

# ¿Qué son los diccionarios y como se definen?

En Python, un diccionario es una estructura de datos que almacena elementos en pares clave-valor (*key-value*), donde cada clave debe ser única. Es similar a una lista o una tupla, pero en lugar de acceder a los elementos por su posición, se accede a ellos a través de su clave. En un diccionario, los valores pueden ser cualquier tipo de objeto, como números, cadenas, listas, tuplas e incluso otros diccionarios. Las claves deben ser objetos inmutables como cadenas, números o tuplas. Son muy útiles porque permiten buscar, agregar y modificar valores rápidamente mediante el uso de claves en lugar de índices.

Para definir un diccionario usaremos las llaves `{}`. IMPORTANTE un valor **nunca** puede existir sin una clave que lo identifique.

Características de los diccionarios:

- No están ordenados: Los elementos en un diccionario no están en ningún orden específico. No se pueden acceder a los elementos de un diccionario utilizando índices, sino que se acceden a ellos a través de las claves.
- Claves únicas: Las claves en un diccionario deben ser únicas. Si se intenta agregar una clave existente a un diccionario, se sobrescribirá el valor anterior con el nuevo valor.
- Mutable: Los diccionarios son mutables, lo que significa que se pueden agregar, modificar y eliminar elementos después de crearlos.
- Flexibles: Los diccionarios pueden contener cualquier tipo de dato como valor, incluso otros diccionarios. Sin embargo, las claves deben ser inmutables, por lo que las cadenas, números y tuplas son buenas opciones de clave.
- Eficientes: Los diccionarios son muy eficientes para buscar y recuperar valores basados en una clave. Debido a su implementación interna basada en tablas hash, los diccionarios son mucho más rápidos para acceder a valores que las listas o las tuplas.

Ya hemos comentado que los diccionarios se definen usando `{}`, pero podemos definirlos de otra forma, estas son:

- Usando llaves `{}` y pares clave-valor separados por coma.
- Usando el constructor `dict()` y pares clave-valor separados por coma.
- Usando una lista de tuplas clave-valor.
- Usando el método `fromkeys()` para crear un diccionario con claves y valores predeterminados.

Veamos todas estas opciones con ejemplos.

### Usando llaves {} y pares clave-valor separados por coma.

```
In [54]: # definimos un diccionario donde las keys sean strings y los values también
diccionario1 = { 'lunes': 'monday', 'martes': 'tuesday', 'miércoles': 'wednesday' }

# también podemos tener diccionarios donde las keys sean strings y los values números
diccionario2 = { 'lunes': 1, 'martes': 2, 'miércoles': 3 }

# o un diccionario donde sean números y los values strings
diccionario3 = { 1: 'monday', 2: 'tuesday', 3: 'wednesday' }

# o incluso diccionarios que tengan listas en sus values
diccionario4 = { 'lunes': [1, "peras"], 'martes': [2, "manzanas"], 'miércoles': [3, "brocoli"] }

print("Este es el primer diccionario:", diccionario1)
print("-----")
print("Este es el segundo diccionario:", diccionario2)
print("-----")
print("Este es el tercer diccionario:", diccionario3)
print("-----")
print("Este es el cuarto diccionario:", diccionario4)

Este es el primer diccionario: {'lunes': 'monday', 'martes': 'tuesday', 'miércoles': 'wednesday'}
-----
Este es el segundo diccionario: {'lunes': 1, 'martes': 2, 'miércoles': 3}
-----
Este es el tercer diccionario: {1: 'monday', 2: 'tuesday', 3: 'wednesday'}
-----
Este es el cuarto diccionario: {'lunes': [1, 'peras'], 'martes': [2, 'manzanas'], 'miércoles': [3, 'brocoli']}
```

### Usando el constructor dict() y pares clave-valor separados por coma. y Usando una lista de tuplas clave-valor

```
In [55]: # definimos el diccionario con "dict" usando el operador "=" y cada par de clave
diccionario5 = dict( jueves = 'thursday', viernes='friday' )

# definimos el diccionario con "dict" usando una lista de tuplas, separando cada par
diccionario6 = dict( [ ('sábado', 'saturday'), ('domingo', 'sunday') ] )

print("Este es el quinto diccionario:", diccionario5)
print("-----")
print("Este es el sexto diccionario:", diccionario6)
print("-----")

Este es el quinto diccionario: {'jueves': 'thursday', 'viernes': 'friday'}
-----
Este es el sexto diccionario: {'sábado': 'saturday', 'domingo': 'sunday'}
```

