😎 Diccionarios

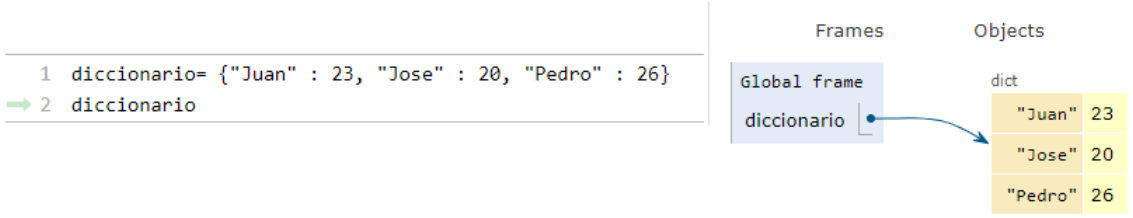
- **Un diccionario es un tipo de datos que sirve para asociar pares de objetos, puede ser visto como una colección de claves (llaves), cada una de las cuales tiene asociada un valor.**
- **Las claves (llaves) no están ordenadas y no hay claves repetidas.**
- **La única manera de acceder a un valor es a través de su clave (llave) a diferencia de las listas y tuplas, cuyos elementos se identifican por su posición.**
- **Las claves suelen ser números enteros o cadenas, aunque cualquier otro objeto inmutable puede actuar como una clave. Los valores, por el contrario, pueden ser de cualquier tipo, incluso otros diccionarios.**

Crear diccionarios

Los diccionarios se crean usando llaves { }. La llave y el valor van separados por dos puntos. En este ejemplo, las llaves son 'Juan', 'Jose' y 'Pedro', y los valores asociados a ellas son, respectivamente, 23, 20 y 26.

In [1]:
```
diccionario= {"Juan" : 23, "Jose" : 20, "Pedro" : 26}
diccionario
```

Out[1]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26}
```



In [2]:
```
type(diccionario)
```

Out[2]:
```
dict
```

```
Un diccionario vacío puede ser creado usando {} o con la función dict():
```

In [3]:
```
diccio = {}
```

In [4]:
```
diccio = dict()
```

In [5]:
```
type(diccio)
```

Out[5]:
```
dict
```

Otras formas de crear diccionarios

In [6]:
```
d = dict(Python=1991, C=1972, Java=1996)
d
```

Out[6]:
```
{'Python': 1991, 'C': 1972, 'Java': 1996}
```

In [7]:
```
gastos = {}
gastos['Luz']=1459.54
gastos['gas']=1978.34
gastos['telefono']=786.21
gastos
```

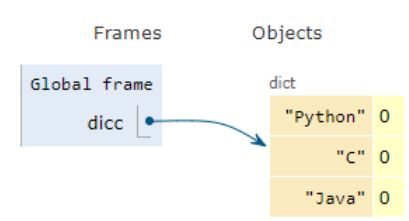
Out[7]:
```
{'Luz': 1459.54, 'gas': 1978.34, 'telefono': 786.21}
```

El método estático **fromkeys()** genera un diccionario a partir de un conjunto de claves y les asigna a todas ellas el mismo valor (None por defecto). El primer argumento puede ser, de hecho, cualquier objeto iterable

In [8]:
```
dict.fromkeys(["Python", "C", "Java"], 0)
```

Out[8]:
```
{'Python': 0, 'C': 0, 'Java': 0}
```

```
1  dicc = dict.fromkeys(["Python", "C", "Java"], 0)
```



Armar un diccionario a partir del contenido de un archivo

In [9]:
```
salarios={}
with open('archs/Base_salarios.csv', 'rt') as f:
    for line in f:
        row = line.split(';')
        salarios[row[0]] =float(row[1])

# salarios
```

Verificar si una clave está en el diccionario:

In [10]:
```
if 'Suarez' in salarios:
    print("Está")
else:
    print("No está")
```

```
Está
```

Restricciones sobre las llaves

No se puede usar cualquier objeto como llave de un diccionario. Las llaves deben ser de un tipo de datos **inmutable**. Por ejemplo, no se puede usar listas:

In [11]:
```
d = {[1, 2, 3]: 'hola'} # error una lista no puede ser clave de diccionario
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
Input In [11], in <cell line: 1>()
----> 1 d = {[1, 2, 3]: 'hola'}

TypeError: unhashable type: 'list'
```

Acceder a los elementos

El valor asociado a la llave k en el diccionario d se puede obtener mediante **diccionario [k]**:

In [12]:
```
diccionario = {"Juan":23,"Jose":20,"Pedro":26}
```

In [13]:
```
diccionario['Jose']
```

Out[13]: 20

A diferencia de los índices de las listas, las llaves de los diccionarios no necesitan ser números enteros. Si la llave no está presente en el diccionario, ocurre un error de llave (KeyError):

In [14]:
```
diccionario['Maria']
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
Input In [14], in <cell line: 1>()
----> 1 diccionario['Maria']

