



ESTRUCTURAS DE DATOS

Manejo de información

01 {

[Estructuras de Datos]

- Tuplas
- Sets
- Diccionarios

}

Tuplas {

En Python, una tupla es un **tipo de dato** que contiene un conjunto de valores ordenados.

Las tuplas se utilizan para almacenar **datos que deben estar juntos y que no deben cambiarse (constantes)**, como los nombres de los días de la semana o los meses del año.

Las tuplas se pueden crear utilizando **paréntesis ()**.



```
tupla = ("Enero", "Jueves", 3.14)
```

Tuplas {

Las tuplas son una estructura de datos similar a las listas, su principal diferencia es que las tuplas son **inmutables**, es decir, no se le pueden agregar ni eliminar elementos.



```
tupla = ("Enero", "Jueves", 3.14)
```

0 1 2

Características de las tuplas {

1. **Ordenadas:** Los elementos tienen un orden definido y ese orden no cambiará.
2. **Inmutables:** No podemos cambiar, agregar o eliminar elementos después de que se haya creado la tupla.
3. **Heterogeneas:** Pueden guardar múltiples tipos de datos al mismo tiempo.
4. **Contienen duplicados:** Permite tener objetos duplicados.

}

Indices {

Los índices se utilizan para **acceder a elementos específicos de una estructura**.

Los índices son **números enteros que indican la posición de un elemento** en la estructura.

En Python, los índices **comienzan en 0**, lo que significa que el primer elemento de la estructura tiene un índice de 0, el segundo elemento tiene un índice de 1, y así sucesivamente.

}

Indices {

Para acceder a un elemento de la tupla mediante índices, debemos primero colocar el **nombre de la tupla**, seguido de **corchetes[]** y dentro de este colocaremos la **posición deseada**.



```
tupla = ("Enero", "Jueves", 3.14)
#           0           1           2
primer_elemento = tupla[0]
print(primer_elemento)
```

Indices negativos {

También se puede acceder a los elementos de la estructura utilizando **índices negativos**, que cuentan desde el final de la lista hacia el principio.

Por ejemplo, **el último elemento de la tupla tiene un índice de -1**, el segundo último tiene un índice de -2, y así sucesivamente.

}

Indices negativos {




```
tupla = ("Enero", "Jueves", 3.14)
#           -3           -2           -1
ultimo_elemento = tupla[-1]
print(ultimo_elemento) # 3.14
```

}

Funcion len() {

La función len() se utiliza para obtener **la longitud de una estructura**, es decir, el **número de elementos** que contiene la estructura.



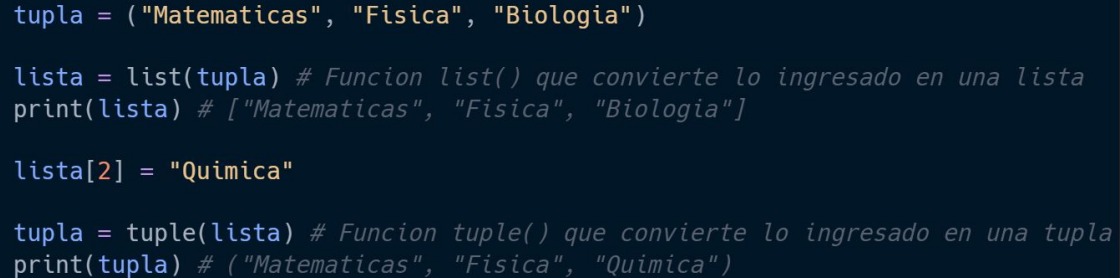
```
tupla = ("Enero", "Jueves", 3.14)
tamanio = len(tupla)
print(tamanio) # 3
```

}

Actualizar tuplas {

Una vez que se crea una tupla, **no puede cambiar sus valores**. Las tuplas son **inmutables**.

