> estructuras de datos

SEMANA 3

# Estructuras de datos –

**list – tuple – set –dict**

## INTRODUCCIÓN

Hasta el momento pudimos ver lo qué son las variables y las estructuras de control de flujo condicional, por lo que ya podemos comenzar a, no solo almacenar valores con las variables, sino que también, ya podemos comenzar a controlar el flujo del programa.
Pero surge algo más que debemos controlar, y es la organización de los datos y la forma de almacenarlos.

**Las estructuras de datos** o también tipos de **datos compuestos** son una forma de organizar y almacenar datos para que puedan ser manipulados y utilizados de manera eficiente. Una estructura de datos se refiere a una colección de valores de datos que están organizados y almacenados de una manera particular para su uso y acceso.

La elección de una estructura de datos adecuada es fundamental para el éxito de cualquier proyecto de programación, ya que una estructura de datos inapropiada puede resultar en un código lento, ineficiente e incluso incorrecto.

Por lo tanto, es importante que los programadores comprendan las diferentes estructuras de datos y su uso para poder elegir la estructura más adecuada para cada situación y problema.

Hay cuatro tipos de estructuras de datos en Python: **listas, tuplas, conjuntos y diccionarios**.

## LISTA – list

Las listas son una colección ordenada y mutable de elementos que pueden ser de diferentes tipos de datos.
Se pueden agregar, eliminar y modificar elementos de una lista.
Las listas se definen utilizando corchetes [] y, los elementosde la lista, se separan por comas.

```
#lista de números
numeros = [1, 2, 3, 4, 5]

#lista de strings
nombres = ["Juan", "Maria", "Pedro"]

#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]
```

Podemos observar que, para crear una lista, basta solo son declarar un nombre para la lista y asignar datos, de un tipo o varios,dentro de corchetes y separados

Cada uno de estos datos se convierte en un elementode la lista, por lo que puedeser accedido mediante un índice.


En las listas cada elemento tiene un índice contando desde el 0 (cero). Es decir que, para acceder al primer elemento de nuestra lista, debemos hacerlo desde el 0. Para mostrar lo que hay en nuestra lista usamos, como ya lo venimos haciendo, la función print().

```
#lista de números
numeros = [1, 2, 3, 4, 5]

#lista de strings
nombres = ["Juan", "Maria", "Pedro"]

#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]

# Acceder a un elemento de la lista
print(numeros[0])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
1
```

De esta manera podemos mostrar lo que tiene nuestra lista ya sea en el elemento 0 (cero) o en el elemento al cual queramos acceder, siempre con el número de índice entre corchetes, luego del nombre de la lista.
Si queremos mostrar todos los elementos que tiene la lista, lo hacemos sin poner un índice.

```
#lista de números
numeros = [1, 2, 3, 4, 5]

#lista de strings
nombres = ["Juan", "Maria", "Pedro"]

#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", 1, 2, 3, True, False, 3.14]

# Acceder a un elemento de la lista
print(numeros)
print(nombres)
print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[1, 2, 3, 4, 5]
['Juan', 'Maria', 'Pedro']
['Juan', 'Maria', 'Pedro', 1, 2, 3, True, False, 3.14]
```

## ¿Y qué más puede contener una lista?

Puede contener otras listas adentro de la lista. En ese caso, esta lista adentro de la otra se convierte en un elemento más, por lo que si queremos acceder a la lista dentro de la lista, lo hacemos con el número de índice, tal cual como lo hacemos para acceder a cualquier elemento, pero ahora veremos una lista completa.

```
#lista de números
numeros = [1, 2, 3, 4, 5]

#lista de strings
nombres = ["Juan", "Maria", "Pedro"]

#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", numeros, True, False, 3.14]

# Acceder a un elemento de la lista
print(lista[3])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[1, 2, 3, 4, 5]
```

En el ejemplo anterior te lo mostramos con la lista “números”, la cual la incluimos en la lista de nombre “lista”, por lo que, si llamamos al índice que corresponde, nos da como resultado, los elementos que se encuentran en la lista “números”.

