3. Diccionarios y Sets

🐍 Apuntes Esenciales de Python: Sets y Diccionarios

1. Introducción a las Estructuras de Datos

En Python, las estructuras de datos son contenedores que organizan la información de manera eficiente. Las **Listas** y **Tuplas** almacenan datos **ordenados** por posición. Por otro lado, los **Sets** y **Diccionarios** están diseñados para colecciones de datos donde el **orden no es el criterio principal** de acceso, sino la unicidad o la asociación.

Diccionarios: se definen con llaves { } y almacenan pares de clave: valor. A diferencia de las listas, no usas índices numéricos sino claves inmutables (strings, números, tuplas).
Ejemplo:

```
perfil = {"nombre": "Juan",
"edad": 30,
"activo": True}
print(perfil["edad"])
```

Output: Resultado: 30

Son:

- Mutable: Puedes modificar, añadir o eliminar elementos
- Acceso directo: Recupera valores mediante sus claves
- Orden de inserción: Python mantiene el orden (versiones 3.7+)
- Enfoque para Análisis de Datos: Los diccionarios simulan registros de bases de datos o documentos JSON, mapeando identificadores (claves) a su información (valores).

Sets: se definen con llaves { } o usando "set()". Su superpoder es eliminar automáticamente elementos duplicados.

```
datos = {1,2, 3, 3, 4}
print(datos)
```

Output: Resultado: {1,2, 3, 4}

Son:

- No ordenado: No mantiene orden de inserción
- No indexable: No puedes usar set[0]
- Solo elementos únicos: Elimina duplicados automáticamente
- Mutable: Puedes añadir y eliminar elementos

📚 Métodos Esenciales de Estructuras de Datos en Python

Estructura	Símbolo	Mutabilidad	Método/Propiedad Más Usado	Función Principal
Lista	`[ ]`	✅ Mutable	`.append(elemento)`	Añade un elemento al final.
(Ordenada)			`.insert(índice, elemento)`	Inserta un elemento en una posición específica.
			`.pop(índice)`	Elimina y devuelve el elemento en un índice (o el último).
			`.remove(elemento)`	Elimina la primera aparición de un valor.
			`[índice]`	Acceso, modificación o *slicing* (subconjunto).
			`len()`	Devuelve el número de elementos.
Tupla	`( )`	❌ Inmutable	`[índice]`	Acceso y *slicing* (no modificación).
(Ordenada)			`.count(elemento)`	Cuenta las veces que aparece un valor.
			`.index(elemento)`	Devuelve el índice de la primera aparición del valor.
			`len()`	Devuelve el número de elementos.

Estructura	Símbolo	Mutabilidad	Método/Propiedad Más Usado	Función Principal
Diccionario	`{clave: valor}`	✅ Mutable	`[clave] = valor`	Añade o modifica el valor de una clave.
(Clave-Valor)			`.get(c, defecto)`	Acceso seguro; devuelve valor por defecto si la clave no existe.
			`.keys()` / `.values()` / `.items()`	Devuelve vistas para iterar sobre claves, valores, o pares.
			`.pop(clave)`	Elimina y devuelve el valor de la clave especificada.
Set	`{ }`	✅ Mutable	`.add(elemento)`	Añade un único elemento.
(Únicos, Desordenado)			`.update(iterable)`	Añade múltiples elementos de un iterable.
			`.discard(elemento)`	Elimina un elemento de forma segura (sin error si no existe).
			`.union()` / `.intersection()`	Realiza operaciones de conjuntos (ej. unir colecciones, encontrar elementos comunes).
Comunes			`len(obj)`	Obtiene el número de elementos/pares.
			`in`	Comprueba la pertenencia (valor en Listas/Tuplas/Sets; clave en Diccionarios).

2. Sets (Conjuntos) 🧩: La Colección de Elementos Únicos

Un **Set** es como un conjunto matemático: es una colección que garantiza que todos sus elementos son diferentes entre sí (únicos).

Propiedades Clave del Set

Propiedad	Descripción	Importancia para el programador
Elementos Únicos	Un set elimina automáticamente cualquier elemento duplicado que se intente añadir.	Ideal para **filtrar listas** y obtener valores sin repetición.
Desordenado	Los elementos **no mantienen un orden fijo**. Por lo tanto, **no se pueden indexar** (no puedes pedir el elemento en la posición 0, por ejemplo).	No debes confiar en que los elementos saldrán en el mismo orden en que los introdujiste.
Mutable	Puedes añadir o eliminar elementos después de crear el set.	Puedes manipular la colección de manera dinámica.
Solo Inmutables	Los elementos dentro de un set deben ser **inmutables** (ej. números, cadenas, tuplas). No puedes almacenar listas, diccionarios u otros sets dentro de un set.	Es un error común intentar meter una lista en un set.

