😎 Tuplas (tuple)

En Python, una **tupla** es un conjunto ordenado e inmutable de elementos del mismo o diferente tipo. Una vez que se crea una tupla, como es inmutable, no se puede cambiar ni su contenido ni su tamaño. Los elementos se escriben entre paréntesis y separados por comas.

In [1]:
```
(1, "z", 3.1416)
```
Out[1]: (1, 'z', 3.1416)

En realidad no es necesario escribir los paréntesis para indicar que se trata de una tupla, basta con escribir las comas, pero Python escribe siempre los paréntesis:

In [2]:
```
1, "z", 3.1416
```
Out[2]: (1, 'z', 3.1416)

La función **len()** devuelve el número de elementos de una tupla:

In [3]:
```
len((1, "z", 3.1416))
```
Out[3]: 3

Hay que tener cuidado cuando le pasamos los parámetros a len()

In [4]:
```
len(1, "z", 3.1416)
```
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [4], in <cell line: 1>()
----> 1 len(1, "z", 3.1416)

TypeError: len() takes exactly one argument (3 given)
```

```
Una tupla puede no contener ningún elemento, es decir, ser una tupla vacía.
```

In [5]:
```
()
```
Out[5]: ()

In [6]:
```
len(())
```
Out[6]: 0

Una tupla puede incluir un único elemento, pero para que Python entienda que nos estamos refiriendo a una tupla es necesario escribir al menos una coma. El ejemplo siguiente muestra la diferencia entre escribir o no, una coma.

In [7]:
```
(3)
```
Out[7]: 3

En el primer caso Python interpreta la expresión como un número.

In [8]:
```
(3,)
```
Out[8]: (3,)

En este segundo Python interpreta como una tupla de un único elemento.

In [9]:
```
(3,)
```
Out[9]: (3,)

In [10]:
```
len((3,))
```
Out[10]: 1

Python escribe una coma al final en las tuplas de un único elemento para indicar que se trata de un tupla, pero esa coma no indica que hay un elemento después.

Operaciones con tuplas

Concatenación

La concatenación es la combinación de tuplas. Ejemplo: Creamos 2 tuplas:

```
In [11]: tuple1 = (1,2,3,4,5)
         tuple2 = (6,7,8,9,10)
```

```
In [12]: tuple1
```

```
Out[12]: (1, 2, 3, 4, 5)
```

```
In [13]: tuple2
```

```
Out[13]: (6, 7, 8, 9, 10)
```

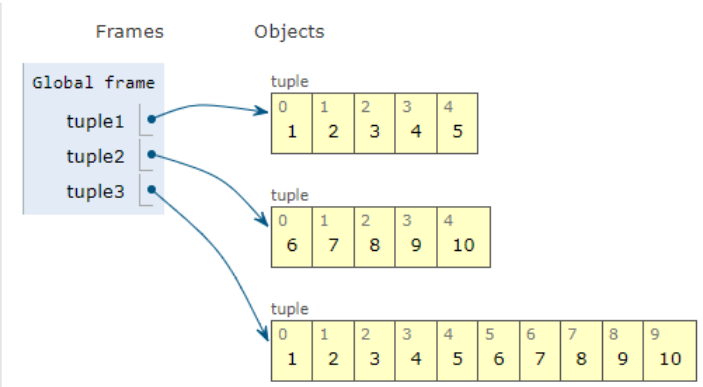
Si queremos concatenar ambas tuplas utilizamos el operador +

```
In [14]: tuple3 = tuple1 + tuple2
```

```
In [15]: tuple3
```

```
Out[15]: (1, 2, 3, 4, 5, 6, 7, 8, 9, 10)
```

```
1  tuple1 = (1,2,3,4,5)
2  tuple2 = (6,7,8,9,10)
3
4  tuple3 = tuple1 + tuple2
```



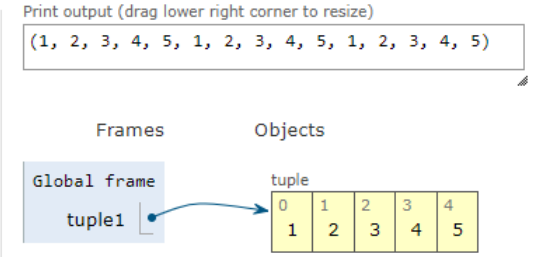
Repetición

La repetición de tuplas se puede llevar a cabo mediante el uso del operador *. Ejemplo:

