



# PYTHON

Módulo 5 – Estructuras de datos

1

# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos - Listas

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Listas

- Son modificables (“mutable”)
    - Declaración e inicialización
      - `diasSemana = []`
      - `estaciones = [“Primavera”, “Verano”, “Otoño”, “Invierno”]`
    - Mostrar lista
      - `print`
      - `for`
    - Acceso (por índice: lista de indexación)
      - `estaciones[posicion]`
      - Permite acceso y asignación
      - No permite acceso ni asignación a elementos inexistentes
      - Admite posiciones negativos
      - Admiten slicing

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Listas
    - Función len()
    - Sentencia del
      - Aplicado con slicing → Eliminación parcial
      - del dias\_semana[5:]
    - Intercambio de valores entre elementos:
      - lista[i1], lista[i2] = lista[i2], lista[i1]

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Listas
    - Operador in y not in
    - Método append() → Aplicado a listas, mete una lista dentro de un elemento de otra lista
    - Método insert(pos, valor)
    - Método remove() → En caso de que haya varios, borra el primero
      - Por posición: `dias_semana.remove(dias_semana[4])`
    - Método sort()
    - Método reverse()
    - Método extend() → similar al operador +
    - Método index() → Obtiene la posición de un elemento
    - Método count()

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Listas
    - ¡Valores (slicing) VS Referencias!
      - Copia por referencia copia = original
      - Copia por valor copia = original[desde:hasta]
    - Listas dentro de listas → Listas bidimensionales y multidimensionales.

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Listas

- Comprensión de listas (*list comprehension*):

- Permite crear listas mediante código muy compacto.

- *nombre\_lista* = [ *expresión for miembro in iterable* ]

```
>>> lista = [i for i in range(10)]
```

```
>>> lista
```

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
>>> dias
```

```
('Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes')
```

```
>>> lista = [dia for dia in dias]
```

```
>>> lista
```

```
['Lunes', 'Martes', 'Miércoles', 'Jueves', 'Viernes']
```

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Listas

- Comprensión de listas (*list comprehension*):

```
>>> def doble(numero):  
...     return numero*2  
...  
>>> lista = [doble(n) for n in range(5)]  
>>> lista  
[0, 2, 4, 6, 8]
```



# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos - Tuplas

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Tuplas

- Es una estructura de datos similar a una cadena de caracteres (string)
    - Es inmutable
    - Contiene elementos arbitrarios (usar conjuntamente con isinstance)
    - nombreTupla = () → Tupla vacía
    - nombreTupla = ("elemento1", elemento2)
    - nombreTupla = ("elemento1",) → Para indicar que sólo hay un elemento

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Tuplas

- Admite slicing [:] → `print(tupla[2:])`
    - Admite concatenaciones + → Modificaciones mediante creación de nuevas tuplas
    - Se recorren con bucle for
    - Operador in
    - Función `len()`
    - Función `max()`
    - Función `min()`
    - Admiten comparaciones
    - Sentencia `list` → `list(tupla)`
    - Función `tuple()` → Convierte una lista en tupla

# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos - Conjuntos

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Conjuntos
    - Estructura de datos sin orden
    - Un elemento sólo puede estar una vez en un conjunto
    - No admite acceso por índice ni slicing.
    - Inmutable.
    - No admite duplicados.
    - Función is()
    - Operador in y not in
    - Sentencia list → list(conjunto). Conversión en lista.
    - Construcción:
      - Conjunto = {"manzana", "pera", "naranja"}

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Conjuntos
    - Método `isdisjoint()` → Determina si dos conjuntos son disjuntos.

