El futuro digital es de todos

MinTIC

# ESTRUCTURA DE DATOS E INTERFACES

Hechos QUE CONECTAN

Universidad Industrial de Santander

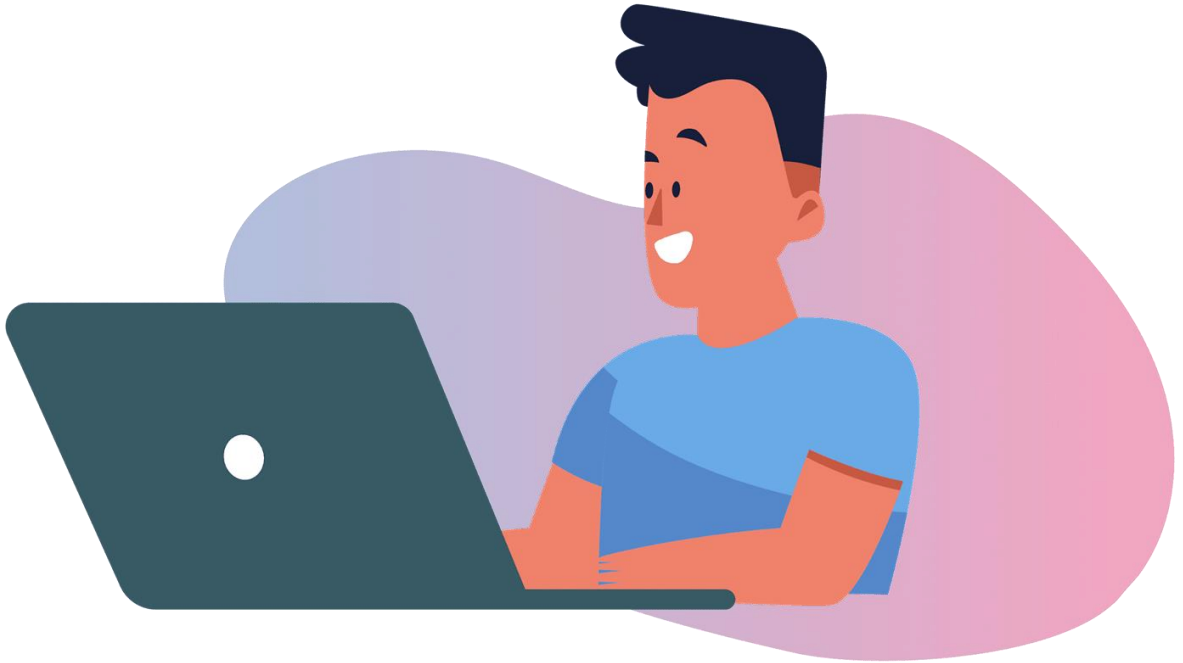
UIS

MisionTIC 2022

# 5.1. VARIABLES ESTRUCTURALES DE PYTHON

## 5.1.1 Creación de una lista en Python

Para crear una lista en Python, se nombra la variable y se asigna con el signo igual el conjunto de datos que conforman la lista, **separados por comas y encerrados entre corchetes**.



Ejemplo:

```
dato_v = [10, 20, 30, 40]

dato_x = [“Juan”, “Pedro”, “José”, “Silvia”]

dato_s = [“Camisas”, “Pantalones”, “Zapatos”, “Corbata”]
```

La lista `dato_s` contiene ítems de tipo texto o cadena de caracteres (string). Es por esto que cada dato de tipo string **debe ir entre comillas.**

**Es posible crear listas vacías sin especificar sus elementos**, ya que al ser mutables sus elementos o ítems pueden ser agregados posteriormente.

Para crear una lista vacía, la debemos declarar de la misma forma pero sin elementos, siempre teniendo en cuenta la sintaxis de los **corchetes.**

