Tema 6. Estructuras de datos complejos

Apuntes extras

Índice

[**Estructuras de datos 2**](#_8dl8jucq2zd4)

[**Listas 2**](#_8x9b5bhildwv)

[Comportamiento de la lista 2](#_hirvjroxtrg)

[¿Cómo ir por las listas? 3](#_7b1t1lu76evt)

[Insertar elementos a una lista 4](#_kns2hhuv7tvi)

[Eliminar elementos de la lista 4](#_qhlupwve28q2)

[Métodos comunes para las listas 5](#_20lxbl8pcz38)

[Listas por comprensión 5](#_xsy5jm21l758)

[**Diccionarios 7**](#_o404ov9zp5i)

[Acceso a los datos de un diccionario 7](#_wj2x4xqqw9pf)

[Métodos y funciones 10](#_vpc6974086ek)

[**Tuplas 11**](#_jsxtw6e5s24)

[Rangos de índices 11](#_4831z5hejtjs)

[Índices negativos 12](#_7p3f9ymzem2w)

[Operador in 12](#_bjdvbal80svh)

[Unir tuplas 12](#_p8m2yyfwqvwc)

[Asignación múltiple 13](#_z92wcswr4zmv)

[Iteración 13](#_8l7jkmcpo060)

[Métodos y funciones 14](#_z5fpz6ruthee)

[- - - - - - - Puede ayudar en el examen (posiblemente) 14](#_v1lpckgz7qyj)

[**Conjuntos 15**](#_t8ymehraf8dr)

[Operaciones posibles sobre los conjuntos 15](#_b3krp6nosfag)

[Ir por sus elementos (iteración) 15](#_8avquarcj3w0)

[Métodos principales 16](#_lcyfyirchu29)

[**Objetos mutables e inmutables 18**](#_19f1lyg5iu4u)

[**Paso por valor o por referencia (parámetros) 18**](#_o91lihg1uh5t)

# Estructuras de datos

Hasta ahora hemos manipulado datos sencillos, incluyendo números, texto y booleanos.

Pero hará falta el uso de colecciones de datos

Una estructura de datos es una agrupación de datos estructurada. Al final de la jerarquía se compone por datos sencillos, es decir, una lista (complejo) puede componerse de una biblioteca (complejo), pero la biblioteca tendrá esos datos sencillos.

Una estructura de datos puede ser parte de una expresión y puede ser asignado a una variable.

Mirar documentación: <https://docs.python.org/3/tutorial/datastructures.html>

# Listas

**Concepto: colección ordenada de datos**

Equivalente a vectores o arrays (depende del lenguaje de programación).

**Sintaxis en Python:** entre corchetes y separados por comas. Se guardan en variables.

Ejemplo:

my\_list = [1, 14, 5, 22]

Una lista puede contener **cualquier tipo de valor.**

Por ejemplo: [5, 3.14, None, True, [“hello”, 3], “text”]

Compuesto por: Entero, real (decimal), valor especial /ausencia de valor, booleano, otra lista (compuesta por texto y un número entero) y texto.

## Comportamiento de la lista

Cada elemento de una lista se puede tratar como una variable.

Operaciones posibles

* Darle valor o modificarlo (set/assign/update)
* Acceder al contenido de la variable (retrieve/get value/check)

**El acceso es hecho señalando a un índice numérico.** Empieza en 0, 1, 2, 3…

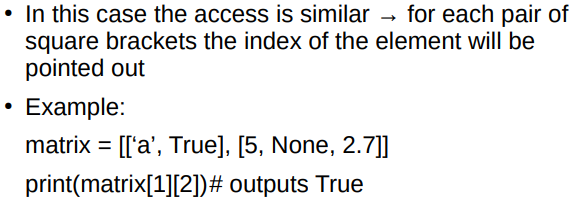
Ejemplo:

my\_list = [“more”, “than”, “meets”, “the”, “eye”]

my\_list[2] → “meets”

Para modificar un elemento de la lista se trata como una variable:  
my\_list[2] = “Julia”

Es posible manejar **multi-listas**, es decir, una lista dentro de otra lista, como una matriz bidimensional ¿? preguntar a Dani si eso.