```
>>> frutas = {'manzana', 'pera', 'naranja'}  
>>> alimentos = {"filete", "patata", "lechuga"}  
>>> caprichos = {"filete", "pera"}  
>>> frutas.isdisjoint(alimentos)  
True  
>>> frutas.isdisjoint(caprichos)  
False
```

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Conjuntos

- Método `intersection()` → Obtiene los elementos comunes.
    - Método `difference()` → Obtiene los elementos no contenidos. **Importa el orden de la invocación.**

```
>>> dias={"lunes","martes","miercoles","jueves","viernes","sabado","domingo"}
>>> misdias = {"martes","jueves","friday"}
>>> dias.intersection(misdias)
{'martes', 'jueves'}
>>> dias.difference(misdias)
{'miercoles', 'domingo', 'lunes', 'viernes', 'sabado'}
```

# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos - Diccionarios



# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Diccionarios
    - También conocidos como mapas, “tabla hash” o array asociativo.
    - Están compuestos de pares clave-valor.
    - Creación:
      - `diccionario = {}`
      - `diccionario = dict()`
      - `diccionario = {clave1:valor1,clave2:valor2,...}`
      - `diccionario = dict([(clave1, valor1), (clave2, valor2),...])`

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Diccionarios

- Asignación de valores:

- `diccionario[clave_n]=valor_n`

- `diccionario.setdefault(clave_n, valor_n)` → Sólo asigna si la clave no existe.

- Acceso:

- `diccionario[clave]` → Provoca error si no existe la clave.

- `Diccionario.get(clave)` → No da error si no existe la clave. Admite valor por defecto. Devuelve None si no se indica un valor por defecto.

- Sentencia `list` → `list(diccionario)`

- Sentencia `del` → `del diccionario[clave]`

- Método `pop(clave)` → Devuelve y elimina un elemento dado (admite valor por defecto)

- Método `popitem()` → Devuelve y elimina el último elemento

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Diccionarios
    - Operadores in y not in
    - Métodos keys, values y ítems.
    - Recorrer diccionario:
      - for k, v in diccionario.items():
    - diccionario.clear() → Borra el diccionario

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Diccionarios

- Crear un diccionario a partir de dos listas:

```
>>> dias = ["lunes","martes","miercoles"]  
>>> valores=[5,8,10]  
>>> zip_iterator = zip(dias, valores)  
>>> diccionario = dict(zip_iterator)  
>>> diccionario  
{'lunes': 5, 'martes': 8, 'miercoles': 10}
```

# MÓDULO 5: ESTRUCTURAS DE DATOS

Empaquetado y desempaquetado  
(*Packing y unpacking*)

# MÓDULO 5: ESTRUCTURAS DE DATOS

## ■ Desempaquetado:

- Permite asignar a variables independientes el contenido de una estructura de datos.
- Los elementos se asignan por orden.
- Debe haber concordancia en el número.