### Usando el método fromkeys() para crear un diccionario con claves y valores predeterminados.

```
In [56]: # definimos una lista con lo que serán nuestras futuras keys
claves = ['jueves', 'viernes', 'sábado']
```

```
# establecemos el valor predeterminado de lo que serán nuestros futuros values. Es
valor_predeterminado = []

# definimos el diccionario usando
diccionario7 = dict.fromkeys(claves, valor_predeterminado)

print("Este es el séptimo diccionario:", diccionario7)
print("-----")
```

```
Este es el séptimo diccionario: {'jueves': [], 'viernes': [], 'sabado': []}
-----
```

## Propiedades de los diccionarios

Al igual que en las listas y en las tuplas, en los diccionarios vamos a tener una serie de métodos que nos ayuden a conocer las principales propiedades de los diccionarios, las más importantes son:

- `len()` : al igual que hasta ahora, este método se utiliza para obtener la cantidad de elementos que hay en un diccionario.
- `keys()` : este método devuelve una lista con todas las claves del diccionario.
- `values()` : este método devuelve una lista con todos los valores del diccionario.
- `items()` : este método devuelve una lista de tuplas, donde cada tupla contiene una clave y su valor correspondiente.

Veámoslo en más detalle:

```
In [57]: # Lo primero que vamos a definir es un diccionario. En este caso vamos a crear uno
# Los values serán listas donde almacenaremos los nombres de las alumnas y sus notas
diccionario_alumnas = {"nombres": ["Lola", "Marta", "Lorena"], "notas": [8, 9, 6]}
print("El diccionario que hemos creado es: ", diccionario_alumnas)

# vamos a usar el método len para saber cuantos pares de clave-valor tenemos en el diccionario
print("La cantidad de pares de clave-valor del diccionario es: ", len(diccionario))

# vamos a sacar todas las claves del diccionario
print("Las keys del diccionario son: ", diccionario_alumnas.keys())

# vamos a sacar todas los valores del diccionario
print("Las values del diccionario son: ", diccionario_alumnas.values())

# vamos a sacar todos los pares de claves-valores que tenemos en el diccionario
print("Los pares de clave-valor del diccionario son: (ESTO NOS DEVUELVE UNA LISTA)")
```

El diccionario que hemos creado es: { 'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6]}

La cantidad de pares de clave-valor del diccionario es: 2

Las keys del diccionario son: dict\_keys(['nombres', 'notas'])

Las values del diccionario son: dict\_values([('Lola', 'Marta', 'Lorena'), [8, 9, 6]])

Los pares de clave-valor del diccionario son: (ESTO NOS DEVUELVE UNA LISTA DE TUPLAS) dict\_items([('nombres', ['Lola', 'Marta', 'Lorena']), ('notas', [8, 9, 6])])

Al igual que en las listas y en las tuplas podemos usar el operador `in` y `not in` en diccionarios. Veamos como lo haríamos:

```
In [58]: print("¿Esta la clave 'nombres' en mi diccionario?", "nombres" in diccionario_alumno)
# si nos fijamos , por defecto nos lo hace sobre las claves

print("¿Esta la clave 'patata' en mi diccionario?", "patata" in diccionario_alumno)

# pero ¿qué pasaría si quisieramos hacerlo sobre los values? en este caso tendríamos
print("¿Esta 'Lola' en mi los valores de mi diccionario?", "Lola" in diccionario_alumno)
# Vaya... 😱, esto nos devuelve False, y es que lo que estamos evaluando es si

print("¿Esta 'Lola' en mi los valores de mi diccionario?", ['Lola', 'Marta', 'Lorena'].count('Lola'))
```

¿Esta la clave 'nombres' en mi diccionario? True  
¿Esta la clave 'patata' en mi diccionario? False  
¿Esta 'Lola' en mi los valores de mi diccionario? False  
¿Esta 'Lola' en mi los valores de mi diccionario? True

# Métodos de los diccionarios

Al igual que el resto de los tipos de datos que hemos visto en Python, los diccionarios tienen una serie de métodos que nos van a resultar muy útiles para poder modificarlos. Aquí hay algunos de los métodos más comunes que se pueden usar con diccionarios:

