Métodos de listas y funciones útiles en Python

Método / Función	Descripción	Ejemplo en código
`append(x)`	Agrega un elemento al final de la lista.	`nums = [1, 2] nums.append(3) # [1, 2, 3]`
`extend(iterable)`	Agrega todos los elementos de otro iterable al final.	`nums = [1, 2] nums.extend([3, 4]) # [1, 2, 3, 4]`
`insert(i, x)`	Inserta un elemento en la posición indicada (desplaza los demás).	`letras = ['a', 'b'] letras.insert(1, 'z') # ['a', 'z', 'b']`
`remove(x)`	Elimina la **primera** aparición del elemento indicado.	`nums = [1, 2, 2, 3] nums.remove(2) # [1, 2, 3]`
`pop([i])`	Quita y devuelve el elemento en el índice dado. Si no se indica, quita el último.	`letras = ['a', 'b', 'c'] letras.pop(1) # 'b' → ['a', 'c']`
`clear()`	Elimina todos los elementos de la lista.	`nums = [1, 2, 3] nums.clear() # []`
`index(x)`	Devuelve el índice de la **primera** aparición de `x`. Error si no está.	`letras = ['a', 'b', 'c'] letras.index('b') # 1`
`count(x)`	Devuelve cuántas veces aparece un valor en la lista.	`nums = [1, 2, 2, 3] nums.count(2) # 2`
`sort()`	Ordena la lista en el lugar (puede recibir `reverse=True` o `key=`).	`nums = [3, 1, 2] nums.sort() # [1, 2, 3]`
`reverse()`	Invierte el orden de la lista **en el lugar**.	`nums = [1, 2, 3] nums.reverse() # [3, 2, 1]`
`copy()`	Devuelve una **copia superficial** de la lista.	`nums = [1, 2, 3] copia = nums.copy()`

Funciones integradas de Python

Función	Descripción	Ejemplo en código
`len(lista)`	Devuelve la cantidad de elementos de la lista.	`len([1, 2, 3]) # 3`
`sum(lista)`	Devuelve la suma de los elementos (si son numéricos).	`sum([1, 2, 3]) # 6`
`max(lista)`	Devuelve el valor máximo.	`max([1, 5, 3]) # 5`
`min(lista)`	Devuelve el valor mínimo.	`min([1, 5, 3]) # 1`
`sorted(lista)`	Devuelve una **nueva lista ordenada**, no modifica la original.	`sorted([3, 1, 2]) # [1, 2, 3]`
`list(iterable)`	Convierte un iterable en lista.	`list("hola") # ['h','o','l','a']`
`enumerate(lista)`	Devuelve un iterable de pares `(índice, valor)`. Útil en bucles.	`for i, v in enumerate(['a','b']):\n print(i, v)`
`zip(lista1, lista2)`	Combina elementos de varias listas en tuplas.	`list(zip([1,2],[3,4])) # [(1,3), (2,4)]`

Métodos propios de las tuplas

Las **tuplas** en Python son estructuras de datos **inmutables**, lo que significa que, a diferencia de las listas, **no se pueden modificar** una vez creadas (no podés agregar, quitar o cambiar elementos).

Por eso, los **métodos disponibles** para tuplas son mucho más limitados. Aun así, se pueden usar varias **funciones integradas de Python** que trabajan sobre tuplas (igual que sobre listas y otros iterables).

Método	Descripción	Ejemplo en código
`count(x)`	Devuelve cuántas veces aparece un valor en la tupla.	`t = (1, 2, 2, 3) \n t.count(2)  # 2`
`index(x)`	Devuelve el índice de la **primera** aparición de `x`. Error si no está.	`t = ('a','b','c') \n t.index('b')  # 1`

Funciones integradas útiles con tuplas

Función	Descripción	Ejemplo en código
`len(tupla)`	Devuelve la cantidad de elementos.	`len((1,2,3))  # 3`
`max(tupla)`	Devuelve el valor máximo (si son comparables).	`max((3,1,5))  # 5`
`min(tupla)`	Devuelve el valor mínimo.	`min((3,1,5))  # 1`
`sum(tupla)`	Devuelve la suma de los elementos (si son numéricos).	`sum((1,2,3))  # 6`
`tuple(iterable)`	Convierte un iterable en tupla.	`tuple("hola")  # ('h','o','l','a')`
`sorted(tupla)`	Devuelve una **lista ordenada** con los elementos de la tupla.	`sorted((3,1,2))  # [1,2,3]`
`enumerate(tupla)`	Devuelve pares `(índice, valor)`.	`for i,v in enumerate(('a','b')):\n    print(i,v)`
`zip(t1, t2)`	Combina elementos de varias tuplas en tuplas nuevas.	`list(zip((1,2),(3,4)))  # [(1,3),(2,4)]`

La gran diferencia con listas es que **no existen métodos como `append`, `insert`, `remove`, `sort`, etc.**, porque las tuplas **no se pueden modificar**.

Métodos de los diccionarios

Los **diccionarios** en Python son estructuras de datos que almacenan **pares clave–valor**. Son **mutables**, lo que significa que se pueden modificar, agregar y eliminar elementos.