Pero hay una solución, se puede **convertir la tupla en una lista**, cambiar la lista y volver a convertir la **lista en una tupla**.



```
tupla = ("Matematicas", "Fisica", "Biologia")

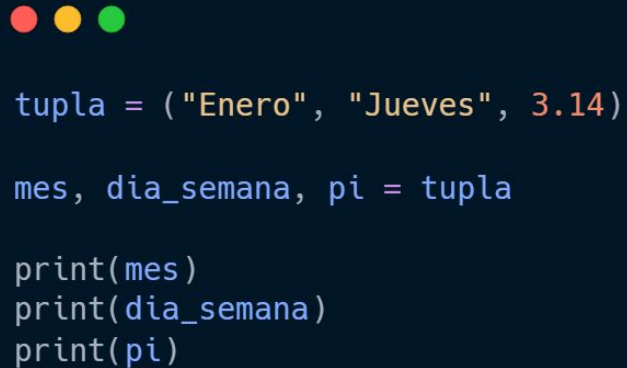
lista = list(tupla) # Funcion list() que convierte lo ingresado en una lista
print(lista) # ["Matematicas", "Fisica", "Biologia"]

lista[2] = "Quimica"

tupla = tuple(lista) # Funcion tuple() que convierte lo ingresado en una tupla
print(tupla) # ("Matematicas", "Fisica", "Quimica")
```

Desempaquetar tuplas {

Es el proceso de **asignar los valores de una tupla a variables individuales**.




```
tupla = ("Enero", "Jueves", 3.14)

mes, dia_semana, pi = tupla

print(mes)
print(dia_semana)
print(pi)
```

Desempaquetar listas {

Es el proceso de **asignar los valores de una lista a variables individuales**.



```
lista = ["Enero", "Jueves", 3.14]

mes, dia_semana, pi = lista

print(mes) # "Enero"
print(dia_semana) # "Jueves"
print(pi) # 3.14
```

Recorrer tuplas {

Podemos recorrer una tupla usando el **ciclo for**, tal como hicimos con las listas.



```
tupla = ("Enero", "Jueves", 3.14)
```

```
for valor in tupla:  
    print(valor)
```

Enero
Jueves
3.14

Recorrer tuplas {

Podemos recorrer una tupla usando el **ciclo for**, tal como hicimos con las listas pero ahora utilizando **range()**.



```
semana = ("Lunes", "Martes", "Miercoles", "Jueves", "Viernes")
```

```
for i in range(len(tupla)): # Desde 0 hasta tamaño - 1  
    print(semana[i]) # Accedemos a cada elemento con los indices
```

Lunes
Martes
Miercoles
Jueves
Viernes

}

Recorrer tuplas {

Podemos recorrer una tupla usando el **ciclo while** utilizando **len()**.



```
semana = ("Lunes", "Martes", "Miercoles", "Jueves", "Viernes")

i = 0 # indice actual

while i < len(semana): # Mientras indice sea menor al tamaño de la tupla
    print(semana[i])
    i += 1 # Aumentamos el indice para no causar un bucle infinito!!!
```

Lunes
Martes
Miercoles
Jueves
Viernes

Unión de tuplas {

Podemos usar el **operador de suma +** para unir tuplas:



```
semana = ("Lunes", "Martes", "Miercoles", "Jueves", "Viernes")
fin_de_semana = ("Sabado", "Domingo")

semana_completa = semana + fin_de_semana

print(semana_completa)
# ('Lunes', 'Martes', 'Miercoles', 'Jueves', 'Viernes', 'Sabado', 'Domingo')
```

Métodos de tuplas {

En Python, las tuplas son una estructura de datos **poco flexible**, por lo cual tiene una pequeña cantidad de métodos integrados para realizar operaciones.

}

Método `.count()` {

El método `count()` devuelve el **número de veces que aparece un valor especificado** en la tupla.



```
tupla = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
```

```
apariciones = tupla.count(7)
```

```
print(apariciones) # 2
```

}

Método `.index()` {

El método `index()` devuelve la **posición del valor especificado** que haya sido encontrado **primero** en la tupla.



```
tupla = (1, 3, 7, 8, 7, 5, 4, 6, 8, 5)
```

```
posicion = tupla.index(5)
```

```
print(posicion) # Posicion 5, no toma en cuenta el 5 en la posicion 9
```

Operadores de pertenencia {

in y **not in** son operadores de pertenencia que se utilizan en Python para verificar **si un elemento está o no en una estructura iterable.**

}

Operador in {



```
semana = ("Lunes", "Martes", "Miercoles", "Jueves", "Viernes")

if "Domingo" in semana:
    print("Domingo esta incluido en la tupla semana")
else:
    print("Domingo no esta incluido :(")
```

}

Operador not in {



```
semana = ("Lunes", "Martes", "Miercoles", "Jueves", "Viernes")

if "Lunes" not in semana:
    print("Lunes no esta incluido en la tupla")
else:
    print("Domingo si esta incluido en la tupla :D ")
```

}

Ejercicio 13 {

Se le pide desarrollar un programa que, dada una tupla de opciones, le pida al usuario la opción que desea realizar y que mientras la opcion ingresada no se encuentre en la tupla, siga solicitando que ingrese una opción.