**matrix = [[‘a’, True], [5, None, 2.7]]**

**print(matrix[1][2])# outputs 2.7**

[‘a’, True] es el elemento 0

[5, None, 2.7] es el elemento 1

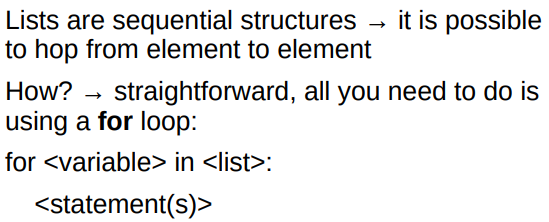
2.7 es el elemento 2 del 1

**Segmentación de listas:** operación de extraer una sublista de la lista base. Se hace usando los dos puntos( : )

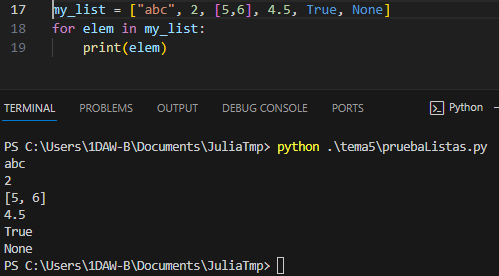


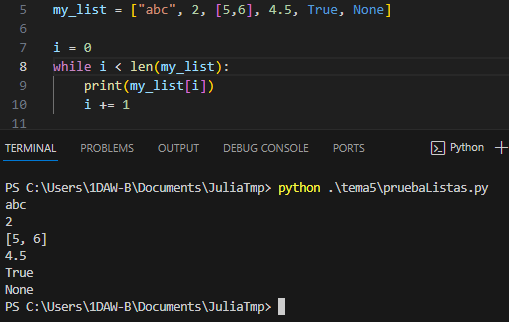
## ¿Cómo ir por las listas?

Metemos la lista en una variable y usamos el for para recorrer los elementos



CON FOR:



CON WHILE:  


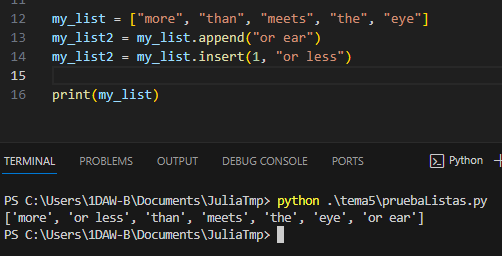
## Insertar elementos a una lista

* Append: al final
* Insert: a la posición indicada

my\_list = [“more”, “than”, “meets”, “the”, “eye”]

my\_list.append(”or ear”)

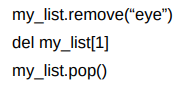
my\_list.insert(1, “or less”)



## Eliminar elementos de la lista

* Remove: señalando el elemento
* Del: señalando el índice/posición
* Pop: elimina y devuelve el último elemento

Ejemplo:



 my\_list.clear()

## Métodos comunes para las listas

Uso: lista.append()

– append → añadir contenido al final

– remove → eliminar un elemento

– index → da el índice/posición de un elemento

– count → contar cuantos elementos iguales al proporcionado hay en una lista

– reverse → cambiar el orden de los elementos

– sort → ordena los elementos de forma alfabética o numérica

– copy → lista1 = lista2.copy()

* Ahora serán dos listas independientes
* No es lo mismo que lista1 = lista2, porque si hacemos lista1.pop() eliminando el último valor en lista2 también se eliminará porque el contenido de la lista está en solo una unidad de memoria pero se le asigna los dos nombres de variable.

– extend → juntar dos listas. lista2.extend(lista1) : lista2 tendrá sus elementos + los elementos de lista1 al final. Sirve no solo para las listas, también para otros elementos iterables (ej: tuplas)

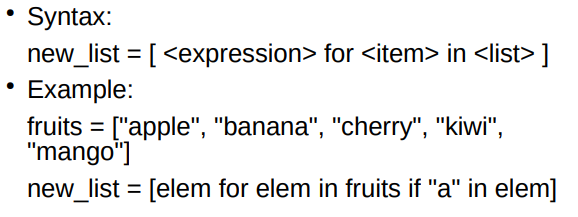
* Otra forma de hacerlo es concatenando:

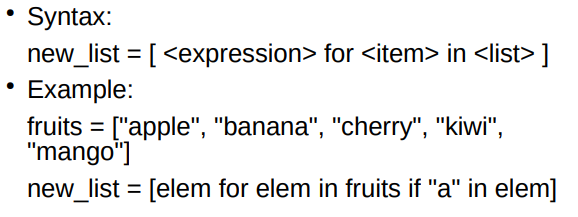
lista2 + lista1

La diferencia entre las dos opciones es que el valor no cambia, solo se muestra la concatenación y no sirve con otros elementos iterables.

