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

Se puede decir que existen dos diferencias fundamentales entre una “lista” y una “tupla” en Python. La primera, desde el punto de vista de la sintaxis, es que las listas se crean mediante corchetes “[ ] ”, y en el caso de las tuplas se utilizan los paréntesis “ ( ) ”.

> Ejemplo de lista: etiquetas = ['roca', 'nieve', 'hielo']
>
> Ejemplo de tupla: escalada = (‘Manual de Escalada’, ‘Introducción a la Escalada’, ‘Conclusiones’)

Por otro lado, y quizás sea esta la diferencia más importante, es que, a nivel estructural, las listas son mutables mientras que las tuplas son inmutables. Por ello, a la hora de desarrollar código la pregunta que nos debemos hacer es: ¿voy a implementar una estructura de datos que necesito cambiar o es algo que quiero mantener igual? O lo que sería lo mimo ¿mi estructura de datos tiene que ser mutable o inmutable? Defendiendo de la respuesta seleccionaremos hacer una lista o una tupla.

Siguiendo con el análisis de las diferencias otra manera de comprenderlas es ver someramente las características propias de cada uno de este tipo de datos de Python.

En este sentido, una lista es como una matriz, es en definitiva una colección de datos. Además, en una lista si se pueden mezclar y combinar diferentes tipos de datos e incluso se pueden poner listas dentro de una lista.

Por otro lado, hay que ser conscientes de que la estructura de las tuplas es muy parecida a la de las listas, pero con las diferencias fundamentales que antes he descrito.

Por último, hay que ser conscientes que una tupla también puede contener estructuras anidadas y estas pueden ser una lista que contenga una serie de etiquetas. Un ejemplo de esto sería:

```
escalada = (‘Manual de Escalada’, ‘Introducción a la Escalada’, ‘Conclusiones’)

etiquetas = ['roca', 'nieve', 'hielo']
```

```
escalada += (etiquetas,)

print(escalada)

('Manual de Escalada', 'Introducción a la Escalada', 'Conclusiones', ['roca', 'nieve', 'hielo'])
```

2. ¿Cuál es el orden de las operaciones?

Python es un lenguaje de programación muy eficaz a la hora de trabajar cálculos numéricos. También es flexible permitiendo sumar tipos de número diferentes. Por ejemplo, si sumamos dos tipos de datos numéricos diferentes, un número flotante (12.99) mas un número entero (100) el resultado es un número flotante (112.99).

Del mismo modo, mediante Python se pueden realizar un buen número de operaciones matemáticas como, por ejemplo: restas (-), divisiones (/), divisiones de base (//) o multiplicaciones (*) entre otras. Eso sí, en Python, el orden de las operaciones de cálculo es fundamental.

Nos referimos a operaciones matemáticas y de aritmética estándar, por ello el orden de los factores de las diferentes operaciones es muy importante. En consecuencia, si no seguimos el orden establecido obtendremos resultados incorrectos. En este sentido, según la información ofrecida en la guía del curso, el orden de prelación de las operaciones en Python es el siguiente:

1º Paréntesis.

2º Exponencial.

3º Multiplicación.

4º División.

5º Suma.

6º Resta.

Este y no otro es el orden del proceso completo que utiliza Python para ejecutar los cálculos. Si lo vemos mediante un ejemplo real, los diferentes pasos del cálculo serían así:

calculo = 4 + 1 * 2 – (3 + 2) ** 2

Primer paso, suma del paréntesis: 4 + 1 * 2 – 5 ** 2

Segundo paso, exponencial: 4 + 1 * 2 – 25

Tercer paso, multiplicación: 4 + 2 – 25

Cuarto paso, suma: 6 – 25

Quinto paso, resta: - 19

3. ¿Qué es un diccionario Python?

En el lenguaje de programación Python los diccionarios son almacenes de datos con valores clave. En este sentido, en los diccionarios podemos generar no sólo elementos como en una lista, sino que podemos crear una “clave” con un valor correspondiente.

Desde el punto de vista de la sintaxis, para crear un diccionario en leguaje Python se utilizan las “{ }”. En estructuras de diccionarios largas utilizaremos múltiples renglones. Un ejemplo de todo ello sería:

```
jugadores =  {
        “Delantero” : “Messi”,
        “Medio” : “Iniesta”,
        “Defensa” : “Cucurella”,
        “Portero” : “Oblak”,
}
```

En los diccionarios, la estructura es diferente a la de una lista, aunque ambas tienen elementos en el interior. Pero en el caso de los diccionarios en vez de trabajar con un índice (“index”) como sucede con las listas, se utiliza una estructura de valor “clave”. Esto significa que tenemos que pasar por la clave para poder acceder al valor.

A modo de explicación, siguiendo el ejemplo anterior, las cuatro claves del diccionario serían: delantero, medio, defensa y portero. Por otro lado, los cuatro valores serían: Messi, Iniesta, Cucurella y Oblak.

