

INTRODUCCIÓN A LA PROGRAMACIÓN

CON  python

TUPLAS

Estructuras de datos

Una **estructura de datos** es la **colección de datos** que se caracterizan por su **organización** y las **operaciones** que se definen en ella.

Estructuras de datos

En general tenemos:

- cadenas
- listas
- **tuplas**
- diccionarios
- conjuntos

Nombre	Tipo	Ejemplo	Descripción
Lista	<code>list</code>	<code>[1, 2, 3]</code>	Secuencia heterogénea de datos mutable
Tupla	<code>tuple</code>	<code>1, 2, 3</code> o <code>(1, 2, 3)</code>	Secuencia heterogénea de datos inmutable
Rango	<code>range</code>	<code>range(1, 20, 2)</code>	Secuencia de enteros inmutable
Cadena de caracteres	<code>str</code>	<code>'Hola'</code>	Secuencia de caracteres inmutable
Diccionario	<code>dict</code>	<code>{'a':1, 'b':2, 'c':3}</code>	Tabla asociativa de valores únicos (clave, valor)
Conjunto	<code>set</code>	<code>{1, 2, 3}</code>	Colección sin orden de valores únicos

Tuplas

El concepto de tupla es muy similar al de lista. Aunque hay algunas diferencias menores, lo fundamental es que, mientras una lista es mutable y se puede modificar, una tupla no admite cambios y por lo tanto, es **inmutable**.


Las tuplas se definen utilizando paréntesis en lugar de corchetes.

```
>>> una_tupla = ()
>>> otra_tupla = ("a",1,True)

>>> matriz = ((1, 2, 3), (4, 5, 6))
```

Tuplas


Se caracterizan por:

- Tienen orden.
 - Pueden contener elementos de distintos tipos.
 - Son inmutables, es decir, no pueden alterarse durante la ejecución de un programa.
- 

Tuplas

El acceso a los elementos de una tupla se realiza del mismo modo que en las listas.

```
>>> vocales = ("a","e","i","o","u")  
>>> vocales[0]  
'a'
```



Tuplas vs Listas

Las tuplas a diferencia de las listas son inmutables. No se pueden añadir ni remover elementos.

```
>>> tupla = (1,2,3)
>>> tupla[1] = 5
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

Tuplas de un elemento

Hay que prestar especial atención cuando vamos a crear una tupla de un único elemento. La intención primera sería hacerlo de la siguiente manera:

```
>>> tupla_un_elemento = ("a")
>>> tupla_un_elemento
'a'
>>> type(tupla_un_elemento)
<class 'str'>
```

Tuplas de un elemento

Realmente, hemos creado una variable de tipo str (cadena de texto). Para crear una tupla de un elemento debemos añadir una coma al final:

```
>>> tupla_un_elemento = ("a",)
>>> tupla_un_elemento
('a',)
>>> type(tupla_un_elemento)
<class 'tuple'>
```

Tuplas sin paréntesis

Según el caso, hay veces que nos podemos encontrar con tuplas que no llevan paréntesis. Quizás no está tan extendido, pero a efectos prácticos tiene el mismo resultado. Veamos algunos ejemplos de ello:

```
>>> tupla_un_elemento = "papá noel",
>>> reyes_magos = "gaspar", "melchor", "baltazar"
>>> coordenadas_tenerife = 28.46824, -16.25462
```

Conversión

Para convertir otros tipos de datos en una tupla podemos usar la función `tuple()`:

```
>>> compra = ["agua","aceite","arroz"]
>>> type(compra)
<class 'list'>
>>> tuple(compra)
('agua', 'aceite', 'arroz')
>>> type(compra)
<class 'list'>
```

Esta conversión es válida para aquellos tipos de datos que sean *iterables*: cadenas de caracteres, listas, diccionarios, conjuntos, etc. Un ejemplo que no funciona es intentar convertir un número en una tupla:

```
>>> tuple(5)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'int' object is not iterable
```

Operaciones

Con las tuplas podemos realizar todas las operaciones que vimos con listas **salvo las que conlleven una modificación** «in-situ» de la misma:

- `reverse()`
- `append()`
- `remove()`
- `clear()`
- `sort()`
- etc

Empaquetado de tuplas

Reunir varias variables o valores en una tupla

