# LISTAS

En esta sección del curso de [Python para principiantes de Microsoft](https://learn.microsoft.com/es-es/training/modules/intro-python-lists/), que ya realizasteis, se encuentra la definición y uso básico de las listas.

En esta sección profundizaremos en el uso de este tipo de estructuras.

## Listas multidimensionales

Una lista puede tener tipos simples (textos, números, etc) o tipos complejos (como por ejemplo, otras listas, o tuplas, u objetos de clases como veremos más adelante). Aquí vemos una lista que internamente contiene otras listas:

datos = [

["Nacho", 40, 233],

["Pepe", 70, 231],

...

]

En este caso (cuando una lista contiene otras listas, o tuplas), la posición numérica dentro de la lista nos llevará a la lista que ocupa esa posición, con lo que necesitaremos otro índice numérico para indicar qué dato de esa lista o nos interesa. Así, para el ejemplo anterior, datos[0][0] haría referencia al primer elemento de la primera lista (el nombre "Nacho"), y datos[1][2] al tercer dato de la segunda lista (el valor 231).

**AE306.1 Operaciones con matrices en Python.**

En esta práctica se trabajará con matrices representadas mediante listas anidadas en Python. Todo el código debe ir acompañado de comentarios y capturas de pantalla que muestre su ejecución en el terminal.

Crea un archivo denominado matrices.py que contendrá un conjunto de funciones relacionadas con matrices.

* Crea la función es\_cuadradaM (matriz), que devuelva un True si la matriz es cuadrada, False, en otro caso.

**def es\_cuadradaM (matriz):**

# una matriz implementada por listas es una lista de listas.

# primero debo saber cuántos elementos (filas) tiene la matriz [(len)](https://geekflare.com/es/python-len-function/)

# comparamos el valor de los numero de filas con el de las columnas.

necesito saber cuantas filas tiene la matriz y comparar

En Python, casi cualquier tipo de dato puede evaluarse como True o False en un contexto booleano, siguiendo estas reglas generales:

* + False: Se considera False en Python:
* None
* 0 (tanto 0, 0.0, como 0+0j en números complejos)
* Cadenas vacías ("")
* Listas, tuplas, conjuntos y diccionarios vacíos ([], (), set(), {})
  + True: Todo lo que no esté en la lista anterior se evalúa como True.
* Crea la función identidad(matriz) que indique si una matriz es identidad, es decir, cuadrada y tiene en su diagonal todo 1.

## Construcción de listas

Podemos crear listas en la propia definición de la variable, como hemos hecho en el ejemplo anterior. Sin embargo, puede haber casos en los que se necesite crear de manera dinámica, es decir, añadiendo o eliminando elementos.

Si queremos añadir un elemento **al final** de la lista usamos la instrucción **append**. En el caso de querer insertar un elemento **en una posición determinada**, usamos la instrucción **insert**, indicando en qué posición queremos insertar, y el dato que queremos insertar. Automáticamente, todos los elementos a la derecha de esa posición se desplazarán para hacer hueco al nuevo elemento. Finalmente, la instrucción **extend** permite añadir toda una lista al final de otra existente.

Finalmente, si queremos **eliminar** un dato de la lista:

* Usamos la instrucción **del** o **pop** para indicar la posición del elemento a borrar. Además, con pop podemos obtener el elemento borrado. La función pop sin argumentos borra y devuelve el último elemento de la lista. Con argumento, borra y devuelve el elemento que ocupa esa posición.
* Usamos **remove** para indicar el valor del elemento a borrar. Se borrará la primera ocurrencia de ese dato.
* Usamos **clear** para eliminar todos los elementos de la lista

Aquí vemos un ejemplo completo de estas instrucciones:

datos = [1, 2, 3]

datos.**append**(1000) # [1, 2, 3, 1000]

datos.**insert**(1, 20) # [1, 20, 2, 3, 1000]

datos.**extend**([2000, 3000]) # [1, 20, 2, 3, 1000, 2000, 3000]

datos[0] = 100 # [100, 20, 2, 3, 1000, 2000, 3000]

**del** datos[2] # [100, 20, 3, 1000, 2000, 3000]

datos.**remove**(20) # [100, 3, 1000, 2000, 3000]

borrado = datos.pop(0) # datos = [3, 1000, 2000, 3000], borrado = 100

datos.clear() # []

La instrucción list(valor) **convierte un valor secuenciable en una lista de valores**. Entendemos por valor secuenciable aquel que se puede convertir en una secuencia de elementos. Por ejemplo, un texto se puede convertir en una secuencia de letras o caracteres.