Además, podemos incluir una lista que no declaramos antes, y esto lo hacemos solamente abriendo corchetes e incluyendo los elementos que queremos en la lista, como en el siguiente ejemplo.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

# Acceder a un elemento de la lista
print(lista[3])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
[6, 7, 8, 9]
```

Además, podemos acceder a un rango de elementos de la lista, como te mostraremos en el siguiente ejemplo. Esto se llama Slicing.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

# Acceder a un rango de elementos de la lista
print(lista[1:4])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Maria', 'Pedro', [6, 7, 8, 9]]
```

Como se puede observar, esto se hace escribiendo el rango al que queremos acceder (escribiendo el número de la primer posición, separado por : (dos puntos) y, el número de la última posición a la que queremos acceder). En este caso accedimos al elemento de la posición 1 (es decir, el segundo elemento de la lista) hasta el elemento de la posición 3 (que sería el elemento 4). Volvemos a recordarte que, al momento de acceder, Python cuenta desde el 0 en adelante.
También, se puede dar un rango desde una posición hasta el final o, desde el principio hasta una posición.

*Ejemplo:*
*desde una posición hasta el final de la lista*

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

# Acceder a un rango de elementos de la lista
print(lista[2:])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Pedro', [6, 7, 8, 9], True, False, 3.14]
```

*Ejemplo:*
*desde el principio de la lista hasta una posición*

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

# Acceder a un rango de elementos de la lista
print(lista[:3])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Juan', 'Maria', 'Pedro']
```

Y otra forma de acceder a los elementos de la lista, es desde el final hacia adelante. Esto lo hacemos usando números negativos.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

#acceder al último elemento la lista
print(lista[-1])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
3.14
```

O, al anteúltimo, y así sucesivamente, podemos recorrer la lista desde atrás.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]

#acceder al anteúltimo elemento la lista
print(lista[-2])
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
False
```

Para modificar un elemento de la lista solo basta son asignar el nuevo valor a la posición del elemento que deseamos cambiar.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
lista[1] = "Ana"

#Imprimir la lista
print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Juan', 'Ana', 'Pedro', [6, 7, 8, 9], True, False, 3.14]
```

Y si asignamos nuevos valores a la lista, éstos nuevos valores reemplazan a todos los elementos que había antes en la lista. Todo esto que venimos haciendo es gracias a que las listas son mutables.

```
#lista de varios tipos de datos
lista = ["Juan", "Maria", "Pedro", [6, 7, 8, 9], True, False, 3.14]
lista = ["Hola", "Mundo", 1, True]

#Imprimir la lista
print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Hola', 'Mundo', 1, True]
```

Además, podemos acceder a un rango de elementos de la lista, como te mostraremos en el siguiente ejemplo. Esto se llama Slicing.

## Métodos para listas

Además, podemos hacer otras cosas con las listas, como agregar elementos, eliminarlos, etc., mediante métodos, la cual, es una nueva palabra que estamos introduciendo al vocabulario, por lo que te daremos una pequeña explicación y su diferencia con funciones (las cuales venimos usando desde el primer día).

### Funciones

Una función es un bloque de código que se puede llamar desde cualquier parte de un programa para realizar una tarea específica. Esta puede recibir parámetros (valores que se le pasan a la función) y devolver un resultado. Por ejemplo, la función print a la cual le pasamos un parámetro y muestra por pantalla ese valor ----> print(variable)

### Métodos

Un método es una función que está asociada con un objeto en particular. En Python, los objetos tienen métodos que les permiten realizar acciones específicas. Por ejemplo, una lista tiene un método llamado append(), que se utiliza para agregar elementos al final de la lista. En este caso, la lista es el objeto al que pertenece el método append(), y el método solo puede ser utilizado con objetos de tipo lista.

Así que podemos decir que, la principal diferencia entre una función y un método en Python, es que
una función es independiente y puede ser llamada desde cualquier parte del programa (o incluso
desde otros programas), mientras que un método está asociado con un objeto en particular y solo
puede ser utilizado por objetos de ese tipo.

Explicado brevemente esto (más adelante vas a comprender aún mejor esto), vamos a ver los
métodos más comunes que se puede usar con listas.

- **append(elemento)**: Agrega un elemento al final de la lista.
- **extend(iterable)**: Agrega los elementos de un iterable al final de la lista.
- **insert(i, elemento)**: Inserta un elemento en una posición específica de la lista.
- **remove(elemento)**: Elimina la primera aparición de un elemento de la lista.
- **pop([i])**: Elimina el elemento en la posición especificada y lo devuelve. Si no se especifica una posición, elimina y devuelve el último elemento de la lista.
- **clear()**: Elimina todos los elementos de la lista.

- **index(elemento[, inicio[, fin]])**: Devuelve la posición de la primera aparición de un elemento en la lista. Se pueden especificar los índices de inicio y fin para hacer un slicing.
- **count(elemento)**: Devuelve el número de veces que un elemento aparece en la lista.
- **reverse()**: Invierte el orden de los elementos en la lista.
- **copy()**: Devuelve una copia superficial de la lista.
- **sort(key=None, reverse=False)**: Ordena los elementos de la lista en orden ascendente (a menos que se especifique lo contrario con el parámetro reverse=True). Se puede especificar una función de clave opcional para definir el orden de clasificación.
- **len()**: devuelve el número de elementos en la lista.

Tres ejemplos en código de cómo usar alguno de estos métodos

## append()