Métodos Esenciales del Set

Método	Uso	Ejemplo y Comportamiento
`.add(elemento)`	Añade un único elemento al set.	`mi_set.add('rojo')`. Si 'rojo' ya existe, no hace nada.
`.update(iterable)`	Añade múltiples elementos de una lista o tupla.	`mi_set.update(['verde', 'azul'])`.
`.remove(e)`	Elimina un elemento.	**Cuidado**: Si el elemento no existe, lanza un error (`KeyError`).
`.discard(e)`	Elimina un elemento.	**Seguro**: Si el elemento no existe, no hace nada (no lanza error).
`.union()` o `|`	Combina dos sets, eliminando duplicados.	`set1.union(set2)`.
`.intersection()` o `&`	Encuentra los elementos comunes entre dos sets.	`set1.intersection(set2)`.

---

3. Diccionarios (Dictionaries) 📒: La Colección Clave-Valor

Un **Diccionario** es una estructura de datos que asocia **claves** únicas con **valores**. Piensa en ello como una agenda telefónica: el nombre (clave) te lleva al número (valor).

Propiedades Clave del Diccionario

Propiedad	Descripción	Importancia para el programador
Clave-Valor	Los datos se guardan en pares `clave: valor`.	Permite acceder a la información de forma *semántica* (por nombre), no por posición.
Claves Únicas e Inmutables	Cada clave debe ser única (si pones la misma clave, se sobrescribe el valor) y debe ser un objeto **inmutable** (ej. cadenas, tuplas, números).	**Cuidado**: Nunca uses una lista o un diccionario como clave.
Valores Libres	Los valores pueden ser **cualquier tipo de dato** (incluso listas, otros diccionarios, o sets).	Permite construir estructuras de datos complejas (ej. un registro de usuario completo).
Mutable	Puedes agregar, modificar o eliminar pares clave-valor después de la creación.	La estructura es muy flexible.

Métodos Esenciales del Diccionario

Método/Sintaxis	Uso	Ejemplo y Comportamiento
`mi_dict[clave]`	**Acceso/Modificación por clave**.	`datos['edad'] = 30`. Si 'edad' no existe, la crea. Si existe, la cambia.
`.get(c, defecto)`	**Acceso Seguro**.	`datos.get('ciudad', 'Desconocida')`. Devuelve el valor o el predeterminado, **evitando el error** `KeyError` si la clave no está.
`.keys()`	Devuelve una "vista" de todas las claves.	Útil para iterar solo sobre los nombres de los datos.
`.values()`	Devuelve una "vista" de todos los valores.	Útil para operar solo sobre los datos.
`.items()`	Devuelve una "vista" de los pares `(clave,`	La forma más común de iterar sobre un diccionario.

Método/Sintaxis	Uso	Ejemplo y Comportamiento
	`valor)`.	
`.pop(clave)`	Elimina y devuelve el valor de la clave especificada.	`nombre = datos.pop('nombre')`.
`.update(otro_dict)`	Fusiona o agrega los pares clave-valor de otro diccionario.	`datos.update({'email': 'test@a.com'})`.

🔑 El uso de `.append()` en Diccionarios

Recuerda que los métodos como `.append()` **no pertenecen al diccionario en sí**, sino a los objetos que contiene. Si un valor del diccionario es una **Lista**, puedes modificar esa lista directamente:

Python

```
mi_dict = {'nombre': 'Ana', 'hobbies': ['leer', 'cantar']}

# Accedemos al valor 'hobbies' (que es una lista) y usamos su método
.append()
mi_dict['hobbies'].append('programar')

# Resultado: mi_dict ahora es {'nombre': 'Ana', 'hobbies': ['leer',
'cantar', 'programar']}
```

---

4. Concepto Crucial: Tipos de Copias (Superficial vs. Profunda)

Este punto es vital al trabajar con estructuras de datos mutables que contienen otras estructuras mutables (ej. un diccionario con una lista como valor).

Tipo de Copia	Método	Descripción y Consecuencia
Copia Superficial	**`.copy()`**	Crea un objeto nuevo en la memoria, pero solo copia las **referencias** a los objetos anidados. **Consecuencia**: Modificar un objeto mutable anidado en la copia, **también lo modifica en el original**.

Tipo de Copia	Método	Descripción y Consecuencia
Copia Profunda	`import copy; copy.deepcopy(obj)`	Crea una **copia totalmente independiente** del objeto principal y de todos sus objetos anidados, sin compartir ninguna referencia.

---

5. 🆚 Diferencia con Listas y Tuplas

Listas y Tuplas son estructuras de datos basadas en la **secuencia y el orden**, lo que las diferencia fundamentalmente de Sets y Diccionarios.