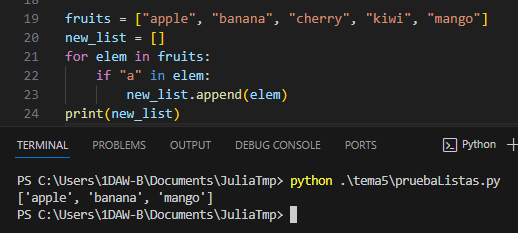
## Listas por comprensión

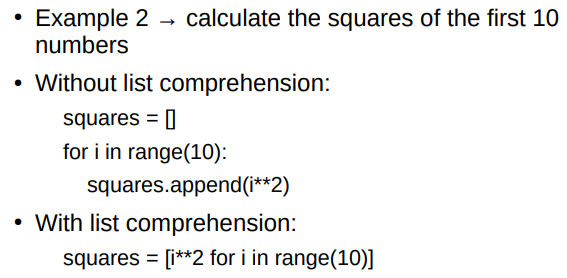
Sintaxis especial para crear una lista a partir de otra cogiendo los elementos que cumplan la condición.





Si no tuviéramos esta herramienta podríamos hacerlo de la siguiente manera:





# Diccionarios

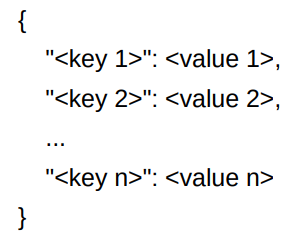
**Concepto:** es una estructura asociativa. Contiene pares de los elementos/valor y su clave.

Para acceder a un elemento/valor en esta estructura es necesario saber su clave

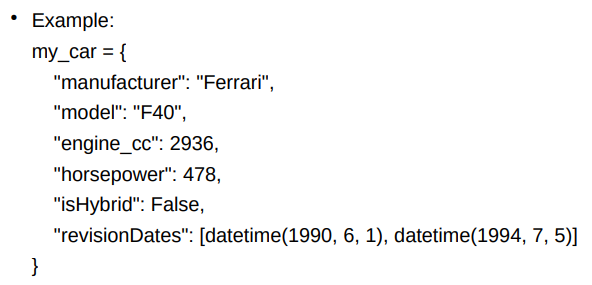
**Nombramiento**: valor\_by\_clave



**Sintaxis:**

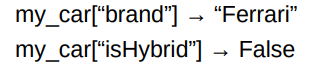


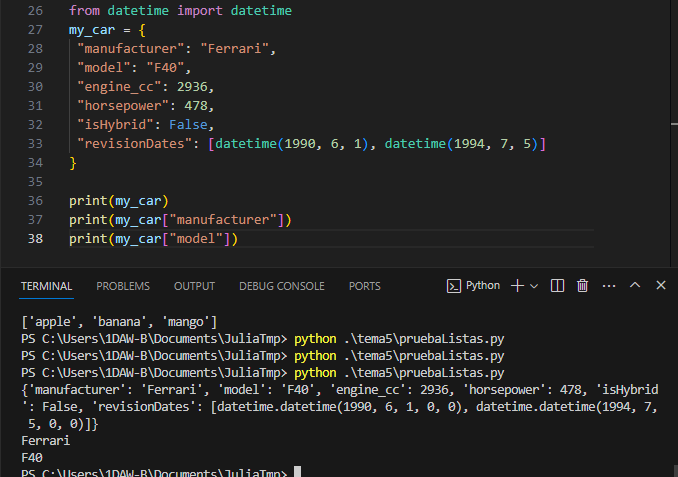
La clave es un texto que va entre comillas, no pueden ser números, booleanos o listas y va seguido de dos puntos. Mientras que el valor sí pueden ser texto, números, booleanos o listas.



## Acceso a los datos de un diccionario

Hay que conocer la clave del dato que deseamos conocer. Las comillas en las claves son necesarias.