}

Sets {

En Python, un **set** es una **colección no ordenada de objetos únicos**.

Los **conjuntos** se utilizan para **almacenar datos que no deben repetirse**, como las letras del alfabeto.

Los conjuntos se pueden crear utilizando **llaves {}**.



```
set = {"rojo", "verde", "azul"}
```

}

Características de los sets {

1. **Desordenados:** Los elementos de un conjunto **no tienen un orden definido**. Los elementos establecidos pueden aparecer en un orden diferente cada vez que los usa y no se puede hacer referencia a ellos por índice o clave.
2. **Inmutables:** Los elementos del conjunto **no se pueden cambiar**, lo que significa que no podemos cambiar los elementos después de que se haya creado el conjunto.
3. **Heterogeneas:** Pueden guardar múltiples tipos de datos al mismo tiempo.
4. **No permiten duplicados:** Los conjuntos no pueden tener dos elementos que sean iguales.

}

Advertencia {

En los sets los valores **True** y **1**, se consideran el mismo **valor** en conjuntos y se tratan como **duplicados**. Lo mismo ocurre con **False** y **0**.



```
set = {True, 1, False, 0}

print(set) # {False, True}

set2 = {1, True, 0, False}

print(set2) # {0, 1}
```

Recorrer sets {

Es importante tener en cuenta que los conjuntos **no están ordenados**, por lo que no se puede garantizar que los valores se devuelvan en el mismo orden en el que se agregaron.

Por lo tanto, lo recomendable es utilizar el **ciclo for** para recorrerlos.



```
set = {"rojo", "verde", "azul"}
```

```
for color in set:  
    print(color)
```

Pregunta{

Se debe tener en cuenta que los **sets** son **estructuras no ordenadas**. Teniendo esto en cuenta:

Si ejecutamos el siguiente código, ¿que cree que ocurriría?



```
set = {"rojo", "verde", "azul"}  
print(set[0])
```

Funcion len() {

La función len() se utiliza para obtener **la longitud de una estructura**, es decir, el **número de elementos** que contiene la estructura.



```
set = {"rojo", "verde", "azul"}
```

```
print(len(set)) # 3
```

}

Operador in {



```
numeros = {1, 2, 3, 4}
```

```
if 2 in numeros:
```

```
    print("El numero 2 existe en el conjunto.")
```

```
else:
```

```
    print("El numero 2 no pertenece al conjunto")
```

```
}
```

Operador not in {



```
colores = {"rojo", "verde", "azul"}
```


```
if "rosado" not in colores:  
    print("Hay escasez de rosado por la pelicula de Barbie.")  
else:  
    print("Queda rosado en el conjunto de colores.")
```

```
}
```


Metodo .add() {

Una vez que se crea un conjunto, no puede cambiar sus elementos, pero puede agregar nuevos elementos.

Para agregar un elemento a un conjunto, podemos utilizar el **método add()**.



```
numeros = {1, 2, 3, 4}
```

```
numeros.add(5)
```

```
print(numeros)
```

Eliminar elemento {

Para eliminar un elemento de un conjunto, podemos utilizar el método **remove()** o **discard()**.



```
numeros = {1, 2, 3, 4}
```

```
numeros.remove(3)
```

```
print(numeros) # {1, 2, 3}
```

Ejercicio 14 {

Se le pide desarrollar un programa que, dado un conjunto(set) de concursantes, escoja tres nombres aleatorios para ser los **ganadores de un sorteo**.

```
concursantes = {"Maria", "Luis", "Paola", "Antonio", "Laura"}
```

```
}
```

Diccionarios {

En Python, un diccionario es una **estructura de datos** que almacena datos en pares de **llave:valor**, en donde **a cada dato se le asigna una llave para acceder a este.**

Las claves pueden ser de cualquier tipo de objeto, y los valores también pueden ser de cualquier tipo de objeto.

