UD4 - Listas y tuplas

Programación

Introducción

En las dos siguientes UD vamos a estudiar las principales estructuras de datos de Python: listas y tuplas en esta UD y mapas y conjuntos en la siguiente UD.

Estructura de datos: forma organizada de almacenar y gestionar datos para facilitar su acceso, manipulación y uso eficiente en un programa o sistema informático.

Listas

Lista: estructura de datos ordenada, existe una relación de orden entre los elementos. Todo elemento de la lista es identificado de forma inequívoca por un número que recibe el nombre de índice.

Un ejemplo de lista en la vida real sería la clasificación de la liga de fútbol:

1	Real Madrid	21	15	4	2	50	20	30	**49**
2	Atlético Madrid	21	13	6	2	35	14	21	**45**
3	Barcelona	21	13	3	5	59	24	35	**42**
4	Athletic	21	11	7	3	31	18	13	**40**
5	Villarreal	21	9	7	5	39	32	7	**34**
6	R.C.D. Mallorca	21	9	3	9	19	26	-7	**30**
7	Rayo Vallecano	21	7	8	6	25	24	1	**29**
8	Girona	21	8	4	9	29	29	0	**28**
9	Real Sociedad	21	8	4	9	17	17	0	**28**
10	Betis	21	7	7	7	23	26	-3	**28**
11	Osasuna	21	6	9	6	25	30	-5	**27**
12	Sevilla	21	7	6	8	24	30	-6	**27**

En dicha lista los elementos son equipos de fútbol y cada uno de los elementos (equipos de fútbol) es identificado de forma inequívoca por un índice (posición en la liga).

Crear listas

Para crear una lista utilizamos los corchetes []. Dentro de los corchetes ponemos los elementos que formarán la lista separados por comas.

```
# Creo una lista de números
lista1 = [4, 6, 9, 3, 2]

# Creo una lista con cadenas de texto
lista2 = ["Granada", "Málaga", "Almería"]

# Creo una lista de booleanos
lista3 = [True, False, True, False]

# En Python las listas pueden contener elementos de distintos tipos
# Por ejemplo la siguiente lista contiene cadenas de texto y números
# En concreto contiene el nombre de las provincias de Andalucía seguido por su población
andalucia = ["Almería", 753920, "Cádiz", 1250539, "Córdoba", 773997, "Granada", 930181, "Huelva", 530824, "Jaén", 620242, "Málaga", 1751600, "Sevilla", 1957210]
```

Ejercicio. Crea una lista que contenga el nombre de 5 países de Europa.
Ejercicio. Crea una lista con el once inicial del equipo de fútbol del equipo que más te guste.
Ejercicio. Crea una lista que contenga elementos de todos los tipos de datos estudiados hasta el momento.

De momento nuestras listas contendrán elementos de tipos base: número, texto y booleanos, pero los elementos de una lista pueden ser de cualquier tipo, por ejemplo objetos de clases (lo veremos en la UD6) o incluso otras listas (listas dentro de listas, lo veremos al final del tema).

Lectura y escritura de listas

Para acceder a los elementos de una lista especificamos el nombre de la lista seguido de unos corchetes y dentro de los corchetes especificamos la posición (índice) del elemento teniendo en cuenta que el primer elemento tiene asociado el índice 0.

```
# Lista con el nombre de las provincias de Andalucía
andalucia = ["Almería", "Cádiz", "Córdoba", "Granada",
"Huelva", "Jaén", "Málaga" , "Sevilla"]

# Accedemos al primer elemento
andalucia[0]

# Hemos accedido pero no hemos hecho nada con el elemento
# Vamos a imprimirlo por pantalla
print(andalucia[0])

# Accedemos al cuarto elemento y lo mostramos por pantalla
print(andalucia[3])
```

Ejercicio. Crea una lista con tus 5 frutas favoritas ordenadas de más favorita a menos favorita. Muestra tu fruta favorita y la menos favorita de las 5.
Ejercicio. Si una lista tiene n elementos. ¿Cuáles son los índices de la lista?

