

# Listas en Python

Las listas en Python son un tipo de dato que permite almacenar datos de cualquier tipo. Son [mutables](#) y dinámicas.

## Crear listas Python

Las listas en Python son uno de los tipos o estructuras de datos más versátiles del lenguaje, ya que permiten almacenar un conjunto arbitrario de datos. Es decir, podemos guardar en ellas prácticamente lo que sea.

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

También se puede crear usando `list` y pasando un objeto [iterable](#).

```
lista = list("1234")
```

Una lista sea crea con `[]` separando sus elementos con comas `,`. Una gran ventaja es que pueden almacenar tipos de datos distintos.

```
lista = [1, "Hola", 3.67, [1, 2, 3]]
```

Algunas propiedades de las listas:

- Son **ordenadas**, mantienen el orden en el que han sido definidas
- Pueden ser formadas por tipos **arbitrarios**
- Pueden ser **indexadas** con `[i]`.
- Se pueden **anidar**, es decir, meter una dentro de la otra.
- Son **mutables**, ya que sus elementos pueden ser modificados.
- Son **dinámicas**, ya que se pueden añadir o eliminar elementos.

## Acceder y modificar listas

Si tenemos una lista `a` con 3 elementos almacenados en ella, podemos acceder a los mismos usando corchetes y un índice, que va desde `0` a `n-1` siendo `n` el tamaño de la lista.

```
a = [90, "Python", 3.87]
print(a[0]) #90
print(a[1]) #Python
print(a[2]) #3.87
```

Se puede también acceder al último elemento usando el índice `[-1]`.

```
a = [90, "Python", 3.87]
print(a[-1]) #3.87
```

De la misma manera, al igual que `[-1]` es el último elemento, podemos acceder a `[-2]` que será el penúltimo.

```
print(a[-2]) #Python
```

Y si queremos modificar un elemento de la lista, basta con asignar con el operador `=` el nuevo valor.

```
a[2] = 1
print(a) #[90, 'Python', 1]
```

Un elemento puede ser eliminado con diferentes métodos como veremos a continuación, o con `del` y la lista con el índice a eliminar.

```
l = [1, 2, 3, 4, 5]
del l[1]
print(l) #[1, 3, 4, 5]
```

También podemos tener **listas anidadas**, es decir, una lista dentro de otra. Incluso podemos tener una lista dentro de otra lista y a su vez dentro de otra lista. Para acceder a sus elementos sólo tenemos que usar `[]` tantas veces como niveles de anidado tengamos.

```
x = [1, 2, 3, ['p', 'q', [5, 6, 7]]]
print(x[3][0]) #p
print(x[3][2][0]) #5
print(x[3][2][2]) #7
```

También es posible crear sublistas más pequeñas de una más grande. Para ello debemos de usar `:` entre corchetes, indicando a la izquierda el valor de inicio, y a la izquierda el valor final que no está incluido. Por lo tanto `[0:2]` creará una lista con los elementos `[0]` y `[1]` de la original.

```
l = [1, 2, 3, 4, 5, 6]
print(l[0:2]) #[1, 2]
print(l[2:6]) #[3, 4, 5, 6]
```

Y de la misma manera podemos modificar múltiples valores de la lista a la vez usando `:`.

```
l = [1, 2, 3, 4, 5, 6]
l[0:3] = [0, 0, 0]
print(l) #[0, 0, 0, 4, 5, 6]
```

Hay ciertos operadores como el `+` que pueden ser usados sobre las listas.

```
l = [1, 2, 3]
l += [4, 5]
print(l) #[1, 2, 3, 4, 5]
```

Y una funcionalidad muy interesante es que se puede asignar una lista con `n` elementos a `n` variables.

```
l = [1, 2, 3]
x, y, z = l
print(x, y, z) #1 2 3
```

## Iterar listas

En Python es muy fácil iterar una lista, mucho más que en otros lenguajes de programación.

```
lista = [5, 9, 10]
for l in lista:
    print(l)
#5
#9
#10
```

Si necesitamos un índice acompañado con la lista, que tome valores desde `0` hasta `n-1`, se puede hacer de la siguiente manera.

```
lista = [5, 9, 10]
for index, l in enumerate(lista):
    print(index, l)
#0 5
#1 9
#2 10
```

O si tenemos dos listas y las queremos iterar a la vez, también es posible hacerlo.

```
lista1 = [5, 9, 10]
lista2 = ["Jazz", "Rock", "Pop"]
for l1, l2 in zip(lista1, lista2):
    print(l1, l2)
#5 Jazz
#9 Rock
#10 Pop
```

