

NIVEL 3

OPERACIONES SOBRE LISTAS



COMPARACIÓN DE LISTAS

- Los operadores de comparación también trabajan con listas
- Si las listas son de tamaño diferente, entonces las listas son diferentes
- Si son del mismo tamaño, se compara elemento a elemento de izquierda a derecha y:
 - Las dos listas son iguales si todos sus elementos son iguales, y diferentes si hay algún elemento distinto

```
Terminal 2/A ✕  
  
In [30]: [1, 2, 3] == [1, 2]  
Out[30]: False  
  
In [31]: [1, 2, 3] == [1, 2, 3]  
Out[31]: True  
  
In [32]: [1, 2, 3] == [1, 2, 4]  
Out[32]: False
```

COMPARACIÓN DE LISTAS



Los operadores `<`, `>`, `<=` y `>=` también funcionan con listas.
¿Cómo?

1. El primer criterio de comparación se realiza comparando elemento por elemento de izquierda a derecha.
2. Si no hay elementos diferentes se procede a revisar el tamaño de las listas para determinar cuál es mayor/menor.

```
In [1]: a = [2,3,1,5]
```

```
In [2]: b = [2,3,4]
```

```
In [3]: b < a
```

```
Out[3]: False
```

```
In [5]: [1, 10, 100] < [2, 10, 100]
```

```
Out[5]: True
```

```
In [6]: [1, 10, 100] < [1, 10, 100]
```

```
Out[6]: False
```

```
In [7]: [1, 10, 100] <= [1, 10, 100]
```

```
Out[7]: True
```

MODIFICACIÓN DE LISTAS

Podemos asignar valores a elementos particulares de una lista gracias al operador de indexación

No nos podemos salir del tamaño original de la lista

```
Terminal 3/A x

In [2]: a = [1, 2, 3]

In [3]: a
Out[3]: [1, 2, 3]

In [4]: a[1] = 10

In [5]: a
Out[5]: [1, 10, 3]

In [6]: a[3] = 20
Traceback (most recent call last):

  File "<ipython-input-6-668cfa6b3a75>", line 1, in
    <module>
      a[3] = 20
IndexError: list assignment index out of range
```



