📝 Apuntes de Programación: Las Tuplas (Tuples) en Python

Las Tuplas son colecciones de datos **ordenadas** e **inmutables**. A diferencia de las listas, una vez creadas, su tamaño y contenido (los elementos que las componen) no pueden ser modificados.

1. Creación y Características Esenciales

A. Formas de Declarar una Tupla

Las tuplas se definen usando paréntesis `()`, pero a menudo los paréntesis son opcionales.

Código de Ejemplo	Descripción y Uso
`tupla = (1, 2, 3)`	La forma más clara y común, usando paréntesis.
`tupla4 = 4, 5, 6`	**Creación sin paréntesis (Tupla Implícita):** Python interpreta una secuencia de valores separados por comas como una tupla.
`tupla3 = ("Maria", 28.5, False)`	Son **heterogéneas**, pueden contener distintos tipos de datos.

B. Tuplas con Un Solo Elemento (¡Cuidado!)

Para que Python reconozca una tupla con un solo elemento, es **obligatorio** incluir una coma (`comma trailing`) después del valor.

- **Incorrecto (Es un número):** `pi = (3.14159)` → Python lo trata como un simple número entre paréntesis.
- **Correcto (Es una Tupla):** `p2 = (3.14159,)` → La coma final indica que es una tupla.

2. Inmutabilidad y Mutabilidad Anidada

La característica definitoria de las tuplas es su inmutabilidad.

A. Inmutabilidad de la Tupla

Una vez declarada, no se pueden añadir, eliminar o reasignar elementos de la tupla. Esto las hace más seguras y rápidas que las listas.

B. El Caso Especial de la Mutabilidad Interna

Aunque la tupla es inmutable, si contiene un objeto mutable (como una lista), ese objeto interno SÍ puede ser modificado.

```
tupla7 = (1, 2, (1, 3, 4), 5, [1, 2, 3], 7) # Contiene una lista [1, 2, 3] en el índice
print(tupla7[4]) # Imprime [1, 2, 3]

# Se modifica el contenido de la lista interna (el elemento [1, 2, 3])
tupla7[4][0] = 23
```

```
print(tupla7[4]) # Imprime [23, 2, 3]
```

- **Explicación:** La tupla no cambia su estructura (sigue teniendo 6 elementos), solo ha cambiado el *contenido* de uno de los objetos a los que apunta.

3. Acceso, Recorrido y Conversión

A. Acceso Posicional

El acceso por índice funciona igual que en las listas, incluyendo los índices negativos.

```
print(tupla7[-2]) # Accede al penúltimo elemento.
```

B. Conversión entre Tipos

Las tuplas se pueden convertir fácilmente a y desde otros tipos de colecciones, perdiendo la inmutabilidad al convertirse en lista.

Conversión	Función	Propósito
Lista a Tupla	`tuple(lista)`	Convierte la lista en una tupla (inmutable).
Tupla a Lista	`list(tupla)`	Convierte la tupla en una lista (mutable).
Tupla a Cadena	`str(tupla)`	Convierte la tupla completa en una representación de cadena.

4. Consejos para Mejorar la Programación con Tuplas (Examen Pro-Tip)

Las tuplas son más que simples listas inmutables. Son la forma que tiene Python de agrupar datos relacionados.

🎯 Principio 1: Usar Tuplas para Agrupar Datos Relacionados

- **Cuando usar Tuplas:** Usa tuplas para agrupar elementos que son lógicamente inseparables y que representan un único registro de datos (ej. coordenadas geográficas, una fecha, las credenciales de un usuario: `(lat, long)`, `(año, mes, día)`, `(usuario, clave)`).
- **Desempaquetado (Unpacking):** Las tuplas son excelentes para el desempaquetado.

```
coordenadas = (40.41, -3.70) # Tupla
latitud, longitud = coordenadas # Desempaqueta y asigna en una sola línea.
print(f"Latitud: {latitud}, Longitud: {longitud}")
```

🎯 Principio 2: Las Tuplas son más Rápidas

- Debido a que las tuplas son inmutables, Python puede optimizar el código que las usa. En general, son ligeramente **más rápidas y consumen menos memoria** que las listas.

🎯 **Principio 3: Uso en Diccionarios (Claves)**

- La inmutabilidad es un requisito. Por lo tanto, si alguna vez necesitas usar una colección como **clave en un diccionario**, debes usar una tupla (ya que las listas son mutables y no pueden ser