```
In [16]: tuple1 * 3
```

```
Out[16]: (1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
```

```
1  tuple1 = (1,2,3,4,5)
2
3  print(tuple1 * 3)
```



Comprobar elementos

Para comprobar si un elemento existe en la tupla, simplemente hay que utilizar la palabra clave in. Ejemplo:

```
In [17]: tuple1
```

```
Out[17]: (1, 2, 3, 4, 5)
```

```
In [18]: 5 in tuple1
```

```
Out[18]: True
```

```
In [19]: 7 in tuple1
```

```
Out[19]: False
```

Buscar

Si queremos saber en qué índice está localizado un valor concreto solo hay que usar index. Ejemplo:

```
In [20]: tuple1
```

```
Out[20]: (1, 2, 3, 4, 5)
```

```
In [21]: tuple1.index(3)
```

```
Out[21]: 2
```

Contador

Se puede contar el número de veces que un elemento existe en la tupla usando count().Ejemplo:

```
In [22]: tuple3 = (65,67,5,67,34,76,67,231,98,67)
         tuple3
```

```
Out[22]: (65, 67, 5, 67, 34, 76, 67, 231, 98, 67)
```

```
In [23]: tuple3.count(67)
```

```
Out[23]: 4
```

```
In [24]: tuple4 = (4,6,7,6,5,8,9,6)
```

```
In [25]: tuple4.count(6)
```

```
Out[25]: 3
```

Indexación

La indexación es el proceso de acceder al elemento de una tupla mediante un índice.

Si queremos acceder a la quinta posición de tupla, hay que recordar que los índices de la tupla comienzan en 0 (siempre que no esté vacía):

```
In [26]: tupla = (65, 67, 5, 67, 34, 76, 67, 231, 98, 67)
         tupla [5]
```

```
Out[26]: 76
```

Un índice también puede ser negativo, si el recuento lo iniciamos desde la derecha de la tupla se inicia con -1. Si se escribe -0 es lo mismo que indicar 0

```
In [27]: tupla [-3]
```

```
Out[27]: 231
```

```
In [28]: tupla [-0]
```

```
Out[28]: 65
```

Qué pasaría si hacemos referencia a un índice fuera de rango?

```
In [29]: tupla[-12]
```

```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
Input In [29], in <cell line: 1>()
----> 1 tupla[-12]

IndexError: tuple index out of range
```

La tuplas no tienen métodos, append(), extend() para agregar elementos, ni remove() o pop() para eliminar porque son inmutables.
Tampoco se pueden modificar.

```
In [30]: tupla = ('a', 1, 'Python', 4, 5.6, 'Juanito')
         tupla
```

```
Out[30]: ('a', 1, 'Python', 4, 5.6, 'Juanito')
```

In [31]:
```
tupla.append('tupla')
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [31], in <cell line: 1>()
----> 1 tupla.append('tupla')

AttributeError: 'tuple' object has no attribute 'append'
```

In [32]:
```
tupla.remove(5.6)
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
Input In [32], in <cell line: 1>()
----> 1 tupla.remove(5.6)

AttributeError: 'tuple' object has no attribute 'remove'
```

In [33]:
```
tupla[2]
```

Out[33]:
```
'Python'
```

In [34]:
```
tupla[2] = 987
```

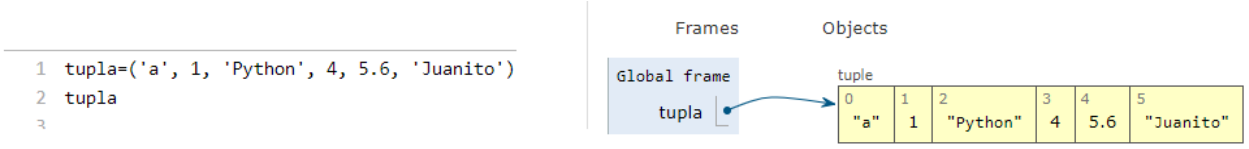
```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [34], in <cell line: 1>()
----> 1 tupla[2] = 987