datos = list('123')

print(datos) # ['1', '2', '3']

La operación lista1 **+** lista2 **concatena** los datos de dos listas.

datos1 = [1, 2, 3, 4]

datos2 = [4, 5, 6]

datosTotales = datos1 + datos2 # [1, 2, 3, 4, 4, 5, 6]

La operación lista \* N genera una nueva lista donde los elementos de la lista original aparecen **repetidos N veces**

datos = [4, 5, 6]

datosRepetidos = datos \* 3 # [4, 5, 6, 4, 5, 6, 4, 5, 6]

### Construcción de listas con for

Podemos usar el bucle *for* para generar fácilmente listas que cumplan un patrón determinado. Es lo que se conoce como ***list comprehension***.

Ejecuta e indica que hacen los siguientes instrucciones:

importa random

nueva\_lista = [ random.randint(1,9) for c in range(5)]

multiplos = [x for x in range(101) if x % 5 == 0 and x % 7 == 0]

numeros = list(range(1, 11))

**AE306.2 Operaciones con matrices en Python.**

* Crea la función crear\_matrizv1(nfil,ncol,val), que devuelva una matriz de nfilas con ncolumnas y con valores aleatorios entre 1 y val.
* Refactoriza la función anterior utilizando *list comprehension*.
* Crea la función matriz\_transpuestav1 (matriz), que devuelva la matriz transpuesta.

## Añadir, modificar y borrar elementos

La expresión n in lista comprueba **si el dato *n* está en la lista**

datos = [4, 5, 6]

if 4 in datos:

print("Existe el dato 4")

Las instrucciones **max**(lista) y **min**(lista) obtienen el **mayor / menor valor de la lista**, respectivamente. La función sum calcula la suma total de los elementos indicados.

datos = [4, 5, 6]

print(max(datos)) # 6

print(sum(datos)) # 15

La instrucción lista.count(objeto) obtiene el **número de apariciones** del objeto en la lista.

datos = [4, 5, 6, 4]

print(datos.count(4)) # 2

La instrucción lista.**index**(objeto) obtiene la **posición donde aparece** por primera vez el objeto en la lista. Si el elemento no se encuentra, se produce una excepción en el programa. Podemos pasarle como segundo dato desde qué posición queremos seguir buscando.

datos = [4, 5, 6, 4]

print(datos.index(4)) # 0

print(datos.index(4, 1)) # 3

La instrucción lista.**sort**(funcion) **ordena una lista** según el criterio especificado en la función indicada. Para listas de datos simples (listas de enteros, de strings...) no hace falta indicar ninguna función: las listas se ordenan automáticamente de menor a mayor. Si queremos una ordenación inversa, usamos un parámetro adicional llamado reverse:

datos = [4, 2, 7, 5]

datos.sort() # [2, 4, 5, 7]

datos.sort(reverse=True) # [7, 5, 4, 2]

Como vemos, la instrucción sort deja la lista original ordenada según el criterio establecido. Si queremos mantener intacta la lista original y dejar el orden en otra lista paralela, es mejor usar la instrucción **sorted**, de este modo:

datos\_ordenados = sorted(lista)

La instrucción lista.**reverse**() **invierte el orden de la lista**. Este método NO devuelve una nueva lista, sino que afecta a la original. Si queremos mantener el orden original y que la lista invertida sea otra nueva, podemos usar la instrucción lista[::-1] y asignar el valor a otra variable.

datos = [1, 2, 3, 4]

datos.reverse() # [4, 3, 2, 1]

datosInvertidos = datos[::-1] # [1, 2, 3, 4]

Podemos también **obtener sublistas** de una lista, igual que se obtienen subcadenas de una cadena: indicando entre corchetes, separados por dos puntos, la posición inicial y la final que obtener (la posición final NO se incluye en la sublista). Estas posiciones pueden indicarse con números positivos o negativos (en este caso, se cuentan desde el final de la lista). También podemos no indicar uno de los dos extremos (con lo que se obtiene desde el inicio o hasta el final de la lista, respectivamente), o no indicar ningún extremo (con lo que se obtiene una copia de la lista original).

datos = [1, 10, 100, 1000]

sublista = datos[1:3] # [10, 100]

sublista2 = datos[2:] # [100, 1000]

sublista3 = datos[:2] # [1, 10]

sublista4 = datos[:] # Copia de la lista original

sublista5 = datos[-3:] # 3 últimos = [10, 100, 1000]

**AE306.3 Operaciones con matrices en Python.**

* RETO: Define una función que permita calcular el determinante de una matriz de n dimensiones; determinante\_matriz(matriz1)