LAS LISTAS SON MUTABLES

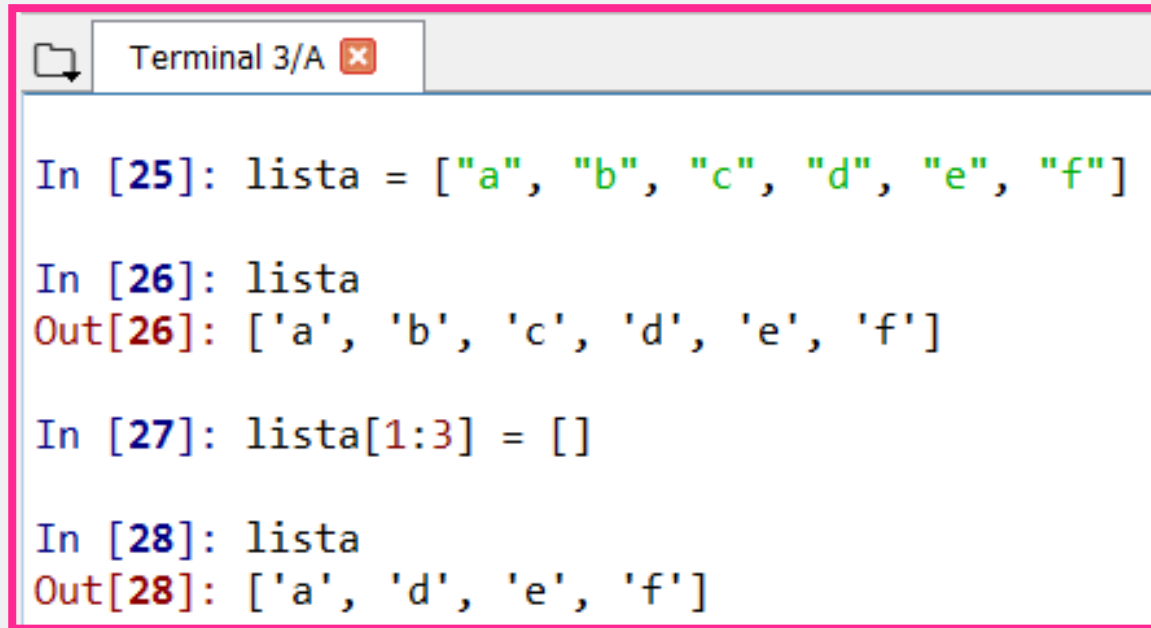
MODIFICACIÓN DE LISTAS

Con el operador de corte podemos actualizar una sublista

```
Terminal 3/A ✕  
  
In [13]: lista = ["a", "b", "c", "d", "e", "f"]  
  
In [14]: lista  
Out[14]: ['a', 'b', 'c', 'd', 'e', 'f']  
  
In [15]: lista[1:3] = ["x", "y"]  
  
In [16]: lista  
Out[16]: ['a', 'x', 'y', 'd', 'e', 'f']
```

MODIFICACIÓN DE LISTAS

Podemos también eliminar «pedazos» de una lista asignándoles una lista vacía:

A screenshot of a Jupyter Notebook terminal window titled "Terminal 3/A". The window shows a sequence of four code cells. The first cell defines a list 'lista' with elements 'a', 'b', 'c', 'd', 'e', and 'f'. The second cell prints the list. The third cell assigns an empty list to the slice 'lista[1:3]', effectively removing elements 'b' and 'c'. The fourth cell prints the updated list, which now contains 'a', 'd', 'e', and 'f'.

```
Terminal 3/A ✕  
  
In [25]: lista = ["a", "b", "c", "d", "e", "f"]  
  
In [26]: lista  
Out[26]: ['a', 'b', 'c', 'd', 'e', 'f']  
  
In [27]: lista[1:3] = []  
  
In [28]: lista  
Out[28]: ['a', 'd', 'e', 'f']
```

MODIFICACIÓN DE LISTAS

Podemos también adicionar elementos a una lista insertándolos en un «pedazo», en la posición deseada

```
Terminal 3/A ✕  
  
In [30]: lista = ["a", "d", "f"]  
  
In [31]: lista  
Out[31]: ['a', 'd', 'f']  
  
In [32]: lista[1:1] = ["b", "c"]  
  
In [33]: lista  
....:  
Out[33]: ['a', 'b', 'c', 'd', 'f']  
  
In [34]: lista[4:4] = ["e"]  
  
In [35]: lista  
Out[35]: ['a', 'b', 'c', 'd', 'e', 'f']
```

ELIMINANDO ELEMENTOS DE UNA LISTA

Podemos eliminar elementos de una lista con la instrucción `del`

Podemos usar la instrucción `del` con un slice para borrar una sublista

```
Terminal 3/A x

In [37]: a = ["uno", "dos", "tres"]

In [38]: a
Out[38]: ['uno', 'dos', 'tres']

In [39]: del a[1]

In [40]: a
Out[40]: ['uno', 'tres']
```

```
Terminal 3/A x

In [46]: lista = ["a", "b", "c", "d", "e", "f"]

In [47]: lista
Out[47]: ['a', 'b', 'c', 'd', 'e', 'f']

In [48]: del lista[1:5]

In [49]: lista
Out[49]: ['a', 'f']
```


REFERENCIAS A LISTAS Y EL OPERADOR IS

Analicemos las siguientes instrucciones:

```
Terminal 4/A x  
In [2]: a = [1, 2, 3]  
In [3]: b = [1, 2, 3]  
In [4]: c = a  
In [5]: a  
Out[5]: [1, 2, 3]  
In [6]: b  
Out[6]: [1, 2, 3]  
In [7]: c  
Out[7]: [1, 2, 3]
```