Los diccionarios **se utilizan para almacenar datos que están relacionados entre sí**, como los nombres de los estados y sus capitales, o los nombres de las personas y sus cédulas.

}

Diccionarios {

Los diccionarios se pueden crear utilizando **llaves {}**.



```
carro = {  
    "marca": "Ford",  
    "modelo": "Mustang",  
    "anio": 1964  
}  
print(carro)  
# {'marca': 'Ford', 'modelo': 'Mustang', 'anio': 1964}
```

Características de los diccionarios {

1. **Mutable:** Podemos cambiar, agregar o eliminar elementos después de que se haya creado el diccionario..
2. **Heterogeneas:** Pueden guardar múltiples tipos de datos al mismo tiempo.
3. **No permiten duplicados:** Los diccionarios **no pueden tener dos elementos con la misma clave.**

}

Acceder a un dato {

Para acceder a un elemento de un diccionario en Python, debemos colocar el **nombre del diccionario** seguido de **corchetes []**, y dentro de estos tenemos que colocar la **llave** del dato al cual queremos acceder.

Si intentamos acceder a un valor que no existe, entonces obtendremos un **error**.



`diccionario[llave]`

Acceder a un dato {



```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 20,  
}  
  
nombre = persona["nombre"]  
apellido = persona["apellido"]  
print(nombre + " " + apellido) # Daniela Perez
```

}

Metodo .get() {

El método **.get()** en diccionarios en Python **devuelve el valor asociado a una clave determinada.**

Si la clave no existe en el diccionario, el método **.get()** devuelve el valor predeterminado especificado.

De esta forma, evitamos el error al intentar acceder a un elemento que no existe.

}

Metodo .get() {

```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 20,  
}  
  
nombre = persona.get("nombre")  
print(nombre) # Daniela  
  
cedula = persona.get("cedula")  
print(cedula) # None, porque no existe
```

Funcion len() {

La funcion **len()** con diccionarios en Python **nos devolverá la cantidad de pares llave:valor.**



```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 20,  
}
```

```
print(len(persona)) # 3, nos indica el numero de pares llave:valor
```

}

Agregar par llave:valor {

Para agregar un par **llave:valor** al diccionarios tenemos que colocar el nombre del diccionario seguido de **corchetes []** que contengan dentro la llave, e igualar esto a el valor deseado.



```
diccionario[nueva_llave] = nuevo_valor
```

}

Agregar par llave:valor {

```
1
2
3
4
5 pais = {
6     "nombre": "Venezuela",
7     "capital": "Caracas",
8     "poblacion": 32075994,
9     "superficie": 916445
10 }
11
12
13
14 pais["idioma"] = "español" # Agregamos un nuevo par llave:valor
print(pais)
```

```
{
  "nombre": "Venezuela",
  "capital": "Caracas",
  "poblacion": 32075994,
  "superficie": 916445,
  "idioma": "español"
}
```

Modificar par llave:valor {

Seria igual a lo que hicimos para agregar, solo que en este caso **la llave especificada debe existir**.

```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 26,  
}  
persona["nombre"] = "Paola" # Modificamos  
print(persona["nombre"]) # Paola
```

Eliminar par llave:valor {

Para borrar un par **key:value** de un diccionario disponemos de la **sentencia del**.

Debemos colocar primero la **sentencia del**, seguido del nombre del diccionario, se abre **corchetes []** y dentro de estos se coloca la llave del par a eliminar.



```
del diccionario[llave]
```

}

Eliminar par llave:valor {



```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 26,  
}  
  
del persona["edad"] # Eliminamos el par con llave edad  
  
print(persona)  
  
"""  
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez"  
}  
"""
```


Recorrer diccionarios {

Para recorrer diccionarios debemos utilizar el **ciclo for**.

```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 26,  
}  
  
for llave in persona:  
    print(llave)
```

nombre
apellido
edad

Recorrer diccionarios {

Si observamos lo que acabamos de obtener con el **ciclo for**, nos daremos cuenta de que **solo se imprimieron las llaves.**

Entonces, accederemos a los valores de la siguiente manera:

}

Recorrer diccionarios {



```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 26,  
}  
  
for llave in persona:  
    valor = persona[llave]  
    print(llave + ": " + str(valor))
```

```
nombre: Daniela  
apellido: Perez  
edad: 26
```

