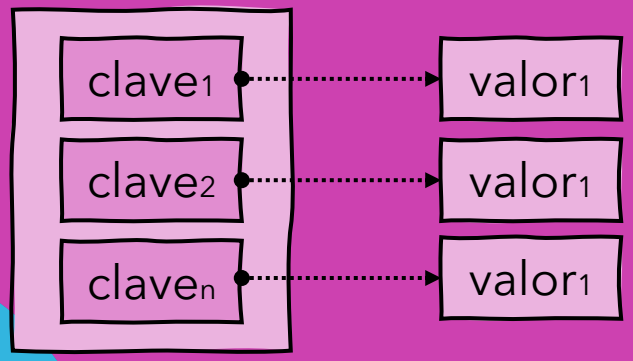
DICCIONARIOS

Algoritmos y estructuras de datos I

DICCIONARIOS

```
diccionario = {
              clave1 : valor1,
              clave2 : valor2,
              claven : valorn
              }
```



diccionario

Son un tipo de estructuras de datos que permite guardar un conjunto no ordenado de pares *clave:valor*, siendo las claves únicas dentro de un mismo diccionario.

Los diccionarios son mutables, es decir, es posible modificar su longitud, podemos agregar o quitar elementos de él; de igual forma todos los valores almacenados en el diccionario pueden ser modificados.

Los diccionarios no se rigen por la regla de los índices, en este caso todos los valores que se almacenen en el diccionario no corresponderán a un índice, si no a una clave (llave o *key*). Todos los valores necesitan tener una clave y cada clave necesita tener un valor.

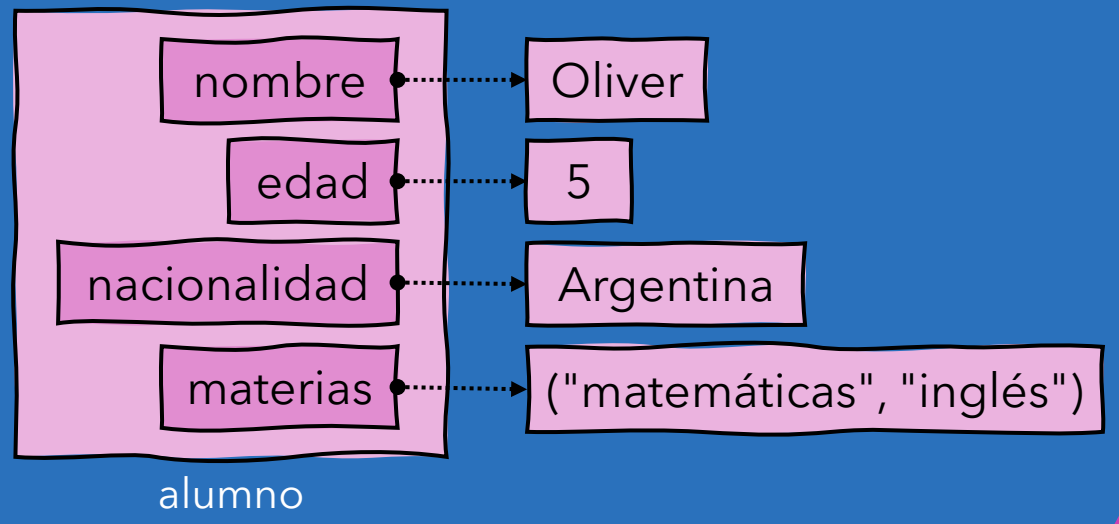
CREACIÓN

A continuación, observamos un ejemplo en el que creamos un diccionario para almacenar datos de un alumno.

En el mismo, estamos almacenando cuatro valores con sus correspondientes llaves (nombre, edad, nacionalidad y materias).