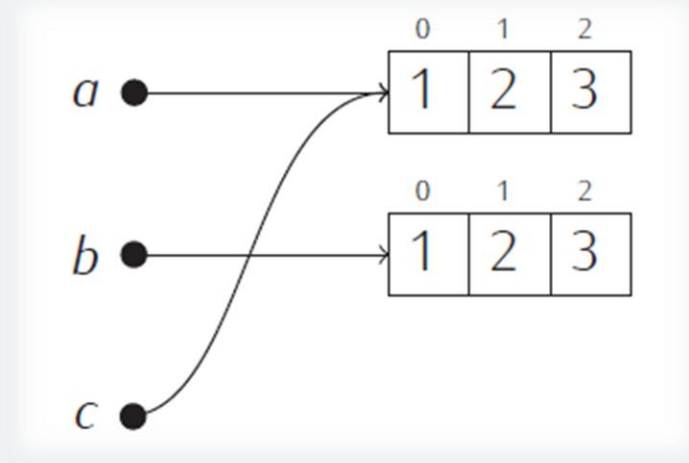
El contenido de las 3
listas es el mismo:

```
In [8]: a == b  
Out[8]: True  
  
In [9]: a == c  
Out[9]: True  
  
In [10]: b == c  
Out[10]: True
```

PERO...

Esto es lo que sucede en memoria:

```
Terminal 4/A ✕  
  
In [2]: a = [1, 2, 3]  
In [3]: b = [1, 2, 3]  
In [4]: c = a  
In [5]: a  
Out[5]: [1, 2, 3]  
In [6]: b  
Out[6]: [1, 2, 3]  
In [7]: c  
Out[7]: [1, 2, 3]
```



Las variables **a** y **b** referencian a listas diferentes (aunque tengan el mismo contenido), mientras que la variable **c** referencia a la misma lista que referencia la variable **a**

Y ESTO SE SABE CON EL OPERADOR IS

```
In [12]: a == b  
Out[12]: True
```

```
In [13]: b == c  
Out[13]: True
```

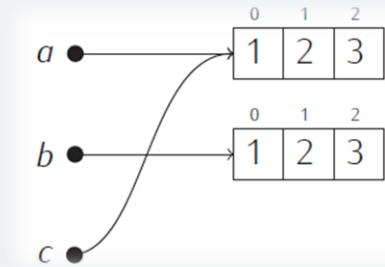
```
In [14]: a == c  
Out[14]: True
```

```
In [15]: a is b  
Out[15]: False
```

```
In [16]: b is c  
Out[16]: False
```

```
In [17]: a is c  
Out[17]: True
```

El operador `is` devuelve `True` si dos variables apuntan al mismo objeto, es decir, si ambas apuntan a la misma zona de memoria, y `False` en caso contrario



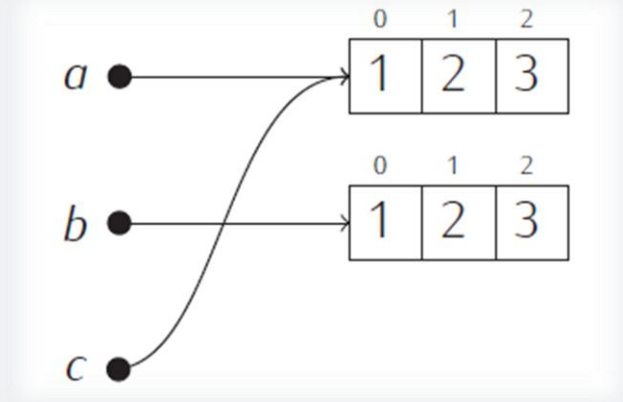
Otro ejemplo:

```
Terminal 4/A x  
  
In [26]: a = [1, 2]  
  
In [27]: a == [1, 2]  
Out[27]: True  
  
In [28]: a is [1, 2]  
Out[28]: False
```