Y por supuesto, también se pueden iterar las listas usando los índices como hemos visto al principio, y haciendo uso de `len()`, que nos devuelve la longitud de la lista.

```
lista1 = [5, 9, 10]
for i in range(0, len(lista)):
    print(lista1[i])
#5
#9
#10
```

## Métodos listas

El método `append()` añade un elemento al final de la lista.

```
l = [1, 2]
l.append(3)
print(l) #[1, 2, 3]
```

El método `extend()` permite añadir una lista a la lista inicial.

```
l = [1, 2]
l.extend([3, 4])
print(l) #[1, 2, 3, 4]
```

El método `insert()` añade un elemento en una posición o índice determinado.

```
l = [1, 3]
l.insert(1, 2)
print(l) #[1, 2, 3]
```

El método `remove()` recibe como argumento un objeto y lo borra de la lista.

```
l = [1, 2, 3]
l.remove(3)
print(l) #[1, 2]
```

El método `pop()` elimina por defecto el último elemento de la lista, pero si se pasa como parámetro un `índice` permite borrar elementos diferentes al último.

```
l = [1, 2, 3]
l.pop()
print(l) #[1, 2]
```

El método `reverse()` invierte el orden de la lista.

```
l = [1, 2, 3]
l.reverse()
print(l) #[3, 2, 1]
```

El método `sort()` ordena los elementos de menos a mayor por defecto.

```
l = [3, 1, 2]
l.sort()
print(l) #[1, 2, 3]
```

Y también permite ordenar de mayor a menor si se pasa como parámetro `reverse=True`.

```
l = [3, 1, 2]
l.sort(reverse=True)
print(l) #[3, 2, 1]
```

El método `index()` recibe como parámetro un objeto y devuelve el índice de su primera aparición. Como hemos visto en otras ocasiones, el índice del primer elemento es el `0`.

```
l = ["Perisferia", "Intervalos", "Monumentos"]
print(l.index("Intervalos"))
```

También permite introducir un parámetro opcional que representa el índice desde el que comenzar la búsqueda del objeto. Es como si ignorara todo lo que hay antes de ese índice para la búsqueda, en este caso el `4`.

```
l = [1, 1, 1, 1, 2, 1, 4, 5]
print(l.index(1, 4)) #5
```



## El ciclo for en Python

El ciclo for se utiliza para recorrer los elementos de un objeto iterable (lista, tupla, conjunto, diccionario, ...) y ejecutar un bloque de código.

En cada paso de la iteración se tiene en cuenta a un único elemento del objeto iterable, sobre el cuál se pueden aplicar una serie de operaciones.

```
for <elem> in <objeto iterable>:  
    Bloque de código.
```

Aquí, elem es la variable que toma el valor del elemento dentro del iterador en cada paso del ciclo.

Este finaliza su ejecución cuando se recorren todos los elementos.

Como se indicaba, es muy frecuente usar el ciclo for para iterar sobre los elementos de listas, tuplas o diccionarios.

```
nums = [4, 78, 9, 84]
```

```
for n in nums:
```

```
    print(n)
```



## Ejercicios

- 1) Escribir un programa que almacene el abecedario en una lista, elimine de la lista las letras que ocupen posiciones múltiplos de 3, y muestre por pantalla la lista resultante.
- 2) Escribir un programa que almacene en una lista los números del 1 al 10 y los muestre por pantalla en orden inverso separados por comas.
- 3) Escribir un programa que pida al usuario una palabra y muestre por pantalla si es un palíndromo.
- 4) Escribir un programa que almacene en una lista los siguientes precios, 50, 75, 46, 22, 80, 65, 8, y muestre por pantalla el menor y el mayor de los precios.
- 5) Escriba un programa que permita crear una lista de palabras. Para ello, el programa tiene que pedir un número y luego solicitar ese número de palabras para crear la lista. Por último, el programa tiene que escribir la lista.
- 6) Escriba un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y diga cuántas veces aparece esa palabra en la lista.
- 7) Escriba un programa que permita crear una lista de palabras y que, a continuación, pida dos palabras y sustituya la primera por la segunda en la lista.
- 8) Escriba un programa que permita crear una lista de palabras y que, a continuación, pida una palabra y elimine esa palabra de la lista.
- 9) Escriba un programa que permita crear dos listas de palabras y que, a continuación, escriba las siguientes listas (en las que no debe haber repeticiones):
  - Lista de palabras que aparecen en las dos listas.
  - Lista de palabras que aparecen en la primera lista, pero no en la segunda.
  - Lista de palabras que aparecen en la segunda lista, pero no en la primera.
  - Lista de palabras que aparecen en ambas listas.

Nota: Para evitar las repeticiones, el programa deberá empezar eliminando los elementos repetidos en cada lista.

