**LISTAS**

Permiten realizar operaciones como ordenamiento, filtrado y cálculos estadísticos en conjuntos de datos. Se pueden modificar añadiendo y eliminando elementos.

Se crean con [] y separados por ,.

**Funciones:**

- **Juntar varias listas** en una sola usamos +.
  Ej: nueva_lista = [lista1 + lista2]

- Para juntar listas haciendo una **lista de listas** usamos ,.
- Ej: nueva_lista = [lista1 , lista2]

- **len**(): conocer el **número de elementos** de una lista → len(lista)
- **max**(): conocer el **valor más alto** de una lista → max(lista)
- **min**(): conocer el **valor más bajo** de una lista → min (lista)

Estas funciones se pueden usar si todos los elementos son números (int o float), si hubiera str daría error.

Para listas de str se ordenan por orden alfabético. min A, max Z.

- **in, not in**: verificar si un **elemento está presente** o no en la lista. Devuelve True si está, False si no está

- **Indexar**: **acceder a un elemento específico** en una lista.

El primer elemento tiene un índice 0, el segundo elemento tiene un índice 1 y así sucesivamente.
Ej: para saber el tercer elemento de una lista → nombre_lista[2]
También se puede sacar el índice empezando por el **final de la lista**, el último elemento comenzaría por -1.
Ej: para saber el tercer elemento por el final de una lista de 365 elementos → nombre_lista[-363].
**Indexar**/sacar **varios elementos de una lista**, por el principio y por el final: Ej:
Elementos_extraidos = [nombre_lista[0], nombre_lista[1], nombre_lista[2], nombre_lista[3], nombre_lista[4]].

Elementos_extraidos = [nombre_lista[-1], nombre_lista[-2], nombre_lista[-3], nombre_lista[-4], nombre_lista[-5]].
También se puede usando **slicing**. **(esto mejor)**
Ej1 lista con 5 elementos: nombre_lista[0:4] es igual que nombre_lista[:4] Extrae los 4 primeros elementos porque el último se excluye.
Ej2 lista con 4 elementos: nombre_lista[-5:-1] Aquí no se extraerían el 1º elemento del final porque se excluye. nombre_lista[-5:] aquí sí se extraen todos

- **Start-stop-step** técnica para **acceder a elementos específicos** de una lista.
  **-** Start se refiere al índice de inicio de la lista. Si se omite, se considera 0.
  **-** Stop se refiere al índice de fin de la lista (excluyéndolo). Si se omite, se considera la longitud de la lista.
  **-** Step se refiere al número de elementos que se salta entre cada elemento en la lista. Si se omite, se considera 1.

Ej1: en una lista de 8 elementos, para conocer los que están en posición par nombre_lista[::2].
Ej2: Para saber los que están en posición impar desde el 1 hasta el 7 nombre_lista[1:8:2].

Para **invertir el orden** de los elementos **sin guardar la lista** se puede usar [::-1]

- **.Index()** se usa para **saber** en qué **posición** está un elemento que le demos. Ej: lista_letras = [‘A’, ‘B’, ‘C’, ‘D’, ‘E’]. Para saber la posición de ‘D’ print(lista_letras.index(D)). El resultado saldría 3.

- **.Copy** hace una **copia exacta** de una lista con los mismos elementos de la lista original → nueva_lista = lista_original.copy()

- **.Append**: **Agrega un elemento** a la lista y sobreescribe → nombre_lista.append(‘elementoaañadir’)

- **.Extend**: **Agrega los elementos de una lista al final de otra lista**. Ej: nombre_lista.extend(nombre_lista_añadida)

- **Insert**: **Inserta** un elemento en una **posición específica** de la lista. Ej: nombre_lista.insert(nºposición, “elementoañadir”)

- **.Remove**: **Elimina el primer elemento de la lista que coincida con el valor especificado**. Si queremos seguir eliminando ese valor repetido en la lista habrá que repetir remove. → nombre_lista.remove(elementoaeliminar)

- **.Pop**: **Elimina y devuelve el último** elemento de la lista. También elimina el elemento de un **índice concreto**. → nombre_lista.pop() . nombre_lista.pop(nº posición)

- **Index**: Para saber en qué **posición** está el **elemento especificado** en la lista → nombre_lista.index(elementoacontar)

- **Count**: cuenta el **número de veces que aparece el elemento** especificado en la lista → nombre_lista.count(elementoacontar)

- **.Sort**: **Ordena los elementos** y **modifica** la lista original.
  -**ascendente** → nombre_lista.sort()
  -**descendente** → nombre_lista.sort(reverse=True)

- **Sorted(): Ordena los elementos** y **no modifica** la lista original, para modificar habría que crear una nueva variable.
  -**ascendente** → sorted(nombre_lista)
  -**descendente** →sorted(nombre_lista, reverse=True)

- **.Reverse**: **Invierte el orden** de los elementos de la lista y queda modificada → nombre_lista.reverse()

- **.Clear**: **Elimina** todos los elementos de la lista y la deja **vacía** → nombre_lista.clear()

- **Para modificar un elemento por otro:** nombre_lista[nºposicióna modificar]=nuevoelemento

**TUPLAS**

Proporcionan una estructura para almacenar información que **no pueden modificarse agregando ni eliminando elementos**.