```
#coordenadas
x = 0
y = 1
z = 3

#empaquetado
punto = x, y, z
print(punto)
```

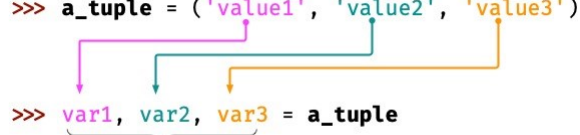
La ejecución de este código muestra por pantalla

```
(0, 1, 3)
```

Desempaquetado de tuplas

El desempaquetado es una característica de las tuplas que nos permite asignar una tupla a variables independientes:

```
>>> a_tuple = ('value1', 'value2', 'value3')
>>> var1, var2, var3 = a_tuple
```



También es una tupla

Desempaquetado de tuplas

Veamos un ejemplo con código:

```
>>> reyes_magos = ("gaspar", "melchor", "baltazar")
>>> rey1, rey2, rey3 = reyes_magos
>>> rey1
'gaspar'
>>> rey2
'melchor'
>>> rey3
'baltazar'
```

Desempaquetado de tuplas

Python proporciona la función `divmod()` que devuelve el cociente y el resto de una división usando una única llamada. Lo interesante (para el caso que nos ocupa) es que se suele utilizar el desempaquetado de tuplas para obtener los valores:

```
>>> cociente, resto = divmod(7,3)
>>> cociente
2
>>> resto
1
```


¿Python permite retornos múltiples?

La siguiente función recibe segundos y retorna el equivalente en horas, minutos y segundos

```
def obtenerHMS (s):
    horas = s // 3600
    minutos = ( s % 3600 ) // 60
    segundos = s % 60
    return horas, minutos, segundos

# llamado a la función
h , m , s = obtenerHMS (4523)
print (f'{h}:{m}:{s}')
```

1:15:23

Intercambio de valores

A través del desempaqueado de tuplas podemos llevar a cabo el intercambio de los valores de dos variables de manera directa:

```
>>> valor1 = 40
>>> valor2 = 30
>>> valor1, valor2 = valor2, valor1
>>> valor1
30
>>> valor2
40
```

A priori puede parecer que esto es algo «natural», pero en la gran mayoría de lenguajes de programación no es posible hacer este intercambio de forma «directa» ya que necesitamos recurrir a una tercera variable «auxiliar» como almacén temporal en el paso intermedio de traspaso de valores.

Tuplas por compresión

El desempaquetado de tuplas es extensible a cualquier tipo de datos que sea **iterable**. Veamos algunos ejemplos de ello. Los tipos de datos mutables (*listas, diccionarios y conjuntos*) sí permiten comprensiones pero no así los tipos de datos inmutables como *cadenas de texto y tuplas*.

Tuplas vs Listas

```
>>> tupla = (1,2,3,4,5,6,7,8,9)
>>> lista = [1,2,3,4,5,6,7,8,9]
```

Aunque puedan parecer estructuras de datos muy similares, sabemos que las tuplas carecen de ciertas operaciones, especialmente las que tienen que ver con la modificación de sus valores, ya que son inmutables. Si las listas son más flexibles y potentes, ¿por qué íbamos a necesitar tuplas? Veamos ventajas del uso de tuplas frente a las listas:

- Las tuplas ocupan **menos espacio** en memoria.
- En las tuplas existe **protección** frente a cambios indeseados.
- Las tuplas son más rápidas que las listas
- Si estás definiendo un conjunto constante de valores y todo lo que vas a hacer con él es recorrerlo, utiliza una tupla en lugar de una lista.

Tuplas

Las tuplas son ideales para cuando hay una colección de valores que están conectados de alguna manera. Por ejemplo, cuando hay que manejar las coordenadas x e y de un punto, una tupla es una opción natural, porque las coordenadas siempre estarán formadas por dos valores:


punto = (10, 20)

Técnicamente es posible, por supuesto, utilizar también una lista para almacenarlos:

Punto = [10, 20]



Actividad

1. Crea una función que reciba una lista de usuarios (tuplas: nombre, apellido, correo, edad) y **muestre**:
 - El número de usuarios.
 - El nombre del usuario más viejo.
 - El nombre del usuario más joven.
 - El nombre del usuario con apellido más largo.
 - La edad promedio de los usuarios.
- 

Actividad

2. Crea una función que reciba una lista de puntos (tuplas: x,y, z) y retorne la lista ordenada según la distancia entre cada punto y el origen.

Función para calcular distancia entre pares de puntos:

$$\sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2 + (z_1 - z_2)^2}$$