}

Metodo .keys() {



```
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 20,  
}  
llaves = persona.keys()  
print(llaves) # dict_keys(['nombre', 'apellido', 'edad'])
```

}

Metodo .keys() {

Sin embargo, se debe tener lo siguiente en cuenta:

```
• • •  
  
persona = {  
    "nombre": "Daniela",  
    "apellido": "Perez",  
    "edad": 20,  
}  
llaves = persona.keys()  
print(llaves) # dict_keys(['nombre', 'apellido', 'edad'])  
print(type(llaves)) # El tipo de dato es dict_keys. No es una lista.  
  
# Debemos castear a lista  
llaves = list(llaves)  
print(llaves) # ['nombre', 'apellido', 'edad']
```

Metodo .keys() {

```
● ● ●

persona = {
    "nombre": "Daniela",
    "apellido": "Perez",
    "edad": 26,
}
llaves = persona.keys() # Secuencia de llaves
lista_llaves = list(persona.keys()) # Casteamos a lista

print("ANTES")
print(llaves) # dict_keys(['nombre', 'apellido', 'edad'])
print(lista_llaves) # ['nombre', 'apellido', 'edad']

# Agregamos otro elemento
persona["cedula"] = 25763954

print("DESPUES")
print(llaves) # dict_keys(['nombre', 'apellido', 'edad', 'cedula'])
print(lista_llaves) # ['nombre', 'apellido', 'edad']
```

Ejercicio 15 {

Se le pide que dado un diccionario de productos, realice un programa que indique el producto con mayor precio.

```
productos = {  
    "Leche": 1.50,  
    "Pan": 0.50,  
    "Huevos": 0.75,  
    "Queso": 2.00,  
    "Fruta": 1.00  
}
```



ESTRUCTURAS COMBINADAS

Estructuras complejas

03 {

[Estructuras combinadas]

- Resumen
- Lista de Listas.
- Lista de Diccionarios

}

Resumen {

EDD	Ordenada	Mutable	Constructor	Ejemplo
Lista	SI	SI	[]	["Hola", 10]
Tupla	SI	NO	()	(1, 3.14, "N")
Set	NO	SI	{ }*	{1, 2, 3}
Diccionario	NO	NO**	{ }	{"a": 10, "b": -6}

}

Lista de listas {



```
lista_de_estudiantes = [  
    ['Juan', 27, 'Calle 123, Ciudad'],  
    ['María', 29, 'Calle 456, Pueblo']  
]
```

}

Lista de listas {

Las listas de listas también **se pueden usar para representar matrices.**

Por ejemplo, si está almacenando datos sobre una matriz de números, podría usar una **lista de listas para almacenar los números de la matriz.**

La lista principal contendría una lista para cada fila de la matriz, y cada lista dentro de la lista principal contendría los números de una fila de la matriz.

}

Listas de listas {



```
matriz = [  
    [1, 2, 3, 4, 5],  
    [6, 7, 8, 9, 10],  
    [11, 12, 13, 14, 15],  
    [16, 17, 18, 19, 20],  
    [21, 22, 23, 24, 25]  
]
```

}

Recorrer lista de listas {

Para recorrer una lista de listas en Python, podemos usar un **bucle for anidado** (dentro).

El bucle exterior iterará sobre las listas principales, y el bucle interior iterará sobre las listas anidadas.

Se recomienda primero imprimir cada elemento al cual accedemos con el primer **ciclo for**, antes de usar el **bucle for anidado**.

}

Recorrer lista de listas {

```
lista_de_estudiantes = [  
    ['Juan', 27, 'Calle 123, Ciudad'],  
  
    ['María', 29, 'Calle 456, Pueblo']  
]  
  
for estudiante in lista_de_estudiantes:  
    print(estudiante)
```

```
['Juan', 27, 'Calle 123, Ciudad']  
['María', 29, 'Calle 456, Pueblo']
```

Recorrer lista de listas {

Visualizando lo obtenido, podemos notar que lo que se imprimió por cada elemento fue una **lista**.

Ahora sí entendiendo esto, podemos intentar iterar (recorrer) sobre cada una de estas **listas internas**.