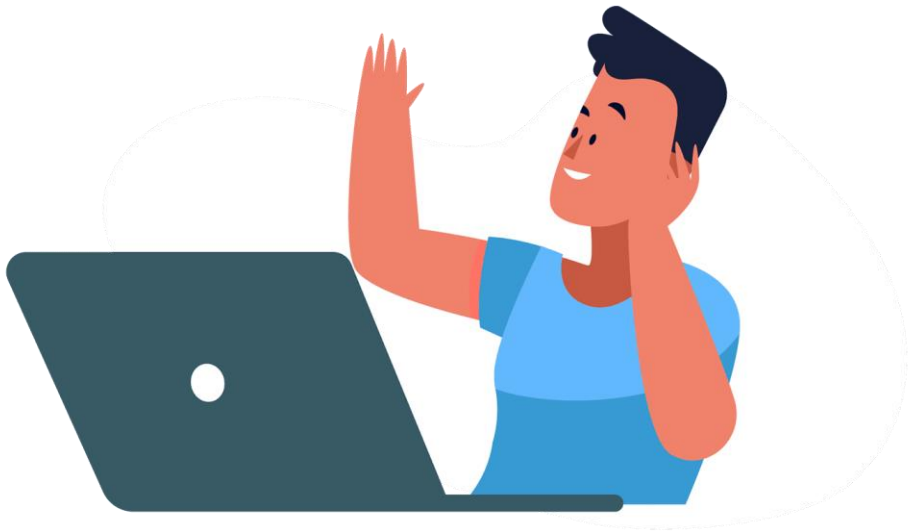
```
lista_vacia = []
```

# 5.1.2 Creación de una tupla en Python

A diferencia de las listas, las tuplas usan **paréntesis** para definir sus elementos. A continuación, se dan dos ejemplos que muestran la creación de tuplas.

```
Cosas = ("Apartamento", "Ascensor", "Batería", "Libro", "Alfombra")

Numeros = (10, 20, 30, 40, 50)
```



# 5.1.3 Accediendo a los elementos de listas y tuplas

La lista:

```
Objetos = ["Celular", "Vehículo", "Escritorio"]
```

se puede representar con sus posiciones así:

| Objetos = | [ “Celular”, | “Vehículo”, | “Escritorio” ] |
|---|---|---|---|
| | **0** | **1** | **2** |

En el ejemplo anterior tenemos **tres** elementos indexados: **0, 1, 2**

```
misValores= [1, 22, 333, 4444, 55555]
```

| misValores= | [ 1, | 22, | 333, | 4444, | 55555 ] |
|---|---|---|---|---|---|
| | **0** | **1** | **2** | **3** | **4** |

En el ejemplo anterior tenemos **cinco** elementos indexados: **0, 1, 2, 3, 4**

Entonces si se desea acceder a una posición de la Lista "`misValores`", se debe especificar la lista y seguido entre corchetes cuadrados el número de la posición que se requiere acceder:

```
print( misValores[1] )
```

**Resultado: 22**

En la otra lista, llamada “Objetos”, se procede de la misma forma para obtener el string “Celular”.

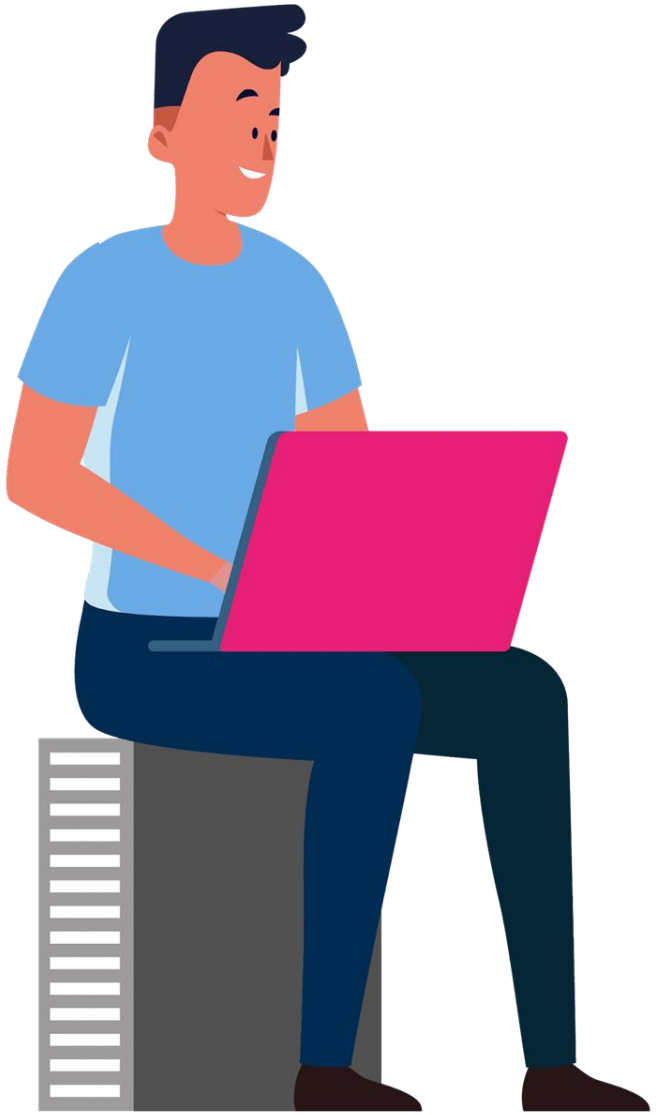
```
print( Objetos[0] )
```

**Resultado: Celular**

Si se requiriera hacer el mismo proceso para la estructura de tuplas, se realizaría de la misma manera, **usando para el índice, corchetes cuadrados**

```
print( Cosas[2] )
```

