Lección 08: Tuplas

Índice

Anterior

Definición

Hay cuatro estructuras de datos compuestas en Python: *Listas*, *Tuplas*, *Conjuntos* y *Diccionarios*.

Las tuplas : - Se utilizan para agrupar valores. - Son inmutables, una vez creadas no se pueden modificar, ni añadir o quitar elementos. - Se pueden ordenar. - Permite valores duplicados. - Pueden estar vacías. - Puede contener valores de distintos tipos, es decir cualquier tipo de objeto.

> **Nota:** Se puede pensar que una tupla es una lista que no se puede cambiar. INMUTABLE. Por eso, las tuplas tienen muy pocos métodos.

Se pueden crear usando el contructor tuple() o con paréntesis ().

```
# Creación de tuplas vacías

mi_tupla = () # Esto es una tupla vacía
mi_otra_tupla = tuple() # Esto es otra tupla vacía

print(mi_tupla)
print(mi_otra_tupla)

print(type(mi_tupla))
print(type(mi_otra_tupla))

# Creación de tuplas con valores iniciales

mi_tupla = (3,4,5,6, 6, 6, 6) # Pueden tener elementos repetidos. No se pueden cambiar, pero
mi_otra_tupla = ("queso", "plátano", " ", 35, 44, 2.2, None, "uno") # Puenden contener eleme

print(mi_tupla)
print(mi_otra_tupla)

print(type(mi_tupla))
print(type(mi_otra_tupla))
```

Se puede utilizar la función *len()* para saber el número de elementos de una tupla.

```
# Número de elementos de una tupla
```

```
print("Tupla:", mi_tupla)
print(f"Número de elementos: {len(mi_tupla)}")

print("Tupla:", mi_otra_tupla)
print(f"Número de elementos: {len(mi_otra_tupla)}")
```

Accediendo a los elementos de una tupla

- Se puede acceder a los elementos mediante el índice de cada uno de ellos.
- Al igual que en las cadenas el índice del primer elemento es 0.
- Se pueden utilizar índices negativos.

```
# Accediendo a los elementos de una tupla

primer_elemento = mi_tupla[0]
ultimo_elemento = mi_tupla[-1] # Se pueden utilizar índices negativos

print(f"Tupla: {mi_tupla}")
print(f"El primer elemento de la tupla es {primer_elemento}")
print(f"El último elemento de la tupla es {ultimo_elemento}")

tercer_elemento = mi_otra_tupla[2]

print(f"Tupla: {mi_otra_tupla}")
print(f"El tercer elemento de la tupla es {tercer_elemento}")
```

- Se pueden desempaquetar los elementos. > **Nota: NO** se pueden cambiar los elementos de una tupla.

```
# Desempaquetando elementos de una tupla
print(f"Tupla inicial: {mi_otra_tupla}")
primero, segundo, tercero, cuarto, quinto, sexto, septimo, octavo = mi_otra_tupla
print(f"El tercer elemento de la tupla es {tercero}")

# No se pueden cambiar los elementos de una tupla

#mi_otra_tupla[2]="globo" # Esto da ERROR
#print(f"tupla cambiada: {mi_otra_tupla}")
primero, segundo, tercero, cuarto, quinto, sexto, septimo, octavo = mi_otra_tupla
print(f"El tercer elemento de la tupla es {tercero}")
```

Accediendo a partes de una tupla - Slicing

- Se puede acceder u obtener un subconjunto de los elementos de una tupla mediante slicing. Igual que con las cadenas, utilizando [*inicial*:*final*:*pasos*].
- Por defecto, *inicial* será 0 (el primer elemento), *final* será -1 (el último elemento) y *pasos* será 1.

- Se pueden utilizar índices negativos.

```
# Accediendo a partes de una tupla - Slicing

print(f"La tupla completa es: {mi_otra_tupla}")

print(f"Dos primeros elementos: {mi_otra_tupla[:2]}")
print(f"Todos los elementos desde el tercero hasta el final: {mi_otra_tupla[2:]}")
print(f"Uno de cada dos elementos desde el primero: {mi_otra_tupla[::2]}")
print(f"Accediendo a la tupla invertida: {mi_otra_tupla[::-1]}")

# NO se puede cambiar una parte de la_tupla
# mi_otra_tupla[:2] = ["uno", "dos"]  # Esto da ERROR
# print(f"La tupla modificada es: {mi_otra_tupla}")
```

De tupla a lista y viceversa

Se puede crear una lista que contenga los elementos de una tupla y veceversa.
Se utilizan los constructores *list()* o *tuple()* según se necesite.

```
# De tupla a lista y viceversa

print(f"Esto es una tupla: {mi_otra_tupla}")
mi_otra_lista = list(mi_otra_tupla) # De tupla a lista
print(f"Esto es una lista: {mi_otra_lista}")
print(type(mi_otra_lista))

mi_otra_nueva_tupla = tuple(mi_otra_lista) # De lista a tupla
print(type(mi_otra_nueva_tupla))
```

Comprobando si un elemento está en una tupla

- Se puede utilizar el operador *in*.

```
# Comprobando si un elemento está en la tupla

globo_en_tupla = "globo" in mi_otra_tupla
print(globo_en_tupla)

none_en_tupla = None in mi_otra_tupla
print(none_en_tupla)

num_en_tupla = 35 in mi_otra_tupla
print(num_en_tupla)

num_en_tupla = 2.2 in mi_otra_tupla
print(num_en_tupla)
```

```
cabra_en_tupla = "cabra" in mi_otra_tupla
print(cabra_en_tupla)
```

Concatenar tuplas

- Se pueden concatenar tuplas utilizando el operador *+*.

```
# Concatenar tuplas

mi_tupla_completa = mi_tupla + mi_otra_tupla
print(f"Mi tupla completa es: {mi_tupla_completa}")
```

Eliminando una tupla

No se pueden eliminar elementos individuales, pero sí se puede eliminar/borrar una tupla completa utilizando *del*.

```
# Eliminar una tupla

del mi_tupla_completa

#del mi_tupla_completa # Ejecutarlo una segunda vez daría error porque ya no está definida
```

Siguiente