



# Aprenda a programar con Python 3

## Secuencias: Listas, Tuplas, Diccionarios y Conjuntos

Carolina Mañoso, Ángel P. de Madrid y Miguel Romero

# Índice

---

- ◆ Secuencias o colecciones
- ◆ Listas
  - Práctica: Tipo Listas
- ◆ Tuplas
  - Práctica: Tipo Tuplas
- ◆ Diccionarios
  - Práctica: Tipo Diccionarios
- ◆ Conjuntos
  - Práctica: Tipo Conjuntos
- ◆ Referencias



# Secuencias o Colecciones

- ◆ En Python tenemos varios tipos de datos para almacenar colecciones de datos o sucesión de elementos.
- ◆ Se diferencian por su sintaxis y por la forma en la que se manipulan los datos.
- ◆ Una cadena es una secuencia de caracteres.
- ◆ Otros tipos son:
  - Listas
  - Tuplas
  - Diccionarios
  - Conjuntos

# Listas (1/5)

- ◆ Una lista es una secuencia de datos ordenada (en otros lenguajes es equivalente al array o vector).
- ◆ Los elementos de la lista deben ir separados por comas y todo el conjunto entre corchetes

```
>>> listaColores = ['rojo', 'verde', 'azul']
```

- ◆ Es una **estructura de datos mutable** porque además de poder acceder a los elementos, pueden suprimirse, añadirse y modificarse
- ◆ Para convertir al tipo Lista usamos `list()`

```
>>> a = list('hola')
```

```
>>> type(a)
```

## Listas (2/5)

- ◆ Se pueden recorrer, indexándolas. Para acceder a un elemento: `nombre de la lista[índice del elemento]`. El primer elemento tiene índice 0.

```
>>> colorB = listaColores[2]
```

- Permite índices negativos (`[-1]` hace referencia al último elemento)

```
>>> colorB = listaColores[-1]
```

- Permite seleccionar porciones de la lista:

- ◆ `[inicio:fin]`, la porción va desde `inicio` hasta `fin`, sin incluir el último.
- ◆ `[inicio:fin:salto]`, `salto` indica cada cuanto se selecciona un elemento desde `inicio` hasta `fin` sin incluir el último.
- ◆ `[:]`, representa toda la lista
- ◆ `[x:]`, la porción va desde `x` hasta `fin`.
- ◆ `[:y]`, la porción va desde `inicio` hasta la posición `y`.

## Listas (3/5)

◆ Las listas pueden contener cualquier tipo de datos: números, cadenas, ... y, también, listas.

- No tienen por que ser del mismo tipo
- Con una lista de listas hacemos una matriz:

```
>>>list=[["a","b","c"],["d","e","f"],["g","h","i"],["j","k","l"]]
```

```
list[0][0] = "a"   list[0][1] = "b"   list[0][2] = "c"
list[1][0] = "d"   list[1][1] = "e"   list[1][2] = "f"
list[2][0] = "g"   list[2][1] = "h"   list[2][2] = "i"
list[3][0] = "j"   list[3][1] = "k"   list[3][2] = "l"
```

# Listas (4/5)

◆ Python proporciona operadores y funciones similares para trabajar con tipos de datos similares. Así:

- El operador `+` concatena listas

```
>>> lista = [1, 2] + [3, 4] # lista = [1, 2, 3, 4]
```

- El operador `*` repite la lista el número indicado

```
>>> lista = [1, 2] * 2 # lista = [1, 2, 1, 2]
```