**Resultado: Batería**



Ya que se decidió obtener el ítem del índice 2, nos debe mostrar el string “Batería”.

```
print( misValores[2])
```

**Resultado: 333**

Como resultado, se imprime el número 333 que corresponde al contenido del índice 2. Recuerda que el indexamiento en Python empieza con CERO.

Como se pudo observar en el momento de acceder a cualquiera de las dos estructuras, **ya sean listas o tuplas, se deben usar corchetes cuadrados**. Solo utilizamos paréntesis cuando se crea una tupla.

Es importante aclarar que,  si se intenta acceder a un índice mayor al último índice del último elemento, ocurrirá un error.

# 5.1.4 Acceder a más de un elemento en tuplas y listas

Para acceder a más de un elemento se debe especificar, entre corchetes, el **índice de inicio y el índice final separados por dos puntos.**

Los elementos obtenidos de la lista serán los correspondientes **desde el índice de inicio hasta el índice final menos 1.**

Ejemplos:

```
lista = [1, 2.5, 'DevCode', [5,6] ,4]

print (lista[1:3])                  # devuelve [2.5, 'DevCode']

print (lista[1:6])                  # devuelve [2.5, 'DevCode', [5, 6], 4]

print (lista[1:6:2])              # devuelve [2.5, [5, 6]]

print (lista[::-1])               # devuelve la lista invertida
```

# 5.1.5 Recorrer una Lista o una Tupla

Los elementos de una Lista o de una Tupla, pueden obtenerse utilizando una instrucción iterativa **for**.

```
lista = [1, 2.5, 'Enya Isabel', [5,6] ,99]

for element in lista:

    print (element)
```

La salida del anterior segmento de código es:

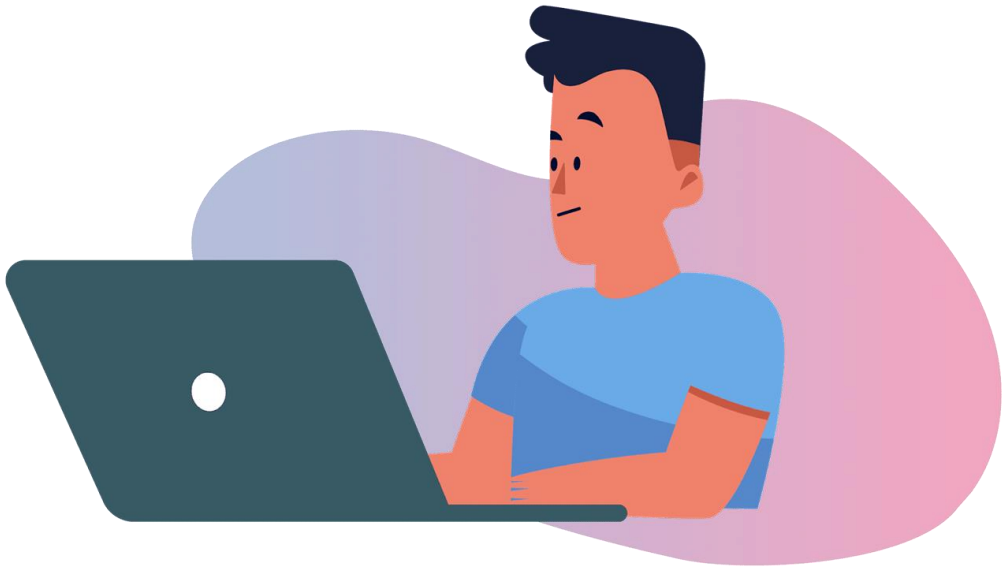
```
1

2.5

"Enya Isabel"

[5,6]

99
```



# 5.1.6 Métodos para modificar Listas en tiempo de ejecución

**El método append** permite añadir un ítem al final de una lista en Python. Es necesario crear una lista antes de modificarla.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']

print(Nombres)      # Resultado ['David', 'Elkin', 'Juliana', 'Silvia']

