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

Tanto la tupla como la lista son elementos en los que se pueden estructurar datos de distinta forma.
Las listas se establecen encerrando los elementos en corchetes:
`una_lista = [“hola”, “adiós”]`
y por el contrario las tuplas se definen con paréntesis
`una_tupla = (“esto”, “es”, “una”, “tupla”)`.

Ambas pueden almacenar distintos tipos de datos y la única diferencia entre ellas es que las listas son mutables mientras que las tuplas son inmutables. Esto quiere decir que mientras que a una lista podemos añadirle elementos, eliminarlos o intercambiarlos sin que cambie la “identidad” de esa lista, es decir, a ojos del sistema será la misma lista tras los cambios que realicemos en ella, las tuplas no permiten este tipo de comportamiento, es decir, una vez que definimos una tupla no podemos modifiarla y cualquier cambio que hagamos en ella hará que se cree otra tupla con otro ID, es decir, que el sistema identificará como algo distinto a la original.

¿Cuál es el orden de las operaciones?

El orden de operaciones es la organización que se debe seguir cuando en un cálculo hay distintas operaciones. Este orden establece la prioridad en la que se debe llevar a cabo cada una de estas operaciones para que el resultado sea correcto. El orden es el siguiente:

1º: Paréntesis ()
2º: Exponentes **
3º: Multiplicación * y División /
4º: Sumas + y Restas -

¿Qué es un diccionario Python?

Un diccionario Python es otro tipo de estructura de datos, como las listas y los tuples. En el caso de los diccionarios se nos permite almacenar los datos en forma de llave y valor o “key” y “value”, cada elemento del diccionario estará compuesto por una llave y un valo que va asociado a esta y cada par se separa del siguiente con una coma al final. Los elementos dentro del diccionario pueden ser cualquier tipo de dato (incluso otro diccionario).

Existen tres posibles formas de crear un diccionario en python, estas son: con paréntesis {} y con dict(). Por ejemplo:

```
my_dictionary = {
        “first”: “hola”,
        “second”: [1, 2, 3, 4, 5],
        “third”: “adiós”
}

another_dict = dict([
        (“first”, “hola”),
        (“second”, [1, 2, 3, 4, 5]),
        (“third”, “adiós”)
])
```

```
tercer_dic = dict(first= “hola”,
        second=[1, 2, 3, 4, 5],
        third=”adiós”)
```

Podemos encontrar los elementos a través de las llaves. La ventaja de los diccionarios es que te permiten trabajar con los elementos o con las llaves de manera aislada así como con ambos a la vez. También son mutables, es decir, se pueden modificar los elementos, cambiarlos, añadir y eliminarlos.

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

Existen distintas maneras de ordenar una lista, con la función sort() podemos ordenarla alfabéticamente y también en orden alfabético reverso.
Esta función afecta a la lista y la modifica. Sin embargo, el cambio no se puede guardar en otra variable para futuras operaciones y por eso cuando intentamos llamar esta nueva variable (equivalente a la anterior pero con la función sort()), en la consola aparece None, es decir, ese comando no devuelve nada por que aunque la acción se ha realizado, esta no se ha guardado en una variable como una operación estandar. Esto se debe a que python es muy particular con los tipos de valores que devuelve cuando se cambian todos los elementos dentro de una lista. Esta función se llama poníendola tras la variable: `my_variable.sort()`

Para cuando queremos ordenar una lista alfabéticamente y guardar esos valores en una nueva variable sin modificar la lista original existe el método sorted(), que tiene el mismo tipo de comportamiento que sort() pero este te permite guardar el valor modificado en otra variable. Es decir, este no altera los valores de la lista original, la deja intacta para que la podamos usar en cualquier otra parte del programa con los mismos valores. Este método se llama desde delante de la variable: `sorted(my_variable)`

¿Qué es un operador de reasignación?

Como ya hemos comentado anteriormente, las tuplas son inmutables, esto quiere decir que cuando modificamos elementos dentro del objeto, su identidad cambia y se genera otro ID distinto al anterior. Sin embargo existe la manera de incluir, modificar o eliminar elementos dentro de un tipo de dato inmutable y esto se puede conseguir gracias al método de reasignación. Esto es, creamos una variable con un nombre específico. Por ejemplo my_tuple = (“Este”, “es”, “mi tuple”) y si quisiéramos añadir al final la frase “de hoy”, deberíamos hacer lo siguiente:

`my_tuple = (“Este”, “es”, “mi tuple”)`
`my_tuple = my_tuple + (“de hoy”,)` o lo que es lo mismo usando un operador de reasingnación: `my_tuple += (“de hoy”,)`

De esta forma la nueva tupla tendría el mismo nombre que la anterior (aunque no el mismo ID) y cuando la utilicemos estará incluida la nueva frase.

Los operadores de reasignación son los operadores que se usan en este método y son abreviaturas de ecuaciones en las que se omite el valor duplicado. Para comprenderlo mejor se pueden ver los siguientes ejemplos:

`mi_num = mi_num + 4` sería `mi_num+= 4`

mi_num = mi_num - 4 sería mi_num**-=** 4
mi_num = mi_num * 4 sería mi_num***=** 4
mi_num = mi_num / 4 sería mi_num**/=** 4

Los operadores por tanto son 4: **+=, -=**, ***=, /=**