- La función `len()` devuelve la longitud de la lista

```
>>> len(lista) #4
```

# Listas (5/5)

## ◆ Algunas operaciones sobre listas con métodos específicos:

- `lista.append(item)`: suma un item al final de la lista
- `lista.extend(lista_2)`: junta la `lista_2` al final de la lista
- `lista.pop(x)`: devuelve y elimina el término `x`
- `lista.insert(x, item)`: Inserta `item` en la posición `x`
- `lista.sort()`: ordena la lista
- `lista.index(item)`: retorna la posición de la primera ocurrencia
- `lista.count(item)`: retorna cuantas veces aparece `item`
- `lista.remove(item)`: elimina la primera ocurrencia del `item`

```
>>> lista = [3, 2, 1]
```

```
>>> lista.append(0) → [3, 2, 1, 0]
```

```
>>> lista.extend([0, -1]) → [3, 2, 1, 0, -1]
```

```
>>> lista.insert(1, 99) → [3, 99, 2, 1]
```

```
>>> lista.sort() → [1, 2, 3] >>> lista.index(2) → 1
```

```
>>> lista.count(2) → 1
```

```
>>> lista.remove(2) → [3, 1]
```



# Práctica: Tipo Listas

## ◆ Introduzca en el intérprete de comandos:

```
>>> lista1 = ['uno', 2, True] #declara una lista heterogénea
>>> lista2 = [1, 2, 3, 4, 5]#declara lista numérica homogénea
>>> lista3 = ['nombre', ['calle', 'ciudad']]#lista en otra
>>> print(lista1[0]) # uno, muestra el primer elemento
>>> print(lista2[-1]) # 5, muestra el último elemento
>>> print(lista2[0:3:1]) # [1, 2, 3], patrón inicio:fin:paso
>>> lista1[2] = False # cambia el valor de un elemento
>>> lista2.append(6) # añade un elemento
>>> lista2.extend([7,8,9]) añade una lista al final
>>> elemento = lista2.pop(8)
>>> lista2.insert(3,8)
>>> lista2.count(8)
>>> lista2.remove(8)
```

# Tuplas

- ◆ La tupla agrupa elementos de forma **inmutable**, es decir, no permite añadir, ni eliminar, ni modificar sus elementos una vez que se define.
- ◆ Para declararla se separan los elementos por comas y, estos van entre paréntesis (opcional, aunque recomendable).

```
tuplaColoresRGB = ('rojo', 'verde', 'azul')
```

- Es necesario la coma para tuplas de un elemento:

```
tuplaColorR = ('rojo',)
```

- Para referirnos a un elemento de la tupla utilizamos, `[]`, al igual que en las listas

- Algunas operaciones vistas en las listas que no cambian los valores pueden ser usadas con tuplas.

- Para convertir a tupla usamos `tuple`

```
>>> a = tuple('hola')
```

```
>>> type(a)
```

# Práctica: Tipo Tuplas

## ◆ Introduzca en el intérprete de comandos:

```
>>> tupla1 = (1, 2, 3) # declara una tupla (con paréntesis)
>>> tupla2 = 1, 2, 3 # ... pueden declararse sin paréntesis
>>> tupla3 = (100,) # con un solo un elemento, poner ","
>>> tupla4 = tupla1, 4, 5, 6 # anida tuplas
>>> tupla5 = () # declara una tupla vacía
>>> len(tupla1) # 3
>>> tupla2[0:3] # (1, 2, 3), accede a los valores desde:hasta
>>> tupla[1]=4
```

Traceback (most recent call last):

File "<pyshell#54>", line 1, in <module>

a[1]=4

TypeError: 'tuple' object does not support item assignment

# Diccionarios (1/2)

◆ Los Diccionarios o matrices asociativas: son colecciones de parejas donde el primer valor es la clave (`keys`) y el segundo es el valor asociado a la clave (`values`).

- Los pares clave-valor están separados por dos puntos y las parejas por comas y todo el conjunto se encierra entre llaves.

```
>>>dic={'piso1':'Juan','piso2':'Pepe','piso3':'Luis'}
```

- No se puede repetir ninguna clave y es un valor **inmutable**, podríamos usar números, booleans, tuplas o cadena pero no listas, mientras que el valor puede ser un número, una cadena, una lista o una tupla.
- Para sumar un nuevo item (clave/valor) basta con especificar la nueva clave y asignarle su valor

```
>>> dic['piso4']='Manolo'
```

## Diccionarios (2/2)

- Se accede a los valores almacenados:

