientes para trabajar con ellos, permitiendo leer grandes volúmenes de datos y escribir resultados de programas de manera compatible con otras aplicaciones y lenguajes.

Objetivo del Proyecto

Desarrollar una aplicación de consola en Python 3 que permita gestionar información de distintos países, aplicando los conceptos vistos en la materia: listas, diccionarios, funciones, condicionales, bucles, filtros, ordenamientos y estadísticas.

El programa debe:

- Leer y escribir datos desde un archivo CSV.
- Permitir altas, búsquedas, actualizaciones y filtrados.
- Generar estadísticas como promedios y máximos/mínimos.
- Mostrar información en pantalla con un menú de opciones.

Diseño del Caso Práctico

En el caso práctico, lo primero que hicimos fue leer bien la consigna. Si bien era parecida a los parciales anteriores, había detalles que teníamos que pensar con cuidado. Por eso, comenzamos creando la base del proyecto, y lo organizamos por módulos. De esta manera, pudimos dividirnos las tareas, y trabajar cada uno a su ritmo y en sus tiempos. Solo en la parte del menú nos dividimos distintas partes del código.

Elegimos utilizar listas, diccionarios, funciones y condicionales, ya que son herramientas que permiten organizar mejor la información y hacer que el código sea más claro y fácil de mantener.

El módulo de persistencia fue el que más temor nos generaba al principio, porque queríamos asegurarnos de que funcionara correctamente. Sin embargo, durante todo el desarrollo fuimos corrigiendo errores, y ajustando cosas que nos ayudaron a desarrollarlo bien hasta que quedó estable.

Durante todo el desarrollo fuimos corrigiendo errores, ajustando detalles y buscando formas más simples o más limpias de resolver algunas funciones.

Al final realizamos muchos testeos, agregando países, ordenándolos, modificando datos y verificando los resultados, para asegurarnos de evitar posibles errores y que el programa funcione correctamente.

Metodología Utilizada

El código se fue desarrollando de manera modular. Cada módulo tenía una función clara:

- **`main.py`**: controla el menú principal y la interacción con el usuario.
- **`persistencia.py`**: se encarga de leer y escribir los datos en el archivo CSV.
- **`validaciones.py`**: controla los errores y valida las entradas.
- **`consultas.py`**: contiene las búsquedas, filtros y ordenamientos.
- **`estadisticas.py`**: realiza los cálculos de promedios y estadísticas.

Resultados Obtenidos

El programa final cumple con todas las funcionalidades pedidas en la consigna. Permite agregar, buscar, filtrar, ordenar y modificar países, así como también obtener estadísticas generales de los datos cargados.

Se logró que el menú sea claro y fácil de usar, con mensajes que orientan al usuario y validaciones que evitan errores comunes al ingresar datos.

El manejo del archivo CSV resultó estable: los datos se leen, se procesan y se guardan correctamente. Además, las funciones implementadas permiten que el programa sea flexible y fácil de mantener o ampliar en el futuro.

En general, el trabajo fue una buena experiencia de aprendizaje. Nos permitió aplicar todo lo visto en clase y entender mejor cómo se organiza un proyecto real de programación en Python, trabajando con datos, módulos y validaciones.

Conclusiones

1. Qué aprendimos:

Aprendimos que trabajar con el código separado en módulos cambia completamente la forma de programar. Nos ayudó a organizarnos mejor, a entender qué hacía cada parte y a trabajar más tranquilos sin mezclar el trabajo de los demás. También notamos que así es mucho más fácil encontrar errores o hacer cambios sin romper el resto del programa.

2. Por qué nos parecen importantes estas herramientas:

Nos parecen importantes porque hacen que todo sea más simple y entendible. Los condicionales sirven mucho para controlar lo que hace el programa, pero las funciones, las listas y los diccionarios son las que realmente te facilitan el trabajo. Gracias a ellas entendimos mejor cómo "piensa" Python y cómo se pueden resolver las cosas de una manera más limpia y ordenada.

3. Justificación del uso de las herramientas y ejemplos:

Usamos estas herramientas porque eran las más adecuadas para lo que necesitábamos. Las listas y los diccionarios fueron clave para guardar y manejar los datos de los países, mientras que las funciones nos permitieron separar bien cada parte: la búsqueda, los filtros, las estadísticas, etc. Los condicionales y los bucles también nos ayudaron a validar datos y recorrer el CSV sin problemas. En conjunto, todo eso hizo que el programa quede más claro y fácil de entender.

4. Cómo organizamos el trabajo:

Desde el principio nos repartimos las tareas, así cada uno podía avanzar tranquilo con su parte. Nos íbamos hablando por chat para contarnos cómo veníamos, compartir los avances o pedir ayuda cuando algo no salía. No siempre coincidíamos en horarios, pero igual nos mantuvimos en contacto y nos fuimos ayudando entre todos. Al final, cuando juntamos todo, las partes encajaron bien y el programa funcionó como queríamos.

Referencias

- Universidad Tecnológica Nacional (U.T.N.). *Tecnicatura Universitaria en Programación* (2025) – Material de cátedra de la materia Programación
- Python Software Foundation (n.d.). *The Python Tutorial* (sección "Data Structures" y "Modules") – Documentación oficial de Python 3.
- Python Software Foundation (n.d.). *The Python Standard Library* (módulos `csv` y `statistics`) – Documentación de la biblioteca estándar de Python.
- Wikipedia (2023). *Ordenamiento por burbuja* – enciclopedia en línea (es.wikipedia.org).
- Wikipedia (2023). *Valores separados por comas (CSV)* – enciclopedia en línea (es.wikipedia.org).