Tipo de datos secuencia: listas

Si queremos guardar un conjunto de valores, en pseudocódigo utilizamos loas arreglos. Un array (o arreglo) es una estructura de datos con elementos homogéneos, del mismo tipo, numérico o alfanumérico, reconocidos por un nombre en común.

Hay muchos lenguajes que implementan los arrays, pero esta estructura tiene dos limitaciones: son homogéneas, es decir sólo se pueden guardar datos del mismo tipo, y son estáticas, a la hora de declarar se indican las posiciones y la longitud del array no se puede cambiar durante la ejecución del programa.

En Python no existen los arrays, tenemos varios tipos de datos que nos permiten guardar conjuntos de informaciones. En esta unidad vamos a estudiar las **Listas**. Las listas (`list`) me permiten guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos. Además esta estructura es dinámica, en cualquier momento de la ejecución del programa puedo añadir o eliminar elementos de la lista.

Construcción de una lista

Para crear una lista puedo usamos los caracteres `[` y `]`:

```
>>> lista1 = []
>>> lista2 = ["a",1,True]
```

Operaciones básicas con listas

Las listas son secuencias, a las que podemos realizar las siguientes operaciones. Vamos a ver distintos ejemplos partiendo de la siguiente lista:

```
lista = [1,2,3,4,5,6]
```

- Las listas se pueden recorrer:

```
>>> for num in lista:
...   print(num,end="")
123456
```

Con la instrucción `for` podemos recorrer más de una listas, utilizando la función `zip`. Veamos un ejemplo:

```
>>> lista2 = ["a","b","c","d","e"]
>>> for num,letra in zip(lista,lista2):
...     print(num,letra)
...
1 a
2 b
3 c
4 d
5 e
```

- Operadores de pertenencia: Se puede comprobar si un elemento pertenece o no a una lista con los operadores `in` y `not in`.

```
>>> 2 in lista
 True
 >>> 8 not in lista
 True
```

- Concatenación: El operador `+` me permite unir datos de tipos listas:

```
 >>> lista + [7,8,9]
 [1, 2, 3, 4, 5, 6, 7, 8, 9]
```

- Repetición: El operador `*` me permite repetir un dato de una lista:

```
 >>> lista * 2
 [1, 2, 3, 4, 5, 6, 1, 2, 3, 4, 5, 6]
```

- Indexación: Puedo obtener el dato de una secuencia indicando la posición en la secuencia.

```
 >>> lista[3]
 4
```

  Cada elemento tiene un índice, empezamos a contar por el elemento en el índice 0. Si intento acceder a un índice que corresponda a un elemento que no existe obtenemos una excepción `IndexError`.

```
 >>> lista1[12]
 Traceback (most recent call last):
   File "<stdin>", line 1, in <module
 IndexError: list index out of range
```

  Se pueden utilizar índices negativos:

```
 >>> lista[-1]
 6
```

- Slice (rebanada): Puedo obtener una subsecuencia de los datos de una lista. Funciona de forma similar como en las cadenas, veamos algunos ejemplos:

```
>>> lista[2:4]
 [3, 4]
 >>> lista[1:4:2]
 [2, 4]
 >>> lista[:5]
```

```
[1, 2, 3, 4, 5]
>>> lista[5:]
[6, 1, 2, 3, 4, 5, 6]
>>> lista[::-1]
[6, 5, 4, 3, 2, 1, 6, 5, 4, 3, 2, 1]
```

Funciones predefinidas que trabajan con listas

```
>>> lista1 = [20,40,10,40,50]
>>> len(lista1)
5
>>> max(lista1)
50
>>> min(lista1)
10
>>> sum(lista1)
150
>>> sorted(lista1)
[10, 20, 30, 40, 50]
>>> sorted(lista1,reverse=True)
[50, 40, 30, 20, 10]
```

Listas multidimensionales

A la hora de definir las listas hemos indicado que podemos guardar en ellas datos de cualquier tipo, y evidentemente podemos guardar listas dentro de listas.

```
>>> tabla = [[1,2,3],[4,5,6],[7,8,9]]
>>> tabla[1][1]
5

>>> for fila in tabla:
...   for elem in fila:
...      print(elem,end="")
...   print()

123
456
789
```

Las listas son mutables

Al igual que las cadenas el tipo de datos lista es una **clase**, cada vez que creamos una variable de la clase lista estamos creando un **objeto** que además de guardar un conjunto de datos, posee un conjunto de **métodos** que nos permiten trabajar con la lista.

¿Qué significa que las listas son mutables?