SI DOS VARIABLES APUNTAN A LA MISMA LISTA, AL MODIFICAR UNA, SE CAMBIA LA OTRA



```
Terminal 4/A ✕  
  
In [46]: a = [1, 2, 3]  
In [47]: b = [1, 2, 3]  
In [48]: c = a  
  
In [49]: a  
Out[49]: [1, 2, 3]  
  
In [50]: b  
Out[50]: [1, 2, 3]  
  
In [51]: c  
Out[51]: [1, 2, 3]
```



```
In [52]: c[0] = 4  
  
In [53]: a  
....:  
Out[53]: [4, 2, 3]  
  
In [54]: b  
Out[54]: [1, 2, 3]  
  
In [55]: c  
Out[55]: [4, 2, 3]
```

LO MISMO PASA CON LOS PARÁMETROS

Un parámetro de tipo `list` es en realidad una referencia a la lista. Veamos un ejemplo:

```
EjemploParametroLista.py ✕  
1 def duplicar(lista: list) -> None:  
2     for i in range(0, len(lista)):  
3         lista[i] = lista[i] * 2  
4
```

Resultado de la ejecución

```
Terminal 6/A ✕  
  
In [6]: numeros = [1, 2, 3, 4, 5]  
  
In [7]: numeros  
Out[7]: [1, 2, 3, 4, 5]  
  
In [8]: duplicar(numeros)  
  
In [9]: numeros  
Out[9]: [2, 4, 6, 8, 10]
```

CLONACIÓN DE LISTAS

- Si queremos modificar una lista y mantener una copia de la lista original, tenemos que sacar una copia de la lista misma, no sólo de su referencia
- Este proceso es llamado «cloning» y se implementa con el operador de corte (slice)



b es una copia
(clon) de a

Tienen el
mismo
contenido

Pero no
referencian a la
misma lista

```
Terminal 5/A ✕

In [3]: a = [1, 2, 3]

In [4]: b = a[:]

In [5]: a
Out[5]: [1, 2, 3]

In [6]: b
Out[6]: [1, 2, 3]

In [7]: a == b
Out[7]: True

In [8]: a is b
Out[8]: False
```

FUNCIONES PROPIAS DE LISTAS (MÉTODOS DE LA CLASE LIST)

append: Adiciona un elemento al final de la lista

```
Terminal 6/A ✕  
  
In [11]: mi_lista=[]  
In [12]: mi_lista.append(5)  
In [13]: mi_lista.append(27)  
In [14]: mi_lista.append(3)  
In [15]: mi_lista.append(12)  
  
In [16]: mi_lista  
Out[16]: [5, 27, 3, 12]
```

insert: Inserta un elemento en una posición dada de la lista

```
In [16]: mi_lista  
Out[16]: [5, 27, 3, 12]  
  
In [17]: mi_lista.insert(1, 12)  
  
In [18]: mi_lista  
Out[18]: [5, 12, 27, 3, 12]
```

FUNCIONES PROPIAS DE LISTAS (MÉTODOS)

count: Cuenta cuántas veces aparece un elemento en la lista

extend: Inserta una lista entera al final de la lista

```
Terminal 6/A ✕  
  
In [18]: mi_lista  
Out[18]: [5, 12, 27, 3, 12]  
  
In [19]: mi_lista.count(12)  
Out[19]: 2
```

```
Terminal 6/A ✕  
  
In [21]: mi_lista  
Out[21]: [5, 12, 27, 3, 12]  
  
In [22]: mi_lista.extend([5, 9, 5, 11])  
  
In [23]: mi_lista  
Out[23]: [5, 12, 27, 3, 12, 5, 9, 5, 11]
```


