LISTAS

Una lista es una colección en la que los elementos que pueden ser de distintos tipos, están ordenados.

Características de las listas:

- No todos los elementos tienen que ser del mismo tipo.
- La lista es dinámica y puede cambiar de tamaño, es decir, podemos añadir y quitar elementos.
- Los elementos están ordenados, en cuanto a su posición en la lista. No es que por defecto se les aplique un orden al insertarlos. Además dado que pueden ser de distintos tipos, no siempre se puede establecer un orden según su valor.

Si has programado en otro lenguaje, las listas te pueden recordar a los arrays, pero en Python son mucho más dinámicas y versátiles.

Para declararla lo haremos de la siguiente forma:

```
mi_lista = [ 10, "hola", "Pepe",25]
```

Accederemos a cada elemento usando su posición, empezando en 0 y terminando en el elemento N-1, siendo N el número de elementos de la lista.

```
print(mi_lista[2])
```

Podemos modificar el elemento que está en una posición dada:

```
mi_lista = [ 10, "hola", "Pepe",25]

print("inicialmente",mi_lista[2])

mi_lista[2] = "Juan"

print("ahora es",mi_lista[2])
```

También podemos usar un índice negativo siendo **-1** el elemento de la última posición y así hasta llegar al primero.

Si intentamos acceder a un elemento más allá de la última o de la primera obtendremos una excepción **IndexError**

Podemos crear una lista vacía con *lista = []*

SubListas

También podemos recuperar varios elementos de una lista, es lo que se llama una **sublista**. Para ello indicaremos un rango de valores, que son 2 valores numéricos separados por ":", de la siguiente forma **principio:final**, siendo **principio** el índice del primer elemento de la sublista y **final** el índice posterior al último que vamos a recuperar.

```
>>> print(mi_lista[0:2])
[10, 'hola']
```

Si omitimos el primer valor se entenderá que nos referimos al primer valor de la lista

```
>>> print(mi_lista[:2])
[10, 'hola']
```

y si omitimos el segundo que queremos llegar hasta el elemento final

```
>>> print(mi_lista[2:])
['Juan', 25]
```

Si omitimos los dos tendremos la lista entera

```
>>> print(mi_lista[:])
[10, 'hola', 'Juan', 25]
```

También podemos incluir un tercer número en la sublista (separado por otro **:**) que indicará cada cuantos elementos contamos al elegirlos (lo que sería el "paso"). Por defecto este número es 1, pero podría ser otro.

```
>>> print(mi_lista[::2])
[10, 'Juan']
```

Una utilidad de este tercer parámetro es invertir el orden de nuestra lista usando **-1**. Por ejemplo:

```
>>> print(mi_lista[::-1])
[25, 'Juan', 'hola', 10]
```

Operaciones sobre listas

Si nuestra lista está formada por elementos homogéneos (todo números o todo texto) podremos realizar una serie de operaciones sobre ellas:

- **Ordenación**: podemos ordenar una lista con *mi_lista.sort()*. Una vez ordenada sabremos que el primero o el último serán el mínimo o el máximo valor.
    - Podemos ordenarla en orden inverso con *mi_lista.sort(reverse=True)* para . Conviene recalcar que la operación se realiza sobre la propia lista, no es que la llamada nos devuelva una lista ordenada.
    - Por defecto la ordenación de cadenas distingue entre mayúsculas y minúsculas,

```
>>> palabras = ['hola','ola','Hola','hOLA']
>>> palabras.sort()
>>> palabras
['Hola', 'hOLA', 'hola', 'ola']
```

- Podemos hacer que no dependa de mayúsculas/minúsculas usando una función de ordenación diferente, lo que indicaremos con el argumento **key**, por ejemplo usando la función *str.lower* (que usará todos los caracteres como minúsculos antes de hacer la comparación)

```
>>> palabras = ['hola','ola','Hola','hOLA']
>>> palabras.sort(key=str.lower)
>>> palabras
['hola', 'Hola', 'hOLA', 'ola']
```

- Para que no dependa de si la primera letra es mayúscula usaremos *str.capitaliza* que usa el primer carácter de cada cadena como mayúscula

```
>>> palabras = ['Hola','ola','hola','hOLA']
>>> palabras.sort(key=str.capitalize)
>>> palabras
['Hola', 'hola', 'hOLA', 'ola']
```

- Obtener el máximo aplicando la función **max** *max(mi_lista)*.
- Obtener el mínimo aplicando la función **min** *min(mi_lista)*.
- Calcular su longitud con **len** *len(mi_lista)*.

