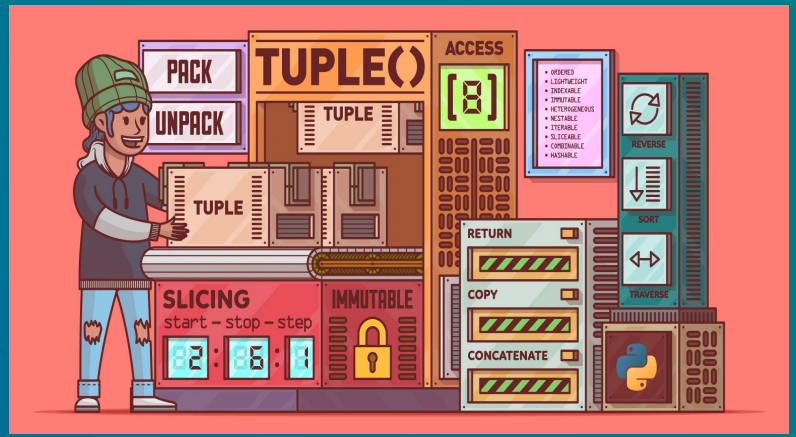
Germán Scarafilo - Lucas Ferrini - Alex Roberts

Tipos de datos avanzados: Tuplas, sets y diccionarios

Tuplas: tuple

Las Tuplas son muy similares a las listas, son una colección de elementos ordenados e inmutables, lo que significa que no pueden ser modificadas una vez declaradas.
Las Tuplas son útiles para almacenar datos que no deben cambiar a lo largo del ciclo de vida del programa.



Tuplas: tuple

Características de las Tuplas:

- ***Inmutabilidad***: Una vez creada, no se pueden modificar sus elementos.
- ***Ordenada***: Los elementos de una tupla están ordenados y mantienen el orden en el que se añadieron.
- ***Pueden contener cualquier tipo de dato***: Una tupla puede contener elementos de diferentes tipos de datos.

Tuplas: tuple

¿Que ocurre en el siguiente caso?

```
lista = tuple([1, 'Hola', 3.67])
print(type(lista))   # <class 'tuple'>
print(lista[1])      # Hola
lista[1] = 'Chau'    # TypeError: 'tuple'
                     # object does not support item assignment
```

Tuplas: tuple

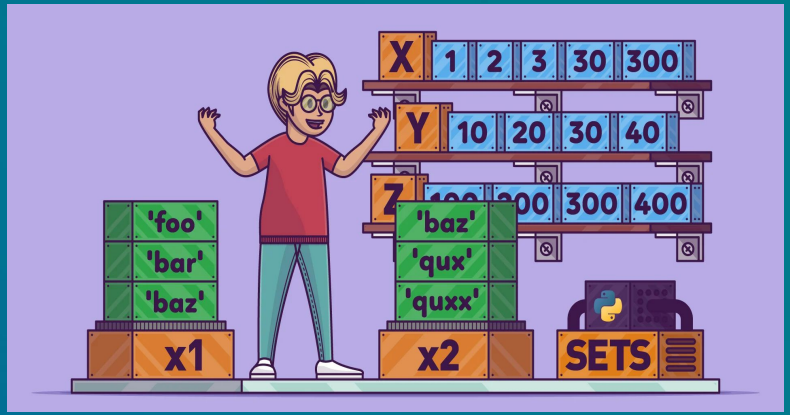
Desempaquetado de Tuplas:

- En Python, podemos asignar los elementos de una tupla a variables individuales en un solo paso (a esto se llama “desempaquetado de tuplas”).

```
mi_tupla = ("Juan", "Perez", 30)
nombre, apellido, edad = mi_tupla
print(nombre)     # Resultado: Juan
print(apellido)   # Resultado: Perez
print(edad)       # Resultado: 30
```

Conjuntos: set

Los elementos de un set son únicos, no contiene elementos duplicados.
Los set no respetan el orden que tenían al ser declarados.
Sus elementos son mutables.