Tabla Comparativa de las 4 Estructuras

Estructura	Símbolo	Ordenada	Mutable	Elementos Únicos	Acceso	Propósito Principal
Lista	`[ ]`	✅ Sí	✅ Sí	❌ No	Por índice `[0]`	Colección general, flexible y ordenada.
Tupla	`( )`	✅ Sí	❌ No	❌ No	Por índice `[0]`	Almacenar datos fijos y proteger la integridad de los datos.
Set	`{ }`	❌ No	✅ Sí	✅ Sí	Por pertenencia (`in`)	**Filtrado de duplicados** y operaciones de conjuntos.
Diccionario	`{ clave: valor }`	❌ No	✅ Sí	✅ Sí (solo en claves)	Por clave `['clave']`	**Asociar un valor a un identificador único** (mapeo).

Resumen de Uso para Evitar Confusiones

- **Usa Sets** cuando: Quieras saber si un elemento existe en una colección o cuando necesites **eliminar duplicados** rápidamente, sin importar el orden.

- **Usa Diccionarios** cuando: La información tiene un **nombre** o etiqueta única y necesitas acceder a su valor asociado de forma instantánea (ej. buscar el ID de un usuario por su nombre).
- **Usa Listas** cuando: El **orden** de los elementos es importante y los elementos pueden cambiar (ej. un *log* de eventos o una cola de tareas).
- **Usa Tuplas** cuando: El orden es importante, pero la colección de datos **nunca debe cambiar** después de su creación (ej. coordenadas geográficas, fechas).

MAS INFO EN INGLES

🧠 Comprehensive Command & Function Reference (Sets + Dictionaries)

Category	Command / Function	Detailed Description	Example Code
Set Creation	`{}`	Creates a set directly with curly braces.	`alergenos = {'Leche', 'Huevos', 'Trigo'}`
	`set(iterable)`	Converts a list, tuple, or string into a set.	`set(['Leche', 'Huevos' 'Huevos'])`
Set Inspection	`len(set)`	Returns number of unique elements.	`len(alergenos)`
	`in` / `not in`	Checks if an element exists in the set.	`'Huevos' in alergenos`
Set Modification	`.add(element)`	Adds a single element to a set.	`alergenos.add('Mostaza`
	`.update(iterable)`	Adds multiple elements (from list,	`alergenos.update(['Brc 'Sulfitos'])`

Category	Command / Function	Detailed Description	Example Code
		tuple, or set).	
	`.remove(element)`	Removes an element. Raises `KeyError` if not found.	`alergenos.remove('Huev`
	`.discard(element)`	Removes element if it exists; does nothing otherwise.	`alergenos.discard('Api`
	`.pop()`	Removes and returns a random element.	`alergenos.pop()`
	`.clear()`	Removes all elements from the set.	`alergenos.clear()`
	`.copy()`	Returns a shallow copy of the set.	`nuevo = alergenos.copy`
Set Operations	`.union(other)` or `	`	Returns a new set with all elements from both sets.
	`.intersection(other)` or `&`	Returns common elements.	`a & b`
	`.difference(other)` or `-`	Returns elements in one set but not the other.	`a - b`

Category	Command / Function	Detailed Description	Example Code
	`.symmetric_difference(other)` or `^`	Elements in either set, but not both.	`a ^ b`

Category	Command / Function	Detailed Description	Example Code
Dictionary Creation	`{}`	Defines a dictionary with key–value pairs.	`dic = {'lunes': 'monday'}`
	`dict()`	Creates a dictionary using keyword arguments.	`dict(lunes='monday', martes='tuesday')`
	`dict([(k1,v1), (k2,v2)])`	Creates a dictionary from list of tuples.	`dict([('sábado','saturday'), ('domingo','sunday')])`
	`dict.fromkeys(keys, value)`	Creates dict with given keys and same value.	`dict.fromkeys(['a','b'], 0)`
Dictionary Inspection	`len(dict)`	Returns number of key-value pairs.	`len(diccionario)`
	`.keys()`	Returns all keys.	`diccionario.keys()`
	`.values()`	Returns all values.	`diccionario.values()`

Category	Command / Function	Detailed Description	Example Code
	`.items()`	Returns list of (key, value) tuples.	`diccionario.items()`
	`in` / `not in`	Checks if key exists.	`'lunes' in diccionario`
Dictionary Access	`dict[key]`	Returns value for given key.	`dic['lunes']`
	`.get(key, default)`	Returns value or default if missing.	`dic.get('sabado', 'no existe')`
Dictionary Modification	`dict[key] = value`	Adds or updates entry.	`dic['jueves'] = 'thursday'`
	`.update(other)`	Updates dict with another dict or pairs.	`dic.update({'viernes':'friday'`
	`.pop(key)`	Removes key and returns value.	`dic.pop('lunes')`
	`.popitem()`	Removes and returns last added pair.	`dic.popitem()`
	`del dict[key]`	Deletes key from dictionary.	`del dic['lunes']`
	`.clear()`	Empties entire dictionary.	`dic.clear()`
	`.copy()`	Returns shallow copy.	`nuevo = dic.copy()`
Advanced Use	Nested dictionaries	Stores dicts or lists	`{'nombres':['Lola','Marta'], 'notas':[8,9]}`