TypeError: 'tuple' object does not support item assignment
```

Particionado

El particionado (en inglés **slicing**) de una tupla funciona como en las listas. Una partición de una tupla es una nueva tupla con los elementos seleccionados. Lo que diferencia a las tuplas de las listas es que las primeras no se pueden modificar. Se puede particionar una tupla porque en realidad se crea una nueva tupla.

In [35]:
```
tupla=('a', 1, 'Python', 4, 5.6, 'Juanito')
tupla
```

Out[35]:
```
('a', 1, 'Python', 4, 5.6, 'Juanito')
```



In [36]:
```
tupla[1:3]
```

Out[36]:
```
(1, 'Python')
```

In [37]:
```
tupla[:3]
```

Out[37]:
```
('a', 1, 'Python')
```

In [38]:
```
tupla[0:3]
```

Out[38]:
```
('a', 1, 'Python')
```

In [39]:
```
tupla[3:]
```

Out[39]:
```
(4, 5.6, 'Juanito')
```

In [40]:
```
tupla[3:0]
```

Out[40]:
```
()
```

In [41]:
```
tupla[2:4]
```

Out[41]:
```
('Python', 4)
```

Las tuplas se pueden utilizar en un contexto booleano

In [42]:
```
def es_True_o_False(tupla):
    if (tupla):
        print("Es verdadero")
    else:
        print("Es falso")
```

Una tupla vacía siempre es false en un contexto booleano.

In [43]:
```
es_True_o_False(())
```

```
Es falso
```

Una tupla con al menos un valor vale true

In [44]:
```
es_True_o_False(tupla)
```

```
Es verdadero
```

Para crear una tupla con un único elemento, hay que escribir una coma después del valor.

In [45]:
```
es_True_o_False(   ('es tupla?',)    )
```

```
Es verdadero
```

In [46]:
```
type(('es tupla?',))
```

Out[46]:
```
tuple
```

Sin la coma Python asume que lo que se está haciendo es poner un par de paréntesis a una expresión, por lo que no se crea una tupla.

In [47]:
```
type(('es tupla?'))
```

Out[47]:
```
str
```

Empaquetado y desempaquetado de tuplas

Empaquetado

(pack) Si a una variable se le asigna una secuencia de valores separados por comas, el valor de esa variable será la tupla formada por todos los valores asignados. A esta operación se la denomina empaquetado de tuplas.

In [48]:
```
a = 12.34
b = "@#"
c = "Madrid"
tupla = a, b, c
len(tupla)
tupla
```

Out[48]:
```
(12.34, '@#', 'Madrid')
```

In [49]:
```
tupla = 1, "z", 3.1416
tupla
```

Out[49]:
```
(1, 'z', 3.1416)
```

Desempaquetado:

(unpack) Si se tiene una tupla de longitud k, se puede asignar la tupla a 'k' variables distintas y en cada variable quedará una de las componentes de la tupla. A esta operación se la denomina desempaquetado de tuplas.

Tupla tiene 6 elementos:

In [50]:
```
tupla = ('a', 1, 'Python', 4, 5.6, 'Juanito')
tupla
```

Out[50]:
```
('a', 1, 'Python', 4, 5.6, 'Juanito')
```

(r, s, t, x, y, z) es una tupla de 6 variables:

In [51]:
```
(r, s, t, x, y, z) = tupla
```

Al asignar una tupla a otra lo que sucede es que cada una de las variables se queda con el valor de los elementos de la primera tupla que corresponde a su posición.