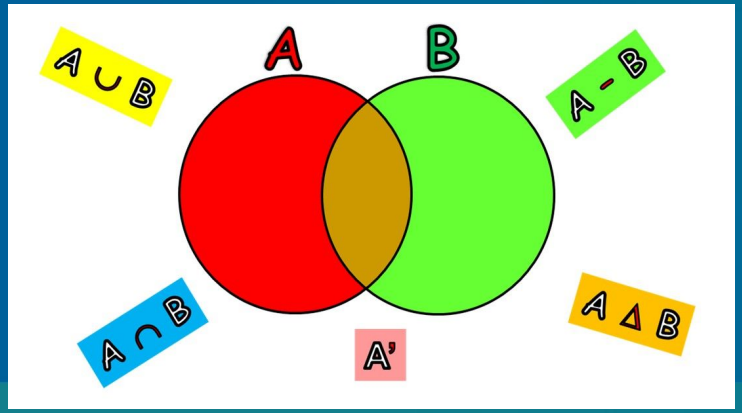
Conjuntos: set

¿Que ocurre en el siguiente caso?

```
set = { 3, 5, 9, 5, 3, 4, 3 }
print(type(set))    # <class set>
print(set)          # 9, 3, 4, 5
```

Conjuntos: set

Un set también se puede crear haciendo uso de la palabra reservada **set** la cual permite transformar cualquier objeto iterable en un **set**



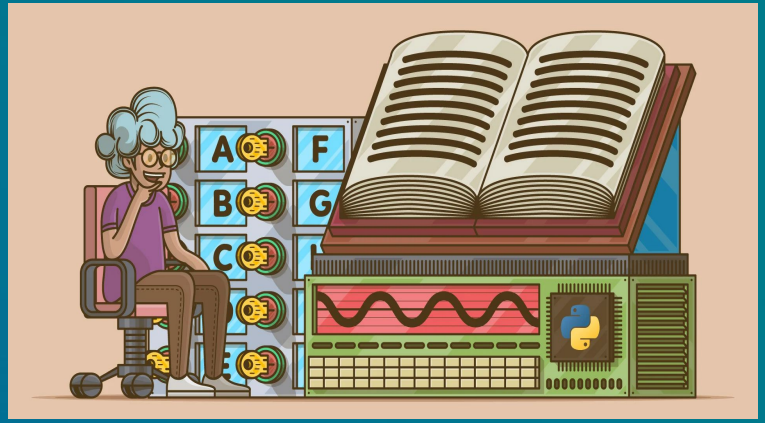

```
set_uno = { 1, 2, 3 }
set_dos = { 3, 4, 5 }
union_set = set_uno.union(set_dos)
print(union_set)      # Salida: {1, 2, 3, 4, 5}
```

Métodos más Importantes

add()	Agrega un elemento al set.
remove()	Elimina un elemento específico del set. Da error si el elemento no existe.
discard()	Elimina un elemento sin dar error si el elemento no está presente.
pop()	Elimina y devuelve un elemento aleatorio del set.
clear()	Vacía el set, eliminando todos sus elementos.
union()	Devuelve un nuevo set con los elementos de dos sets.
intersection()	Devuelve un nuevo set con los elementos que están en ambos sets.
update()	Añade varios elementos de otro set o lista al set original

Diccionarios: dict

Un diccionario es una colección de elementos compuesto por una clave (key), que es única, y un valor (value). Contenidos entre { } llaves.
La ventaja en el uso de diccionario es que se pueden acceder a sus valores utilizando su clave en lugar de su posición en la estructura, lo cual facilita enormemente la búsqueda y manipulación de los datos.



Diccionarios: dict

Existen dos forma de crear un diccionario.
Una de las formas es utilizando -> **dict( ):**

```
diccionario = dict([
    ('Nombre', 'Sara'),
    ('Edad', 27),
    ('Documento', 1003883),
])

print(diccionario)
#           key      value    key  value    key       value
# Output: {'Nombre': 'Sara', 'Edad': 27, 'Documento': 1003883}
```

Diccionarios: dict

La segunda forma es:

```
diccionario = {
        'nombre': 'Juan',
        'edad': 21,
        'ciudad': 'Buenos Aires'
}
print(type(diccionario))        # <class 'dict'>
print(diccionario['nombre'])    # Juan
print(diccionario['edad'])      # 21
print(diccionario['ciudad'])    # Buenos Aires
```

Diccionarios: dict

Puedes usar estos métodos para iterar diccionario en Python:

- keys():
- values():
- items():

Diccionarios: dict

keys() :

- Utiliza el método keys() cuando solo necesitas las claves del diccionario.
- Devuelve una vista iterable de las claves.
- Útil para verificar si una clave específica existe en el diccionario.

```
diccionario = {'nombre': 'Juan', 'edad': 21}
print(diccionario.keys())
# dict_keys(['nombre', 'edad'])
```

Diccionarios: dict

values() :

