INFORMATORIO

TALLER DE PROGRAMACIÓN WEB

LISTAS

Subsecretaría de Empleo
Chaco Gobierno de todos

Ministerio de Producción, Industria y Empleo
Chaco Gobierno de todos

CHACO
Gobierno de todos

Tuplas

Las **tuplas** en Python, son muy similares a las listas, la principal diferencia es que las tuplas son inmutables, es decir, no se puede modificar o reordenar su contenido.

En Python los elementos de una tupla se separan por coma y pueden ir encerrados por paréntesis (**`No es necesario, pero muy habitual verlas de esta manera`**).

Ejemplos de tuplas:

```
tupla = "a", "b", 1, 3, True
```

o lo que sería igual

```
tupla = ("a", "b", 1, 3, True)
```

Para crear una tupla simplemente separamos los elementos con comas, similar a como hacíamos con listas, pero sin los [ ]

```
>>> dias = (“lunes”, “martes”, “miercoles”, “jueves”, “viernes”, “sabado”, “domingo”)
```

Si queremos crear una tupla con un solo elemento, se debe poner una coma al final sino python lo va interpretar como el elemento en sí y no un elemento de una tupla

```
>>> tupla = 5,
>>> print(tupla)
(5,)
>>> otra_tupla = (3,)
>>> print(tupla)
(3,)
>>> cinco = 5
>>> print(cinco)
5
>>> tres = (3)
>>> print(tres)
3
```

Acceder a los elementos de una tupla al igual que en las lista, también lo hacemos por medio de sus índices.

```
>>> dias =
(“lunes”,”martes”,”miercoles”,”jueves”,”viernes”,”sabado”,”domingo”)
>>> print(dias[0])
lunes
>>> print(dias[1])
martes
>>> print(dias[2])
miercoles
```

También podremos al igual que en las listas obtener los elementos con [:]

```
>>> dias[2:4]
("miercoles", "jueves")
```

Varias de las funciones que vimos en listas como len(), min(), max() y sum() por ejemplo, también funcionan en tuplas.

```
>>> numeros = (6,7,2)
>>> longitud = len(numeros)
>>> print(longitud)
3
>>> minimo = min(numeros)
>>> print(minimo)
3
>>> maximo = max(numeros)
>>> print(maximo)
7
```

También contamos con los métodos count() e index()

```
>>> colores = ("azul", "verde", "rojo", "amarillo", "azul")
>>> colores.count("azul")
2
>>> colores.count("rojo")
1
>>> colores.index("amarillo")
3
```

La principal diferencia que veremos frente a las listas será cuando tratemos modificar algún elemento de la tupla.

```
>>> tupla = ("a","b","c")
>>> tupla[1] = "y"
TypeError: 'tuple' object does not support item assignment
```

Tratar de modificar algún valor nos producirá un error. Recordándonos que las tuplas no son mutables, no se pueden modificar.

Tampoco contaremos con métodos como append(), insert() o sort() que si teníamos en las listas que sí podían ser modificadas.

Empaquetar y Desempaquetar Tuplas
Así como podemos asignar varios elementos en una tupla a una variable (empaquetar)

```
>>> tupla = 10, 20, 30
```

También podemos desempaquetar una tupla en distintas variables

```
>>> x, y, z = tupla
>>> print(x)
10
>>> print(y)
20
>>> print(z)
30
```

Para desempaquetar una tupla, la cantidad de variables tiene que ser la misma cantidad que elementos haya en la tupla de lo contrario nos dará error.
Como por ejemplo:

```
>>> x,y = ("a","b","c")
ValueError: too many values to unpack
```

Podríamos usar tuplas por ejemplo para intercambiar valores entre variables, sin necesidad de usar una variable auxiliar:
por ejemplo si queremos el valor de una variable "a" y pasarla a una variable "b", y el contenido de "b" pasarlo a "a"

usando una variable temporal se podría hacer:

```
>>> temp = a
>>> a = b
>>> b = temp
```

pero con tuplas podríamos crear una tupla con los valores (a,b) y desempaquetarlo en las variables "b" y "a" de la siguiente manera:

```
>>> b, a = (a, b)
```

Tuplas como retorno de una función
Las funciones también pueden devolver múltiples valores en forma de tupla.

Por ejemplo

```
def multiples_valores():
      return "Hola Mundo", 1, True, 25.6
```

```
>>> resultado = multiples_valores()
>>> a = resultado[0]
>>> b = resultado[1]
>>> c = resultado[2]
>>> d = resultado[3]
>>> print(a)
>>> print(b)
>>> print(c)
>>> print(d)
```

```
>>> x, y, w, z = multiples_valores()
>>> print(x)
>>> print(y)
>>> print(w)
>>> print(z)
```

Crear listas a partir de Tuplas, y tuplas a partir de listas

con la función list() podemos crear listas a partir de tuplas

```
>>> tupla = (2,4,6)
>>> lista = list(tupla)
>>> print(lista)
[2, 4, 6]
```

también poder usarla para crear listas a partir de una cadena de caracteres

```
>>> lista = list("hola mundo")
>>> print(lista)
['h', 'o', 'l', 'a', ' ', 'm', 'u', 'n', 'd', 'o']
```

Similarmente podemos crear tuplas a partir de listas con tuple()

```
>>> lista = ["a","b","c"]
>>> tupla = tuple(lista)
>>> print(tupla)
(‘a’,’b’,’c’)
```

también desde una cadena

```
>>> tupla = tuple("hola")
>>> print(tupla)
('h', 'o', 'l', 'a')
```

List comprehesion

La comprensión de listas (list comprehesion) es una manera concisa para crear listas a partir de iterables en una forma legible y funcionalmente eficiente.

Supongamos que necesitamos crear una lista con números del 0 al 100

Una manera de hacerlo podría ser:

```
>>> lista = [ ]
>>> for valor in range(0,101):
>>>   lista.append(valor)
>>> print(lista)
```

Pero en python podemos también hacerlo de la siguiente manera:

```
>>> lista = [ valor for valor in range(0,101) ]
>>> print(lista)
```

o por ejemplo queremos una lista que tenga 50 veces la palabra "hola":

```
>>> lista = [ "hola" for valor in range(0,51) ]
>>> print(lista)
```

o supongamos tenemos una lista de nombres

```
>>> nombres = ["juan", "pedro", "jorge","maria", "analia", "sandra"]
```

pero queremos una lista que contenga esos nombres con la primer letra en mayúscula

```
nombre_inicial_mayuscula = [ ]
>>> for nombre in nombres:
>>>     nombre_inicial_mayuscula.append(nombre.capitalize())
>>> print(nombre_inicial_mayuscula)
```

eso lo podríamos hacer así:

```
>>> nombre_inicial_mayuscula = [nombre.capitalize() for nombre in nombres]
>>> print(nombre_inicial_mayuscula)
```

Supongamos una lista de números

```
numeros = [2,5,10]
```

y queremos una lista con los triples de esos numeros

```
>>> triples = [x*3 for x in numeros]
>>> print(triples)
```

Podemos agregar condiciones, por ejemplo:

Supongamos que queremos tener todos los números pares desde el 0 al 100 en una lista:

```
>>> pares = [x for x in range(0,101) if x %2 == 0]
>>> print(pares)
```

En resumen, la sintaxis básica de una lista de comprensión es

```
[ expression for elemento in iterable if condicion ]
```