- `copy()` : Devuelve una copia superficial del diccionario.
- `clear()` : Elimina todos los elementos del diccionario.
- `update()` : Actualiza el diccionario con pares clave-valor de otro diccionario o de una lista de tuplas (clave, valor).
- `get()` : Devuelve el valor de la clave dada si está presente en el diccionario, de lo contrario devuelve el valor predeterminado (opcional).
- `setdefault()` : Si la clave está en el diccionario, devuelve su valor. Si no, inserta la clave con el valor predeterminado (opcional) y devuelve ese valor.
- `pop()` : Elimina la clave y devuelve su valor. Si la clave no está presente y se proporciona un valor predeterminado, se devuelve ese valor. Si la clave no está presente y no se proporciona un valor predeterminado, se produce una excepción `KeyError`.
- `popitem()` : Elimina y devuelve un par clave-valor arbitrario del diccionario. Si el diccionario está vacío, se produce una excepción `KeyError`.

## copy()

El método `copy()` es un método incorporado en Python que se usa para crear una copia superficial de un diccionario. Esto significa que el método crea un nuevo diccionario que contiene los mismos pares clave-valor que el diccionario original, pero que no está vinculado al diccionario original en ningún otro sentido.

Veámoslo con un ejemplo:

```
In [59]: # primero recordemos el último diccionario de alumnas que habíamos creado previa
print("El diccionario de las alumnas de clase es: ", diccionario_alumnas)

# haciendo uso del método copy, nos haremos una copia exacta del diccionario ant
diccionario_alumnas_copia = diccionario_alumnas.copy()
print("La copia del diccionario de las alumnas de clase es: ", diccionario_alumn
```

El diccionario de las alumnas de clase es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6]}  
La copia del diccionario de las alumnas de clase es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6]}

## clear()

El método `clear()` es un método incorporado en Python que se usa para eliminar todos los elementos de un diccionario. En otras palabras, el método `clear()` borra todo el contenido de un diccionario, lo que hace que el diccionario quede vacío.

```
In [60]: # mostramos La copia del diccionario que nos hicimos en la celda anterior
print("La copia del diccionario de las alumnas de clase es: ", diccionario_alumn

# vaciamos todo el contenido del diccionario usando el método clear()
diccionario_alumnas_copia.clear()

print("El diccionario de las alumnas de clase después de haberlo vaciado usando

# muy relacionado con el método anterior y para entender mejor el concepto de La
print("El diccionario de las alumnas de clase es: ", diccionario_alumnas)
```

```
La copia del diccionario de las alumnas de clase es: {'nombres': ['Lola', 'Marta
', 'Lorena'], 'notas': [8, 9, 6]}
El diccionario de las alumnas de clase después de haberlo vaciado usando el métod
o 'clear()' es: {}
El diccionario de las alumnas de clase es: {'nombres': ['Lola', 'Marta', 'Loren
a'], 'notas': [8, 9, 6]}
```

## update()

El método `update()` es un método incorporado en Python que se utiliza para actualizar los valores de un diccionario con los valores de otro diccionario o de un iterable de pares clave-valor. Este método modifica el diccionario original y puede ser muy útil cuando necesitas actualizar un diccionario con nuevos valores.

Imaginemos que ahora queremos añadir un nuevo par de clave-valor en el diccionario, en concreto, queremos añadir una clave que nos diga si la alumna tiene hermanos o no. En esta situación podríamos simplemente volver a crear un diccionario de nuevo o usar el método `update()`. Veamos como hacerlo:

```
In [61]: print("El diccionario original de las alumnas de clase es: ", diccionario_alumna

# añadimos un nuevo par de clave-valor a nuestro diccionario.
# para eso lo primero que tenemos que crear un diccionario, con el nuevo par de

nuevo_elemento = {"hermanos": ["Si", "No", "Si", "Si", "Si"]}

# utilizamos el método update para añadir la nueva pareja de key-value. FIJAOS Q
diccionario_alumnas.update(nuevo_elemento)
print("El diccionario de las alumnas de clase después de añadir la nueva pareja
```

```
El diccionario original de las alumnas de clase es: {'nombres': ['Lola', 'Marta'
, 'Lorena'], 'notas': [8, 9, 6]}
El diccionario de las alumnas de clase después de añadir la nueva pareja de key-v
alue es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6], 'hermanos'
: ['Si', 'No', 'Si', 'Si', 'Si']}
```

