**## \*\*Tuplas\*\***

**Es una estructura que puede almacenar varios elementos de distintos tipos, pero una vez declarada, no se puede modificar. Le llamamos una estructura \*inmutable\*.**

**Podemos declarar tuplas de la siguientes maneras:**

**t = (1, 2, 3)**

**t2 = ('a', 1, 2.1, [1, 2, 3])**

**t3 = tuple([1, 2, 3]) # A partir de una lista**

**Su tamaño está dado por la función len():**

print(len(t), len(t2), len(t3))

**Tiene algunas funciones básicas como count() e index():**

nums = (1, 1, 1, 2, 2, 2, 3, 3, 3)

print(nums.count(2)) # Cuenta el numero de elementos que son 2.

print(nums.index(3)) # Retorna el indice del primer 3 que encuentra.

**La igualdad de las tuplas se puede evaluar con el operador ==**

print((1, 2, 3) == (1, 2, 3)) # True / Son iguales.

print((1, 2) == (1,)) # False / No son iguales.

**Las tuplas se pueden iterar de una manera similar a las listas, pero no se pueden modificar los valores.**

**## \*\*Conversión o Casteo Entre Estructuras de Datos\*\***

**Las funciones generadores de cada estuctura se usan para convertir una estructura en otra:**

**```**

**list(), tuple()**

**```**

Veamos algunos ejemplos.

lst = [1,2,2,2,3,3,4] # Generamos una lista

print(lst)

tp = tuple(lst) # La convertimos a tupla

print(tp)

**## \*\*Comprehensions o Comprensiones\*\***

**Hay algunas maneras cortas de inicializar listas o diccionarios que llamamos comprensiones:**

**# Comprensiones de Listas**

**s = [ i for i in range(0, 11) ] # Genera una lista del 0 al 10**

**print(s)**

**t = [ i\*\*2 + 2 for i in range(-10, 11, 2) ] # Evalúa i^2 + 2 de -10 a 10 con paso de 2 y guarda el resultado como lista.**

**print(t)**

**La ventaja de usar una comprension es indicar al interpretador una operacion iterativa en una sola linea. De esta manera, se evita usar ciclos for "tradicionales", que son menos eficientes, para generar listas.**

**## \*\*Copias de Estructuras de Datos\*\***

**En muchas ocasiones es necesario crear copias de estructuras de datos. En Python, hay que usar ciertos métodos según la estructura para realizar copias. Veamos el siguiente caso para comprender como Python maneja las estructuras de datos:**

**# Creamos una lista:**

**lst = [1,2,3,4]**

**print("lst =", lst)**

**# Intentamos (ingenuamente) hacer una copia**

**lst\_copy = lst**

**# Cambiamos datos en nuestra copia**

**lst\_copy[1] = 4**

**lst\_copy[3] = 8**

**lst\_copy.extend([5,6,7,8])**

**print("lst\_copy = ", lst\_copy)**

**# Veamos que paso con la original.**

**print("lst = ", lst)**

**# Podemos verificar que es la misma estructura con el operador is:**

**print(lst\_copy is lst)**

**Vemos que al modificar la copia se modifica la original. Esto es porque al ejecutar:**

**```**

**lst\_copy = lst**

**```**

**lst\_copy guarda una referencia a lst de modo que trabajamos con la misma estructura. Esto hace muchas cosas eficientes en cuanto a uso de memoria.**

**Este comportamiento lo encontraremos en muchos contextos de Python, y es causa de muchos "bugs" cuando se comienza con este lenguaje. Por lo pronto, para estructuras de datos, lo resolvemos con el método copy. Las listas, tuplas, diccionarios, y sets tienen su método copy. Veamos lo aplicado al ejemplo de lista de arriba:**

**# Creamos una lista:**

**lst = [1,2,3,4]**

**print("lst =", lst)**

**# Hacemos una copia.**

**lst\_copy = lst.copy()**

**# Cambiamos datos en nuestra copia**

**lst\_copy[1] = 4**

**lst\_copy[3] = 8**

**lst\_copy.extend([5,6,7,8])**

**print("lst\_copy = ", lst\_copy)**

**# La original queda intacta.**

**print("lst = ", lst)**

**# Podemos verificar que no es la misma estructura con el operador is:**

**print(lst\_copy is lst)**

**\*\*Tip de Pythonista\*\***

**Los métodos copy de listas, tuplas, diccionarios y sets hace una copia "shallow" o "no profunda". ¿Qué significa? Si hay estructuras anidadas, se copian referencias a las estructuras anidadas. Veamos un ejemplo:**

**# Definimos una matriz de 2x2 como una lista que contiene dos listas.**

**M = [[1, 2], [3, 4]]**

**print("M = ", M)**

**# Copiamos**

**M\_copy = M.copy()**

**# Hacemos cambios en la copia**

**M\_copy[0][1] = 4**

**M\_copy[1][1] = 8**

**print("M = ", M)**

**print("M\_copy = ", M\_copy)**

**Esto se resuelve con la función deepcopy del módulo copy**

**import copy**

**# Definimos una matriz de 2x2 como una lista que contiene dos listas.**

**M = [[1, 2], [3, 4]]**

**print("M = ", M)**

**# Copiamos**

**M\_copy = copy.deepcopy(M)**

**# Hacemos cambios en la copia**

**M\_copy[0][1] = 4**

**M\_copy[1][1] = 8**

**print("M = ", M)**

**print("M\_copy = ", M\_copy) # La original queda intacta.**

**Para más información sobre el módulo copy, consultar: https://docs.python.org/3/library/copy.html**