```
7   #lista de varios tipos de datos
8   lista = ["Hola", "Mundo", 1, True]
9
10  #Imprimir la lista
11  print(lista)
12
13  # Agregar un elemento a la lista
14  lista.append("otro elemento")
15
16  #Imprimir la lista
17  print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Hola', 'Mundo', 1, True]
['Hola', 'Mundo', 1, True, 'otro elemento']
```

## remove()

```
7   #lista de varios tipos de datos
8   lista = ["Hola", "Mundo", 1, True]
9
10  #Imprimir la lista
11  print(lista)
12
13  # Eliminar un elemento de la lista
14  lista.remove("Mundo")
15
16  #Imprimir la lista
17  print(lista)
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
['Hola', 'Mundo', 1, True]
['Hola', 1, True]
```

## len()

```
7   #lista de varios tipos de datos
8   lista = ["Hola", "Mundo", 1, True]
9
10  #Imprimir la lista
11  print(len(lista))
```

PROBLEMAS SALIDA CONSOLA DE DEPURACIÓN TERMINAL

```
4
```

## TUPLA -tuple

Una tupla es una colección de elementos ordenados e inmutables. Las tuplas son similares a las listas,
pero a diferencia de las listas, una vez que se crea una tupla, no se pueden agregar, eliminar o modificar elementos.

Las tuplas se pueden crear utilizando paréntesis () y los elementos se separan por comas. Incluso si una tupla tiene un solo elemento, se debe incluir una coma al final para indicar que es una tupla en lugar de un valor entre paréntesis.

```
tupla = ("manzana", "naranja", "banana")
```

Como se puede observar, tiene prácticamente la estrustura de una lista para crearla, pero en vez de corchetes, lleva paréntesis. Sin embargo, las tuplas pueden ser creadas sin paréntesis también (siguiendo el mismo orden de separar los elemcntos por coma), pero por convención usamos paréntesis.

```
tupla = ("manzana", "naranja", "banana")
tupla1 = "manzana", "naranja", "banana"

print(type(tupla))
print(type(tupla1))
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

<class 'tuple'>
<class 'tuple'>
```

En el ejemplo anterior te mostrábamos lo que decíamos sobre poder crear una tupla sin paréntesis, sin embargo, por convención, vamos a crear las tuplas con paréntesis.

Algo que también aplicamos es la función type() para ver el tipo de estructura que tenemos (lo usamos para ver los tipos de datos simples en las variables), y esta función también se puede usar para listas, conjuntos, diccionarios y tuplas, como es este caso.

Mencionamos antes también, que si queremos crear una tupla de un solo elemento, siempre debemos poner la coma al final.

```
tupla = ("manzana",)

print(type(tupla))
```

```
PROBLEMAS  SALIDA  CONSOLA DE

<class 'tuple'>
```

## Una tupla es inmutable

Cuando nos referimos a esto, decimos que las tuplas no pueden ser modificadas, a diferencia de las listas, las cuales si queríamos modifisar un elemento, podíamos asignar un nuevo valor ingresando el nuevo elemento mediante su índice. Con las tuplas no ocurre esto.
Si bien podemos mostrar por pantalla un elemento de la tupla mediante su índice (como lo hacíamos en las listas), no podemos modificar un elemento.
Lo vemos en el siguiente ejemplo:

```
tupla = ("manzana", "pera", "banana")

#accediendo mediante índice
print(tupla[1])
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL

pera
```

*Aquí accedemos mediante el índice a un elemento de la tupla. En este caso al segundo elemento.*