Búsquedas

Una funcionalidad que usaremos muchos son las búsquedas, que podremos hacer con la palabra reservada **in**. Obtendremos como resultado un booleano, siendo *True* que se ha encontrado y *False* en caso contrario.

Conviene destacar que se distingue entre mayúsculas y minúsculas.

```
>>> mi_lista = ['Juan', 'Dionisio', 'Pepe', 'Manolo']
>>> 'Juan' in mi_lista
True
```

Una vez que sabemos que el elemento está en la lista podemos preguntar en qué posición con **index**:

```
>>> posicion = mi_lista.index("Juan")
0
```

Si intentamos buscar un elemento que no está se produce una excepción de tipo **ValueError**.

También podemos contar el número de veces que se repite un valor con **count** :

```
>>> mi_lista.count("Juan")
1
>>> mi_lista.count("juan")
0
```

Como vemos podemos usar la función *count* para ver si un elemento está o no en una lista.

Edición en listas

Hemos visto cómo cambiar, buscar, contar o acceder a elementos de una lista. Pero se pueden hacer más operaciones como las siguientes, a las que llamaremos a partir de nuestra lista original

- **append(elemento)**: Añade un elemento al final de la lista.
- **clear()**: borra todos los elementos de una lista.
- **insert(posicion, valor)**: inserta en *posicion* el valor indicado, desplazando los siguientes.
- **remove(elemento)**: elimina "**elemento**" de la lista. Para esto mismo podemos usar el "arcaísmo" **del**:

```
del lista[4]
```

- **copy()**: devuelve una copia de la lista. Podemos usarlo para crear una nueva si no queremos alterar la original.
- **reverse(lista)**: invierte el orden de la lista.
- **pop()**: Devuelve el último elemento de la lista y lo retira de la lista. También podemos usarlo para eliminar un elemento indicando el índice

```
lista.pop(2)
```

Esta función nos muestra que podemos usar una lista como soporte para un funcionamiento más complejo como sería una cola o pila, donde vamos eliminando el último elemento.

- **extend(lista2)**: añade a la lista actual el contenido de lista2. Es equivalente a hacer una suma de las dos listas:

```python lista1.extend(lista2) lista1 = lista1 + lista2

* **sort()**: Ordena la lista. Si tenemos una lista que mezcla distintos tipos, por ejemplo números y letras nos dará una excepción de tipo **TypeError**.

También podemos ordenar una lista con la función **sorted**:
```python
sorted(list)
```

Recorrer una lista: enumerar

Uno de los usos más frecuentes de las listas es recorrer sus elementos. Ya dijimos que los bucles *for* están pensado para recorrer colecciones. Veamos cómo hacerlo con una lista:

```
amigos = ['Juan', 'Dionisio', 'Pepe', 'Manolo']
for amigo in amigos:
    print(f'Hola {amigo}')
```

A este proceso de recorrer una lista se le denomina enumerar o iterar. A veces necesitamos además de iterar el indicar la posición del elemento. Podemos hacerlo con la palabra reservada **enumerate**, obteniendo tanto el elemento como su posición:

```
amigos = ['Juan', 'Dionisio', 'Pepe', 'Manolo']
for posicion, amigo in enumerate(amigos): # Crea una variable "posicion" y otra "amigo" donde va poniendo el índice y el valor de los elementos
    print(posicion, amigo)
```

Este ejemplo nos muestra que podemos utilizar varias variables en un mismo bucle for, algo que haremos con frecuencia.

Creación dinámica de listas

En ocasiones necesitamos generar una lista a partir de otra, realizando una operación sobre sus elementos. Podemos hacerlo con lo que ya sabemos, recorriendo nuestra lista y añadiendo los elementos a una nueva lista:

```
pi = 3.14
```
```

```
radios = [2.0, 35, 7.1]
circunferencia = [] # Creamos la lista vacía
for radio in radios:
    circunferencia.append(radio*pi)
```

Pero como ésto es algo muy frecuente, Python incorpora una manera muy cómoda de hacerlo, donde especificamos dentro de la lista la manera en la que calculamos cada elemento a partir de un bucle for:

```
pi = 3.14
radios = [2.0, 35, 7.1]
circunferencias = [pi * num for num in radios]
```

MATRICES

Hemos dicho que una lista puede contener cualquier cosa. De hecho podemos hacer una lista que contenga listas, es lo que llamamos una matriz. Veamos un ejemplo

```
fila1 = [1, 3, 5]
fila2 = [9, 3, 5]
fila3 = [7, 3, 8]

matriz = [fila1, fila2, fila3]

print(matriz) # imprime la matriz completa

print(matriz[1][2]) # imprime un solo elemento el 3º de la fila 2º
```

Como vemos necesitamos 2 índices para acceder a un elemento dado.