## get()

El método `get()` es un método incorporado en Python que se utiliza para obtener el valor de una clave en un diccionario, si lo comparamos con lo que conocemos hasta ahora sería como la indexación de las listas o las tuplas.

```
In [62]: # este método nos sirve para acceder a los valores de una key específica.
# Lo primero que vamos a hacer es recordar el diccionario de las alumnas
print("El diccionario original de las alumnas de clase es: ", diccionario_alumnas)

# imaginemos ahora que queremos extraer solo los valores de los nombres de la clase
print("Los valores de la clave 'nombres' es: ", diccionario_alumnas.get("nombres"))

# otra forma de acceder a los valores de una clave es usando la sintaxis ([]), vemos
print("Los valores de la clave 'nombres' es: ", diccionario_alumnas["nombres"])
```

El diccionario original de las alumnas de clase es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si']}  
 Los valores de la clave 'nombres' es: ['Lola', 'Marta', 'Lorena']  
 Los valores de la clave 'nombres' es: ['Lola', 'Marta', 'Lorena']

👉 FIJAOS COMO EL OUTPUT DE LO QUE NOS DEVUELVE ES EXACTAMENTE EL MISMO!  
 Pero entonces... ¿qué diferencia existe entre una forma y otra para acceder?

A diferencia de la notación de corchetes ([]), que devuelve un error si la clave no existe en el diccionario, el método `get()` devuelve `None` si la clave no está presente en el diccionario o un valor predeterminado que se puede especificar como argumento.

```
In [63]: # en este caso vamos a buscar la clave 'edad', la cual NO existe en nuestro diccionario
print("Los valores de la clave 'edad' es: ", diccionario_alumnas.get("edad"))

# otra forma de acceder a los valores de una clave es usando la sintaxis ([]), vemos
print("Los valores de la clave 'edad' es: ", diccionario_alumnas["edad"])
```

Los valores de la clave 'edad' es: None  
 -----
   
**KeyError** Traceback (most recent call last)  
`/tmp/ipykernel_322/1325725148.py` in <module>  
 3  
 4 # otra forma de acceder a los valores de una clave es usando la sintaxis ([]), veamos un ejemplo:  
 ----> 5 print("Los valores de la clave 'edad' es: ", diccionario\_alumnas["edad"])

**KeyError: 'edad'**

Como decíamos, el método `.get()` nos devuelve un `None` en caso de que estemos buscando una clave que no está en el diccionario. Pero imaginaos que en vez de querer que nos devuelva `None`, nos devuelva otro mensaje, por ejemplo, "la clave que estas buscando no esta en el diccionario". Para esto, le podemos añadir un parámetro al método `'get()'` en el que pogamos este mensaje o valor predeterminado

```
In [64]: # buscamos la clave 'edad', y le vamos a añadir un valor predeterminado con el método get()
# si nos fijamos en el output, en este caso ya no aparece None si no que aparece el valor que le dimos
print("Los valores de la clave 'edad' es: ", diccionario_alumnas.get("edad", 'la clave no existe'))
```

```
# sin embargo, esta segunda forma de acceder a una clave no tiene la opción de p
print("Los valores de la clave 'edad' es: ", diccionario_alumnas["edad"])
```

Los valores de la clave 'edad' es: la clave que estas buscando no esta en el diccionario

```
-----
KeyError                                                 Traceback (most recent call last)
/tmp/ipykernel_322/1124467818.py in <module>
    4
    5 # sin embargo, esta segunda forma de acceder a una clave no tiene la opción de pasar este parámetro, por lo tanto, nunca podremos evitar este error.
--> 6 print("Los valores de la clave 'edad' es: ", diccionario_alumnas["edad"])

KeyError: 'edad'
```

In [65]: # hagamos otros ejemplos con el método get. Saquemos los valores de la clave hermanos  
print("Los valores de la clave 'hermanos' es: ", diccionario\_alumnas.get("hermanos"))  
# que es lo mismo que:  
# hagamos otros ejemplos con el método get. Saquemos los valores de la clave hermanos  
print("Los valores de la clave 'hermanos' es: ", diccionario\_alumnas["hermanos"])

Los valores de la clave 'hermanos' es: ['Si', 'No', 'Si', 'Si', 'Si']  
Los valores de la clave 'hermanos' es: ['Si', 'No', 'Si', 'Si', 'Si']

En resumen, el método `get()` es una manera fácil y segura de obtener el valor de una clave en un diccionario, ya que no generará un error si la clave no está presente en el diccionario. Además, el valor predeterminado que se puede especificar como argumento es útil si desea devolver un valor personalizado en lugar de `None` si la clave no existe.