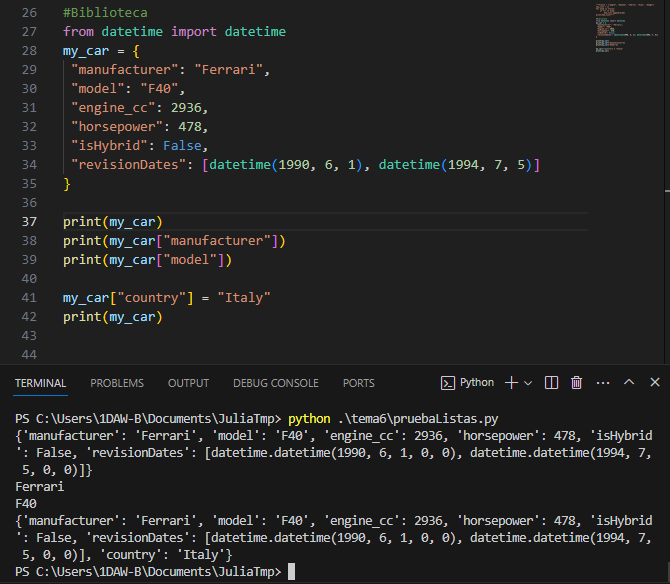


Las claves deben ser ÚNICAS.

Para cambiar el valor se hace así (asignar valor a la clave como si fuera una variable):

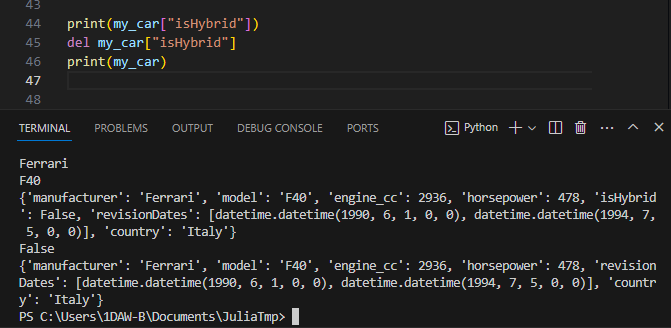


Si hacemos referencia a una clave que no existe saldrá error, pero si le damos valor se creará.



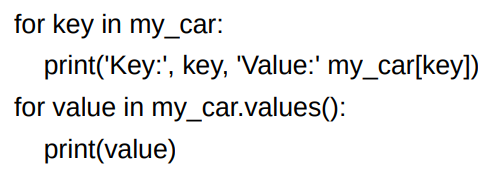
Eliminamos elementos con del:

del my\_car[“isHybrid”]



Como una biblioteca es iterable se puede meter en un for:

La primera forma devuelve la clave y los valores, la segunda solo los valores



## Métodos y funciones

– keys → devuelve las claves

– values → devuelves los valores

– items → muestra los pares de clave/valor que hay en formato de lista con tuplas

– clear → vacía la biblioteca, no la elimina solo la deja sin contenido

– get, update, pop

* get → my\_car.get(“model”) hace lo mismo que my\_car[“model”]. Hay veces que en mejor usar el get
* update → actualiza el contenido en formato de lista o diccionario: my\_car.update({‘country’: ‘Italy’}). Hace lo mismo que my\_car[“country”] = Italy, solo que es posible actualizar varios elementos a la vez.

my\_car.update({‘cylinders’ : 8, ‘year’ : 1990}), pueden ser campos nuevos

* pop → elimina el elemento indicado: my\_car.pop(‘year’)

– copy / dict→ dictionaries are assigned by reference

* copy → copia el diccionario en otra variable y serán independientes la una de la otra.
* dict → ¿?

# Tuplas

**Concepto**: casi lo mismo que las listas, es una colección de valores ordenados que se pueden repetir. La diferencia a parte de la sintaxis es que *no son objetos mutables*, es decir que a una lista se le pueden cambiar los valores pero a una tupla no, sus valores son constantes y una vez declarada no cambiará.

Su uso más común es para darle más de un valor a una variable.

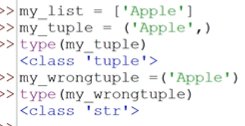
**Sintaxis:**  
Se escriben con paréntesis.



La excepción es que si la tupla contiene solo un valor es necesario poner una coma antes de cerrar paréntesis:



Ejemplo de diferencias y cómo declararla correctamente:



Se utilizan para manipular múltiples valores en una variable o devolver varios valores de una función.