```
tupla = ("manzana", "pera", "banana")

#intentando modificar una tupla
tupla[1] = ("pomelo")
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL  Python

Traceback (most recent call last):
  File "c:\Users\Pc\Desktop\Info2023\estructuras_de_datos_tuplas.py", line 4, in <module>
    tupla[1] = ("pomelo")
    ~~~~~^^^
TypeError: 'tuple' object does not support item assignment
```

*Aquí estamos intentando modificar la tupla como lo hacíamos con las listas, sin embargo, vemos que nos da un error al intentar realizar esto.*

Esto no quiere decir que una vez creada la tupla no se le pueda modificar más, o reutilizarla en todo caso, lo que no se puede modificar es por medio de índices, es decir, no se puede modificar un solo valor de la tupla, pero si podemos reescribir completamente todos los valores.

```
1   tupla = ("manzana", "pera", "banana")
2   print(tupla)
3
4   #escribiendo valores nuevos en la tupla
5   tupla = ("pomelo", "melón", "kiwi")
6   print(tupla)
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

('manzana', 'pera', 'banana')
('pomelo', 'melón', 'kiwi')
```

*Y si bien las tuplas no tienen tantos métodos para usarse sobre ellas como las listas, te invitamos a que puedas probar algunos de los métodos qus te dimos para las listas y verás que algunos sirven también para tuplas y otros no. También te animamos a que búsques y practiques más métodos para tuplas.*

*Práctica, más práctica y mucha más práctica te a ayudarán a ser un buen programador!*

## CONJUNTO - set

Un conjunto es una colección de elementos desordenados y únicos. Así como vimos que las listas y tuplas son estructuras ordenadas, ya que por medio de índices podemos acceder a los elementos que contienen, aquí no ocurre lo mismo.

Cada elemento que contiene el conjunto, no tiene índice y no puede accederse como lo veníamos haciendo anteriormente. Otra diferencia es que son únicos, ya que sus elementos no pueden repetirse. Esta es una función útil cuando queremos eliminar valores, o elementos, duplicados.
También son muy útiles para realizar operaciones de conjunto, como la unión, la intersección, la diferencia y la comprobación de la pertenencia de elementos. Aquí para crearlos utilizaremos llaves **{ }**

```
1   conjunto = {1, 5, "Hola", "Mundo", 3.14, True}
2   print(conjunto)
3   print(type(conjunto))
```

```
PROBLEMAS   SALIDA   CONSOLA DE DEPURACIÓN   TERMINAL

{1, 'Mundo', 3.14, 5, 'Hola'}
<class 'set'>
```

Como se puede observar, se mantiene la misma estructura básica de como veníamos creando las otras estructuras, con la diferencia de el uso de llaves. Pero, vamos a ver una de las características del conjunto, y es que son únicos.

*En este ejemplo estamos repitiendo los valores "Hola", 1 y 5, sin embargo,cuando mostramos por pantalla, estos valores duplicados no salen, a diferencia de las listas o tuplas en las que si podíamos ingresar valores duplicados.*
*Aquí el conjunto está cumpliendo con su característica de eliminar la duplicidad de datos. Y para probar la otra característica de desordenado, te invitamos a copiar este código y ejecutarlo varias veces. Vas a poder apreciar que todas las veces que se muestren estos datos por pantalla, no van a estar en el mismo orden.*

## DICCIONARIO - dict

Y aquí llegamos al último apartado de estructuras de datos. Un diccionario es una estructura de datos que permite almacenar información en pares clave - valor **(key - value)**. Es decir, cada elemento del diccionario consta de una clave y un valor asociado a dicha clave.

La clave debe ser única dentro del diccionario, mientras que los valores pueden ser de cualquier tipo de datos. De la misma forma que con los conjuntos, se utilizan las llaves {} para crearlo, sin embargo, difiere un poco en la forma de hacerlo. Vamos a ver un ejemplo.

```
1  diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2  print(diccionario)
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
{'Juan': 24, 'Ana': 30, 'Cristian': 28}
```

Vemos que comenzamos de la misma forma que las demás estructuras, con el nombre que va a tener, el signo de asignación, una llave de apertura y al final una llave de cierre, pero los datos se agregan de otra forma.
Aquí debe ir una clave (de cualquier tipo de dato), : (dos puntos) y un valor (de cualquier tipo de dato), y para seguir agregando datos, los separamos por comas como el resto de las estructuras.
Para acceder a un dato del diccionario, se debe usar su clave y obtendremos el valor.