En lo general, haremos uso de llaves de un mismo tipo, comúnmente *string*, sin embargo, si por algún motivo necesitamos almacenar llaves con diferentes tipos de datos, es posible hacerlo.

```
alumno = {
            "nombre" : "Oliver",
            "edad" : 5,
            "nacionalidad" : "Argentina",
            "materias" : ("matemáticas", "inglés")
          }
```



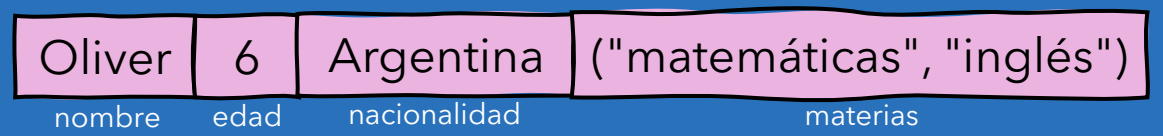
CREACIÓN Y ACCESO

Para poder agregar, obtener o modificar algún valor del diccionario haremos uso de corchetes.

El acceso a un valor se realiza mediante indexación de la clave. Para ello, simplemente encierra entre corchetes la clave del elemento alumno["edad"] siguiendo el ejemplo del código anterior. Si la clave ya existe en el diccionario, se actualiza su valor.

En caso de que la clave no exista, se lanzará la excepción *KeyError*.

```
alumno = {}

#Agregamos elementos
alumno["nombre"] = "Oliver"
alumno["edad"] = 5
alumno["nacionalidad"] = "Argentina"
alumno["materias"] = ("matemáticas", "inglés")

alumno["edad"] = 6 #Modificamos un valor

print(alumno["nombre"]) #Oliver
print(alumno["edad"]) #6
```



```
alumno = {
        "nombre":"Oliver",
        "edad":5,
        "nacionalidad":"Argentina",
        "materias":("matemáticas", "inglés")
      }

print(len(alumno)) #4
```

CANTIDAD DE ELEMENTOS

Para determinar la cantidad de elementos o pares clave-valor que contiene un diccionario, se utiliza la función *len()*.

Esta función toma como argumento el diccionario y devuelve el número total de pares clave-valor presentes en él.

Al utilizar *len()* con un diccionario, se obtiene un valor que representa la cantidad de elementos que contiene, lo que puede ser útil para diversas operaciones y análisis de datos.

COMPARACIÓN

En Python se puede utilizar el operador de igualdad == para comparar si dos diccionarios son iguales. Dos diccionarios son iguales si contienen el mismo conjunto de pares *clave:valor*, independientemente del orden que tengan.

Otro tipo de comparaciones entre diccionarios no están permitidas. Si se intenta, el intérprete lanzará la excepción *TypeError*.

```
alumno = {
            "nombre":"Oliver",
            "edad":5
          }

alumno2 = {
            "nombre":"Oliver",
            "edad":6
          }

if (alumno == alumno2):
   print("Son iguales")
else:
   print("Son distintos")
```

```
alumno = {
            "nombre":"Oliver",
            "edad":5,
            "nacionalidad":"Argentina",
            "materias":("matemáticas", "inglés")
           }

print(alumno.get("nombre")) #Oliver
print(alumno.get("edad")) #5
print(alumno.get("apellido", "-")) #-
print(alumno.get("añoIngreso")) #None
```

MÉTODO GET

Este método devuelve el valor correspondiente a la clave. En caso de que la clave no exista no lanza ningún error, sino que devuelve el segundo argumento valor por defecto. Si no se proporciona este argumento, se devuelve el valor *None*.

optativo

```
diccionario.get( clave, valor por defecto )
```

En el ejemplo, observaremos su utilización.

