**Descripción de la Estructura de Datos Utilizada – REST COUNTRIES**

**La estructura de datos abordada será anidada basada en listas y diccionarios**.

**La solución implementada utiliza una estructura de datos anidada basada en listas y diccionarios, diseñada para manejar información jerárquica y estructurada proveniente de la API REST Countries. Esta estructura permite organizar, procesar y analizar los datos con eficiencia, siguiendo principios de programación estructurada y popularización descritos en los apuntes de programación básica en Python.**

**Estructura Principal**

La estructura se compone de:

* **Lista de Diccionarios:** 
  + Cada país se representa como un **diccionario** con claves que describen atributos específicos (ej.: nombre, población, región). La colección completa se almacena en una **lista** para facilitar iteraciones y operaciones en lote.

**[**

**{**

**"Nombre": "Colombia",**

**"Población": 50882891,**

**"Área (km²)": 1141748,**

**"Densidad (hab/km²)": 44.56,**

**"Región": "América",**

**"Subregión": "Sudamérica",**

**"Idiomas": "Español",**

**"Monedas": "COP (Peso colombiano)"**

**},**

**...**

**]**

* **Jerarquía Anidada:**
  + **Claves Simples:** Valores atómicos (ej.: **"Nombre"**, **"Población"**).
  + **Claves Compuestas:** Listas o cadenas concatenadas para campos múltiples (ej.: **"Idiomas"** es una cadena separada por comas).

**Razones para Usar esta Estructura**

1. **Flexibilidad y Accesibilidad:**
   * Los diccionarios permiten acceso directo a valores mediante claves descriptivas (ej.: **pais["Nombre"]**).
   * Las listas facilitan iteraciones secuenciales (ej.: bucles **for** para procesar todos los países).
2. **Compatibilidad con APIs y JSON:**
   * La **API REST Countries** devuelve datos en formato **JSON**, que se mapea naturalmente a listas y diccionarios en Python.
3. **Escalabilidad:**
   * Se pueden añadir nuevas claves (ej.: **"Zona horaria"**, **"Límites fronterizos"**) sin modificar funciones existentes.
4. **Integración con Librerías Externas:**
   * **pandas:** Convierte fácilmente la lista de diccionarios en un DataFrame para análisis estadístico y exportación a Excel.
   * **matplotlib:** Extrae valores de claves específicas (ej.: **"Población"**) para generar gráficos.

**Operaciones Realizadas sobre la Estructura**

1. **Filtrado con Expresiones Regulares:**

* Usa listas por comprensión para seleccionar países cuyo nombre cumple un patrón (**re.search(patrón, nombre)**).

[pais for pais in datos\_estructurados if re.search(patrón\_regex, pais["Nombre"])]

1. **Análisis Estadístico:**

* Extrae valores numéricos (ej.: **"Población"**, **"Área (km²)"**) para calcular estadísticas básicas (**mean**, **median**, **stdev**).

valores = [pais[campo] for pais in datos\_estructurados if pais.get(campo, 0) > 0]

1. **Visualización Gráfica:**

* Separa ejes X e Y (ej.: nombres de países vs. población) para alimentar gráficos de barras o líneas.

valores\_x = [pais[campo\_x] for pais in datos]

valores\_y = [pais[campo\_y] for pais in datos]

1. **Exportación a Excel:**

* Convierte la lista de diccionarios en un DataFrame de **pandas** y lo guarda como archivo **.xlsx**.

df = pd.DataFrame(datos\_estructurados)

df.to\_excel("datos\_paises.xlsx", index=False)

**Ventajas Respecto a las Fases del Curso**

1. **Fase I (Fundamentos):**
   * Aplica variables, ciclos (**for**, **while**) y condicionales (**if-else**) para procesar datos.
   * Ejemplo:

for pais in datos\_estructurados[:5]:

print(f"- {pais['Nombre']}: {pais['Población']} habitantes")

1. **Fase II (Funciones y Algoritmos):**
   * Modulariza funcionalidades en funciones independientes (**obtener\_datos\_paises**, **analizar\_estadisticas**).
   * Usa algoritmos de ordenamiento implícitos en **sorted(..., key=...)** para mostrar top 10 países.
2. **Fase III (Aplicación en Ciencia y Tecnología):**
   * Integra bibliotecas externas (**requests, pandas, matplotlib**) para automatización y visualización.
   * Ejemplo de conexión a API:

respuesta = requests.get(url, timeout=10)

return respuesta.json()

**Desafíos y Soluciones**

1. **Datos Incompletos o Nulos:**
   * Uso de **.get(clave, valor\_predeterminado)** para evitar errores por claves inexistentes.

poblacion = pais.get("population", 0)

1. **Alta Dispersión en Estadísticas:**
   * Cálculo del coeficiente de variación (**desviación\_std / media**) para interpretar dispersión.
2. **Visualización de Grandes Conjuntos de Datos:**
   * Límite de filas mostradas en gráficos (ej.: top 10 países) para mejorar legibilidad.

**Conclusión**

La estructura de datos utilizada combina la simplicidad de listas y diccionarios con la potencia de bibliotecas modernas, cumpliendo con los objetivos pedagógicos del curso:

* **Programación Estructurada:** Uso claro de secuencias, decisiones y bucles.
* **Modularización:** Funciones reutilizables y responsabilidades bien definidas.
* **Aplicación Práctica:** Integración con APIs, análisis estadístico y visualización gráfica.

Esta base permite escalar el proyecto agregando nuevas funcionalidades (ej.: búsqueda binaria en listas ordenadas, uso de conjuntos para eliminar duplicados) mientras mantiene un código legible y mantenible.