Nombres.append ('Ivon')

print(Nombres) # Resultado ['David', 'Elkin', 'Juliana', 'Silvia', 'Ivon']
```

**El método extend** se utiliza para agregar elementos iterables como un *string* **u otra lista** con sus elementos uno a uno.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']

print(Nombres)

# Resultado ['David', 'Elkin', 'Juliana', 'Silvia']

Nombres.extend('MARIA')

print(Nombres)

# Resultado ['David', 'Elkin', 'Juliana', 'Silvia','M','A','R','I','A']
```

Un ejemplo de agregar una lista a otra lista:

```
Numeros1 = [1, 2, 3]

Numeros2 = [4, 5, 6]

Numeros1.extend(Numeros2) #Añadimos la lista Numeros2 como extensión de la lista Numeros1

print(Numeros1) # Resultado [1, 2, 3, 4, 5, 6]
```

**El método insert** permite añadir un elemento en una posición o índice específico.

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']

print(Nombres)    # Resultado ['David', 'Elkin', 'Juliana', 'Silvia']

Nombres.insert(1, 'Ivon')

print(Nombres) #Resultado ['David', 'Ivon', 'Elkin', 'Juliana', 'Silvia']
```

**Primera posición (0):**

```
l = [1,2,3]

l.insert(0,0)      # Resultado [0, 1, 2, 3]
```

**Penúltima posición (-1):**

```
l = [5,10,15,25]

l.insert(-1,20) # Resultado [5, 10, 15, 20, 25]
```

**Última posición en una lista con len():**

```
n = len(l)

l.insert(n,30) # Resultado [5, 10, 15, 20, 25, 30]
```

**Una posición fuera de rango** añade el elemento al final de la lista:

```
l.insert(999, 35) # Resultado [5, 10, 15, 20, 25, 30, 35]
```

**El método pop** quita un elemento de la lista dado su índice

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']

print(Nombres) #Resultado ['David', 'Elkin', 'Juliana', 'Silvia']

Nombres.pop(1)

print(Nombres) #Resultado ['David', 'Juliana', 'Silvia']
```

En este ejemplo, el elemento con índice 1 (‘Elkin’) se elimina de la lista. Si no se indica el índice, se eliminará el último elemento de la lista.

**El método remove** quita un elemento de la lista nombrando el elemento mismo y no su índice

```
Nombres = ['David', 'Elkin', 'Juliana', 'Silvia']

Nombres.remove('Elkin')

print(Nombres)      #Resultado ['David', 'Juliana', 'Silvia']
```

Si el ítem a eliminar no se encuentra en la lista se mostrará un error.

# Material de estudio complementario

Links:

https://www.youtube.com/watch?v=MHvrJhshEU0

https://www.youtube.com/watch?v=xdCJa2QXmJ8

https://www.youtube.com/watch?v=0NTaCJQUE1I

# Referencias bibliográficas


- Pythones, "Listas, Tuplas y Range en Python 3: Similitudes y diferencias" [Online]. Recuperado de: *https://pythones.net/listas-tuplas-python/*.

# 5.2. CONJUNTOS

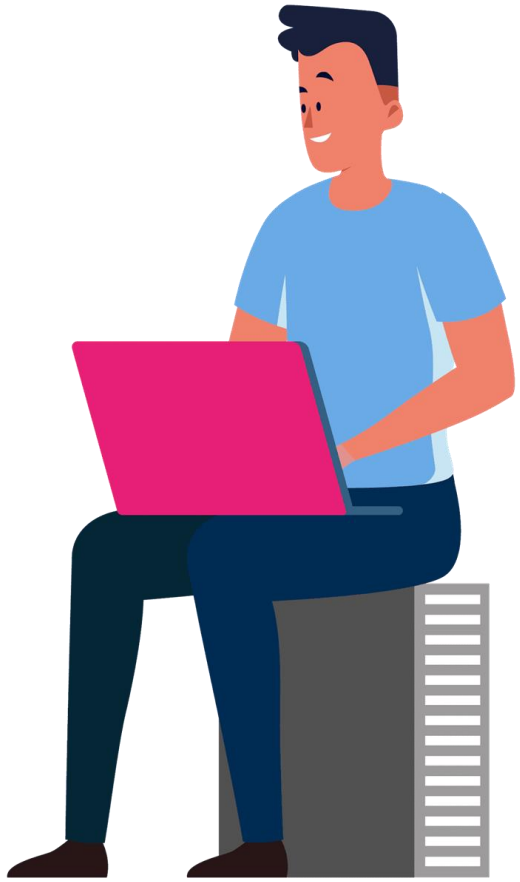
## 5.2.1 Creación de Conjuntos

```
mi_conjunto = {1, 2, 3}
print(mi_conjunto)# Resultado: {1, 2, 3}


mi_conjunto = {1.0, "Hello", (1, 2, 3)}
print(mi_conjunto) #Resultado: {1.0, (1, 2, 3), 'Hello'}


mi_conjunto = {1, 2, 3, 4, 3, 2}
print(mi_conjunto) # Resultado: {1, 2, 3, 4} (Se omiten elementos duplicados)


mi_conjunto = set([1, 2, 3, 2])
print(mi_conjunto) # Resultado: {1, 2, 3}
mi_conjunto = {1, 2, [3, 4]} # Mostrará un error
```