KeyError: 'Maria'
```

Una forma alternativa para obtener un valor es el método get(), indicando la clave como argumento. Agregando is None -cuando la clave no existe- el valor retornado es True

In [15]:
```
diccionario = {'Juan': 'Ingeniero', 'José': 'Medico', 'Pedro': 'Abogado'}
diccionario.get("José")
```

Out[15]: 'Medico'

In [16]:
```
diccionario.get("Pepe") is None
```

Out[16]:
```
True
```

In [17]:
```
diccionario.get("Pepe", "No esta en el diccionario")
```

Out[17]:
```
'No esta en el diccionario'
```

Agregar, modificar y eliminar elementos

Agregar

Se puede agregar una llave nueva simplemente asignándole un valor.

In [18]:
```
diccionario = {"Juan":23,"Jose":20,"Pedro":26}
diccionario
```

Out[18]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26}
```

In [19]:
```
diccionario["Pepe"] = 28
diccionario
```

Out[19]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26, 'Pepe': 28}
```

Modificar

Si se asigna un valor a una llave que ya estaba en el diccionario, el valor anterior se sobreescribe. Recuerde que un diccionario no puede tener llaves repetidas pero sí el valor.

In [20]:
```
diccionario["Pedro"] = 19
diccionario
```

Out[20]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 19, 'Pepe': 28}
```

Eliminar

Para borrar una llave, se puede usar la sentencia **del**

In [21]:
```
del diccionario['Pedro']
diccionario
```

Out[21]:
```
{'Juan': 23, 'Jose': 20, 'Pepe': 28}
```

Otros métodos

El método **clear()** elimina todas las claves y valores

In [22]:
```
diccionario = {'Juan': 23, 'Jose': 20, 'Pedro': 26}
diccionario
```

Out[22]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26}
```

In [23]:
```
diccionario.clear()
diccionario
```

Out[23]:
```
{}
```

La función **len()** entrega cuántos pares llave-valor hay en el diccionario

In [24]:
```
diccionario = {'Juan': 23, 'Jose': 20, 'Pedro': 26}
diccionario
```

Out[24]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26}
```

In [25]:
```
len(diccionario)
```

Out[25]:
```
3
```

Podemos actualizar un diccionario o bien unir dos de ellos con **update()**:

In [26]:
```
diccionario1 = {"Juan":"Ingeniero","Caro":"Medico","Pedro":"Abogado"}
diccionario2 = {"Maria":23,"Pepa":22,"Kika":27}
```

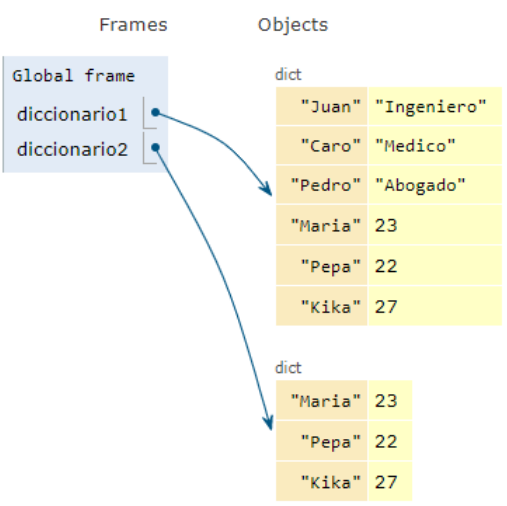
diccionario1, diccionario2

```
In [27]: diccionario1.update(diccionario2)
```

```
In [28]: diccionario1
```

```
Out[28]: {'Juan': 'Ingeniero',
          'Caro': 'Medico',
          'Pedro': 'Abogado',
          'Maria': 23,
          'Pepa': 22,
          'Kika': 27}
```

```
1  diccionario1 = {"Juan":"Ingeniero","Caro":"Medico","Pedr
2  diccionario2 = {"Maria":23,"Pepa":22,"Kika":27}
3
4  diccionario1.update(diccionario2)
```