Se crean con **()** y separados por **,**.

**Peculiaridades**:

- Si definimos una tupla de **un elemento con ()** NO será una tupla, será variable de str, int o float.
- Si definimos **un elemento sin ()** y **con ,** SÍ será una tupla Ej: 3,
- Si definimos una variable **sin ()** separados **con ,** SÍ será una tupla Ej: 3, 'gato', 'mango', 6.5

**Funciones:**

- Para **convertir una lista** con números y letras **en tupla** usamos el método **tuple()**.
  Ej: lista1 = [3, gato, pera, 6.5] → tupla1 = tuple(lista1)

- Al igual que las listas, se puede **indexar** usando la misma forma para **acceder a un elemento específico** en una tupla.

- **Indexar**/extraer varios elementos de una tupla, igual que en las listas.
  Ej1 tupla con 5 elementos: nombre_tupla[0:4] es igual que nombre_ tupla[:4]
  Extrae los 4 primeros elementos porque el último se excluye.
  Ej2 tupa con 5 elementos: nombre_ tupla[-5:-1] Aquí no se extraerían el 1º elemento del final porque se excluye. nombre_ tupla[-5:] aquí sí se extraen todos

- **len**(): conocer el **número de elementos** de una tupla → len(tupla)
- **max**(): conocer el **valor más alto** de una tupla → max(tupla)
- **min**(): conocer el **valor más bajo** de una tupla → min (tupla)

Estas funciones se pueden usar si todos los elementos son números (int o float), si hubiera str daría error.

Para tuplas de str se ordenan por orden alfabético. min A, max Z.

- **Count**: para saber el **número de veces que aparece el elemento** especificado en la tupla → nombre_ tupla.count(elementoacontar)

- **Index** se usa para **saber** en qué **posición** está un elemento que le demos. Ej: tupla_letras = ('A', 'B', 'C', 'D', 'E'). Para saber la posición de 'D' print(tupla_letras.index(D)). El resultado saldría 3.

- Como las tuplas no se pueden **modificar**, para hacerlo hay 2 opciones:
  **Opción 1**: **convertir** las tuplas **en lista** y luego volver a pasarla a tupla. Pasos:
  1. Definimos una tupla con nombres de alumnas,
  tupla_nombres = ("Lola", "Paula", "Lorena")
  2. Convertimos la tupla a lista usando el método list()
  lista_nombres = list(tupla_nombres)
  3. Cambiamos el nombre de "Lola" por el de otra alumna
  lista_nombres[0] = "Marta"
  4. Convertimos la lista a tupla usando el método tuple()
  tupla_nombres = tuple(lista_nombres)
  **Opción 2: crear variables separadas** para cada valor de la tupla y crear una nueva tupla con los valores separados.
  1. Definimos una tupla con nombres de alumnas,
  tupla_nombres = ("Lola", "Paula", "Lorena")
  2. Generamos 3 variables que correspondan a cada elemento de la tupla
  a, b, c = tupla_nombres
  3. creamos una nueva tupla usando las variables que hemos creado
     tupla_nombres3 = ("Maria", b, c)
  4. print(tupla_nombres3) imprime ("María", "Paula", "Lorena")

- **Concatenar** tuplas usando + → nueva_tupla = tupla1 + tupla2

- **Zip():** función que combina dos o más objetos iterables (listas, duplas, diccionarios, etc.) en una sola estructura de datos. Creará parejas agrupando el primer elemento de cada lista, el segundo de cada lista y así sucesivamente.

Pasos:

1. Para convertir en zip, por ejemplo, las tuplas definidas primero hay que pasarlas a lista
   zip1 = list(zip(tupla1, tupla2))
2. Los zip solo se pueden usar una vez, por lo que si queremos usarlo habrá que combinarlo de nuevo
   zip2 = list(zip(tupla1, tupla2)
3. Ahora, por ejemplo, podemos ordenar el resultado del zip usando sort → zip2.sort. Y se ordenaría por el primer elemento, por lo que el orden en el que combinemos las tuplas sí influye en este resultado.