CHECKPOINT 4

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

Aunque ambas pertenecen a las colecciones de datos, tienen diferencias importantes entre ellas.

Las listas se definen utilizando corchetes [ ] : lista = [‘a’,’b’,’c’], son mutables, se pueden añadir, modificar o eliminar elementos. Poseen métodos (append(), pop(),remove()) que permiten hacer estas operaciones de forma sencilla. Se utilizan cuando se necesita una estructura de datos flexible. Ejemplos:

```
#Crear una lista:
        lista  = [1,2,3,4]
#Añadir elementos una lista:
        lista.append(5)
#Modificar un elemento:
        lista[0] = 6
#Eliminar el ultimo elemento:
        lista.pop()
```

Las tuplas se definen utilizando paréntesis (): tupla = (‘a’,’b’,’c’), son inmutables, por lo tanto no se pueden modificar. Tambien son mas rápidas que las listas y se utilizan para almacenar datos cuyo valor no va a cambiar. Ejemplos de operaciones con tuplas:

```
#Crear una tupla
        tupla = (1,2,3,4)
#Acceder a un elemento por su índice (primer elemento)
        elemento_tupla = tupla[0]
#Obtener el tamaño de una tupla
        tamaño_tupla= len(tupla)
```

¿Cuál es el orden de las operaciones?

El orden de las operaciones matemáticas se conoce con el nombre de PEMDAS (PEDMAS también es valido) y es:

Paentesis ()

Exponente **

Multiplicacion *

Division /

Suma +

Resta –

-Ejemplo-

Operación: **resultado = (8 / 2) * (2 + 2) - 3 ** 2**

Si seguimos PEMDAS:

1. Paréntesis: **(2 + 2)** se evalúa primero y da 4.
2. Exponentes: **3 ** 2** se calcula después y da 9.
3. División y multiplicación: Se evalúan de izquierda a derecha. **(8 / 2)** da 4, y luego **4 * 4** da 16.
4. Sustracción: Finalmente, **16 - 9** da el resultado final: 7.

¿Qué es un diccionario Python?

Es una estructura para almacenar datos en formato par [clave : valor], que forma parte de las colecciones de datos.

Son mutables, por lo tanto permiten modificar sus elementos o valores.

La clave ha de ser única, mientras que el valor puede repetirse.

Se pueden anidar diccionarios dentro de diccionarios, lo cual las convirte en una estructura de datos muy flexible y potente.

Poseen métodos propios para trabajar con ellos de manera sencilla: keys(), values(), ítems().

Veamos algunos ejemplos:

```
# Crear un diccionario
        mi_diccionario = {"nombre": "Juan", "edad": 25, "ciudad": "Madrid"}
# Acceder a un valor
        valor = mi_diccionario["nombre"]
# Añadir un nuevo par clave-valor
        mi_diccionario["profesión"] = "Cocinero"
# Modificar un valor existente
        mi_diccionario["edad"] = 26
# Eliminar un par clave-valor
        del mi_diccionario["ciudad"]
# Recorrer claves y valores
        for clave, valor in mi_diccionario.items():
           print(clave, valor)
# Obtener solo las claves
        print(mi_diccionario.keys())
# Obtener solo los valores
        print(mi_diccionario.values())
# Verificar si una clave existe
        If "edad" in mi_diccionario:
```

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

La diferencia entre sort() y sorted() esta en como se aplican y trabajan con los datos.

Sort() : En un método que se aplica a objetos de tipo lista[], lo que hace es ordenar alfabéticamente la lista, modificándola. Devuelve el valor 'none'.

Sorted(): Es una función que se puede aplicar sobre cualquier objeto iterable, devuelve un nuevo objeto (copia) ordenado, pero no modifica el original

Por ejemplo:

```
#Crea una lista
        lista  = [1,2,3,4]
#Ordena la lista sin modificar la original utilizando sorted()
        lista_ordenada = sorted(lista)
#Ordenar la lista modificándola (permanentemente) con sort()
        lista.sort()
```

¿Qué es un operador de reasignación?

Es un tipo de operador que permite realizar cálculos de forma acumulativa (sumando su valor en una variable) en una sola línea. Por ejemplo:

```
numero = 6
numero *=2  seria igual a → numero = numero *2
```