```
In [52]: r
Out[52]: 'a'

In [53]: s
Out[53]: 1

In [54]: t
Out[54]: 'Python'

In [55]: x
Out[55]: 4

In [56]: y
Out[56]: 5.6

In [57]: z
Out[57]: 'Juanito'
```

Esta prestación tiene toda clase de usos, por ejemplo si se quiere asignar nombres a un rango de valores, se puede combinar la función range() con la asignación múltiple para hacerlo de una forma rápida.

La función interna range() genera una secuencia de números enteros:

```
In [58]: (LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO) = range ( 7 )
```

```
1 (LUNES, MARTES, MIERCOLES, JUEVES, VIERNES, SABADO, DOMINGO) = range ( 7 )
```

Frames Objects

Global frame

LUNES	0
MARTES	1
MIERCOLES	2
JUEVES	3
VIERNES	4
SABADO	5
DOMINGO	6

Ahora cada variable tiene un valor: LUNES vale 0, MARTES vale 1, etc.

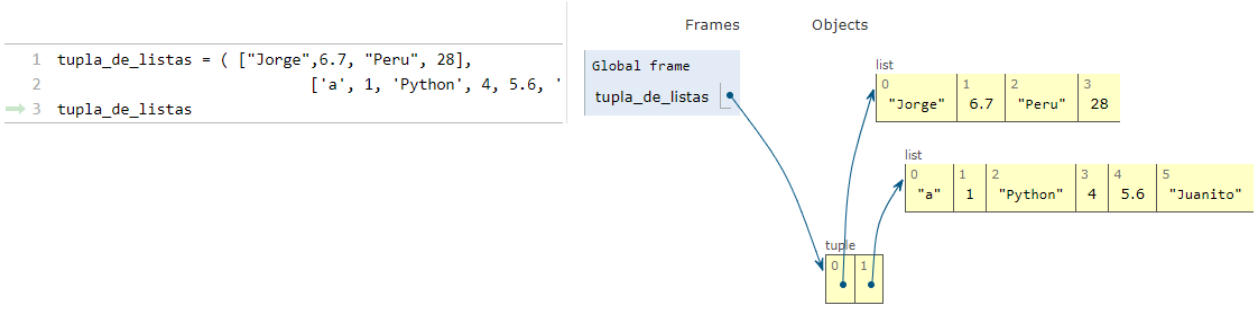
```
In [59]: LUNES
Out[59]: 0

