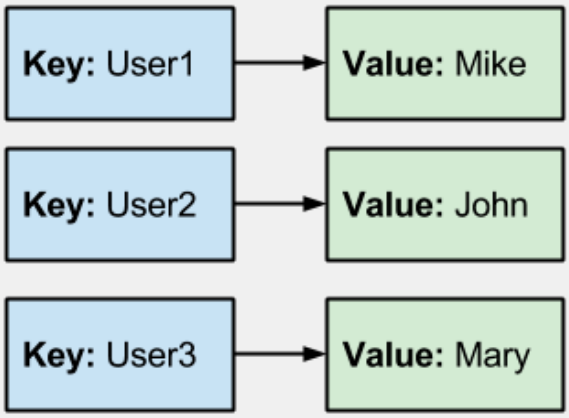
INFORMATORIO

TALLER DE PROGRAMACIÓN WEB

DICCIONARIOS

Diccionarios

En Python, un **diccionario** es una colección no ordenada de valores que son accedidos a través de *keys* (llaves o claves).



Las keys o llaves deben ser únicas, si se asigna un valor a una llave ya existente, se reemplaza el valor anterior.

A diferencia de las listas o tuplas que se acceden a los elementos mediante el índice numérico.

Los diccionarios son mutables, por lo tanto se pueden **modificar, agregar o quitar** elementos.

Para crear diccionarios **encerramos** los elementos y sus keys entre `{ }`, los elementos tendría la estructura **`clave:valor`** y cada uno de los elementos deben estar separados por coma.

Las keys/claves pueden ser de cualquier tipo inmutable, y los valores pueden ser de cualquier tipo.

Veamos algunos ejemplos:

```
coordenadas = {"x":34, "y": 23, "z":45}

agenda_telefonica = {
       "Sandra":456773,
       "Pedro":3784758,
       "Lorena":483758,
       "Juan":578493
       }

seleccion_futbol = {
       1:"Sergio Almirón",
       2:"Sergio Batista",
       3:"Ricardo Enrique Bochini",
       4:"Claudio Borghi",
       5:"José Luis Brown",
       6:"Daniel Passarella",
       7:"Jorge Burruchaga",
       8:"Néstor Clausen",
       9:"José Luis Cuciuffo",
       10:"Diego Maradona",
       11:"Jorge Valdano"
       }

alumno = { "nombre": "Marcos", "materias":["biología", "matemáticas", "diseño"]}
```

Para acceder los elementos utilizamos la clave del diccionario.

```
>>> coordenadas["x"]
34
>>> agenda_telefonica["Sandra"]
456773
>>> seleccion_futbol[10]
"Diego Maradona"
```

Al igual que con lista, podemos declarar diccionarios vacíos de dos maneras.

```
>>> diccionario = { }

O podemos usar esta opción
>>> diccionarios = dict()
```

Para modificar un valor, de manera similar a las listas pero en lugar de usar el índice usamos la clave.

```
>>> usuario = { "nombre":"Juan", "edad":23 }
>>> print(usuario)
{'nombre': 'Juan', 'edad': 23}
>>> usuario["edad"] = 24
>>> print(usuario)
{'nombre': 'Juan', 'edad': 24}
```

Para agregar claves a un diccionario existente, realizamos la acción similar a como modificamos el valor de una clave existente pero con la nueva clave

```
>>> usuario["apellido"] = "Perez"
>>> print(usuario)
>>> {'nombre': 'Juan', 'edad': 24, 'apellido': 'Perez'}
```

Otra manera de insertar elementos es usando el método setdefault(), al cual se le pasa la clave y el elemento a agregar.
Pero solo se agregara el elemento si la llave no existía, si ya se encontraba en el diccionario no agrega el nuevo valor.

```
>>> usuario = { "nombre":"Juan", "edad":23 }
>>> usuario.setdefault("comida_favorita", "ninguna")
>>> print(usuario)
{'nombre': 'Juan', 'edad': 23, 'comida_favorita': 'ninguna'}
```


```
>>> usuario.setdefault("nombre", "Cristobal")
>>> print(usuario)
{'nombre': 'Juan', 'edad': 23, 'comida_favorita': 'ninguna'}
```

Como vimos podemos acceder a un elemento por medio de su llave

```
>>> usuario["nombre"]
Juan
```

pero en caso de tratar acceder con una clave no existente en el diccionario da un error.

```
>>> usuario["color_favorito"]
KeyError: 'color_favorito'
```

para esto los diccionarios incluyen un **método `get()`**, en el que se le pasa la clave como parámetro y devuelve el elemento asociada a la clave o el valor **`None`** si la clave no se encuentra.

```
>>> color = usuario.get("color_favorito")
>>> print(color)
None
```

también podemos opcionalmente pasarle a **`get()`** en otro parámetro el elemento a devolver en lugar de **`None`** en caso de no encontrar la clave en el diccionario.

```
>>> color = usuario.get("color_favorito", "no tiene color favorito")
>>> print(color)
'no tiene color favorito'
```

No podemos obtener partes de un diccionario usando **[:]** como en listas por que ***los diccionarios no tienen los elementos ordenados***.