También puedes utilizar los corchetes para acceder a una letra concreta de una cadena de texto, de hecho las cadenas de texto internamente son listas cuyos elementos son letras:

```
nombre = "Borja"
print(f"Me llamo {nombre} y la primera letra de mi nombre es
{nombre[0]}")
```

Para modificar los elementos de una lista primero los seleccionamos y después le asignamos un nuevo valor.

```
# Lista con mi frutas favoritas
frutas_favoritas = ["Mandarina", "Plátano", "Manzana",
"Fresa", "Pera"]

# De repente sufro una terrible intolerancia a las fresas
# Comer una sola fresa acabaría con mi vida
# Por lo tanto debo actualizar la lista

frutas_favoritas[3] = "Pera"
frutas_favoritas[4] = "Albaricoque"
```

Ejercicio. Dada la siguiente lista: andalucia = ["Almería", "Kádiz", "Córdoba", "Granada", "Huelva", "Jaén", "Málaga" , "Sevilla"], modifica el segundo elemento para que esté bien escrito (Cádiz)

Tamaño de una lista

Habrá muchas ocasiones en las que necesitamos saber el tamaño de una lista. Utilizamos la función len() la cual recibe como argumento una variable compuesta por elementos (en nuestro caso una lista) y nos devuelve el número de elemento de la lista.

```
# Lista con el nombre de las provincias de Andalucía
andalucia = ["Almería", "Cádiz", "Córdoba", "Granada",
"Huelva", "Jaén", "Málaga" , "Sevilla"]

# ¿Cuántas provincias tiene Andalucía?
n_provincias = len(andalucia)
print(f"Andalucía tiene {n_provincias} provincias")
```

Recorrer listas

Para recorrer listas utilizamos bucles for:

```
# Lista con el nombre de las provincias de Andalucía
andalucia = ["Almería", "Cádiz", "Córdoba", "Granada",
"Huelva", "Jaén", "Málaga" , "Sevilla"]

print("Las provincias de Andalucía son:")
for provincia in andalucia:
   print(provincia)
```

Pro tip: Usa siempre nombres descriptivos que indiquen que son los elementos de la lista. En el ejemplo anterior hemos utilizado el nombre provincia porque lo que estamos recorriendo son provincias. Utiliza nombres como x o e (de element) solo cuando los elementos de la lista puedan ser cualquier cosas.

Ejercicio. Dado el siguiente código:

¿Qué opinas de los nombres elegidos para las variables (jugador, x, e)? ¿Están bien escogidos? ¿Pondrías otros?

```
frutas = ["Pera", "Manzana", "Plátano", "Fresa"]
jugadores = ["Iniesta", "Iker Casillas", "Xavi"]
elems = [-2323, "Ax5", True, "tri"]

for fruta in frutas:
   print(fruta)

for x in jugadores:
   print(x)

for e in elems:
   print(e)
```

Ejercicio. Crea una lista con tus 5 películas favoritas. Muéstralas por pantalla utilizando un bucle for para recorrer la lista.

Ejercicio 1. Crea una lista con las provincias de Andalucía y muestra por pantalla solo aquellas que empiezan por la letra C.
Ejercicio 2. Crea una lista con el nombre de los planetas del sistema solar. Muestra por pantalla aquello que empiecen por la letra M.
Ejercicio 3. Crea una lista con 10 números (los que quieras pero que no estén ordenados). Muestra por pantalla solo aquellos números que sean mayores al primero.
Ejercicio 4. Crea una lista con 10 números (los que quieras). Muestra por pantalla la media de dichos números.
Ejercicio 5. Utilizando ChatGPT crea una lista de 1000 números aleatorios. Muestra por pantalla el menor de los números (utilizando bucles, la gracias es que encuentre el menor el ordenador, no tú).
Ejercicio 6. Encuentra el mayor de los números de la lista anterior.
Ejercicio 7. Crea una variable con tu nombre completo (nombre + apellidos) y muestra por pantalla, en orden de aparición, las vocales que hay en tu nombre completo.
Ejercicio 8. Dada la siguiente lista con las precipitaciones en Granada mensuales del año 2023:

```
precipitaciones_granada_2023 = [
    40.2,  # Enero
    35.6,  # Febrero
    45.8,  # Marzo
    50.1,  # Abril
    30.3,  # Mayo
    10.4,  # Junio
    1.2,   # Julio
    5.6,   # Agosto
    20.8,  # Septiembre
    60.5,  # Octubre
    55.3,  # Noviembre
    42.7   # Diciembre
]
```

Encuentra el valor medio de precipitaciones mensuales en Granada

Ejercicio 9. Crea una lista con el nombre de todos los países de Europa. Muestra por pantalla el nombre de aquellos países que tengan 6 o menos letras.
Ejercicio 10. Crea una lista con los nombres de los jugadores de tu equipo de fútbol favorito. Muestra por pantalla el nombre de aquellos jugadores cuyo nombre empiece por la misma letra que acaba.