}

Recorrer lista de listas {

```
lista_de_estudiantes = [  
    ['Juan', 27, 'Calle 123, Ciudad'],  
    ['María', 29, 'Calle 456, Pueblo']  
]  
  
for estudiante in lista_de_estudiantes:  
    for elemento in estudiante: # estudiante representa la lista interna  
        print(elemento)
```

```
Juan  
27  
Calle 123, Ciudad  
María  
29  
Calle 456, Pueblo
```

Recorrer lista de listas {

Otra forma sería utilizar **índices** para recorrer las **listas internas**.

}

Recorrer listas de listas {

```
lista_de_estudiantes = [  
    ['Juan', 27, 'Calle 123, Ciudad'],  
  
    ['María', 29, 'Calle 456, Pueblo']  
]  
contador = 1  
for estudiante in lista_de_estudiantes:  
    nombre = estudiante[0]  
    edad = estudiante[1]  
    direccion = estudiante[2]  
    print(f"""  
    Estudiante Nro {contador}  
        Nombre: {nombre}  
        Edad: {edad}  
        Direccion: {direccion}  
    """)  
    contador += 1  
    edad = estudiante[1]  
    direccion = estudiante[2]
```



Estudiante Nro 1
Nombre: Juan
Edad: 27
Direccion: Calle 123, Ciudad

Estudiante Nro 2
Nombre: María
Edad: 29
Direccion: Calle 456, Pueblo

Lista de diccionarios {

En Python, una lista de diccionarios es una **lista que contiene diccionarios**. Cada elemento de la lista es un diccionario en sí mismo.

Las listas de diccionarios se pueden usar para almacenar datos organizados en una forma jerárquica.

Por ejemplo, si se está almacenando datos sobre estudiantes, se podría usar una lista de diccionarios para **almacenar la información de cada estudiante**. La lista principal contendría un diccionario para cada estudiante, y cada diccionario dentro de la lista principal contendría la información del estudiante, como su nombre, edad, dirección y notas.

}

Lista de diccionarios {

```
lista_de_estudiantes = [  
    {  
        'nombre': 'Juan',  
        'edad': 20,  
        'dirección':  
        'Calle 123, Ciudad',  
        'notas': [9, 8, 7, 6, 5]  
    },  
    {  
        'nombre': 'María',  
        'edad': 21,  
        'dirección': 'Calle 456, Pueblo',  
        'notas': [8, 9, 10, 10, 10]  
    }  
]
```

Recorrer lista de diccionarios {

Debemos evaluar cuál es la estructura externa y según esto determinaremos cómo empezar a recorrer. En este caso, es una **lista**.

Entonces, primero **iteraremos** sobre la lista para observar que hay en cada posición de la misma.

}

Recorrer lista de diccionarios {

```
lista_de_estudiantes = [  
    {  
        'nombre': 'Juan',  
        'edad': 20,  
        'dirección':  
        'Calle 123, Ciudad',  
        'notas': [9, 8, 7, 6, 5]  
    },  
    {  
        'nombre': 'María',  
        'edad': 21,  
        'dirección': 'Calle 456, Pueblo',  
        'notas': [8, 9, 10, 10, 10]  
    }  
]  
  
for estudiante in lista_de_estudiantes:  
    print(estudiante)
```

Recorrer lista de diccionarios {



```
{'nombre': 'Juan', 'edad': 20, 'dirección': 'Calle 123, Ciudad', 'notas': [9, 8, 7, 6, 5]  
{'nombre': 'María', 'edad': 21, 'dirección': 'Calle 456, Pueblo', 'notas': [8, 9, 10, 10, 10]}
```

}

Recorrer lista de diccionarios {

Ahora que podemos acceder a cada diccionario en la lista, podemos imprimir cada par **llave:valor**.

}

Recorrer lista de diccionarios {

```
lista_de_estudiantes = [
    {
        'nombre': 'Juan',
        'edad': 20,
        'dirección':
            'Calle 123, Ciudad',
        'notas': [9, 8, 7, 6, 5]
    },
    {
        'nombre': 'María',
        'edad': 21,
        'dirección': 'Calle 456, Pueblo',
        'notas': [8, 9, 10, 10, 10]
    }
]
contador = 1
for estudiante in lista_de_estudiantes:
    print("Estudiante Nro ", contador)
    contador += 1
    llaves = list(estudiante.keys())
    for llave in llaves:
        print(llave, ": ", estudiante[llave])
```

```
Estudiante Nro  1
nombre : Juan
edad : 20
dirección : Calle 123, Ciudad
notas : [9, 8, 7, 6, 5]
Estudiante Nro  2
nombre : María
edad : 21
dirección : Calle 456, Pueblo
notas : [8, 9, 10, 10, 10]
```

Recorrer lista de diccionarios {

¿Cómo podemos recorrer las notas de
cada estudiante?

