¿Cuál es la diferencia entre una lista y una tupla en Python?

la principal diferencia entre una lista y una tupla es que las listas son estructuras de datos mutables, mientras que las tuplas son inmutables. Esto significa que una lista puede modificarse después de ser creada y se pueden cambiar elementos, mientras que una tupla no puede cambiarse una vez definida, además, las tuplas suelen ser un poco más rápidas que las listas en tiempo de ejecución, y se utilizan cuando necesitas una colección de datos que no debe cambiar, como coordenadas, colores fijos, o constantes. En cambio, las listas se usan cuando necesitas trabajar con datos dinámicos, que pueden cambiar a lo largo del programa.

Ejemplos:
Lista

```python
frutas = ["manzana", "banana", "naranja"]
print(frutas[0])  # manzana

# Puedes modificarla
frutas.append("pera")
frutas[1] = "kiwi"
print(frutas)  # ['manzana', 'kiwi', 'naranja', 'pera']
```

Tupla

```python
colores = ("rojo", "verde", "azul")
print(colores[1])  # verde

# No puedes modificarla
colores[1] = "amarillo"  # ❌ Error: las tuplas son inmutables
```

¿Cuál es el orden de las operaciones?

Orden de operaciones en Python:

En Python, el orden en que se evalúan los operadores sigue una jerarquía, desde los de mayor hasta los de menor prioridad:

1. **Paréntesis:** ()
   Todo lo que esté dentro de paréntesis se evalúa primero.
2. **Exponenciación:** **
   Se utiliza para elevar un número a una potencia.
3. **Signos positivos y negativos:** +x, -x
   Son operadores unarios que indican el signo de un valor.
4. **Multiplicación, división, módulo y división entera:** *, /, %, //
   Estos operadores se evalúan en este nivel de prioridad.

5. **Suma y resta: +, -**
   Se usan para sumar o restar valores.
6. **Operadores de comparación: ==, !=, >, <, >=, <=**
   Sirven para comparar valores y devuelven un resultado booleano (True o False).
7. **Operadores lógicos:**
   - not (negación)
   - and (conjunción)
   - or (disyunción)
8. **Asignación:** =, +=, -=, etc.
   Se usan para asignar valores a variables o modificar su contenido.

   Ejemplo:

```
resultado = 2 + 3 * 4 ** 2
print(resultado)  # 50
```

En este caso se evaluara de la siguiente manera
→ 4 ** 2 = 16
3 * 16 = 48
2 + 48 = 50

¿Qué es un diccionario Python?

es una estructura de datos que almacena pares clave-valor, es similar a un mapa donde cada clave se asocia a un valor, Es muy útil para guardar información organizada y acceder rápidamente a los datos a través de su clave y tienen la ventaja de ser mutables, entonces puedes cambiar cualquier elemento.

```
persona = {
    "nombre": "Ana",
    "edad": 25,
    "ciudad": "Madrid"
}

print(persona["nombre"])  # Ana
```

¿Cuál es la diferencia entre el método ordenado y la función de ordenación?

muchas veces se confunden el método ordenado con la función de ordenación porque ambos sirven para ordenar, tanto sort () como sorted () se utilizan para ordenar elementos, pero funcionan de manera diferente.

El método sort () es exclusivo de las listas y modifica la lista original, ordenando sus elementos directamente, no devuelve una nueva lista, sino que cambia la existente. Por eso, su valor de retorno es None.

En cambio, la función sorted () puede usarse con cualquier tipo de iterable (como listas, tuplas, diccionarios, etc.) y no modifica el objeto original. En su lugar, devuelve una nueva lista ordenada con los elementos del iterable.

En resumen:

- Sort () cambia la lista original y no devuelve nada.

```
frutas = ["naranja", "manzana", "pera"]
frutas.sort()
print(frutas)  # ['manzana', 'naranja', 'pera']
```

- Sorted () deja intacto el original y devuelve una nueva lista ordenada.

```
frutas = ["naranja", "manzana", "pera"]
nueva_lista = sorted(frutas)
print(frutas)       # ['naranja', 'manzana', 'pera']
print(nueva_lista)  # ['manzana', 'naranja', 'pera']
```

¿Qué es un operador de reasignación?

sirve para actualizar el valor de una variable combinando una operación matemática con una asignación.
en lugar de escribir algo como **x = x + 1**, puedes usar un operador de reasignación más corto: **x += 1** y esto nos dará el mismo resultado, este tipo de operadores hace que tu código sea más corto, más limpio y fácil de leer.

EJEMPLO:

```
x = 5
x += 3  # Ahora x es 8
x *= 2  # Ahora x es 16
print(x)
```