Recorrido mediante índice

Habrá ocasiones en las que nos interese recorrer una lista mediante el índice en vez de acceder directamente al elemento. Por ejemplo para resolver el siguiente ejercicio: Dada una lista con lo equipos de la liga ordenados según la clasificación:

```
liga = ["Real Madrid", "Atlético de Madrid", "FC Barcelona", "Athletic Club",  "Villarreal CF", "RCD Mallorca", "Rayo Vallecano", "Girona FC", "Real Sociedad", "Real Betis", "CA Osasuna", "Sevilla FC", "RC Celta", "Getafe CF", "UD Las Palmas", "CD Leganés", "Deportivo Alavés", "RCD Espanyol", "Valencia CF", "Real Valladolid"]
```

Mostrar por pantalla la siguiente información:

```
Clasificación de la li
1. Real Madrid
2. Atlético de Madrid
3. FC Barcelona
4. Athletic Club
5. Villarreal CF
6. RCD Mallorca
7. Rayo Vallecano
8. Girona FC
9. Real Sociedad
10. Real Betis
11. CA Osasuna
12. Sevilla FC
13. RC Celta
14. Getafe CF
15. UD Las Palmas
16. CD Leganés
17. Deportivo Alavés
18. RCD Espanyol
19. Valencia CF
20. Real Valladolid
```

Con lo que sabemos hasta ahora haríamos:

```
liga = ["Real Madrid", "Atlético de Madrid", "FC Barcelona",
"Athletic Club",  "Villarreal CF", "RCD Mallorca", "Rayo
Vallecano", "Girona FC", "Real Sociedad", "Real Betis", "CA
Osasuna", "Sevilla FC", "RC Celta", "Getafe CF", "UD Las
Palmas", "CD Leganés", "Deportivo Alavés", "RCD Espanyol",
"Valencia CF", "Real Valladolid"]

print("Clasificación de la liga:")
pos = 1
for equipo in liga:
   print(f"{pos}. {equipo}")
   pos += 1
```

Pero Python nos ofrece una forma más cómoda de resolverlo:

```
liga = ["Real Madrid", "Atlético de Madrid", "FC Barcelona",
"Athletic Club",  "Villarreal CF", "RCD Mallorca", "Rayo
Vallecano", "Girona FC", "Real Sociedad", "Real Betis", "CA
Osasuna", "Sevilla FC", "RC Celta", "Getafe CF", "UD Las
Palmas", "CD Leganés", "Deportivo Alavés", "RCD Espanyol",
"Valencia CF", "Real Valladolid"]

print("Clasificación de la liga:")
for i, equipo in enumerate(liga):
   print(f"{i+1}. {equipo}")
```

Ejercicio 11. Dada la siguiente lista:

```
liga = ["Real Madrid", "Atlético de Madrid", "FC Barcelona",
"Athletic Club",  "Villarreal CF", "RCD Mallorca", "Rayo
Vallecano", "Girona FC", "Real Sociedad", "Real Betis", "CA
Osasuna", "Sevilla FC", "RC Celta", "Getafe CF", "UD Las
Palmas", "CD Leganés", "Deportivo Alavés", "RCD Espanyol",
"Valencia CF", "Real Valladolid"]
```

Encuentra la posición en la lista del Real Betis

Ejercicio 12. Dada la siguiente lista:

```
precipitaciones_granada_2023 = [
    40.2,  # Enero
    35.6,  # Febrero
    45.8,  # Marzo
    50.1,  # Abril
    30.3,  # Mayo
    10.4,  # Junio
    1.2,   # Julio
    5.6,   # Agosto
    20.8,  # Septiembre
    60.5,  # Octubre
    55.3,  # Noviembre
    42.7   # Diciembre
]
```

Encuentra el nombre del mes en el que más llueve. Para ello crea una segunda lista con el nombre de los meses.

Ejercicio 13. Utilizando las listas del ejercicio 12 encuentra el nombre de los meses en los que llueve más que la media.

Ejercicio 14. Utilizando las listas del ejercicio 12 encuentra el nombre de los meses en los que llovió más que en el anterior mes.