In [60]: MARTES
Out[60]: 1

In [61]: MIERCOLES
Out[61]: 2

In [62]: JUEVES
Out[62]: 3

In [63]: VIERNES
Out[63]: 4

In [64]: SABADO
Out[64]: 5

In [65]: DOMINGO
Out[65]: 6
```

Tuplas que contienen listas

Una tupla puede contener distintos tipo de datos, incluyendo listas.

In [66]:
```
tupla_de_listas = ( ["Jorge",6.7, "Peru", 28], ['a', 1, 'Python', 4, 5.6, 'Juanito'] )
tupla_de_listas
```

Out[66]: (['Jorge', 6.7, 'Peru', 28], ['a', 1, 'Python', 4, 5.6, 'Juanito'])



In [67]:
```
tupla_de_listas[1][4]
```

Out[67]: 5.6

In [68]:
```
tupla_de_listas[1][2:4]
```

Out[68]: ['Python', 4]

In [69]:
```
tupla_de_listas[0]
```

Out[69]: ['Jorge', 6.7, 'Peru', 28]

In [70]:
```
tupla_de_listas[0][2]
```

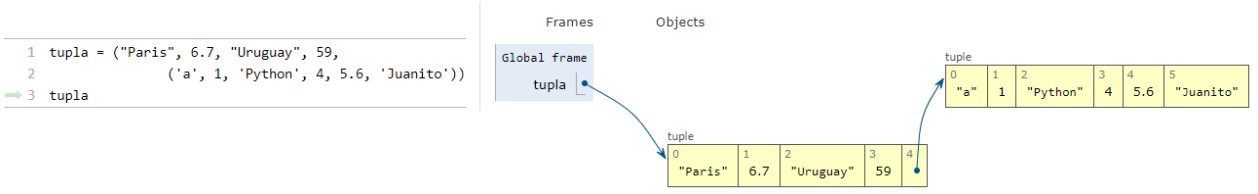
Out[70]: 'Peru'

Tuplas que contienen otras tuplas

Una tupla puede contener distintos tipo de datos, incluyendo tuplas.

In [71]:
```
tupla = ("Paris", 6.7, "Uruguay", 59, ('a', 1, 'Python', 4, 5.6, 'Juanito'))
tupla
```

Out[71]: ('Paris', 6.7, 'Uruguay', 59, ('a', 1, 'Python', 4, 5.6, 'Juanito'))



In [72]:
```
tupla[2]
```

Out[72]: 'Uruguay'

In [73]:
```
tupla[4][1:5]
```

Out[73]: (1, 'Python', 4, 5.6)

In [74]:
```
tupla[4][2:5]
```

Out[74]: ('Python', 4, 5.6)

In [75]:
```
tupla[4][2:4]
```

Out[75]: ('Python', 4)

Por ejemplo, una persona puede ser descripta por su nombre, su cuil y su fecha de nacimiento:

In [76]:
```
persona = ('Fulano', '20-12345678-9', (1980, 5, 14))
persona
```

Out[76]: ('Fulano', '20-12345678-9', (1980, 5, 14))

In [77]:
```
nombre, cuil, (a, m, d) = persona
m
```

Out[77]: 5

In [78]:
```
nombre
```

Out[78]: 'Fulano'

In [79]:
```
a,m,d
```

Out[79]: (1980, 5, 14)

Para evitar crear variables innecesarias, que no se van a utilizar se suele asignar estos valores a la variable **(_)**. Por ejemplo, si sólo nos interesa el mes:

In [80]:
```
 _, _, (_, mes, _) = persona
```

In [81]:
```
mes
```

Out[81]: 5

Conversión

Las tuplas se pueden convertir en listas, y viceversa. La función **tuple** toma una lista y devuelve una tupla con los mismos elementos, y la función **list** toma una tupla y devuelve una lista.

Tupla a Lista

In [82]:
```
tupla = ('a', 1, 'Python', 4, 5.6, 'Juanito')
tupla
```

Out[82]: ('a', 1, 'Python', 4, 5.6, 'Juanito')

In [83]:
```
tupla_a_lista = list(tupla)
tupla_a_lista
```

Out[83]: ['a', 1, 'Python', 4, 5.6, 'Juanito']

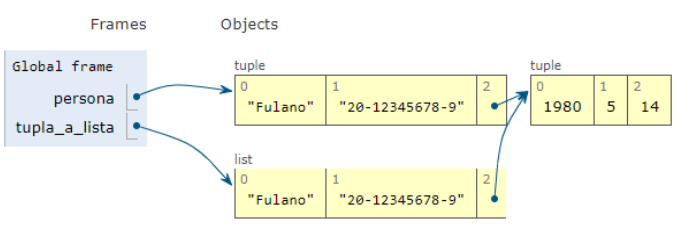
In [84]:
```
persona
```

Out[84]: ('Fulano', '20-12345678-9', (1980, 5, 14))

In [85]:
```
tupla_a_lista = list(persona)
tupla_a_lista
```

Out[85]: ['Fulano', '20-12345678-9', (1980, 5, 14)]

```
1  persona = ('Fulano', '20-12345678-9', (1980, 5, 14))
2  persona
3
4  tupla_a_lista = list(persona)
5  tupla_a_lista
```



Lista a Tupla

In [86]:
```
lista = ['a', 1, 'Python', 4, 5.6, 'Juan']
lista
```

Out[86]: ['a', 1, 'Python', 4, 5.6, 'Juan']

In [87]:
```
tupla = tuple(lista)
tupla
```

Out[87]: ('a', 1, 'Python', 4, 5.6, 'Juan')

In [88]:
```
persona = ('Fulano', '20-12345678-9', (1980, 5, 14))
persona
```

Out[88]: ('Fulano', '20-12345678-9', (1980, 5, 14))

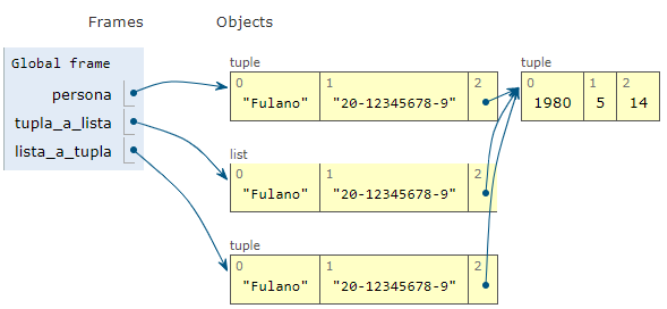
In [89]:
```
tupla_a_lista = list(persona)
tupla_a_lista
```

Out[89]: ['Fulano', '20-12345678-9', (1980, 5, 14)]

In [90]:
```
lista_a_tupla = tuple(tupla_a_lista )
lista_a_tupla
```

Out[90]: ('Fulano', '20-12345678-9', (1980, 5, 14))

