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

La diferencia principal es que una lista es mutable, es decir, se puede modificar después de crearla, mientras que una tupla es inmutable y no se puede cambiar. Por ejemplo:

```
# Ejemplo de lista
mi_lista = [1, 2, 3]
mi_lista.append(4)  # Ahora mi_lista es [1, 2, 3, 4]
print(mi_lista)     # Salida: [1, 2, 3, 4]

# Ejemplo de tupla
mi_tupla = (1, 2, 3)
# mi_tupla.append(4)  # Esto genera un error porque las tuplas no se pueden modificar
print(mi_tupla)     # Salida: (1, 2, 3)
```

- Las tuplas se usan cuando se quiere que los datos no cambien, como en coordenadas o fechas.
    Ejemplo: coordenada = (40.7128, -74.0060)  # Coordenadas de una ciudad

- Las tuplas pueden ser utilizadas como claves en diccionarios, mientras que las listas no, debido a su inmutabilidad.
    Ejemplo: dic = { (1, 2): "punto A", (3, 4): "punto B" }

- Las listas tienen más métodos disponibles, como remove(), pop(), insert().
    Ejemplo: mi_lista = [1, 2, 3]; mi_lista.remove(2); print(mi_lista)  # Salida: [1, 3]

- Las tuplas solo tienen los métodos count() e index().
    Ejemplo: mi_tupla = (1, 2, 2, 3); print(mi_tupla.count(2))  # Salida: 2

Buenas prácticas:
- Usa listas cuando necesites modificar, agregar o eliminar elementos.
- Usa tuplas para datos que no deben cambiar durante la ejecución del programa.
- Utiliza tuplas como claves en diccionarios si necesitas pares únicos e inmutables.

Malas prácticas:
- Usar tuplas si necesitas modificar los datos.
- Modificar listas mientras las recorres con un bucle, ya que puede causar errores inesperados.

¿Cuál es el orden de las operaciones?

Python sigue el mismo orden que en matemáticas, conocido como PEMDAS:
1. Paréntesis ()
2. Potencias **
3. Multiplicación * y División /, División entera //, Módulo %
4. Suma + y Resta -

Por ejemplo:

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

Primero se calcula 4 ** 2 (16), luego 3 * 16 (48), y finalmente 2 + 48 (50).

También existen operadores como el módulo (%) para obtener el residuo y la división entera (//) para obtener el cociente sin decimales. Ejemplo:

```
print(10 % 3)   # Salida: 1
print(10 // 3)  # Salida: 3
```

Buenas prácticas:
- Usa paréntesis para dejar claro el orden de las operaciones y mejorar la legibilidad.
- Escribe expresiones simples y fáciles de entender.

Malas prácticas:
- Escribir expresiones muy largas y sin paréntesis, lo que puede llevar a errores de interpretación.

¿Qué es un diccionario en Python?

Un diccionario es una estructura que guarda datos en pares de clave y valor. Por ejemplo:

```
persona = {'nombre': 'Ana', 'edad': 20}
print(persona['nombre'])  # Salida: Ana
```

Para acceder a un dato, se usa la clave, como persona['nombre']. Los diccionarios son útiles para organizar información relacionada y permiten buscar datos de forma rápida. Puedes agregar o modificar elementos así:

```
persona['ciudad'] = 'Madrid'  # Agrega una nueva clave
persona['edad'] = 21          # Modifica un valor existente
```

Puedes recorrer un diccionario usando un ciclo for:

```
for clave, valor in persona.items():
    print(clave, valor)
```

Buenas prácticas:
- Usa claves descriptivas y consistentes.
- Verifica que la clave existe antes de acceder a ella usando el método get() o el operador in.
- Utiliza métodos como keys(), values() e items() para trabajar con los datos.

Malas prácticas:
- Usar claves poco claras o inconsistentes.
- Acceder a claves que no existen, lo que puede causar errores KeyError.

¿Cuál es la diferencia entre el método sort() y la función sorted()?

El método sort() ordena una lista y la modifica directamente, es decir, cambia el orden de los elementos en la lista original y no devuelve nada. Por ejemplo:

```
lista = [3, 1, 2]
lista.sort()
print(lista)  # Salida: [1, 2, 3]
```

sort() permite parámetros como reverse=True para ordenar de forma descendente y key para ordenar según una función:

```
lista.sort(reverse=True)
print(lista)  # Salida: [3, 2, 1]
```

En cambio, la función sorted() devuelve una nueva lista ordenada y deja la original igual:

```
lista2 = [3, 1, 2]
nueva_lista = sorted(lista2)
print(nueva_lista)  # Salida: [1, 2, 3]
print(lista2)       # Salida: [3, 1, 2]
```

sorted() también acepta reverse y key, y puede usarse con cualquier iterable, no solo listas.

Buenas prácticas:
- Usa sort() solo si no necesitas conservar el orden original de la lista.
- Usa sorted() si necesitas mantener la lista original sin cambios.

Malas prácticas:
- Usar sort() cuando necesitas la lista original sin modificar.
- Asumir que sort() devuelve una nueva lista (devuelve None).

¿Qué es un operador de reasignación?

Un operador de reasignación sirve para actualizar el valor de una variable usando su valor actual y una operación, de forma más corta. Por ejemplo:

```
x = 5
x += 2  # Es lo mismo que x = x + 2
print(x)  # Salida: 7
```

También existen -=, *=, /=, //=, %=, **=. Son útiles para actualizar contadores o acumuladores en bucles:

```
contador = 0
for i in range(5):
    contador += i
print(contador)  # Salida: 10
```

Buenas prácticas:
- Usa operadores de reasignación para simplificar y hacer más legible el código.
- Úsalos en operaciones simples y claras.

Malas prácticas:
- Usar operadores de reasignación en expresiones complejas que dificulten la comprensión del código.
- Abusar de ellos en código poco claro o difícil de entender.