RECORRIDOS

Si queremos recorrer tanto las llaves como sus correspondientes valores haremos uso del método *items*.

```
alumno = {
        "nombre":"Oliver",
        "edad":5,
        "nacionalidad":"Argentina",
        "materias":("matemáticas", "inglés")
      }
for clave, valor in alumno.items():
    print(clave, " --> ", valor)
```

También, podremos recorrer solo las claves o los valores con los métodos *keys* y *values* por separado:

```
for clave in alumno.keys():
    print(clave)

for valor in alumno.values():
    print(valor)
```

ELIMINAR UN ELEMENTO

Si quisiéramos eliminar un elemento del diccionario, utilizamos la función *del*. Si no existe la clave, se lanza una excepción (*KeyError*):

```
alumno = {
        "nombre":"Oliver",
        "edad":5,
        "nacionalidad":"Argentina",
        "materias":("matemáticas", "inglés")
      }

del alumno["nacionalidad"]
print(alumno)
#{'nombre': 'Oliver', 'edad': 5, 'materias': ('matemáticas', 'inglés')}
```

Podemos verificar la existencia de la llave con el operador *in*:

```
if "nacionalidad" in alumno:
    del alumno["nacionalidad"]
```

También podemos eliminar todos los valores del diccionario con el método *clear()*:

```
alumno.clear()
print(alumno) #{}
```

LISTAS DE DICCIONARIOS

Esta capacidad de utilizar diccionarios dentro de listas es particularmente útil cuando es necesario manejar una colección de elementos que requieren almacenar información estructurada y heterogénea.

En el ejemplo, ciudades es una lista que contiene diccionarios, cada uno representando una ciudad con su respectivo código postal y nombre.

```
ciudades = [
                    {
                        "cp": 1425,
                        "nombre": "Capital Federal"
                    },
                    {
                        "cp": 1900,
                        "nombre": "La Plata"
                    }
                ]

#Acceso a un elemento
print("Código postal La Plata: ", ciudades[1]["cp"])

#Agregar un elemento
ciudades.append({"cp":1878, "nombre":"Quilmes"})

#Podemos recorrer por objeto
for ciudad in ciudades:
    print(ciudad["nombre"], "(", ciudad["cp"], ")")
```

LISTAS DE DICCIONARIOS

Podríamos interpretar la visualización de una lista de diccionarios como una grilla; en donde la clave es el título de la columna y, los valores, los registros (filas) que corresponden a cada una de ellas. Cada elemento (intersección de fila y columna) se accederá por la combinación de la posición de la fila (elemento de lista) y la columna (clave).

```
ciudades = [
    {
        "cp": 1425,
        "nombre": "Capital Federal"
    },
    {
        "cp": 1900,
        "nombre": "La Plata"
    }
]

#Agregar un elemento
ciudades.append({"cp":1878, "nombre":"Quilmes"})

#Acceso a un elemento
print("Código postal La Plata: ",ciudades[1]["cp"])
```

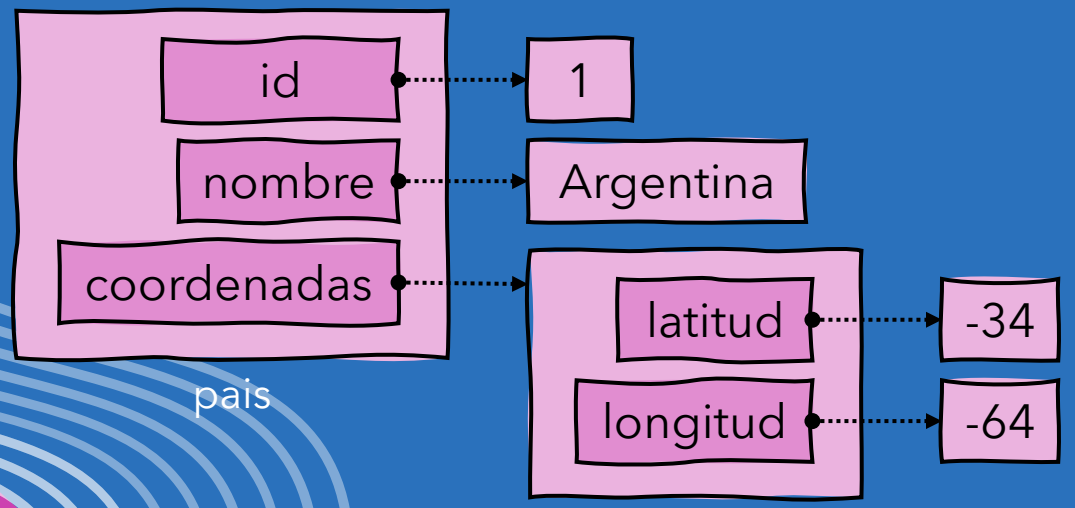
	cp	nombre
0	1425	Capital Federal
1	1900	La Plata
2	1878	Quilmes

ANIDAMIENTO

Es posible, también, crear un diccionario cuyo valor contiene otro diccionario. En dicha situación, se debe acceder a los valores por medio de las distintas claves que lo componen como si fuera una matriz.