FUNCIONES PROPIAS DE LISTAS (MÉTODOS)

index: Busca el índice (posición) de un elemento en la lista

reverse: Invierte los elementos de una lista

```
Terminal 6/A ✕  
In [23]: mi_lista  
Out[23]: [5, 12, 27, 3, 12, 5, 9, 5, 11]  
  
In [24]: mi_lista.index(9)  
Out[24]: 6
```

```
Terminal 6/A ✕  
In [26]: mi_lista  
Out[26]: [5, 12, 27, 3, 12, 5, 9, 5, 11]  
  
In [27]: mi_lista.reverse()  
  
In [28]: mi_lista  
Out[28]: [11, 5, 9, 5, 12, 3, 27, 12, 5]
```

FUNCIONES PROPIAS DE LISTAS (MÉTODOS)

sort: Ordena los elementos de una lista

```
Terminal 6/A ✕  
In [28]: mi_lista  
Out[28]: [11, 5, 9, 5, 12, 3, 27, 12, 5]  
  
In [29]: mi_lista.sort()  
  
In [30]: mi_lista  
Out[30]: [3, 5, 5, 5, 9, 11, 12, 12, 27]
```

remove: Elimina la primera ocurrencia de un elemento dado de la lista

```
Terminal 6/A ✕  
In [30]: mi_lista  
Out[30]: [3, 5, 5, 5, 9, 11, 12, 12, 27]  
  
In [31]: mi_lista.remove(12)  
  
In [32]: mi_lista  
Out[32]: [3, 5, 5, 5, 9, 11, 12, 27]
```

FUNCIONES PROPIAS DE LISTAS (MÉTODOS)

copy: Crea una nueva lista que es una copia de la lista original

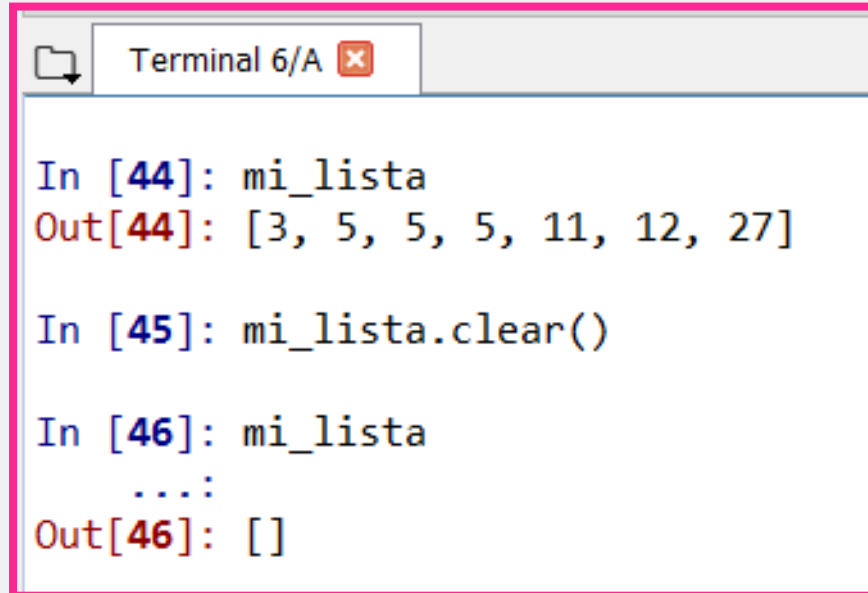
```
Terminal 6/A ✕  
  
In [32]: mi_lista  
Out[32]: [3, 5, 5, 5, 9, 11, 12, 27]  
  
In [33]: otra_lista = mi_lista.copy()  
  
In [34]: otra_lista  
Out[34]: [3, 5, 5, 5, 9, 11, 12, 27]
```

pop: Remueve un elemento de una posición dada de la lista y lo devuelve

```
Terminal 6/A ✕  
  
In [39]: mi_lista  
Out[39]: [3, 5, 5, 5, 9, 11, 12, 27]  
  
In [40]: eliminado = mi_lista.pop(4)  
....:  
  
In [41]: mi_lista  
Out[41]: [3, 5, 5, 5, 11, 12, 27]  
  
In [42]: eliminado  
Out[42]: 9
```