```
>>> lista = [1,2,3]
>>> tupla = (1,2,3)
>>> conjunto = {1,2,3}
>>> diccionario = {1:"Uno",2:"Dos",3:"Tres"}
>>> l1,l2,l3 = lista
>>> t1,t2,t3 = tupla
>>> c1,c2,c3 = conjunto
>>> i1,i2,i3 = diccionario.items()
>>> v1,v2,v3 = diccionario.values()
>>> k1,k2,k3 = diccionario #Keys
```

# MÓDULO 5: ESTRUCTURAS DE DATOS

## ■ Empaquetado:

- Permite generar una lista a partir de elementos individuales.
  - $x, y = 1, 2 \rightarrow \underline{x}$  e  $\underline{y}$  toman los valores 1 y 2 respectivamente.
  - $*x, = 1, 2 \rightarrow \underline{x}$  toma el valor  $[1, 2]$
  - $*x, y = 1, 2, 3 \rightarrow \underline{x}$  toma el valor  $[1, 2]$ ,  $\underline{y}$  toma el valor 3.
  - $x, *y = 1, 2, 3 \rightarrow \underline{x}$  toma el valor 1,  $\underline{y}$  toma el valor  $[2, 3]$ .
  - $*x, y = \text{range}(4) \rightarrow \underline{x}$  toma el valor  $[0, 1, 2]$ ,  $\underline{y}$  toma el valor 3.

# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos - Arrays



# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Arrays

- Mismo tipo de dato.
    - Sólo admite algunos tipos de datos.
    - `from array import array`
    - Declaración:
      - `nombre = array('tipo',[valor1, valor2, ...])`
    - Función `len(nombre)`
    - Acceso  $\rightarrow$  `nombre[índice]`
    - Recorrer con `for-in`.
    - Operador `in`

<https://docs.python.org/3/library/array.html>

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Arrays: tipos

Type code	C Type	Python Type	Minimum size in bytes	Notes
'b'	signed char	int	1	
'B'	unsigned char	int	1	
'u'	wchar_t	Unicode character	2	(1)
'h'	signed short	int	2	
'H'	unsigned short	int	2	
'i'	signed int	int	2	
'I'	unsigned int	int	2	
'l'	signed long	int	4	
'L'	unsigned long	int	4	
'q'	signed long long	int	8	
'Q'	unsigned long long	int	8	
'f'	float	float	4	
'd'	double	float	8	

# **MÓDULO 5: ESTRUCTURAS DE DATOS**

**Estructuras de datos – colas y listas**

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Colas
    - FIFO
      - Estructura sincronizada de datos (permite uso seguro de hilos)
      - `import queue`
      - Clases `Queue`, `SimpleQueue`, `PriorityQueue`, `LifoQueue`
        - `cola = queue.Queue()` → Construcción
        - `cola.qsize()`
        - `cola.empty()`
        - `cola.full()` → Indica si la cola está llena o no (posibles bloqueos)
        - `cola.put()`
        - `cola.get()`

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Colas
    - FIFO – Implementada con una lista.
      - Menos eficiente que queue
      - Inserción: `lista.insert(0,nuevo_elemento)`
      - Acceso: `lista.pop()`

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Pilas

- LIFO – Implementada con una lista.

- Es una lista a la que se accede mediante el método pop (devuelve el último elemento que se agregó con append eliminándolo de dicha lista)
      - `pila.append()` → Agregar
      - `pila.pop()` → Obtener el último agregado (lo elimina de la lista)
      - `len(pila)` → Proporciona la longitud

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos

- Colas

- FIFO y LIFO

- Mediante collections.queue

- Más eficiente.

```
from collections import deque  
cola = deque([1,2,3])  
cola.append(4)  
cola.popleft() → LIFO  
cola.pop() → FIFO
```

# **MÓDULO 5: ESTRUCTURAS DE DATOS**

## **Estructuras de datos - Comparativa**



# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos: Comparativa.

Nombre	Sintaxis	Inmutable	Slicing	Admite duplicados
Lista	[item1, item2, ...]	No	Sí	Sí
Tupla	(item1, item2, ...)	Sí	Sí	Sí
Conjuntos	{item1, item2, ...}	Sí	No	No
Diccionarios	{clave1:valor1, clave2:valor2, ...}	No	No	No (clave única)

# MÓDULO 5: ESTRUCTURAS DE DATOS

## Estructuras de datos – Funciones genéricas

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Estructuras de datos
  - Funciones e instrucciones genéricas.
    - Se pueden utilizar sobre cualquier estructura de datos.
      - print. Aplicable a lista, tupla, conjunto, diccionario.
      - len. Aplicable a lista, tupla, conjunto, diccionario y cadena.
      - in y not in. Aplicable a lista, tupla, conjunto, diccionario y cadena.
      - sorted. Aplicable a lista, tupla, conjunto (devuelve una lista), diccionario (devuelve una lista con las claves) y cadena (devuelve una lista con los caracteres ordenados).
      - del. Aplicable a lista, tupla, conjunto, diccionario y cadena (todo tipo de variables).

# MÓDULO 5: ESTRUCTURAS DE DATOS

- Enlaces:
  - Data structures:
    - <https://docs.python.org/3/tutorial/datastructures.html>
  - Common Python Data Structures:
    - <https://realpython.com/python-data-structures/>
  - Python Data Structures:
    - <https://www.geeksforgeeks.org/python-data-structures/>