Al igual que en las listas, las tuplas pueden combinar valores de diferentes tipos (números, texto, etc)

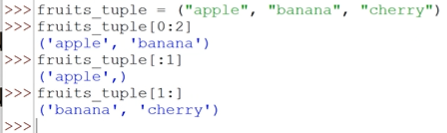
Se accede de la misma forma a su contenido:



**Es importante recordar que no es posible modificar una tupla, tampoco eliminar o añadir nuevos elementos/valores.**

## Rangos de índices

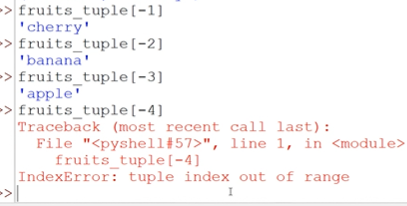
Se pueden usar para obtener sub-tuplas:



## Índices negativos

Sería como empezar desde el final



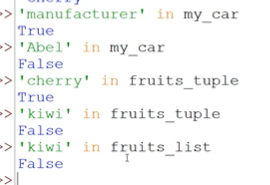


Funciona también con las listas.



## Operador in

Para comprobar si el elemento está en la tupla, lista o biblioteca



## Unir tuplas

Se puede obtener una nueva tupla de juntar dos o más tuplas, para hacerlo se usa el símbolo de concatenación (+)

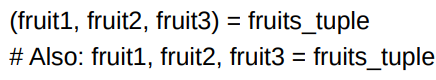
Por ejemplo:

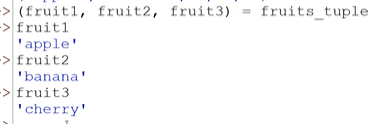


Se deberá meter este valor en una nueva variable.

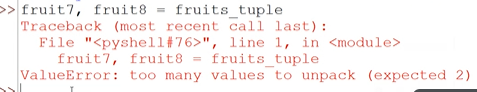
## Asignación múltiple

Pueden ser “desempaquetadas”. Esto significa que es posible asignar cada valor de una tupla en una variable

 (Se puede hacer sin los paréntesis)

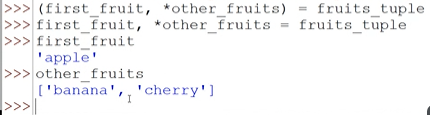


Se deben poner el mismo número de elementos que en la tupla:



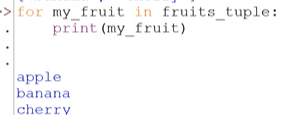
Si se usa el comodín \* se podría hacer lo de arriba (asignar sub-tuplas a las variables en formato de lista).





## Iteración

Se puede iterar sobre sus valores de la misma forma que en las listas

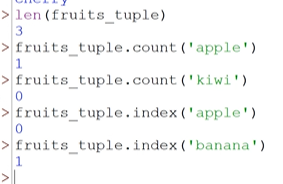


## Métodos y funciones

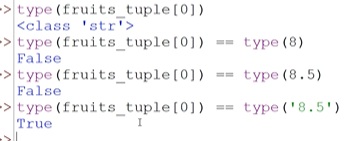
– len() → cuenta los elementos

– count() → cuenta cuantas veces se repite un elemento en la tupla

– index() → dice la posición en la que se encuentra un elemento, empieza en 0



## - - - - - - - Puede ayudar en el examen (posiblemente)



# Conjuntos

**Concepto:** representa una colección de elementos, pero no en un orden particular. No se pueden repetir elementos.

**Sintaxis:**

Se usan las llaves igual que en los diccionarios. La diferencia es que aquí no hay clave y valor, solo valor



Aquí el orden es irrelevante y no existe, los elementos no tienen posición fija, dando igual el orden en el que se han declarado.

