Tuplas

Una **tupla** es una secuencias ordenadas de objetos de distintos tipos.

Se construyen poniendo los elementos entre parentesis `( )` separados por comas.

Se caracterizan por:

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

Se usan habitualmente para representar colecciones de datos una determinada estructura semántica, como por ejemplo un vector o una matriz.

```
# Tupla vacía
type(())
<class 'tuple'>
# Tupla con elementos de distintos tipos
(1, "dos", True)
# Vector
(1, 2, 3)
# Matriz
((1, 2, 3), (4, 5, 6))
```

Creación de tuplas mediante la función `tuple()`

Otra forma de crear tuplas es mediante la función `tuple()`.

- `tuple(c)` : Crea una tupla con los elementos de la secuencia o colección `c`.

Se pueden indicar los elementos separados por comas, mediante una cadena, o mediante una colección de elementos iterable.

```
>>> tuple()
()
>>> tuple(1, 2, 3)
(1, 2, 3)
>>> tuple("Python")
('P', 'y', 't', 'h', 'o', 'n')
>>> tuple([1, 2, 3])
(1, 2, 3)
```

Operaciones con tuplas

El acceso a los elementos de una tupla se realiza del mismo modo que en las listas. También se pueden obtener subtuplas de la misma manera que las sublistas.

Las operaciones de listas que no modifican la lista también son aplicables a las tuplas.

```
>>> a = (1, 2, 3)
>>> a[1]
```

```
2
>>> len(a)
3
>>> a.index(3)
2
>>> 0 in a
False
>>> b = ((1, 2, 3), (4, 5, 6))
>>> b[1]
(4, 5, 6)
>>> b[1][2]
6
```

Diccionarios

Un diccionario es una colección de pares formados por una *clave* y un *valor* asociado a la clave.

Se construyen poniendo los pares entre llaves `{ }` separados por comas, y separando la clave del valor con dos puntos `:`.

Se caracterizan por:

- No tienen orden.
- Pueden contener elementos de distintos tipos.
- Son mutables, es decir, pueden alterarse durante la ejecución de un programa.
- Las claves son únicas, es decir, no pueden repetirse en un mismo diccionario, y pueden ser de cualquier tipo de datos inmutable.

```
# Diccionario vacío
type({})
<class 'dict'>
# Diccionario con elementos de distintos tipos
{'nombre':'Alfredo', 'despacho': 218, 'email':'asalber@ceu.es'}
# Diccionarios anidados
{'nombre_completo':{'nombre': 'Alfredo', 'Apellidos': 'Sánchez Alberca'}}
```

Acceso a los elementos de un diccionario

- `d[clave]` devuelve el valor del diccionario `d` asociado a la clave `clave`. Si en el diccionario no existe esa clave devuelve un error.
- `d.get(clave, valor)` devuelve el valor del diccionario `d` asociado a la clave `clave`. Si en el diccionario no existe esa clave devuelve `valor`, y si no se especifica un valor por defecto devuelve `None`.

```
>>> a = {'nombre':'Alfredo', 'despacho': 218, 'email':'asalber@ceu.es'}
>>> a['nombre']
'Alfredo'
>>> a['despacho'] = 210
>>> a
{'nombre':'Alfredo', 'despacho': 218, 'email':'asalber@ceu.es'}
>>> a.get('email')
```

```
'asalber@ceu.es'
>>> a.get('universidad', 'CEU')
'CEU'
```

Operaciones que no modifican un diccionario

- `len(d)` : Devuelve el número de elementos del diccionario `d`.
- `min(d)` : Devuelve la mínima clave del diccionario `d` siempre que las claves sean comparables.
- `max(d)` : Devuelve la máxima clave del diccionario `d` siempre que las claves sean comparables.
- `sum(d)` : Devuelve la suma de las claves del diccionario `d`, siempre que las claves se puedan sumar.
- `clave in d` : Devuelve `True` si la clave `clave` pertenece al diccionario `d` y `False` en caso contrario.
- `d.keys()` : Devuelve un iterador sobre las claves de un diccionario.
- `d.values()` : Devuelve un iterador sobre los valores de un diccionario.
- `d.items()` : Devuelve un iterador sobre los pares clave-valor de un diccionario.

```
>>> a = {'nombre':'Alfredo', 'despacho': 218, 'email':'asalber@ceu.es'}
>>> len(a)
3
>>> min(a)
'despacho'
>>> 'email' in a
True
>>> a.keys()
dict_keys(['nombre', 'despacho', 'email'])
>>> a.values()
dict_values(['Alfredo', 218, 'asalber@ceu.es'])
>>> a.items()
dict_items([('nombre', 'Alfredo'), ('despacho', 218), ('email',
'asalber@ceu.es')])
```

Operaciones que modifican un diccionario

- `d[clave] = valor` : Añade al diccionario `d` el par formado por la clave `clave` y el valor `valor`.
- `d.update(d2)`. Añade los pares del diccionario `d2` al diccionario `d`.
- `d.pop(clave, alternativo)` : Devuelve del valor asociado a la clave `clave` del diccionario `d` y lo elimina del diccionario. Si la clave no está devuelve el valor `alternativo`.
- `d.popitem()` : Devuelve la tupla formada por la clave y el valor del último par añadido al diccionario `d` y lo elimina del diccionario.
- `del d[clave]` : Elimina del diccionario `d` el par con la clave `clave`.
- `d.clear()` : Elimina todos los pares del diccionario `d` de manera que se queda vacío.

```
>>> a = {'nombre':'Alfredo', 'despacho': 218, 'email':'asalber@ceu.es'}
>>> a['universidad'] = 'CEU'
>>> a
{'nombre': 'Alfredo', 'despacho': 218, 'email': 'asalber@ceu.es',
'universidad': 'CEU'}
>>> a.pop('despacho')
218
>>> a
{'nombre': 'Alfredo', 'email': 'asalber@ceu.es', 'universidad': 'CEU'}
>>> a.popitem()
('universidad', 'CEU')
>>> a
{'nombre': 'Alfredo', 'email': 'asalber@ceu.es'}
>>> del a['email']
>>> a
{'nombre': 'Alfredo'}
>>> a.clear()
>>> a
{}
```

Copia de diccionarios

Existen dos formas de copiar diccionarios:

- **Copia por referencia** `d1 = d2`: Asocia la la variable `d1` el mismo diccionario que tiene asociado la variable `d2`, es decir, ambas variables apuntan a la misma dirección de memoria. Cualquier cambio que hagamos a través de `l1` o `l2` afectará al mismo diccionario.
- **Copia por valor** `d1 = list(d2)`: Crea una copia del diccionario asociado a `d2` en una dirección de memoria diferente y se la asocia a `d1`. Las variables apuntan a direcciones de memoria diferentes que contienen los mismos datos. Cualquier cambio que hagamos a través de `l1` no afectará al diccionario de `l2` y viceversa.

```
>>> a = {1:'A', 2:'B', 3:'C'}
>>> # copia por referencia
>>> b = a
>>> b
{1:'A', 2:'B', 3:'C'}
>>> b.pop(2)
>>> b
{1:'A', 3:'C'}
>>> a
{1:'A', 3:'C'}

>>> a = {1:'A', 2:'B', 3:'C'}
>>> # copia por referencia
>>> b = dict(a)
>>> b
{1:'A', 2:'B', 3:'C'}
>>> b.pop(2)
>>> b
{1:'A', 3:'C'}
>>> a
{1:'A', 2:'B', 3:'C'}
```