}

Recorrer lista de diccionarios {

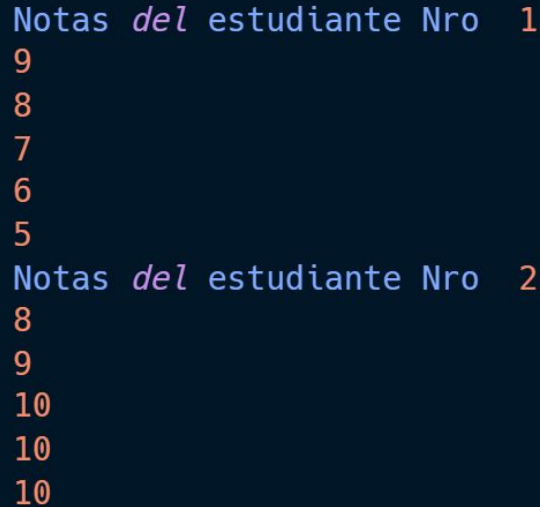


```
contador = 1
for estudiante in lista_de_estudiantes:
    print("Notas del estudiante Nro ", contador)
    contador += 1
    notas = estudiante["notas"]
    for nota in notas:
        print(nota)
```

}

Recorrer lista de diccionarios {

}

A terminal window with a dark blue background and three colored window control buttons (red, yellow, green) at the top left. It displays the output of a program that iterates through a list of dictionaries. The output shows the grades for two students, with the first student having grades [9, 8, 7, 6, 5] and the second student having grades [8, 9, 10, 10, 10].

```
Notas del estudiante Nro 1
9
8
7
6
5
Notas del estudiante Nro 2
8
9
10
10
10
```

Acceder a elementos lista de diccionarios {

```
lista_de_estudiantes = [  
    {  
        'nombre': 'Juan',  
        'edad': 20,  
        'dirección':  
        'Calle 123, Ciudad',  
        'notas': [9, 8, 7, 6, 5]  
    },  
    {  
        'nombre': 'María',  
        'edad': 21,  
        'dirección': 'Calle 456, Pueblo',  
        'notas': [8, 9, 10, 10, 10]  
    }  
]  
  
print(lista_de_estudiantes[0]["notas"]) # [9, 8, 7, 6, 5]
```

Modificar elementos lista de diccionarios {

```
lista_de_estudiantes = [
    {
        'nombre': 'Juan',
        'edad': 20,
        'dirección':
            'Calle 123, Ciudad',
        'notas': [9, 8, 7, 6, 5]
    },
    {
        'nombre': 'María',
        'edad': 21,
        'dirección': 'Calle 456, Pueblo',
        'notas': [8, 9, 10, 10, 10]
    }
]

print(lista_de_estudiantes[0]["notas"]) # [8, 9, 10, 10, 10]

lista_de_estudiantes[0]["notas"] = [18, 19, 20, 20, 20]

print(lista_de_estudiantes[0]["notas"]) # [18, 19, 20, 20, 20]
```

Diccionario de diccionarios {

Un diccionario puede tener diccionarios más pequeños dentro de sí mismo.

Un diccionario de diccionarios es un tipo de estructura de datos que puede almacenar un **conjunto de diccionarios**. Cada diccionario puede contener sus propias **claves y valores**.

Los diccionarios de diccionarios también se pueden usar para representar **tablas**. Por ejemplo, si está almacenando datos sobre una tabla de números, podría usar un diccionario de diccionarios para almacenar los números de la tabla. El diccionario principal contendría un diccionario para cada fila de la tabla, y cada diccionario dentro del diccionario principal contendría los números de una fila de la tabla.

}

Diccionario de diccionarios {

Un diccionario puede tener diccionarios más pequeños dentro de sí mismo.

Un diccionario de diccionarios es un tipo de estructura de datos que puede almacenar un conjunto de diccionarios. Cada diccionario en un diccionario de diccionarios puede contener sus propias claves y valores.