La última línea mostrará un error, ya que no podemos tener elementos mutables en un conjunto como lo es una lista. Es importante recordar que, un conjunto no tiene un orden interno; por lo tanto, las posiciones de los elementos no importan y pueden variar entre la definición y la impresión del conjunto.

# 5.3. DICCIONARIOS

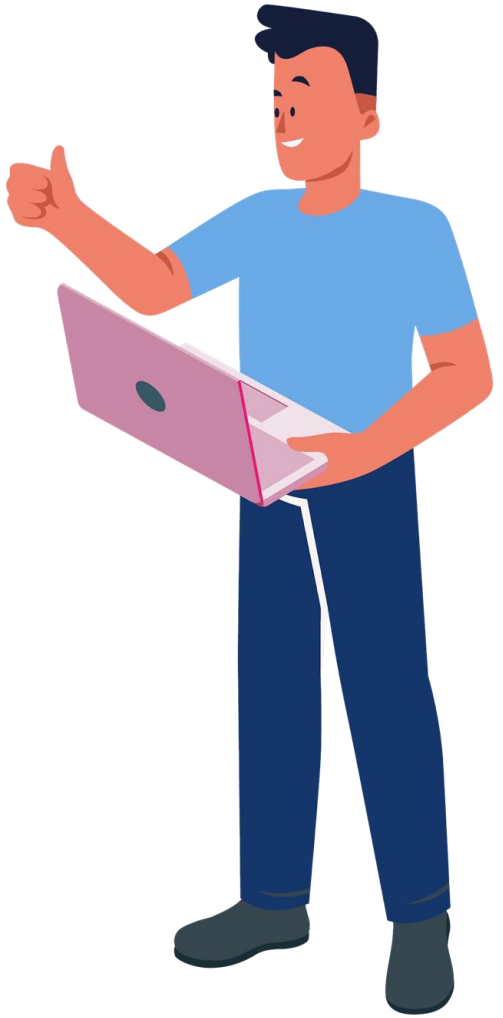
## 5.3.1 Creación de Diccionarios

Para crear un diccionario, empaquetamos entre llaves { } cada par de elementos **Clave:Valor** separadas por comas, así:

```
mi_diccionario = {'nombre': 'david', 'edad': 27, 'genero': 'masculino'}
```

Otro ejemplo:

```
mi_informacion = {
        'nombre' : 'David',
        'apellido' : 'Cortez',
        'sobrenombre' : 'DD',
        'padres' : ["Marín gomez", "Julián Cortez"],
        'edad' : 27,
        'genero' : 'masculino',
        'estado Civil' : 'Soltero',
        'hijos' : 2,
        'mascotas' : 'Perro',
        'nombres de mascotas' : ["chato"]
        }
```



# 5.3.2 Acceso a los elementos de un Diccionario

Es posible almacenar en un valor de una clave elementos complejos como listas. Una vez que almacenamos los datos en el diccionario, vamos a acceder a ellos.

```
print(mi_informacion['Apellido'])                              # Resultado: Cortez

print(mi_informacion.get('Apellido', “No tiene un apellido”))  #Resultado: Cortez

print(mi_informacion.get('celular', “No tiene un celular”))  #Resultado: No tiene un celular
```

Con el método get() de un diccionario, podemos obtener el valor de una clave, pero si no existe la clave devolverá un mensaje en *string* como respuesta.

# 5.3.3 Modificar Diccionarios

Para agregar valores nuevos a un diccionario, se debe realizar de la siguiente forma:

```
mi_informacion['salario'] = 200000
```

De esta forma, la clave será “salario” y el valor 200000.