## setdefault()

El método `setdefault()` es un método incorporado en Python que se utiliza para obtener el valor de una clave en un diccionario y, si la clave no existe, se inserta una nueva clave con un valor predeterminado.

In [66]: # antes hemos visto como al usar el método 'get()' si buscábamos una clave que no existía, nos devolvía None. # veamos que como funciona el método setdefault  
# busquemos primero la clave 'edad', que ya sabemos que no existe en el diccionario # recordemos que, a este método si no le pasabamos el método predeterminado nos devolvería None  
print("Los valores de la clave 'edad' es: ", diccionario\_alumnas.get("edad"))  
# hagamos exactamente lo mismo pero usando el método setdefault  
print("Los valores de la clave 'edad' es: ", diccionario\_alumnas.setdefault("edad", "No"))

Los valores de la clave 'edad' es: None  
Los valores de la clave 'edad' es: None

Como no le hemos añadido ningún parámetro al método `setdefault`, por defecto nos lo pone en `None`. Pero la parte más importante es que en este caso, este método nos devuelve el diccionario con un par de clave valor-nuevo 

In [67]: # profundicemos más en el método setdefault y veamos que es lo que ha pasado # para eso printeemos el diccionario

```
print("El diccionario de las alumnas es", diccionario_alumnas)
```

El diccionario de las alumnas es {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'edad': None}

In [68]: # pogamos ahora otro ejemplo, imaginemos que queremos buscar la clave 'curso' que  
# a diferencia que en el ejemplo anterior, ahora vamos a pasar un parámetro extra  
# hagamos exactamente lo mismo pero usando el método setdefault  
print("Los valores de la clave 'curso' es: ", diccionario\_alumnas.setdefault("cu  
  
# chequeemos ahora el diccionario  
print("El contenido del diccionario es:", diccionario\_alumnas)

Los valores de la clave 'curso' es: no existe

El contenido del diccionario es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'edad': None, 'curso': 'no existe'}

Ahora vemos que se nos ha creado un par de clave-valor nuevo, pero en este caso el valor nuevo no es None, si no el valor del parámetro que hemos puesto, en este caso "no existe". De la misma forma que le hemos puesto el string "no existe", podríamos pasar una lista o cualquier tipo de tipo de dato. En este caso, buscaremos la clave "apellidos"

In [69]: print("Los valores de la clave 'apellidos' es: ", diccionario\_alumnas.setdefault  
  
# chequeemos ahora el diccionario  
print("El contenido del diccionario es:", diccionario\_alumnas)

Los valores de la clave 'apellidos' es: ['no existe']

El contenido del diccionario es: {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas': [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'edad': None, 'curso': 'no existe', 'apellidos': ['no existe']}

En resumen, tanto el método `get()` como el método `setdefault()` se utilizan para obtener valores de un diccionario en Python. Sin embargo, hay algunas diferencias clave entre estos dos métodos. La diferencia principal es lo que sucede cuando la clave no existe en el diccionario. Con el método `get()`, se devuelve el valor predeterminado que se ha pasado como segundo parámetro. Con el método `setdefault()`, se inserta una nueva clave con el valor predeterminado y se devuelve ese valor.

## sorted()

El método `sorted()` se utiliza en Python para ordenar sus elementos, como una lista. En el caso de los diccionarios, el método `sorted()` devuelve una lista de tuplas que representan cada par clave-valor ordenado por la clave.