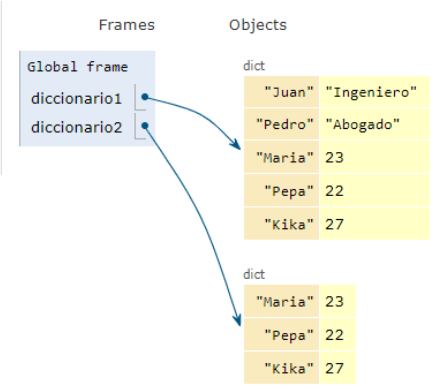


La método **pop()** elimina un elemento una vez retornado

```
In [29]: diccionario1.pop("Caro")
         diccionario1
```

```
Out[29]: {'Juan': 'Ingeniero', 'Pedro': 'Abogado', 'Maria': 23, 'Pepa': 22, 'Kika': 27}
```

```
1  diccionario1 = {"Juan":"Ingeniero","Caro":"Medico","Pedro":"Abogado"}
2  diccionario2 = {"Maria":23,"Pepa":22,"Kika":27}
3  diccionario1.update(diccionario2)
4
5  diccionario1.pop("Caro")
```



El método **popitem()** retorna un elemento aleatorio y después lo elimina

```
In [30]: diccionario1.popitem()
```

```
Out[30]: ('Kika', 27)
```

```
In [31]: diccionario1
```

```
Out[31]: {'Juan': 'Ingeniero', 'Pedro': 'Abogado', 'Maria': 23, 'Pepa': 22}
```

```
In [32]: diccionario1.popitem()
```

```
Out[32]: ('Pepa', 22)
```

Operadores in y not in

k **in** diccionario permite saber si la llave k está en el diccionario:

In [33]:
```
patas = {'gato': 4, 'humano': 2, 'pulpo': 8, 'perro': 4, 'ciempies': 100}
patas
```

Out[33]:
```
{'gato': 4, 'humano': 2, 'pulpo': 8, 'perro': 4, 'ciempies': 100}
```

In [34]:
```
'perro' in patas
```

Out[34]:
```
True
```

In [35]:
```
'loro' in patas
```

Out[35]:
```
False
```

Para saber si una llave no está en el diccionario, se usa el operador **not in**:

In [36]:
```
patas = {'gato': 4, 'humano': 2, 'pulpo': 8, 'perro': 4, 'ciempies': 100}
'lombriz' not in patas
```

Out[36]:
```
True
```

Es posible crear listas de llaves o valores:

In [37]:
```
diccionario = {'Juan': 23, 'Jose': 20, 'Pedro': 26}
lista = list(diccionario)
lista
```

Out[37]:
```
['Juan', 'Jose', 'Pedro']
```

In [38]:
```
valores = list(diccionario.values())
valores
```

Out[38]:
```
[23, 20, 26]
```

In [39]:
```
lista = list(patas)
lista
```

Out[39]:
```
['gato', 'humano', 'pulpo', 'perro', 'ciempies']
```

Iteraciones

Los diccionarios son iterables.

Al iterar sobre un diccionario en un ciclo for, se obtiene las llaves:

In [40]:
```
diccionario = {"Juan":23,"Jose":20,"Pedro":26}
diccionario
```

Out[40]:
```
{'Juan': 23, 'Jose': 20, 'Pedro': 26}
```

In [41]:
```
for k in diccionario:
    print(k)
```

```
Juan
Jose
Pedro
```

In [42]:
```
for k in patas:
    print(k)
```

```
gato
humano
pulpo
perro
ciempies
```

Para iterar sobre las llaves, se usa **values()**

In [43]:
```
for v in diccionario.values():
    print(v)
```

```
23
20
26
```

In [44]:
```
for k in patas.values():
    print(k)
```

```
4
2
8
4
100
```

Para iterar sobre las llaves y los valores simultáneamente, se usa el método **items()**

In [45]:
```
diccionario = {"Juan":23,"Jose":20,"Pedro":26}
diccionario
```

Out[45]: {'Juan': 23, 'Jose': 20, 'Pedro': 26}

In [46]:
```
for k, v in diccionario.items():
    print('La edad de ', k, 'es', v)
```

```
La edad de  Juan es 23
La edad de  Jose es 20
La edad de  Pedro es 26
```

In [47]:
```
for k, v in patas.items():
    print("El ", k, "tiene", v, "patas")
```

```
El  gato tiene 4 patas
El  humano tiene 2 patas
El  pulpo tiene 8 patas
El  perro tiene 4 patas
El  ciempies tiene 100 patas
```

Claves compuestas

Prácticamente cualquier valor puede usarse como clave en un diccionario de Python. La principal restricción es que una clave debe ser de tipo inmutable. Por ejemplo, tuplas:

In [48]:
```
cumples = {}

cumples = {
(19, 2) : 'Pedro',
(24, 5) : 'Rosana',
(13, 10) : "Rubén",
}
```

In [49]:
```
cumples[(24, 5)]
```

Out[49]: 'Rosana'

Las listas, los conjuntos y los diccionarios no pueden ser usados como claves de diccionarios, porque son mutables.