Veamos una tabla con los **métodos principales** y las **funciones integradas más usadas** con diccionarios.

Método	Descripción	Ejemplo en código
`get(clave, [defecto])`	Devuelve el valor de una clave. Si no existe, devuelve `None` o el valor por defecto.	`notas = {"Ana": 8} \n notas.get("Ana")  # 8 \n notas.get("Juan", 0)  # 0`
`keys()`	Devuelve una vista con todas las claves.	`notas = {"Ana": 8, "Juan": 6} \n list(notas.keys())  # ['Ana','Juan']`
`values()`	Devuelve una vista con todos los valores.	`notas.values()  # [8, 6]`
`items()`	Devuelve una vista con pares `(clave, valor)`.	`notas.items()  # [('Ana',8), ('Juan',6)]`
`update(otro_dicc)`	Actualiza el diccionario con pares de otro	`notas = {"Ana": 8} \n notas.update({"Juan": 6})`

Método	Descripción	Ejemplo en código
	(sobrescribe si ya existen).	
`pop(clave, [defecto])`	Elimina y devuelve el valor de la clave. Si no existe, lanza error salvo que se dé un valor por defecto.	`notas = {"Ana": 8} \n notas.pop("Ana")  # 8`
`popitem()`	Elimina y devuelve un **par clave–valor** (último agregado).	`notas = {"Ana": 8, "Juan": 6} \n notas.popitem()`
`setdefault(clave, [defecto])`	Devuelve el valor de una clave. Si no existe, la crea con el valor por defecto.	`notas = {"Ana": 8} \n notas.setdefault("Juan", 10)`
`clear()`	Elimina todos los elementos del diccionario.	`notas.clear()  # {}`
`copy()`	Devuelve una copia superficial del diccionario.	`copia = notas.copy()`

Funciones integradas útiles con diccionarios

Función	Descripción	Ejemplo en código
`len(diccionario)`	Devuelve la cantidad de pares clave–valor.	`len({"a":1,"b":2})  # 2`
`dict(iterable)`	Crea un diccionario a partir de pares `(clave, valor)`.	`dict([("a",1),("b",2)])`
`sorted(diccionario)`	Devuelve una **lista ordenada de las claves**.	`sorted({"c":3,"a":1,"b":2})  # ['a','b','c']`

A diferencia de listas y tuplas, los diccionarios trabajan fuertemente con **claves y valores**, y eso se nota en métodos como `keys()`, `values()` e `items()`.

Métodos de los conjuntos

Los **conjuntos (`set`)** en Python representan colecciones **no ordenadas**, **sin elementos duplicados**, y permiten realizar operaciones de teoría de conjuntos (uniones, intersecciones, diferencias, etc.).

Método	Descripción	Ejemplo en código	
`add(x)`	Agrega un elemento al conjunto (si ya está, no pasa nada).	`s = {1, 2} \n s.add(3)  # {1, 2, 3}`	
`update(iterable)`	Agrega múltiples elementos desde otro iterable.	`s = {1, 2} \n s.update([3, 4])  # {1, 2, 3, 4}`	
`remove(x)`	Elimina un elemento. Error si no existe.	`s = {1, 2, 3} \n s.remove(2)  # {1, 3}`	

Método	Descripción	Ejemplo en código	
`discard(x)`	Elimina un elemento si existe, sin error si no está.	`s = {1, 2, 3} \n s.discard(4)  # {1, 2, 3}`	
`pop()`	Elimina y devuelve un elemento **arbitrario** (no al azar, pero tampoco controlable).	`s = {1, 2, 3} \n s.pop()`	
`clear()`	Vacía el conjunto.	`s = {1, 2} \n s.clear()  # set()`	
`copy()`	Devuelve una copia superficial del conjunto.	`s2 = s.copy()`	
`union(otro)` o `|`	Devuelve un nuevo conjunto con todos los elementos de ambos.	`{1,2}.union({2,3})  # {1,2,3}`	
`intersection(otro)` o `&`	Devuelve un conjunto con los elementos en común.	`{1,2,3} & {2,3,4}  # {2,3}`	
`difference(otro)` o `-`	Devuelve los elementos que están en el primero pero no en el segundo.	`{1,2,3} - {2}  # {1,3}`	
`symmetric_difference(otro)` o `^`	Devuelve los elementos que están en uno u otro, pero no en ambos.	`{1,2,3} ^ {2,3,4}  # {1,4}`	
`issubset(otro)`	Verifica si es subconjunto de otro.	`{1,2} <= {1,2,3}  # True`	
`issuperset(otro)`	Verifica si es superconjunto de otro.	`{1,2,3} >= {2}  # True`	
`isdisjoint(otro)`	Verifica si no tienen elementos en común.	`{1,2}.isdisjoint({3,4})  # True`	

Funciones integradas útiles con conjuntos

Función	Descripción	Ejemplo en código
`len(set)`	Devuelve la cantidad de elementos (sin duplicados).	`len({1,2,2,3})  # 3`
`set(iterable)`	Crea un conjunto a partir de un iterable.	`set("hola")  # {'h','o','l','a'}`
`sorted(set)`	Devuelve una **lista ordenada** con los elementos del conjunto.	`sorted({3,1,2})  # [1,2,3]`