In [70]: # por defecto nos va a ordenar las keys del diccionario 🍀  
print("Las claves del diccionario ordenado son:", sorted(diccionario\_alumnas))  
  
# ¿comó lo podríamos hacer si quisieramos ordenar por los valores? Tendríamos que  
print("Los values del diccionario ordenado son:", sorted(diccionario\_alumnas.val

Las claves del diccionario ordenado son: ['apellidos', 'curso', 'edad', 'hermanos', 'nombres', 'notas']

```

-----
TypeError                                         Traceback (most recent call last)
/tmp/ipykernel_322/1028249886.py in <module>
      3
      4 # ¿comó lo podríamos hacer si quisieramos ordenar por los valores? Tendrí
    amos que usar el método '.values()' que aprendimos en esta lección
----> 5 print("Los values del diccionario ordenado son:", sorted(diccionario_alum
nas.values()))

```

TypeError: '<' not supported between instances of 'int' and 'str'

Vaya... 😬 Nos ha devuelto un error, y es que este error es similar al que teníamos en listas cuando intentábamos ordenar una lista que tenía números y palabras. Python no puede ordenarlo, igual que nosotras no lo podríamos hacer. Hagamos este mismo ejemplo, pero con un diccionario nuevo donde todos los valores son *strings*

```

In [83]: # definimos el diccionario
diccionario_mascotas = {'nombre': 'Poes',
                        'cumpleaños': '01/07',
                        'tipo': 'européo común',
                        'vacunas': 'no tiene'}

# mostramos el contenido del diccionario
print("El diccionario de las mascotas es:", diccionario_mascotas)

# ordenamos los values por orden alfabético de la A a la Z
print("Los valores del diccionario ordenados son:", sorted(diccionario_mascotas.

# y si quisieramos ordenar de la Z-A, usaremos el método reverse = True
print("Los valores del diccionario ordenados son:", sorted(diccionario_mascotas.

```

El diccionario de las mascotas es: {'nombre': 'Poes', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': 'no tiene'}  
 Los valores del diccionario ordenados son: ['01/07', 'Poes', 'européo común', 'no tiene']  
 Los valores del diccionario ordenados son: ['no tiene', 'européo común', 'Poes', '01/07']

## pop()

En Python, el método `pop()` se puede utilizar para eliminar y devolver el valor de una clave especificada en un diccionario. El método `pop()` acepta un argumento, que es la clave del elemento que se desea eliminar. Si la clave existe en el diccionario, el método `pop()` eliminará el elemento y devolverá su valor. Si la clave no existe en el diccionario, el método `pop()` generará un error.

```

In [72]: # mostramos el contenido del diccionario
print("El diccionario de las alumnas es", diccionario_alumnas)

# imaginemos ahora que queremos eliminar la clave de "edad" que tenía un valor d
# este método, .pop() nos devuelve el valor de la clave que estamos eliminando
print("Eliminamos la clave de 'edad' del diccionario:", diccionario_alumnas.pop()

# veamos como queda el diccionario después de eliminar la clave:
print("El diccionario de las alumnas es", diccionario_alumnas)

```

```
El diccionario de las alumnas es {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas' : [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'edad': None, 'curso': 'no existe', 'apellidos': ['no existe']}
Eliminamos la clave de 'edad' del diccionario: None
El diccionario de las alumnas es {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas' : [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'curso': 'no existe', 'apellidos': ['no existe']}
```

## popitem()

El método `popitem()` se puede utilizar para eliminar y devolver **el último par clave-valor agregado al diccionario**. El método `popitem()` no toma ningún argumento y, por lo general, se utiliza cuando se desea eliminar un elemento arbitrario del diccionario.

```
In [73]: # mostramos el contenido del diccionario
print("El diccionario de las alumnas es", diccionario_alumnas)

#imaginemos que quisieramos eliminar el último par de clave-valor del diccionario
print("Eliminamos EL ÚLTIMO PAR DE CLAVE-VALOR del diccionario:", diccionario_a

# veamos como queda el diccionario después de eliminar la clave:
print("El diccionario de las alumnas es", diccionario_alumnas)
```

```
El diccionario de las alumnas es {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas' : [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'curso': 'no existe', 'apellidos': ['no existe']}
Eliminamos EL ÚLTIMO PAR DE CLAVE-VALOR del diccionario: ('apellidos', ['no existe'])
El diccionario de las alumnas es {'nombres': ['Lola', 'Marta', 'Lorena'], 'notas' : [8, 9, 6], 'hermanos': ['Si', 'No', 'Si', 'Si', 'Si'], 'curso': 'no existe'}
```

Por lo tanto, la principal diferencia entre los métodos `pop()` y `popitem()` en los diccionarios de Python es la forma en que se eliminan los elementos del diccionario.