En el ejemplo propuesto, se observa que, si se quiere obtener el valor de la latitud de las coordenadas de un país, se debe acceder a los distintos niveles de claves: coordenadas y, luego, latitud.

```
pais = {
        "id": 1,
        "nombre": "Argentina",
        "coordenadas": {
                        "latitud": -34,
                        "longitud": -64
                        }
        }

print(pais["nombre"])
print(f"({pais["coordenadas"]["latitud"]},
{pais["coordenadas"]["longitud"]})")
```



```
siglasSignificado = {
    "HTML": "Hipertext Markup Lenguaje",
    "CSS": "Cascading Style Sheets",
    "API": "Application Programming Interface",
    "SQL": "Structured Query Language",
    "JSON": "JavaScript Object Notation"
}

siglasOrdenado = dict(sorted(siglasSignificado.items()))

print(siglasOrdenado)
#{'API': 'Application Programming Interface', 'CSS':
'Cascading Style Sheets', 'HTML': 'Hipertext Markup
Lenguaje', 'JSON': 'JavaScript Object Notation', 'SQL':
'Structured Query Language'}
```

ORDENAR LAS CLAVES

Para ordenar un diccionario por sus claves, puedes utilizar la función *sorted()* en combinación con el método *items()* del diccionario. Esto te permite ordenar los elementos del diccionario según las claves en orden alfabético o numérico.

En este ejemplo, utilizamos la función *sorted()* junto con el método *items()* del diccionario para obtener una lista de tuplas *(clave:valor)* ordenadas según las claves. Luego, convertimos esta lista ordenada de tuplas de nuevo en un diccionario utilizando la función *dict()*.

Esto resulta en el diccionario alumnoOrdenado, donde los elementos están ordenados alfabéticamente por las claves.

AGREGAR UN DICCIONARIO A OTRO DICCIONARIO

El método *update()* se utiliza para agregar elementos a un diccionario o para actualizar los valores existentes en el diccionario con nuevos valores. La sintaxis básica es:

```
diccionario.update( {diccionario} )
```

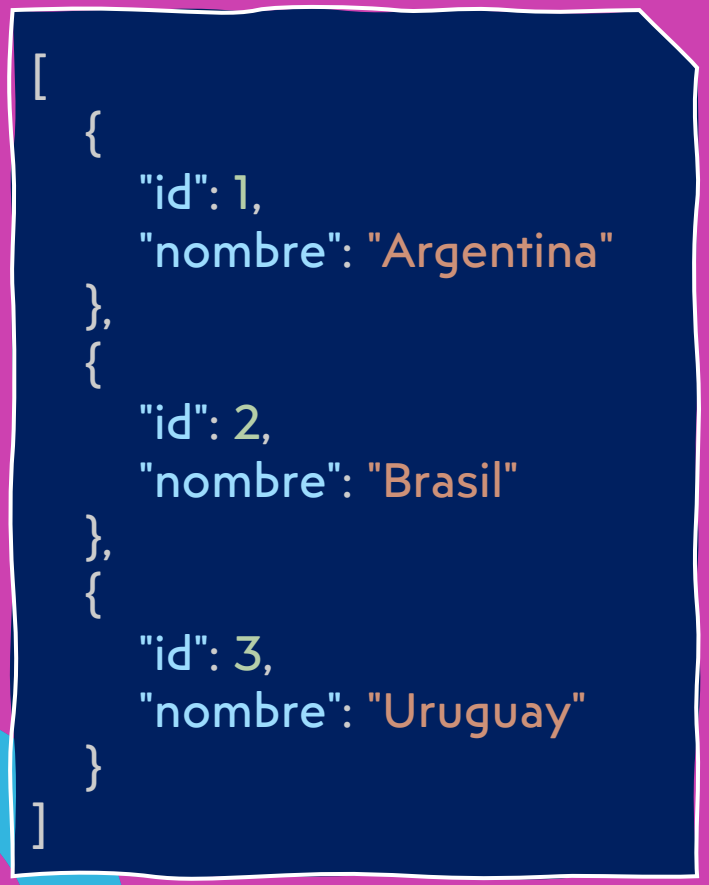
Recibe como parámetro otro diccionario que contiene pares *clave:valor* que se agregarán al diccionario existente.

```
diccionario = {'a': 1, 'b': 2, 'c': 3}
diccionario.update({'d': 4, 'e': 5})
#{'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5}

diccionarioB = {'b': 10}
diccionario.update(diccionarioB)
#{'a': 1, 'b': 10, 'c': 3, 'd': 4, 'e': 5}

print(diccionario)
```

