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

Tanto las listas como las tuplas son estructuras de datos que permiten almacenar colecciones de elementos.

Una variable permite almacenar datos, las tuplas y listas multitud de ellos.

Y para acceder a los elementos, en ambos se utilizan los corchetes con el número del índice que se desea seleccionar, siendo 0 el primer elemento.

La diferencia entre los dos es que una vez definidos, a las tuplas no se les puede manipular los elementos debido a que son inmutables.

Esto quiere decir que, de la misma forma que usamos los corchetes para leer elementos, con las listas se pueden usar para modificarlos, y esto con las tuplas no sucede.

Por ejemplo:

mi_lista = [“elemento1”, “elemento2”, “elemento3”]

Si ahora quisiera acceder al valor del segundo elemento puedo utilizar

print(mi_lista[1])

Y si quiero modificarlos puedo hacer lo siguiente

mi_lista[1] = “modificado”

Esto no es posible con las tuplas por su inmutabilidad.

Por lo tanto, ciertas funciones/métodos (como extend, pop o sort) que permiten manipular las listas no se pueden usar con las tuplas, es como si fueran listas de sólo lectura.

Para modificar el contenido de la tupla hay que volver a definirlo.

mi_tupla = (“elemento1”, “elemento2”, “elemento3”)

mi_tupla = (“elemento1”, “elemento2”, “elemento3”, “elemento4”)

La otra diferencia es la forma en la que se escriben, como se puede ver en el ejemplo anterior, las tuplas usan paréntesis en vez de corchetes.

¿Cuál es el orden de las operaciones?

El orden de las operaciones es igual que en matemáticas, para recordarlo se usa el acrónimo PEMDAS y van de izquierda a derecha.

P = Paréntesis
E = Exponentes
M = Multiplicación
D = División

A = Adición
S = Sustracción

Esto se traduce en que si tenemos la siguiente operación

mi_operacion = 1 * (2 + 3) ** 4 / 5 – 6

Se calcula de la siguiente manera:

1º – Paréntesis 2 + 3 = 5
2º – Exponente 5 ** 4 = 625
3º – Multiplicación 625 * 1 = 625
4º – División 625 / 5 = 125
5º – Sustracción 125 – 6 = 119

mi_operación = 119

Y al igual que en matemáticas, el orden entre multiplicación o división y adición o sustracción es indiferente.

¿Qué es un diccionario Python?

Es otra estructura de datos que permite almacenar colecciones de elementos pero esta vez en pares clave-valor.

Al igual que las listas son mutables por lo que utilizando los corchetes se pueden modificar los valores, pero a diferencia de las primeras, para hacerlo hay que utilizar la clave en vez del índice.

Los valores pueden ser cualquier cosa incluyendo otros diccionarios que quedarían anidados de forma similar a como ocurre con las listas.

Las claves en cambio tienen que ser únicas y son inmutables.

Y para distinguirlos de listas y tuplas, los diccionarios utilizan las llaves.

Ejemplo:

```
mi_diccionario = {
        “clave1”: “valor1”,
        “clave2”: “valor2”,
        “clave3”: “valor3”
        }
```

Digamos que quiero conocer qué hay en la última posición, hacemos lo siguiente:

print(mi_diccionario[“clave3”])

y el resultado seria “valor3”.

Y si quisiéramos modificarlo se haría esto:

mi_diccionario[“clave3”] = “modificado”

y el diccionario quedaría así:

```
mi_diccionario = {
        “clave1”: “valor1”,
        “clave2”: “valor2”,
        “clave3”: “modificado”
        }
```

Y aparte de para poder acceder a los valores de las claves, con los corchetes pueden añadirse más datos directamente sin necesidad de utilizar funciones extra:

```
mi_diccionario = {
        “clave1”: “valor1”,
        “clave2”: “valor2”,
        “clave3”: “valor3”
        }
```

mi_diccionario[“clave4”] = “valor4”

daría como resultado

```
mi_diccionario = {
        “clave1”: “valor1”,
        “clave2”: “valor2”,
        “clave3”: “valor3”,
        “clave4”: “valor4”
        }
```

Y para eliminar las claves se usa del

del mi_diccionario[“clave4”]

lo dejaría como estaba inicialmente.

Los diccionarios son ideales para almacenar datos de forma organizada eficientemente y sobre todo entendible por una persona, son muy flexibles y pueden utilizarse en situaciones más complejas donde una lista se quedaría corta.

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

Los métodos sort y sorted cumplen la misma funcionalidad, que es ordenar elementos.

Sin embargo sort no devuelve ningún valor, o más concretamente devuelve None cuando se asigna a una variable, puesto que sort es un método específico de las listas al cual se accede con el operador .

Esto quiere decir que si tenemos la lista

mi_lista = [“b”, “e”, “c”, “a”]

y la queremos ordenar de menor a mayor se hace lo siguiente

mi_lista.sort()

y quedaría tal que así

mi_lista = [“a”, “b”, “c”, “e”]

Si realizásemos la operación a partir de una variable, como he dicho antes, no retorna ningún valor, sort modifica la lista original:

nueva_lista = mi_lista.sort()
nueva_lista = None

Ahora, si quisiéramos conservar la lista tal y como está originalmente además de tenerla ordenada aparte, sorted es más conveniente y se puede usar en cualquier lugar porque es una función propia de Python y no algo específico de las listas, pero tiene que asignarse a una variable.

Así si cogemos la misma lista de antes y la quisieramos ordenar pero sin modificar la original haríamos lo siguiente:

mi_lista = [“b”, “e”, “c”, “a”]
nueva_lista = sorted(mi_lista)

De esta forma mi_lista se queda tal cual y tenemos nueva_lista con los mismos elementos ordenados alfabéticamente.

¿Qué es un operador de reasignación?

El operador de reasignación es, por decirlo de alguna manera, un atajo para poder escribir una operación y una asignación en la misma variable en un sólo movimiento y escribir menos.

Por ejemplo:

mi_variable = 123

Si ahora quisiera sumarle 1 a la variable usando la forma larga se haría de esta forma:

mi_variable = mi_variable + 1

Pero con el operador de reasignación se puede hacer tal que así:

mi_variable += 1

Y el resultado es el mismo sólo que es más elegante.

También se puede utilizar para concatenar texto:

mi_texto = “Ejemplo”

mi_texto += “ de concatenación”

Y el resultado sería “Ejemplo de concatenación”

Pero no porque haya utilizado el operador + es algo exclusivo de éste, se pueden utilizar el resto de operadores matemáticos (-=, *=, **=, /=, //=, %=)