- Emplea el método values() cuando solo te interesan los valores del diccionario.
- Devuelve una vista iterable de los valores.
- Útil cuando no necesitas las claves, solo los datos asociados.

```
diccionario = {'nombre': 'Juan', 'edad': 21}
print(diccionario.values())
# dict_values(['Juan', 21])
```

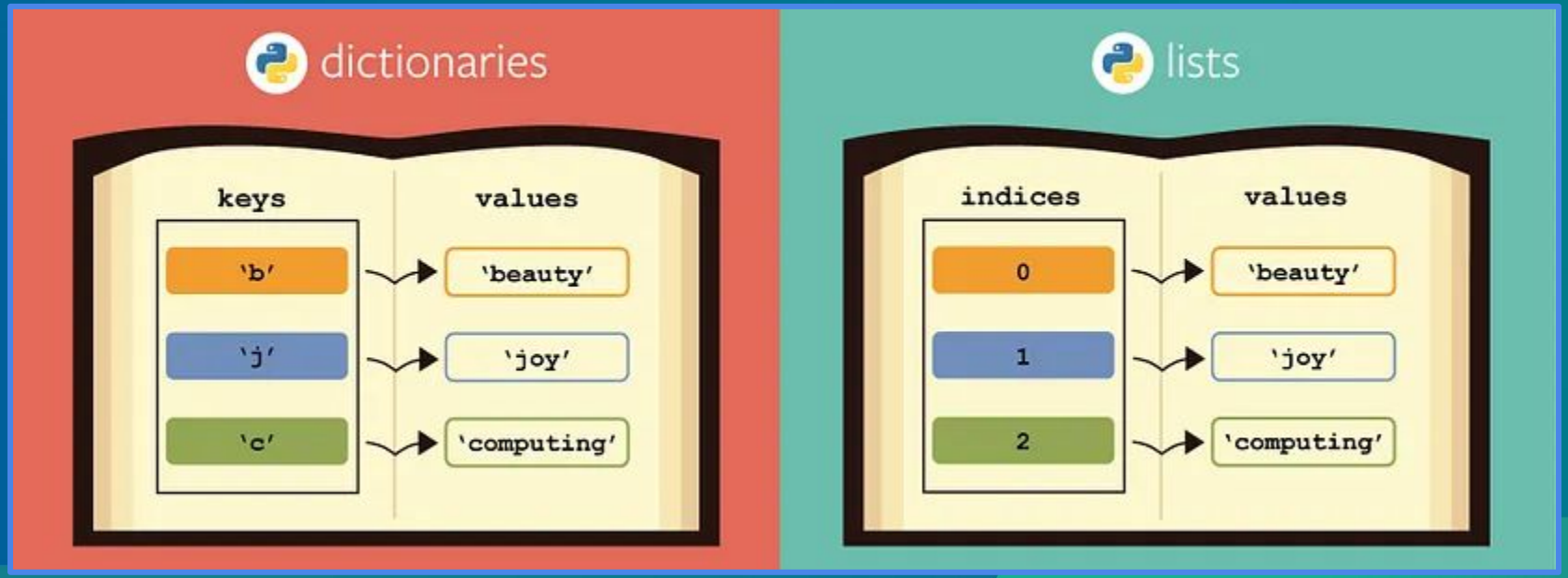
Diccionarios: dict

items():

- Usa items() cuando necesitas tanto las claves como los valores.
- Devuelve una vista iterable de pares clave-valor.
- Ideal para recorrer todo el diccionario y trabajar con ambos componentes.

```
diccionario = {'nombre': 'Juan', 'edad': 21}
print(diccionario.items())
# dict_items([('nombre', 'Juan'), ('edad', 21)])
```

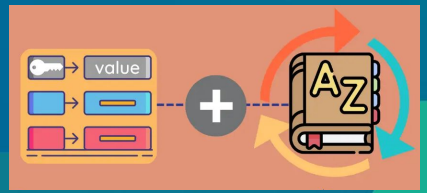
Diccionarios vs Listas



Diccionarios vs Objetos

	Ventajas	Uso ideal
Diccionarios	Simplicidad, flexibilidad, facilidad para manejar datos dinámicos y configurar parámetros.	Configuraciones, datos que cambian frecuentemente, serialización y deserealización.
Objetos	Modelado de entidades complejas. Pilares de la POO.	Entidades con comportamiento definido, sistemas que requieren organización y reutilización de código.

```
config = {
    "host": "localhost",
    "port": 8080,
    "debug": True
}
```



{.json}