- ♦ por su clave:

```
>>> dic['piso1']      #devuelve Juan
```

- ♦ Si se desea todas las claves usamos el método `keys()`

```
>>> dic.keys()
```

- ♦ Si se desea obtener los valores el método `values()`

```
>>> dic.values()
```

- ♦ Si se desea acceder tanto a la clave como al valor: `items()`

```
>>> dic.items()
```

# Práctica: Tipo Diccionarios

## ◆ Introduzca en el intérprete de comandos:

```
>>> cria = {'caballo': 'potro', 'oveja': 'cordero',  
            'gallina': 'pollito'}  
>>> cria['cabra'] = 'cabrito'  
>>> cria.update({'lobo': 'lobezno'})  
>>> print(cria.keys())  
>>> print(cria.values())  
>>> print(cria.items())  
>>> type(cria)
```

# Conjuntos

- ◆ El conjunto es una estructura de datos que nos permite agrupar una colección de datos juntos sin índices, ni claves (sin orden).

```
>>> hierbas = {'cilandro', 'perejil', 'hierbabuena'}
```

```
>>> especias = {'cilandro', 'perejil', 'pimienta'}
```

- Usamos el operador `in` para comprobar que un valor en particular está en el conjunto.
- Las operaciones entre conjuntos son:
  - ◆ `set_1 & set_2` retorna la intersección
  - ◆ `set_1 | set_2` retorna la unión
  - ◆ `set_1 - set_2` retorna la diferencia
  - ◆ `set_1 ^ set_2` retorna los items que están en el primero o en el segundo pero no en ambos

```
>>> hierbas & especias → {'cilandro', 'perejil'}
```

```
>>> hierbas - especias → 'hierbabuena'
```

```
>>> hierbas ^ especias → {'hierbabuena', 'pimienta'}
```

# Práctica: Tipo Conjunto

## ◆ Introduzca en el intérprete de comandos:

```
>>> beb_frias = {'tonica', 'trina', 'cerveza'}
>>> beb_calientes = {'te', 'cafe', 'colacao'}
>>> beb_alcohol = {'cerveza', 'vino', 'ron'}
>>> beb = beb_frias | beb_calientes
>>> beb_frias_alcohol = beb_frias & beb_alcohol
>>> type(beb)
>>> 'atun' in beb_frias
>>> 'trina' in beb_frias
```



# Mutable/Immutable

Inmutable	Mutable
Números	Listas
Cadenas	Diccionarios
Tuplas	Conjuntos

# Referencias

- ◆ La documentación oficial, *Python Tutorial Release 3.5.2*. Guido van Rossum and the Python development team. Python Software Foundation. Junio 25, 2016.  
<https://docs.python.org/3/download.html>
- ◆ *Learning Python 5th edition*. Mark Lutz. O'Reilly 2013
- ◆ *Practical Programming: An Introduction to computer Science using Python 3*. P. Gries, J. Campbell & J. Montoyo. Edited by Lynn Beighley. 2nd Edition. The Pragmatic Bookshelf. 2013
- ◆ *Learning Python with Raspberry Pi*. Alex Bradbury, Ben Everard. Wiley. 2014
- ◆ Tutorial: *Python 3 para impacientes*. Antonio Suárez Jiménez. 2014  
<http://python-para-impacientes.blogspot.com.es/p/indice.html>
- ◆ *Raspberry Pi, User Guide*. Eben Upton, Gareth Halfacree. Wiley. 2012

# Aviso



Aprenda a programar con Python 3 by C. Mañoso, A. P. de Madrid, M. Romero is licensed under a [Creative Commons Reconocimiento-NoComercial-CompartirIgual 4.0 Internacional License](https://creativecommons.org/licenses/by-nc-sa/4.0/).

Esta colección de transparencias se distribuye con fines meramente docentes.

Todas las marcas comerciales y nombres propios de sistemas operativos, programas, hardware, etc. que aparecen en el texto son marcas registradas propiedad de sus respectivas compañías u organizaciones.