El método `pop()` se utiliza para eliminar y devolver el valor asociado con una clave especificada en el diccionario. El método `pop()` toma un argumento, que es la clave del elemento que se desea eliminar. Si la clave existe en el diccionario, el método `pop()` eliminará el elemento y devolverá su valor. Si la clave no existe en el diccionario, el método `pop()` generará un error. Es importante destacar que el método `pop()` permite eliminar elementos de forma selectiva, eligiendo la clave que se desea eliminar.

Por otro lado, el método `popitem()` se utiliza para eliminar y devolver el último par clave-valor agregado al diccionario. El método `popitem()` no toma ningún argumento y, por lo general, se utiliza cuando se desea eliminar un elemento arbitrario del diccionario. Es importante destacar que el método `popitem()` no permite eliminar elementos de forma selectiva, y solo se utiliza para eliminar el último elemento agregado al diccionario.

En resumen, la diferencia principal entre `pop()` y `popitem()` en los diccionarios de Python es que `pop()` se utiliza para eliminar un elemento específico según su clave, mientras que `popitem()` se utiliza para eliminar el último elemento agregado al diccionario, sin importar su clave.

# Algunas notas sobre los diccionarios

Hasta ahora hemos visto un montón de métodos y propiedades de los diccionarios. Los diccionarios tienen una serie de peculiaridades, especialmente relacionadas con la modificación de los valores. Veamoslo en detalle para poder entenderlo mejor:

```
In [88]: # nos hacemos una copia del diccionario original

diccionario_mascotas_copia = diccionario_mascotas.copy()
# trabajaremos con el diccionario de las mascotas que habíamos creado previamente
print("El diccionario de la mascota es:", diccionario_mascotas_copia)

# imaginemos que nos hemos equivocado con en el nombre de nuestra mascota y queremos
# eso lo podremos hacer accediendo a los valores como hemos aprendido con los diccionarios
diccionario_mascotas_copia["nombre"] = "Darwin"

# veamos como ha quedado el diccionario después del cambio. AHORA NUESTRA MASCOTA
print("El diccionario de la mascota es:", diccionario_mascotas_copia)

# incluso podemos cambiar el tipo del valor, por ejemplo de un string a una lista
# pero en este caso lo tenemos como un único string, cuando en realidad queremos
# seguirnos la misma lógica de antes
diccionario_mascotas_copia["vacunas"] = []

# veamos como ha quedado el diccionario después del cambio. AHORA el valor de la clave
print("El diccionario de la mascota es:", diccionario_mascotas_copia)

# ¿Y si quisieramos añadir una vacuna a la lista creada en el paso anterior? Podemos
# las vacunas que tiene la mascota, pero esto puede ser muy tedioso.
# otra forma de hacerlo es usando métodos de listas. Desgranemos este problema:

# si accedemos al valor de la clave "vacunas" nos devolverá una lista
print("El valor de la clave 'vacunas' es: ", diccionario_mascotas_copia['vacunas'])

# si vemos el tipo de este clave será una lista
print("El valor de la clave 'vacunas' es: ", type(diccionario_mascotas_copia['vacunas']))

# como se trata de una lista podremos usar los métodos de las listas que hemos aprendido
diccionario_mascotas_copia["vacunas"].append("gripe")

# veamos como ha quedado el diccionario después del cambio. AHORA el valor de la clave
print("El diccionario de la mascota después de añadir una vacuna a la lista es:")

# esta operación la podemos realizar todas las veces que queramos
# como se trata de una lista podremos usar los métodos de las listas que hemos aprendido
diccionario_mascotas_copia["vacunas"].append("leucemia felina")
print("El diccionario de la mascota después de añadir más vacunas a la lista es:")

# esta operación la podemos hacer con cualquier método de listas!!!
```

El diccionario de la mascota es: {'nombre': 'Poes', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': 'no tiene'}

El diccionario de la mascota es: {'nombre': 'Darwin', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': 'no tiene'}

El diccionario de la mascota es: {'nombre': 'Darwin', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': []}

El valor de la clave 'vacunas' es: []

El valor de la clave 'vacunas' es: <class 'list'>

El diccionario de la mascota después de añadir una vacuna a la lista es: {'nombre': 'Darwin', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': ['gripe']}

El diccionario de la mascota después de añadir más vacunas a la lista es: {'nombre': 'Darwin', 'cumpleaños': '01/07', 'tipo': 'européo común', 'vacunas': ['gripe', 'leucemia felina']}