```
1  diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2  print(diccionario['Juan'])
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
24
```


Y para modificar un valor lo hacemos de la siguiente forma:

```
1  diccionario = {'Juan': 24, 'Ana': 30, 'Cristian': 28}
2  diccionario['Ana'] = 18
3  print(diccionario)
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
{'Juan': 24, 'Ana': 18, 'Cristian': 28}
```

*Accedemos a la clave y asignamos un nuevo valor.*

Otra forma de crear diccionarios es usando la función **dict()**, y se puede usar de la siguiente manera:

```
1  una_lista = [('clave1', 'valor1'), ('clave2', 'valor2'), ('clave3', 'valor3')]
2  un_diccionario = dict(una_lista)
3  print(un_diccionario)
4  print(type(un_diccionario))
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
{'clave1': 'valor1', 'clave2': 'valor2', 'clave3': 'valor3'}
<class 'dict'>
```

O usando la sintaxis de argumentos de palabras clave para especificar las claves y los valores:

```
1  mi_diccionario = dict(clave1='valor1', clave2='valor2', clave3='valor3')
2  print(mi_diccionario)
3  print(type(mi_diccionario))
```

```
PROBLEMAS  SALIDA  CONSOLA DE DEPURACIÓN  TERMINAL
{'clave1': 'valor1', 'clave2': 'valor2', 'clave3': 'valor3'}
<class 'dict'>
```

Sin embargo, la forma más común que vas a ver en la creación de diccionarios es como te mostramos de en la primer forma, con el nombre, la asignación y el uso de llaves.

## Métodos para diccionarios

Éstos son algunos de los métodos más comunes para diccionarios, sin embargo, te invitamos a que los puedas probar para practicar y ver como funciona cada uno.

- **clear()**: Elimina todos los elementos del diccionario.
- **copy()**: Devuelve una copia superficial del diccionario.
- **get()**: Devuelve el valor asociado a una clave dada, o un valor predeterminado si la clave no está presente.
- **items()**: Devuelve una vista de lista de todos los pares clave-valor en el diccionario.
- **keys()**: Devuelve una vista de lista de todas las claves en el diccionario.
- **pop()**: Elimina y devuelve el valor asociado a una clave dada.
- **popitem()**: Elimina y devuelve un par clave-valor aleatorio del diccionario.
- **setdefault()**: Devuelve el valor asociado a una clave dada, o lo crea con un valor predeterminado si la clave no está presente.
- **update()**: Actualiza el diccionario con los pares clave-valor de otro diccionario o iterable.
- **values()**: Devuelve una vista de lista de todos los valores en el diccionario.

## Repasando conceptos

**LISTA - list**
Permite almacenar una colección ordenada y mutable de elementos, que pueden ser accedidos mediante un índice numérico.
Es la estructura que más frecuentemente vamos a utilizar debido a su versatilidad, orden y acceso. Recuerda que para crearla se hace escribiendo un nombre para la lista, signo de asignación y entre corchetes los valores que vamos a guardar separados por coma cada uno. Su nombre en inglés es list.

**TUPLA - tuple**
Permite almacenar una colección ordenada e inmutable de elementos, que también pueden ser accedidos mediante un índice numérico.
Al igual que las listas, la forma de crearlas es similar, pero en vez de usar corchetes, usamos paréntesis. Su nombre en inglés es tuple.

**CONJUNTO - set**
Permite almacenar una colección desordenada y mutable de elementos únicos y no indexados. Los elementos en un conjunto son únicos y no se permiten elementos duplicados. Al igual que las listas o tuplas, la forma de crearlas es similar, pero aquí usamos llaves en vez de corchetes o paréntesis. Su nombre en inglés es set.

**DICCIONARIO - dict**
Permite almacenar una colección no ordenada, mutable e indexada de pares clave-valor.
Las claves deben ser únicas en un diccionario (no debemos repetir el nombre de una clave), mientras que los valores pueden ser de cualquier tipo de datos.
Los diccionarios son muy útiles cuando se necesita acceder rápidamente a un valor basado en una clave. Su nombre en inglés es dict.