Si ahora queremos modificar un valor ya existente, se debe realizar de la siguiente forma:

```
mi_informacion['edad'] = 38
```

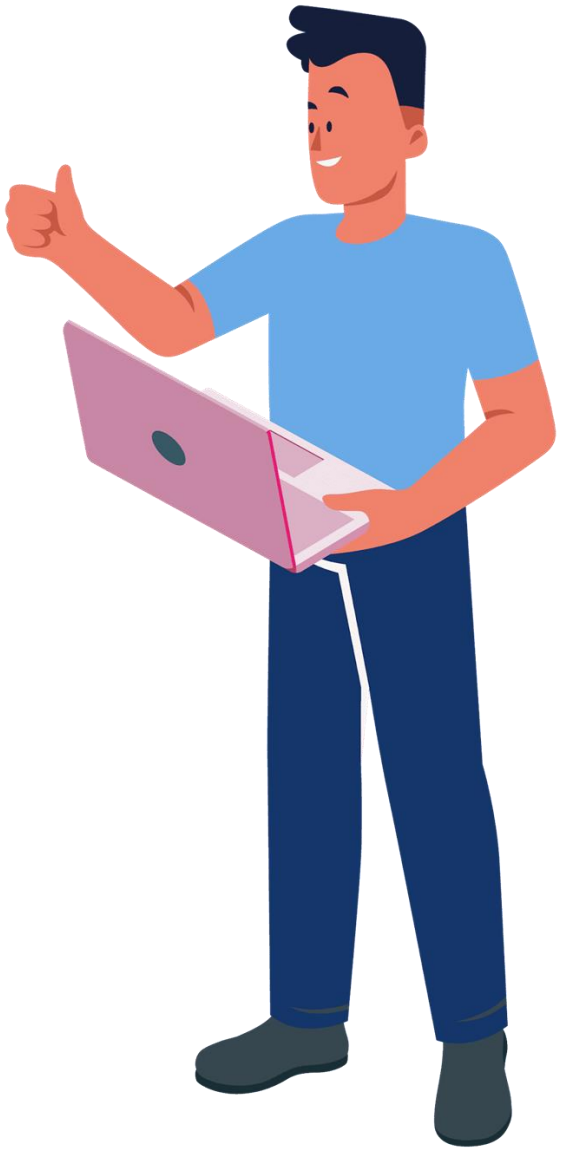
Al ser una clave existente, se modificará sólo el valor.

Si se requiere eliminar un elemento clave:valor se debe utilizar:

```
mi_informacion.pop('hijos')

del(mi_informacion['mascotas'])
```

En la primera línea, utilizamos pop() como una operación del diccionario, y en la segunda línea, usamos la función predefinida del(). De esta manera, se habrá eliminado las **claves** de hijos y mascotas con sus respectivos **valores**.



Para listar todas las claves que tiene un diccionario, podemos utilizar el método keys() de un diccionario en particular.

```
mi_diccionario = {'nombre': 'david', 'edad': 27, 'genero': 'masculino'}

  print(mi_diccionario.keys())     #Resultado:         dict_keys(['nombre', 'edad', 'genero'])
```

Finalmente, para listar todos los valores que tiene un diccionario podemos emplear el método values() de un diccionario en particular.

```
  print(mi_diccionario.values())    #Resultado:    dict_values(['david', 27, 'masculino'])
```

# Material de estudio complementario

Links:

- Programación desde cero. (17 de mayo del 2019). 13.1 Ejercicios resueltos con diccionarios - Curso de programación desde cero (con Python) [vídeo]. YouTube. Obtenido de: https://www.youtube.com/watch?v=uOpW1tKKO8M

- Run Dev. (19 de septiembre del 2020). Diccionarios en Python | Función items, get, dict, keys y values | Código Nativo [Vídeo]. YouTube. Obtenido de: https://www.youtube.com/watch?v=ZRxj3euWzuI

- Codigofacilito. (22 de septiembre del 2016). Curso Python – Diccionarios [Vídeo]. You tuve. Obtenido de: https://www.youtube.com/watch?v=_UELgsIxE7g

# Referencias bibliográficas


- Pythones, "Listas, Tuplas y Range en Python 3: Similitudes y diferencias" [Online]. Recuperado de: *https://pythones.net/listas-tuplas-python/*.

El futuro digital es de todos

MinTIC

Hechos QUE CONECTAN

# CICLO 1

# EJE TEMÁTICO 5

# ESTRUCTURAS DE DATOS

Universidad Industrial de Santander

MisionTIC 2022