Para **recorrer un diccionario** podemos hacerlo de varias maneras

```
>>> diccionario = {"a":11, "b":22, "c":33, "d":44}
>>> for key in diccionario:
>>>   print("la clave es: ", key)
>>>   print("su valor: ", diccionario[key])
```

pero también tenemos un método **`items()`** en los diccionarios, que nos devuelve una lista de tuplas que contienen el par clave:value.

```
>>> diccionario.items()
dict_items([('a', 11), ('b', 22), ('c', 33), ('d', 44)])
```

podemos usar esto para recorrer un diccionario de la siguiente manera

```
>>> for key, value in diccionario.items():
>>>   print(key,":",value)
```

Para verificar si una clave se encuentra en el diccionario, podemos usar el operador *in*

```
>>> diccionario = {"a":11, "b":22, "c":33, "d":44}
>>> "a" in diccionario
True
>>> "f" in diccionario
False

>>> if "b" in diccionario:
>>>   print(diccionario["b"])
```

Para eliminar algún par **`clave:value`** de un diccionario podemos utilizar la palabra reservada ***del***

```
>>> persona = {"nombre":"Marta", "genero":"F"}
>>> print(persona)
{'nombre': 'Marta', 'genero': 'F'}
>>> del persona["genero"]
>>> print(persona)
{'nombre': 'Marta'}
```

También podemos utilizar el método **`pop()`** que vimos anteriormente en listas.
Este método nos quitará el elemento que le indiquemos en la clave y nos retorna el valor quitado.

```
>>> configuracion = { "color_fondo":"negro", "color_texto":"blanco",
"font_size":12}
>>> print(configuracion)
{ "color_fondo":"negro", "color_texto":"blanco", "font_size":12}
>>> quitado = configuracion.pop("color_fondo")
>>> print(quitado)
negro
>>> print(configuracion)
{'color_texto': 'blanco', 'font_size': 12}
```

A diferencia de la lista que podíamos usar el `pop()` sin pasarle un índice, aca obligatoriamente hay que pasarle la key.
En caso de pasarle una clave o key que no esté presente en el diccionario obtendremos un error

```
>>> configuracion.pop("url")
KeyError: 'url'
```

pero podemos pasarle al método `pop()` un valor de retorno en caso que la llave no exista

```
>>> valor = configuracion.pop("idioma", "opcion no configurada")
>>> print(valor)
"opcion no configurada"
```

Si queremos borrar todos los elementos de un diccionario, o en otras palabras limpiar completamente un diccionario, podemos usar el método `clear()`

```
>>> diccionario = {1:("a","b"),2:("c","d")}
>>> print(diccionario)
{1: ('a', 'b'), 2: ('c', 'd')}
>>> diccionario.clear()
>>> print(diccionario)
{ }
```

también podemos extender o actualizar un diccionarios con los elementos de otro diccionario utilizando el método `update()`.
Si las claves del diccionario que pasamos como parámetro existen, se van actualizar los valores de las keys repetidas.

Si las claves no existen, se agregan los nuevos pares key:value

```
>>> diccionario_uno = {"a":100, "b":456, "c":45, "d":88}
>>> diccionario_dos = {"d":110, "f":66, "g":45, "h":99 }
>>> print(diccionario_uno)
{'a': 100, 'b': 456, 'c': 45, 'd': 88}
>>> diccionario_uno.update(diccionario_dos)
>>> print(diccionario_uno)
{'a': 100, 'b': 456, 'c': 45, 'd': 110, 'f': 66, 'g': 45, 'h': 99}
```

vemos como el elemento con clave "d", que existía previamente actualizó su valor del elemento con clave "d" del segundo diccionario. Y se agregaron el resto de claves que no existían previamente en el diccionario con sus respectivo valores del segundo diccionario.

Funciones y métodos

Para saber la cantidad de elementos de un diccionario utilizamos `len()`

```
>>> diccionario = {"hola":"hello", "chau":"bye", "cinco":"five",
"manzana":"apple"}
>>> len(diccionario)
4
```

para tener una lista con las claves de un diccionarios podemos usar el método `keys()`

```
>>> diccionario.keys()
dict_keys(['hola', 'chau', 'cinco', 'manzana'])
```

para recibir una lista con los valores del diccionario tenemos el método `values()`

```
>>> diccionario.values()
dict_values(['hello', 'bye', 'five', 'apple'])
```

podríamos usarlo por ejemplo para ver si un valor se encuentra en un diccionario

```
>>> "hello" in diccionario.values()
True
```

Como vimos anteriormente, podemos recibir una lista con tuplas que contienen el par `key:value` con el método items()

```
>>> diccionario.items()
dict_items([('hola', 'hello'), ('chau', 'bye'), ('cinco', 'five'),
('manzana', 'apple')])
```

Como puedes observar los objetos devueltos por `diccionario.keys()`, `diccionario.values()` y `diccionario.items()` no son como las listas que hemos visto anteriormente, eso es por que realmente son views objects (vista de objetos). Estos proporcionan una manera de ver dinámicamente los elementos del diccionario, esto significa que si el diccionario cambia en la vistas se verán reflejados esos cambios.

```
>>> diccionario = {"a":10,"b":20}
>>> print(diccionario)
{'a': 10, 'b': 20}
>>> valores = diccionario.values()
>>> print(valores)
dict_values([10, 20])
>>> diccionario["b"] = 30
>>> print(diccionario)
{'a': 10, 'b': 30}
>>> print(valores)
dict_values([10, 30])
```

Si necesitamos listas podemos crearlas a partir de estas vistas con la función list(), o si necesitamos tuplas con tuple()

```
>>> diccionario = {"a":10,"b":20}
>>> print(diccionario)
{'a': 10, 'b': 20}
>>> valores = diccionario.values()
>>> print(valores)
dict_values([10, 20])
>>> lista = list(valores)
>>> print(lista)
[10, 20]
```