Category	Command / Function	Detailed Description	Example Code
		inside values.	
	Dictionary comprehension	Create dynamically from iteration.	`{x:x**2 for x in range(3)}`

Would you like me to:

1. **Add an extra column** with *examples and expected outputs side-by-side (code + print results)*,
2. Or **generate a visual PDF/Excel** version with structured formatting and color for learning use?

🧠 Comprehensive Command & Function Reference (Sets + Dictionaries)

Category	Command / Function	Detailed Description	Example Code
Set Creation	`{}`	Creates a set directly with curly braces.	`alergenos = {'Leche', 'Huevos', 'Trigo'}`
	`set(iterable)`	Converts a list, tuple, or string into a set.	`set(['Leche', 'Huevos' 'Huevos'])`
Set Inspection	`len(set)`	Returns number of unique elements.	`len(alergenos)`
	`in` / `not in`	Checks if an element exists in the set.	`'Huevos' in alergenos`

Category	Command / Function	Detailed Description	Example Code
Set Modification	`.add(element)`	Adds a single element to a set.	`alergenos.add('Mostaza`
	`.update(iterable)`	Adds multiple elements (from list, tuple, or set).	`alergenos.update(['Bro 'Sulfitos'])`
	`.remove(element)`	Removes an element. Raises `KeyError` if not found.	`alergenos.remove('Huev`
	`.discard(element)`	Removes element if it exists; does nothing otherwise.	`alergenos.discard('Api`
	`.pop()`	Removes and returns a random element.	`alergenos.pop()`
	`.clear()`	Removes all elements from the set.	`alergenos.clear()`
	`.copy()`	Returns a shallow copy of the set.	`nuevo = alergenos.copy`
Set Operations	`.union(other)` or `	`	Returns a new set with all elements from both sets.

Category	Command / Function	Detailed Description	Example Code
	`.intersection(other)` or `&`	Returns common elements.	`a & b`
	`.difference(other)` or `-`	Returns elements in one set but not the other.	`a - b`
	`.symmetric_difference(other)` or `^`	Elements in either set, but not both.	`a ^ b`

Category	Command / Function	Detailed Description	Example Code
Dictionary Creation	`{}`	Defines a dictionary with key–value pairs.	`dic = {'lunes': 'monday'}`
	`dict()`	Creates a dictionary using keyword arguments.	`dict(lunes='monday', martes='tuesday')`
	`dict([(k1,v1), (k2,v2)])`	Creates a dictionary from list of tuples.	`dict([('sábado','saturday'), ('domingo','sunday')])`
	`dict.fromkeys(keys, value)`	Creates dict with given keys and same value.	`dict.fromkeys(['a','b'], 0)`
Dictionary Inspection	`len(dict)`	Returns number of	`len(diccionario)`

Category	Command / Function	Detailed Description	Example Code
		key-value pairs.	
	`.keys()`	Returns all keys.	`diccionario.keys()`
	`.values()`	Returns all values.	`diccionario.values()`
	`.items()`	Returns list of (key, value) tuples.	`diccionario.items()`
	`in` / `not in`	Checks if key exists.	`'lunes' in diccionario`
Dictionary Access	`dict[key]`	Returns value for given key.	`dic['lunes']`
	`.get(key, default)`	Returns value or default if missing.	`dic.get('sabado', 'no existe')`
Dictionary Modification	`dict[key] = value`	Adds or updates entry.	`dic['jueves'] = 'thursday'`
	`.update(other)`	Updates dict with another dict or pairs.	`dic.update({'viernes':'friday'`
	`.pop(key)`	Removes key and returns value.	`dic.pop('lunes')`
	`.popitem()`	Removes and returns last added pair.	`dic.popitem()`
	`del dict[key]`	Deletes key from dictionary.	`del dic['lunes']`

Category	Command / Function	Detailed Description	Example Code
	`.clear()`	Empties entire dictionary.	`dic.clear()`
	`.copy()`	Returns shallow copy.	`nuevo = dic.copy()`
Advanced Use	Nested dictionaries	Stores dicts or lists inside values.	`{'nombres':['Lola','Marta'], 'notas':[8,9]}`
	Dictionary comprehension	Create dynamically from iteration.	`{x:x**2 for x in range(3)}`