CHECKPOINT 4

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

Hay una diferencia fundamental entre una lista y una tupla, la modificación de los elementos que las componen. En el caso de la lista, podemos modificar los elementos que la componen así como añadir o eliminar dichos elementos. Las listas son ideales cuando necesitamos una estructura de datos mutable. Por el contrario, una tupla no nos permite modificar, añadir o eliminar sus elementos. Esto es indispensable cuando necesitamos una lista que no cambie y queramos asegurarnos de que la lista no se pueda cambiar accidentalmente, es decir, la tupla nos permite garantizar la integridad de los datos.

Otra característica diferencial entre ambas es cómo hay que definirlas, la lista se define utilizando corchetes [ ] y la tupla utilizando paréntesis ( ).

Ejemplo:

```
lista = [1, 2, 3]
tupla = (1, 2, 3)
```

En el caso de la lista, cuando queremos modificar uno de sus elementos tendríamos que hacerlo de la siguiente forma:

```
lista[0] = uno
```

En este caso concreto estaríamos cambiando el primer elemento de la lista ‘1’ por ‘uno’.

¿Cuál es el orden de las operaciones?

El orden de las operaciones en Python es el mismo que se utiliza en matemáticas. Por tanto, en caso de tener en una misma operación varios elementos tendríamos que realizar en primer lugar el exponente (**), en segundo lugar la multiplicación (*) y la división (/) y en último lugar la suma(+) y la resta (-).

Vemos un ejemplo en la siguiente operación

Ejemplo:

```
operacion_matematica= 5 + 2 * 3 ** 2 / 6 - 1
```

Primero se realizaría el exponente, en segundo lugar la multiplicación, en tercer lugar la división, en cuarto lugar la suma y en último lugar la resta.

¿Qué es un diccionario Python?

Un diccionario Python es una estructura de datos que nos permite almacenar conjuntos de elementos, mejor dicho, la clave y el valor de diferentes elementos.

Los diccionarios son fundamentales cuando necesitamos asociar información de manera estructurada, es decir, cuando tenemos por ejemplo una base de datos de artículos y queremos que cada producto este asociado a su precio u otras características propias del producto. Además nos permiten conocer el valor asociado a una clave especifica de manera fácil y rápida, por lo que facilita la búsqueda. También nos da la opción de realizar

modificaciones en la misma, por ejemplo siguiendo el caso anterior, podríamos modificar el precio de un artículo si este varía.

Es importante que no haya dos claves iguales en el mismo diccionario, cada una tiene que ser única, pero los valores si pueden repetirse en más de una ocasión y pueden ser desde cadenas, números hasta tuplas.

Para definir un diccionario tendremos que emplear llaves '{ }' y separar cada clave-valor con dos puntos (':').

Ejemplo:

```
producto= {
        "nombre": "Vestido",
        "precio": 29.99,
        "disponibilidad": TRUE
}
```

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

En este caso estamos hablando de las variables 'sort( )' y 'sorted( )'. Dos variables que a simple vista parecen muy parecidas pero que tienen funciones diferentes.

La primera de ellas, la función 'sort( )' es una función de ordenación que se utiliza para ordenar cualquier variable y nos devuelve una nueva lista ordenada sin modificar la lista original. Es decir, podríamos seguir utilizando ambas listas indistintamente. Este caso es muy útil cuando queremos mantener la lista original, sin modificarla y además tener otra lista con, por ejemplo, nuevos elementos.

Por otro lado, la función 'sorted( )' hace referencia al método de ordenación especifico de las listas en Python. Esta función se utiliza para ordenar los elementos de la lista modificando la lista original. Este caso es muy útil cuando queremos ordenar una lista ya sea alfabéticamente o numéricamente.

Ejemplo:

```
lista_original = [5, 3, 2, 4, 1]
```

Si queremos tener una nueva lista con los elementos ordenados para poder seguir utilizando la anterior tendríamos que usar:

```
nueva_lista = sorted(lista_original)
```

Si por el contrario simplemente queremos tener esa lista ordenada tendríamos que utilizar:

```
lista_original.sort( )
```

En el caso de la primera variable tendríamos tanto la 'lista_original' (5, 3, 2, 4, 1) como nueva_lista (1, 2, 3, 4, 5). En el caso de sort( ) tan solo tendríamos la 'lista_original' (1, 2, 3, 4, 5)

¿Qué es un operador de reasignación?

Un operador de reasignación en Python es un atajo que nos permite realizar una operación aritmética para modificar la variable original. Para ello tendríamos varias opciones dependiendo de la operación que queramos realizar: suma ('+='), resta ('-='),

multiplicación (‘*=’), división (‘/=’), exponente (‘**=’) y módulo (‘%=’). Es decir, consiste en añadir antes del símbolo ‘=’ el símbolo de la operación que queramos realizar.

Si buscamos un caso práctico en el que los operadores de reasignación son muy prácticos, podríamos hablar de una aplicación que cuenta los pasos que damos a diario. Si queremos realizar un seguimiento de los pasos que caminamos, estamos hablando de una variable que va en aumento cada día, por tanto, necesitas añadir nuevos pasos cada día, sumarlos.

Ejemplo:

```
pasos_total = 0

pasos_lunes =7563
pasos_martes = 6539
pasos_miercoles = 9001
pasos_jueves = 25087
pasos_viernes = 2987
pasos_sabado = 3467
pasos_domingo = 8095

#En este caso si quisiéramos modificar los pasos podríamos hacer lo siguiente:

pasos_total += pasos_lunes
pasos_total += pasos_martes
pasos_total += pasos_miércoles
pasos_total += pasos_jueves
pasos_total += pasos_viernes
pasos_total += pasos_sabado
pasos_total += pasos_domingo

print(pasos_total)
(Esto nos daría las suma de los pasos de toda la semana)
```