Clase 3: Colección y Procesamiento de Datos en Python

Subtema: Listas de Más Dimensiones y Tuplas

1. Listas de Más Dimensiones (Matrices) 🧱

En Python, una **matriz** se puede representar mediante **listas de listas**. Es decir, una lista cuyos elementos también son listas. Este tipo de estructura se utiliza para almacenar datos en formato de tabla, con filas y columnas, similar a una hoja de cálculo o a una tabla de una base de datos.

Estructura de una matriz 📋

Una matriz es una colección de listas ordenadas por filas y columnas. Por ejemplo, podemos crear una matriz de 3×3 para almacenar los números del 1 al 9:

```
matriz = [
    [1, 2, 3],  # Primera fila
    [4, 5, 6],  # Segunda fila
```

```
    [7, 8, 9]   # Tercera fila
]
```

En este ejemplo:

- La **fila 0** es `[1, 2, 3]`.
- La **fila 1** es `[4, 5, 6]`.
- La **fila 2** es `[7, 8, 9]`.

Accediendo a elementos específicos de la matriz 🎯

Para acceder a un elemento específico dentro de la matriz, utilizamos **dos índices**: el primero para la **fila** y el segundo para la **columna**. Por ejemplo, para acceder al número **9**, que se encuentra en la posición `[2][2]`:

```
elemento = matriz[2][2]
print(elemento)  # Output: 9
```

Ejemplo práctico: Manejo de datos en una matriz 📝

Imagina que tienes una hoja de cálculo donde estás almacenando las ventas diarias de una tienda en una matriz. Cada fila representa un día de la semana, y cada columna representa diferentes secciones de la tienda.

```
ventas = [
    [100, 200, 150],  # Ventas del lunes
    [80,  220, 300],  # Ventas del martes
    [90,  250, 120]   # Ventas del miércoles
]

# ¿Cuánto se vendió en la segunda sección el miércoles?
ventas_miercoles_seccion2 = ventas[2][1]
print(ventas_miercoles_seccion2)  # Output: 250
```

---

2. Listas de Más Dimensiones y Sublistas Anidadas 🔄

En Python, también podemos tener **listas dentro de listas**, es decir, **listas multidimensionales**. Por ejemplo, si tenemos datos agrupados de manera más compleja, podemos tener una lista de listas de listas.

Ejemplo de sublistas anidadas 🌐

```
data = [
    [[1, 2], [3, 4]],  # Primer grupo de datos
    [[5, 6], [7, 8]]   # Segundo grupo de datos
]
```

Si quisiéramos acceder al número **6**, tendríamos que pasar por varias capas:

1. El primer índice selecciona el conjunto de listas principal (primer o segundo grupo).
2. El segundo índice selecciona la lista interna.
3. El tercer índice selecciona el valor dentro de esa lista.

Accediendo a **6**:

```
numero = data[1][0][1]
print(numero)  # Output: 6
```

---

3. Tuplas: Datos Inmutables 🔒

Una **tupla** en Python es una estructura de datos similar a una lista, pero con la diferencia de que es **inmutable**, es decir, una vez que se crea no se puede modificar.

Definición de una tupla 📦

Para crear una tupla, utilizamos paréntesis `()` o simplemente una secuencia separada por comas. Por ejemplo:

```
mi_tupla = (1, 2, 3, 4, 5)
```

Si intentamos modificar algún valor dentro de la tupla, Python generará un **error** porque las tuplas no permiten modificaciones. Aquí tienes un ejemplo:

```
mi_tupla = (1, 2, 3)
# Intentamos cambiar el primer elemento
mi_tupla[0] = 10  # Esto generará un error
```

¿Por qué usar tuplas en lugar de listas? 🤔

Las tuplas son útiles cuando queremos garantizar que los datos no serán modificados accidentalmente. Algunos casos comunes para utilizar tuplas incluyen:

- Almacenamiento de coordenadas (x, y).
- Retorno de múltiples valores desde una función.
- Representación de datos que no deberían cambiar, como los días de la semana.

Ejemplo práctico: Uso de tuplas en coordenadas de mapas 🌍

Imagina que estás desarrollando una aplicación de mapas y necesitas almacenar las coordenadas (latitud y longitud) de diferentes lugares.

```
coordenadas = (10.1234, -75.1234)  # Coordenadas de una ciuda
d
print(f"Latitud: {coordenadas[0]}, Longitud: {coordenadas
[1]}")
```

---

4. Ejercicios Prácticos 📝

Ejercicio 1: Cálculo de promedio de ventas en una tienda 🏪

Descripción: Tienes una matriz que contiene las ventas diarias de tres productos en una tienda durante cuatro días. Cada fila representa un día, y cada columna representa un producto. Calcula el promedio de ventas de cada producto.

Código:

```
ventas = [
    [100, 200, 150],  # Día 1
    [120, 210, 180],  # Día 2
    [90,  220, 160],  # Día 3
    [110, 190, 170]   # Día 4
]

# Calculamos el promedio de ventas de cada producto
promedios = [sum(columna) / len(ventas) for columna in zip(*v
entas)]
print(promedios)  # Output: [105.0, 205.0, 165.0]
```

Ejercicio 2: Tuplas para datos inmutables 📊

Descripción: Supón que necesitas representar los datos de varias personas en una lista de tuplas, donde cada tupla contiene el nombre, edad y ciudad. Imprime la información de cada persona en formato legible.

Código:

```
personas = [
    ("Carlos", 25, "Bogotá"),
    ("María", 30, "Medellín"),
    ("Pedro", 22, "Cali")
]

for persona in personas:
    nombre, edad, ciudad = persona
    print(f"{nombre} tiene {edad} años y vive en {ciudad}.")
```

Conclusiones Finales 📌

- Las **listas multidimensionales** son útiles para representar matrices y manejar datos de manera estructurada.

- Las **tuplas** son ideales para almacenar datos que no deben cambiar, como coordenadas o constantes.
- Python permite crear y manipular estos tipos de datos de forma eficiente, pero es importante saber cuándo utilizar cada uno según la situación.

Resumen en Imágenes y Códigos

✨ Usa listas cuando necesites colecciones de datos **mutables**.
💡 Usa tuplas cuando los datos deben ser **inmutables**.

Con estos ejercicios y explicaciones, ¡ya tienes una excelente guía para entender y practicar el manejo de listas multidimensionales y tuplas en Python!