Por otro lado, dentro de los diccionarios de Python los valores pueden ser únicos o pueden contener una lista, una tupla, una cadena, un número o incluso otro diccionario. En este sentido, un ejemplo de diccionario con una lista en su interior sería:

```
equipos = {
        “selecciones” : [“EEUU”, “Colombia”, “Ucrania”, “Ghana”]
}
```

Por último, un tema muy importante a la hora de trabajar con diccionarios de Python es el denominado “objeto de vista de diccionario” y hay que tener en cuenta que es un concepto de sintaxis de Python bastante complejo. El mismo, permite observar los “valores”, las “claves” y todos los elementos de los diccionarios. De este modo si queremos sacar las claves del diccionario que he denominado al principio de esta pregunta como “jugadores” utilizaríamos la siguiente sintaxis:

```
print(jugadores.keys())
```

Esto nos dará las claves en una lista:

```
dict_keys([‘Delantero’, ‘Medio’, ‘Defensa’, ‘Portero’])
```

A esto se le denomina objeto de vista de diccionario y específicamente existen tres opciones diferentes de “Objetos de Vista de Diccionarios”:

- dict.keys()
- dict.values()
- dict.items()

El primero nos, como hemos visto en el ejemplo anterior las claves del diccionario, el segundo los valores y el último nos da conjuntamente la “clave” y el “valor” como una tupla. Por último, decir que en Python, no podemos tratar estos “objetos de vista” como listas verdaderas y también debemos tener en

cuenta que los “objetos de vista” son dinámicos, si algo cambia dentro del diccionario, cambia para todos.

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

Algo muy común en el lenguaje de programación Python es la capacidad de ordenar los elementos. En este sentido, tomando como ejemplo la tipología de datos denominada “listas” estas ser ordenan por sus valores de índices, como en el siguiente ejemplo:

datos = [‘cadenas’, ‘numeros’, ‘tuplas’, ‘bibliotecas’]

En el ejemplo anterior de la lista denominada “datos”, según su orden por índice, cadenas sería el ‘0’, números ‘1’, tuplas ‘2’ y bibliotecas el ‘3’. Pero también hay que tener en cuenta que además del método ordenado por un índice, en otras ocasiones podemos querer ordenar los elementos por su valor alfabético.

Así, siguiendo con el ejemplo de la lista anterior y teniendo muy en cuenta que en el lenguaje Python las listas son mutables y que, si hacemos un cambio, este es permanente, si aplicamos la función “sort()” la lista se ordenada alfabéticamente:

datos.sort() sacará [‘bibliotecas’, ‘cadenas’, ‘numeros’, ‘tuplas]

Por el contrario, si queremos que la lista esté ordenada alfabéticamente, pero de modo inverso, utilizaremos la función “sort(reverse=True):

Datos.sort(reverse=True) sacará [‘tuplas’, ‘numeros’, ‘cadenas’, ‘bibliotecas’]

Por último, respecto a estas dos funciones, es importante saber que estas funciones se comportan igual tanto con cadenas alfanuméricas como también con listas compuestas por numeros.

Por otro lado, en el lenguaje Python es importante saber como ordenar una lista sin cambiar la lista original. Para hacerlo utilizaremos el método llamado “sorted”.

El mismo tiene similar comportamiento que la función “sort”, pero nos permite almacenar un valor (variable). Con “sorted” la “lista” permanecerá intacta.

Se ejecutaría de la siguiente manera:

```
puntos = [10,8,3,6,2,9,5,1]
sorted_list = sorted(puntos)
print(sorted_list)
[1,2,3,5,6,8,9,10]
```

En definitiva, a modo de resumen, la función “sort()” y “sort(reverse=True)” son una herramienta muy útil para ordenar elementos de una lista, en orden de valores ascendente y descendente; del mismo modo, con “sorted” también podemos realizar la misma tarea, pero teniendo en cuenta que nos ofrece la posibilidad de reordenar sin tocar la lista original.

5. ¿Qué es un operador de reasignación?

En lenguaje Python un operador de reasignación tiene el objetivo principal de lograr una sintaxis mucho más rápida. Nos ofrece en definitiva la capacidad de realizar un cálculo mientras realizamos una asignación. Así, esta forma de proceder, además de ser más rápida es considerada también una buena práctica en el lenguaje Python; de hecho, es la manera estándar que se utiliza para generar una suma – u otro cálculo – y luego reestablecer y reasignar el valor correspondiente.

Ampliando la información de las guías con documentación de la web, los operadores de reasignación de Python se refieren a un tipo de operador utilizado para actualizar o modificar un valor de una variable que ya existe. Con este operador no solo se asigna un valor a una variable, sino que permiten modificar el valor de una variable en función de su valor actual. A modo de ejemplo:

Modo convencional: total = 10 // Sumar “1” al total // total = total + 1 // total = 11

Operador Reasignación: total=10 // Sumar “1” al total // total += 1// total=11

El resto de operaciones serían: -= ; *= ; **= ; %= ; /=; //=.