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

Python tiene dos tipos de datos principales para almacenar
secuencias de elementos: listas y tuplas.
Ambos tipos de datos pueden almacenar diferentes tipos de objetos como números, cadenas,
listas e incluso otras tuplas. En este artículo, veremos las principales diferencias entre listas y tuplas.
La diferencia más importante entre listas y tuplas es su naturaleza mutable.
Las listas son editables, lo que significa que puede cambiar, agregar o eliminar elementos después de crear la lista.

```
# Ok
my_list = [1, 2, 3, 4]
my_list[0] = 5 # Ok
```

Las tuplas, por otra parte, son inmutables.
Una vez creada una tupla, no se pueden cambiar sus elementos, agregar nuevos ni eliminar los existentes.

```
# Error
my_tuple = (1, 2, 3)
my_tuple[0] = 5 # Error
```

Sintácticamente, las listas y tuplas difieren en los símbolos utilizados para crearlas.
Las listas se crean usando corchetes [] y las tuplas se crean usando paréntesis ().

```
my_list = [1, 2, 3]
my_tuple = (1, 2, 3)
```

Dado que las tuplas son inmutables, tienen algunas ventajas de rendimiento sobre las listas.
Crear tuplas lleva menos tiempo que crear listas con datos idénticos.
Además, las operaciones con tuplas, como el acceso a elementos, suelen ser más rápidas que las operaciones con listas.

Las listas y tuplas están diseñadas para resolver diferentes problemas.
Las listas se utilizan normalmente cuando necesita almacenar una colección de elementos cuyo orden es importante y cuando estos elementos pueden cambiar mientras se ejecuta el programa. Las tuplas se utilizan para almacenar un grupo de valores que juntos representan una entidad y que no deben cambiar.

¿Cuál es el orden de las operaciones?

Las operaciones de mayor prioridad están en la parte superior, en la parte inferior,
con baja prioridad. Los cálculos se realizan de izquierda a derecha, es decir, si una expresión contiene operadores de la misma precedencia, se ejecutará primero el de la izquierda.
El operador de exponenciación es una excepción a esta regla. De los dos operadores **,
primero se ejecutará el derecho y luego el izquierdo.

```
() - Soportes
** - Exponenciación
+x, -x, ~x - Unario más, menos y negación bit a bit
*, /, //, % - Multiplicación, división, división entera, resto de división
+, - - Suma y resta
<<, >> - Cambios de bits
& - poco y
```

^ - OR exclusivo bit a bit (XOR)
| - poco O
==, !=, >, >=, <, <=, is, is not, in, not in - Comparación, verificación de identidad, verificación de ocurrencia
not - lógico NO
and - lógico Y
or - Lógico O

Parans () -> Exponents ** -> Multiplication * -> Division / -> Addition + -> Subtraction -

```
total = (444 + 545) ** 10 / 55 * 666 + (1443 - 8843)
             1       3    4    5     6       2
```

¿Qué es un diccionario Python?

Los diccionarios son un tipo de datos integrado para almacenar objetos.
Se utilizan para asociar un objeto, llamado clave, con otro, llamado valor.
Esta vinculación se llama mapeo. El resultado del mapeo será un par clave-valor.
Los pares clave-valor se agregan al diccionario. Luego puede buscar
la clave en el diccionario y obtener su valor correspondiente.
Sin embargo, no puede, por el contrario, utilizar el valor para encontrar la clave.
Los diccionarios, al igual que las listas, son mutables.
Es decir, puede agregarles nuevos pares clave-valor.
Su utilidad radica en las relaciones entre claves y valores;
hay muchas situaciones en las que será necesario almacenar datos en pares.
Al crear un diccionario, debe usar llaves, separar la clave del valor
con dos puntos y separar los pares clave-valor con comas.
A diferencia de las tuplas, si solo tiene un par clave-valor, no necesita una coma después.
El valor en el diccionario puede ser cualquier objeto.
A diferencia de un valor de diccionario, una clave de diccionario debe ser inmutable.
Una clave de diccionario puede ser una cadena o una tupla,
pero no una lista o un diccionario. Puede eliminar un par clave-valor de un diccionario utilizando la palabra clave del.

```
python_dictionary = {
    'key1': 'value1',
    'key2': 'value2',
    'key3': 'value3'
  }
print(python_dictionary['key2'])
python_dictionary['key4'] = 'value4'
del python_dictionary['key1']
```

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

Método de clasificación:
- sort(): es un método integrado en Python que se utiliza para ordenar listas en su lugar.
- Modifica la lista original: Ordena los elementos de la lista original sin crear una nueva copia.
- No funciona con otros tipos de datos: sólo se aplica a listas.
Ejemplos:

```
my_list = [5, 2, 4, 1, 3]
my_list.sort()
print(my_list)  # [1, 2, 3, 4, 5]
```

Función de clasificación:
- sorted(): es una función incorporada en Python que devuelve una lista

ordenada.
- Crea una nueva copia: no cambia la lista original, pero crea una nueva lista ordenada.
- Funciona con diferentes tipos de datos: se puede aplicar a listas, tuplas, cadenas y otros objetos iterables.
Ejemplos:

```
my_list = [5, 2, 4, 1, 3]
sorted_list = sorted(my_list)
print(my_list)  # [5, 2, 4, 1, 3]
print(sorted_list)  # [1, 2, 3, 4, 5]
```

Cuándo utilizar el método de clasificación:
- Si necesita ordenar la lista en su lugar y no necesita conservar el orden original.
- Si está trabajando con una lista.

Cuándo utilizar la función de clasificación:
- Si necesita mantener el orden original de la lista.
- Si necesita ordenar un objeto que no sea una lista.
- Si necesita obtener una lista ordenada y no modificar la original.

¿Qué es un operador de reasignación?

Con esta operación no cambiamos el objeto, sino que creamos uno nuevo.

```
test_tuple_list = list(test_tuple) # create new variable and assign converted from tuple
test_tuple_list.append("add element")
test_tuple = tuple(test_tuple_list) # convert to tuple
# or
test_tuple += ('new element',) # re-assign too
```