En este ejemplo, se utiliza el método *update()* para agregar dos nuevos pares *clave:valor* al diccionario (*'d': 4 y 'e': 5*).

Finalmente, se utiliza update() para actualizar el valor de la clave 'b' en el diccionario de 2 a 10 (desde una variable).

Lic. Yaps David P.



ARCHIVOS JSON
JAVASCRIPT OBJECT NOTATION

Es una notación para la transferencia de datos que sigue un estándar específico.

Se utiliza comúnmente para representar información estructurada en formato de texto, facilitando la transferencia de datos entre sistemas de manera eficiente.

Es similar a los diccionarios en Python, ya que los datos en JSON se organizan en una colección de pares *clave:valor*. Esta similitud con los diccionarios en Python facilita la manipulación y comprensión de los datos tanto, permitiendo una fácil integración y manipulación de información entre diferentes sistemas y lenguajes de programación.

LEER E INTERPRETAR UN ARCHIVO JSON

A continuación, veamos un ejemplo de cómo realizar la lectura de un archivo JSON.

```json
[
    {
        "id": 1,
        "nombre": "Argentina"
    },
    {
        "id": 2,
        "nombre": "Brasil"
    },
    {
        "id": 3,
        "nombre": "Uruguay"
    }
]
```

archivos/paises.json

```python
import json

try:
    contenido = open("archivos/paises.json", "r")
    lineas = contenido.read()
    contenido.close()

    # Se convierte en un diccionario
    paises = json.loads(lineas)

    print(paises)

    for pais in paises:
        print(pais["id"], "->", pais["nombre"])

except:
    print("No se puede abrir el archivo")
```

Lic. Yaps David P.

MODIFICAR UN ARCHIVO JSON

```python
import json

try:
    contenido = open("archivos/paises.json", "r")
    lineas = contenido.read()
    contenido.close()

    #Se convierte en un diccionario
    paises = json.loads(lineas)
    paises.append({"id":4, "nombre":"Paraguay"})

    #Se convierte el diccionario a JSON
    #indent no es obligatorio, mejora el formateo
    paisesJSON = json.dumps(paises, indent=4)
    try:
        contenido = open("archivos/paises.json", "w")
        contenido.write(paisesJSON)
        contenido.close()
    except:
        print("No se puede grabar el archivo")
except:
    print("No se puede leer el archivo")
```

```json
[
    {
        "id": 1,
        "nombre": "Argentina"
    },
    {
        "id": 2,
        "nombre": "Brasil"
    },
    {
        "id": 3,
        "nombre": "Uruguay"
    },
    {
        "id": 4,
        "nombre": "Paraguay"
    }
]
```

archivos/paises.json

Lic. Yaps David P.

AUTOEVALUACIÓN

La autoevaluación es crucial para identificar mejoras y fortalezas de forma objetiva, facilitando la fijación de metas y la toma de acciones para alcanzarlas.