FUNCIONES PROPIAS DE LISTAS (MÉTODOS)

clear: Elimina todos los elementos de la lista



```
Terminal 6/A ✕  
  
In [44]: mi_lista  
Out[44]: [3, 5, 5, 5, 11, 12, 27]  
  
In [45]: mi_lista.clear()  
  
In [46]: mi_lista  
...:  
Out[46]: []
```

DE CADENAS A LISTAS Y VISCEVERSA

En muchas ocasiones nos encontraremos convirtiendo cadenas en listas y viceversa. Python nos ofrece una serie de utilidades que conviene conocer si queremos ahorrarnos muchas horas de programación



DE CADENA A LISTA ...

Para obtener una lista con todas las palabras de una cadena, usamos el método `split` de string

```
Terminal 7/A ✕  
  
In [11]: cadena = "uno dos tres"  
  
In [12]: cadena  
Out[12]: 'uno dos tres'  
  
In [13]: lista = cadena.split()  
  
In [14]: lista  
Out[14]: ['uno', 'dos', 'tres']
```

DE CADENA A LISTA ...

`split` con delimitador o separador

```
Terminal 7/A ✕  
  
In [16]: cadena = "Hola,cómo,estás?"  
  
In [17]: lista = cadena.split(",")  
        ....:  
  
In [18]: cadena  
Out[18]: 'Hola,cómo,estás?'  
  
In [19]: lista  
Out[19]: ['Hola', 'cómo', 'estás?']
```

DE LISTA A CADENA...

Para obtener una cadena con los valores de una lista, usamos el método **join** de string

```
In [26]: a = " ".join(["uno", "dos", "tres"])

In [27]: a
Out[27]: 'uno dos tres'

In [28]: a = "--".join(["uno", "dos", "tres"])

In [29]: a
Out[29]: 'uno--dos--tres'

In [30]: a = ":".join(["uno", "dos", "tres"])

In [31]: a
Out[31]: 'uno:dos:tres'
```


LIST

Además de ser un tipo es una función que nos sirve para construir una lista

Ya lo habíamos visto con range

```
Terminal 2/A x
In [26]: a = list(range(1, 4))
In [27]: print(a)
[1, 2, 3]
```

También funciona con cadenas de caracteres

```
Terminal 1/A x
In [30]: a = list("1289463027494")
In [31]: a
Out[31]: ['1', '2', '8', '9', '4', '6', '3', '0', '2', '7', '4', '9', '4']
In [32]: b = list("la casa blanca")
In [33]: b
Out[33]: ['l', 'a', ' ', 'c', 'a', 's', 'a', ' ', 'b', 'l', 'a', 'n', 'c', 'a']
```

PERO NO FUNCIONA CON UN NÚMERO



```
Terminal 1/A ✕  
  
In [34]: c = list(1289463027494)  
Traceback (most recent call last):  
  
  File "<ipython-input-34-bc9725c22f55>", line 1, in <module>  
    c = list(1289463027494)  
  
TypeError: 'int' object is not iterable
```



1. Escribe la función `promedio_lista` que recibe una lista de números (positivos y negativos) y retorna el valor promedio de los números positivos



2. Escribe la función `mediana_lista` que recibe una lista de números (positivos y negativos) y retorna el valor de la mediana de los números. Considere el caso en el que la lista tiene una cantidad de números impar y el caso en el que la cantidad de números es par

3. Escribe la función `menor_posicion_lista` que recibe una lista de números (positivos y negativos) y retorna la posición del menor valor que se encuentra en la lista



4. Escribe la función `buscar_numero` que recibe por parámetro una lista de números y un número y retorna la posición donde se encuentra el número. Si el número no existe en la lista, retorna -1.