Los elementos de las listas se pueden modificar:

```
>>> lista1 = [1,2,3]
>>> lista1[2]=4
>>> lista1
[1, 2, 4]
>>> del lista1[2]
>>> lista1
[1, 2]
```

Esto también ocurre cuando usamos los métodos, es decir, los métodos de las listas modifican el contenido de la lista, por ejemplo si usamos el método `append()` para añadir un elemento a la lista:

```
>>> lista1.append(3)
>>> lista1
[1, 2, 3]
```

Como vemos la lista `lista1` se ha modificado.

¿Cómo se copian las listas?

Para copiar una lista en otra no podemos utilizar el operador de asignación:

```
>>> lista1 = [1,2,3]
>>> lista2 = lista1
>>> lista1[1] = 10
>>> lista2
[1, 10, 3]
```

El operador de asignación no crea una nueva lista, sino que nombra con dos nombres distintos a la misma lista, por lo tanto la forma más fácil de copiar una lista en otra es:

```
>>> lista1 = [1,2,3]
>>> lista2=lista1[:]
>>> lista1[1] = 10
>>> lista2
[1, 2, 3]
```

Métodos principales de listas

Métodos de inserción: append, extend, insert

`append()`: añade un elemento a la lista:

```
>>> lista = [1,2,3]
>>> lista.append(4)
>>> lista
[1, 2, 3, 4]
```

`extend()`: Une dos listas:

```
>>> lista2 = [5,6]
>>> lista.extend(lista2)
>>> lista
[1, 2, 3, 4, 5, 6]
```

`insert()`: Añade un elemento en un posición indicada de la lista:

```
>>> lista.insert(1,100)
>>> lista
[1, 100, 2, 3, 4, 5, 6]
```

Métodos de eliminación: pop, remove

`pop()`: elimina un elemento de la lista y lo devuelve. Se puede indicar el índice del elemento que queremos obtener como parámetro, sino se indica se devuelve y elimina el último:

```
>>> lista.pop()
6
>>> lista
[1, 100, 2, 3, 4, 5]

>>> lista.pop(1)
100
>>> lista
[1, 2, 3, 4, 5]
```

`remove()`: Elimina el elemento de la lista indicado por la posición:

```
>>> lista.remove(3)
>>> lista
[1, 2, 4, 5]
```

Métodos de ordenación: reverse, sort,

`reverse()`: Modifica la lista invirtiendo los elementos:

```
>>> lista.reverse()
>>> lista
[5, 4, 2, 1]
```

`sort()`: Modifica la lista ordenando los elementos, se puede indicar el sentido de la ordenación:

```
>>> lista.sort()
>>> lista
[1, 2, 4, 5]

>>> lista.sort(reverse=True)
>>> lista
[5, 4, 2, 1]

>>> lista=["hola","que","tal","Hola","Que","Tal"]
>>> lista.sort()
>>> lista
['Hola', 'Que', 'Tal', 'hola', 'que', 'tal']
```

Métodos de búsqueda: count, index

`count()`: devuelve el número de apariciones de un elemento en la lista:

```
>>> lista.count(5)
1
```

`index()`: Nos devuelve la posición de la primera aparición del elemento indicado. Se puede indicar la posición inicial y final de búsqueda:

```
>>> lista.append(5)
>>> lista
[5, 4, 2, 1, 5]
>>> lista.index(5)
0
>>> lista.index(5,1)
4
>>> lista.index(5,1,4)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ValueError: 5 is not in list
```

Si no encuentra el elemento nos da una **excepción**.

Tipo de datos secuencia: Tuplas

Las tuplas (`tuple`): Sirven para los mismo que las listas (me permiten guardar un conjunto de datos que se pueden repetir y que pueden ser de distintos tipos), pero en este caso es un tipo inmutable.

Construcción de una tupla

Para crear una lista puedo usar los caracteres (y):

```
>>> tupla1 = ()
>>> tupla2 = ("a",1,True)
```

Operaciones básicas con tuplas

En las tuplas se pueden realizar las siguientes operaciones:

- Las tuplas se pueden recorrer.
- Operadores de pertenencia: `in` y `not in`.
- Concatenación: `+`
- Repetición: `*`
- Indexación
- Slice

Entre las funciones definidas podemos usar: `len`, `max`, `min`, `sum`, `sorted`.

Las tuplas son inmutables

```
>>> 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
```

Métodos principales

Métodos de búsqueda: `count`, `index`

```
>>> tupla = (1,2,3,4,1,2,3)
>>> tupla.count(1)
2

>>> tupla.index(2)
1
>>> tupla.index(2,2)
5
```