}

Diccionario de diccionarios {

```
personas = {  
    "Juan": {  
        "nombre": "Juan",  
        "apellido": "Pérez",  
        "edad": 20,  
        "sexo": "masculino",  
        "dirección": "Calle 123, Ciudad"  
    },  
    "María": {  
        "nombre": "María",  
        "apellido": "López",  
        "edad": 21,  
        "sexo": "femenino",  
        "dirección": "Calle 456, Pueblo"  
    }  
}
```

Diccionario de diccionarios {

```
tabla = {  
    "fila1": {  
        "columna1": 1,  
        "columna2": 2,  
        "columna3": 3  
    },  
    "fila2": {  
        "columna1": 4,  
        "columna2": 5,  
        "columna3": 6  
    },  
    "fila3": {  
        "columna1": 7,  
        "columna2": 8,  
        "columna3": 9  
    }  
}
```

Diccionario de diccionarios {

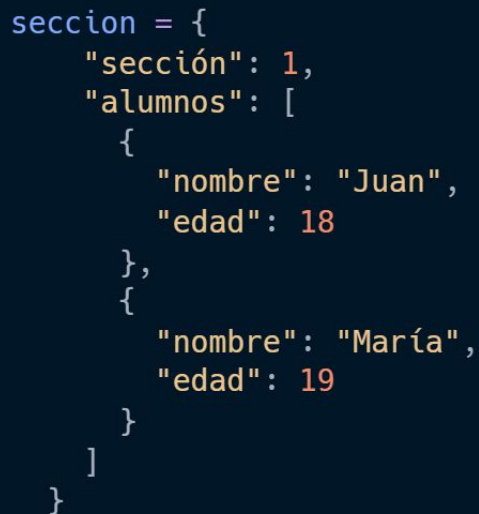
Asi mismo, un diccionario puede contener entre sus elementos **listas**, y estas listas a su vez pueden contener otros diccionarios.

Si bien estas estructuras **pueden crecer en complejidad**, la forma de acceder a cada uno de los elementos siguen siendo las mismas.

}

Diccionario de diccionarios {

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12  
13 }  
14
```



```
seccion = {  
    "sección": 1,  
    "alumnos": [  
        {  
            "nombre": "Juan",  
            "edad": 18  
        },  
        {  
            "nombre": "María",  
            "edad": 19  
        }  
    ]  
}
```

Ejercicio 16 {

Se tiene una **lista de diccionarios** con los datos de los estudiantes de múltiples secciones, escriba un programa que pida al usuario un **número de sección** e imprima los nombres de todos los alumnos que están en dicha sección.

}

Ejercicio 16 {

```
1  
2  
3  
4  
5  
6  
7  
8  
9  
10  
11  
12
```

```
}  
14
```

```
estudiantes = [  
    {  
        "sección": 1,  
        "alumnos": [  
            {  
                "nombre": "Juan",  
                "edad": 18  
            },  
            {  
                "nombre": "María",  
                "edad": 19  
            }  
        ]  
    },  
    {  
        "sección": 2,  
        "alumnos": [  
            {  
                "nombre": "Pedro",  
                "edad": 20  
            },  
            {  
                "nombre": "Susana",  
                "edad": 21  
            }  
        ]  
    }  
]
```

Ejercicio 17 {

Se tiene un **diccionario** que contiene una **lista de diccionarios con los datos de estudiantes**, elabore un programa que permita introducir a cada diccionario de estudiante una nota introducida por el usuario, debe validar que el input sea un número entre 1 y 20, una vez se hayan introducido todas las notas a los diccionarios, se imprime la lista entera.

}

Ejercicio 17 {

}

```
lista_estudiantes = {
    'estudiantes': [
        {
            'name': 'Paola',
            'seccion': '10',
            'cedula': '23463643',
        },
        {
            'name': 'Juan',
            'seccion': '3',
            'cedula': '23463653',
        },
        {
            'name': 'Javier',
            'seccion': '10',
            'cedula': '23679643',
        },
        {
            'name': 'Maria',
            'seccion': '10',
            'cedula': '23463643',
        },
    ],
    'profesor': {
        'nombre': 'Stefani',
        'seccion': '3',
        'materia': "Algoritmos y Programacion"
    },
}
```