## Operaciones posibles sobre los conjuntos

* Comprobar si un elemento está en el conjunto → con el operador **in**

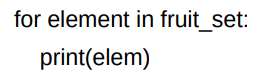


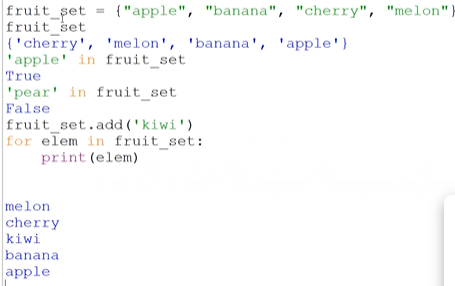
* Añadir nuevos elementos → usando el método **add**

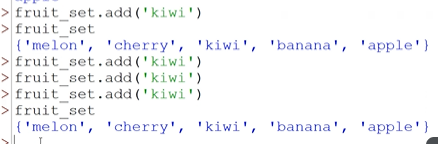
****

## Ir por sus elementos (iteración)

Los conjuntos son objetos iterables, para ir por los elementos que tenga debemos usar **for**







## Métodos principales

– add → añadir

– update → actualizar/cambiar

– remove → eliminar

– discard → igual que remove

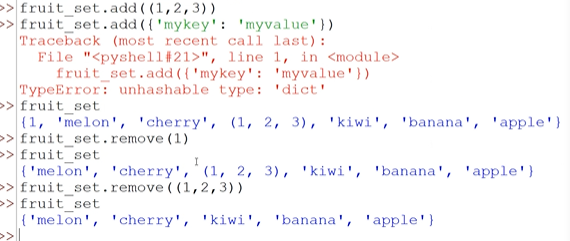
– clear → elimina el contenido

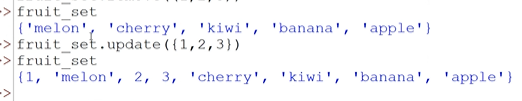
– copy → copiar de forma independiente, también se puede usar set

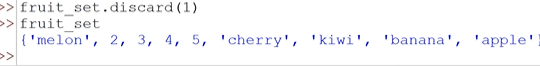
– difference, intersection, union

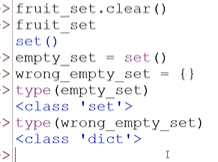
* difference → elementos del primer conjunto que no están en el segundo
* intersection → elementos que coinciden de los dos conjuntos + los diferentes
* union → todos los elemento del primer y segundo conjunto

**Ejemplos de uso:**

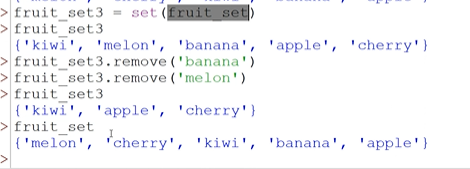


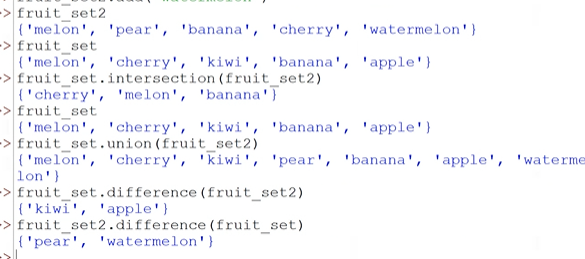




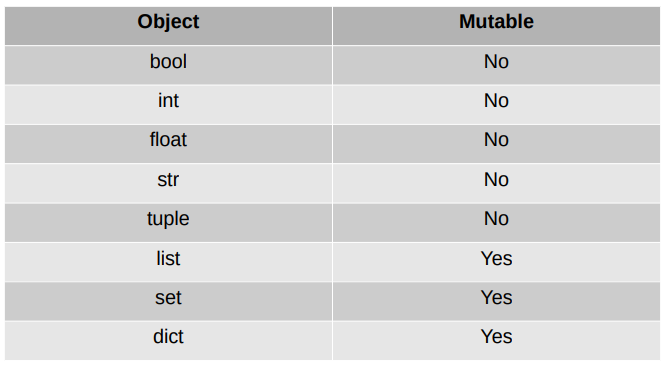








# Objetos mutables e inmutables



# Paso por valor o por referencia (parámetros)

Cuando se define una función también se definen sus parámetros. Podemos manejar los parámetros como si fueran variables, por lo tanto es posible modificar su contenido.

Dicho esto, algunos conservarán los cambios después de que la función finalice (listas, conjuntos y diccionarios), mientras que otros no (el resto).

El primer caso sería pasado por **referencia**

El segundo caso sería pasado por **valor**

**En python todo se pasa por valor**, incluyendo las listas, conjuntos y diccionarios, ya que se indica la dirección de memoria y hace el truco y pasa por valor.