```
1 persona = ('Fulano', '20-12345678-9', (1980, 5, 14))
2 persona
3
4 tupla_a_lista = list(persona)
5 tupla_a_lista
6
7 lista_a_tupla = tuple(tupla_a_lista )
8 lista_a_tupla
```



Comparación de tuplas

Dos tuplas son iguales cuando tienen el mismo tamaño y cada uno de sus elementos correspondientes tienen el mismo valor:

In [91]:
```
(1, 2) == (3/2, 1 + 1)
```

Out[91]: False

In [92]:
```
(6, 1) == (6, 2)
```

Out[92]: False

In [93]:
```
(6, 1) == (6, 1, 0)
```

Out[93]: False

In [94]:
```
(6, 1) == (6, 1)
```

Out[94]: True

Para determinar si una tupla es menor que otra, se utiliza el orden lexicográfico. Si los elementos en la primera posición de ambas tuplas son distintos, ellos determinan el ordenamiento de las tuplas:

In [95]:
```
(1, 4, 7) < (2, 0, 0, 1)
```

Out[95]: True

In [96]:
```
(1, 9, 10) < (0, 5)
```

Out[96]: False

- **La primera comparación es True porque 1 < 2.**
- **La segunda comparación es False porque 1 > 0. No importa el valor que tengan los siguientes valores, o si una tupla tiene más elementos que la otra.**

Si los elementos en la primera posición son iguales, entonces se usa el valor siguiente para hacer la comparación:

In [97]:
```
(6, 1, 8) < (6, 2, 8)
```

Out[97]: True

In [98]:
```
(6, 1, 8) < (6, 0)
```

Out[98]: False

- **La primera comparación es True porque 6 == 6 y 1 < 2.**
- **La segunda comparación es False porque 6 == 6 y 1 > 0**

Si los elementos respectivos siguen siendo iguales, entonces se sigue probando con los siguientes uno por uno, hasta encontrar dos distintos. Si a una tupla se le acaban los elementos para comparar antes que a la otra, entonces es considerada menor que la otra:

In [99]:
```
(1, 2) < (1, 2, 4)
```

Out[99]:
```
True
```

In [100]:
```
(1, 3) < (1, 2, 4)
```

Out[100]:
```
False
```

- **La primera comparación es True porque 1 == 1, 2 == 2, y ahí se acaban los elementos de la primera tupla.**
- **La segunda comparación es False porque 1 == 1 y 3 < 2; en este caso sí se alcanza a determinar el resultado antes que se acaben los elementos de la primera tupla.**

Iteración sobre tuplas

Una tabla de datos puede representarse como una lista de tuplas. Por ejemplo, la información de los alumnos que están cursando una carrera puede ser representada así:

In [101]:
```
alumnos = [
('Pachi', 'Mengano', '20-451189132-5', 'Biomédica'),
('Tato', 'Sultano', '20-251199122-6', 'Industrial'),
('Paqui', 'De Tal', '27-303149987-7', 'Electrónica'),
]
```

En este caso, se puede desempaquetar los valores automáticamente al recorrer la lista en un ciclo for:

In [102]:
```
for nombre, apellido, cuil, carrera in alumnos:
    print (f"{nombre} estudia {carrera}")
```

```
Pachi estudia Biomédica
Tato estudia Industrial
Paqui estudia Electrónica
```

Y como el apellido y el cuil no son usados:

In [103]:
```
for nombre, _, _, carrera in alumnos:
    print (f"{nombre} estudia {carrera}")
```

```
Pachi estudia Biomédica
Tato estudia Industrial
Paqui estudia Electrónica
```

Al igual que las listas, las tuplas son iterables:

In [104]:
```
for valor in (6, 1):
    print(f"{valor ** 2}")
```

```
36
1
```