Ejercicio 15. Dadas las siguientes listas (la segunda contiene la masa de los planetas), encuentra el nombre del planeta de mayor masa y el de aquellos planetas cuya masa es superior a la masa media del sistema solar.

```
# Lista con los nombres de los planetas del sistema solar
planetas_sistema_solar = [
    "Mercurio", "Venus", "Tierra", "Marte",
    "Júpiter", "Saturno", "Urano", "Neptuno"
]

# Lista con las masas de los planetas en kilogramos (valores aproximados)
masas_sistema_solar = [
    3.3011e23,  # Mercurio
    4.8675e24,  # Venus
    5.97237e24, # Tierra
    6.4171e23,  # Marte
    1.8982e27,  # Júpiter
    5.6834e26,  # Saturno
    8.6810e25,  # Urano
    1.02413e26  # Neptuno
]
```

El programa deberá mostrar:

```
Los planetas cuya masa es mayor a la masa media (3.334468362500001e+26) son:
Júpiter
Saturno
```

Métodos de las listas

En la UD6 estudiaremos a fondo qué es un método, para esta UD nos basta con saber que las listas tienen métodos que nos permiten realizar acciones con las listas como insertar o eliminar elementos. Para usar los métodos de las listas usamos la sintaxis nombre_lista.nombre_metodo(parámetros…).

.append(x)

Añade un elemento al final de la lista.

```
import random

# Ejemplo de cómo crear una lista con 1000 números aleatorios
entre 0 y 10
# Comenzamos creando una lista vacía
nums = []

# Ahora creamos 1000 números aleatorios entre 0 y 10 y los vamos
insertando en la lista
for _ in range(1000):
   nums.append(random.randint(1, 10))

print(nums)
```

Ejercicio 16. Crea un programa con una lista que contenga el resultado de haber lanzado 10 veces una moneda al aire (cara o cruz).
Ejercicio 17. Crea un programa que lea números hasta que insertes un 0. Al terminar de leer números deberá mostrar por pantalla aquellos números insertados que sean superiores a la media.
Ejercicio 18. Escribe un programa que genere una lista con los pares comprendidos entre a y b. Los parámetros a y b son leídos desde el teclado. (Hacerlo sin listas y con listas)

.insert(i, x)

Inserta un elemento x en la posición i.

```
equipos = ["Real Madrid", "FC Barcelona", "Villareal"]
# Insertamos el Atlético de Madrid entre Real Madrid y FC
Barcelona
equipos.insert(1, "Atlético de Madrid")
print(equipos)
```

Ejercicio 19. Crea una lista llamada nombres con los valores ["Ana", "Carlos", "Lucía"]. Usa insert() para agregar "Beatriz" entre "Ana" y "Carlos". Usa insert() para agregar "Diego" al final de la lista. Imprime la lista después de cada operación.

Ejercicio 20. Crea un programa para gestionar una lista de tareas pendientes. La lista comenzará con las siguientes tareas: `"Comprar fruta"` `"Estudiar programación"` `"Desinstalar el LOL"` . El programa permitirá ver la lista de tareas pendientes e insertar nuevas tareas en la posición que se desee.

Al iniciarse el programa:

```
Bienvenido a la app de listas to-do más avanzada del universo
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
```

Al mostrar la lista hazlo con el siguiente formato:

```
Bienvenido a la app de listas to-do más avanzada del universo
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
1
--------------------
LISTA TAREAS PENDIENTES:
1. Comprar fruta
2. Estudiar programación
3. Desinstalar el LOL
--------------------
```

Para insertar una tarea:

```
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
2
¿Qué tarea quieres insertar?Pincharle las ruedas al coche del profe
¿En que posición?2
```

Ahora cuando mostremos de nuevo la lista veremos la nueva tarea:

```
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
2
¿Qué tarea quieres insertar?Pincharle las ruedas al coche del profe
¿En que posición?2
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
1
----------------------
LISTA TAREAS PENDIENTES:
1. Comprar fruta
2. Pincharle las ruedas al coche del profe
3. Estudiar programación
4. Desinstalar el LOL
----------------------
```

Ejercicio 21. Crea un programa que inserte números en una lista hasta que insertes el número 0 para terminar. Los números deberán estar ordenados de menor a mayor. Prohibido utilizar el método .sort().
Ejemplo de ejecución:

```
Inserta un número: 10
Inserta un número: 5
Inserta un número: 15
Inserta un número: 2
Inserta un número: 1
Inserta un número: 7
Inserta un número: 33
Inserta un número: 12
Inserta un número: 0
Los números insertados ordenados de menor a mayor son[1, 2, 5, 7, 10, 12, 15, 33]
```

.extends([x])

Añade los elementos de la lista x pasada como argumento al final de la lista que invoca el método.

```
andalucia_oriental = ["Granada", "Jaén", "Almería", "Málaga"]
andalucia_occidental = ["Sevilla", "Cádiz", "Huelva",
"Córdoba"]
andalucia = []
andalucia.extend(andalucia_occidental)
andalucia.extend(andalucia_oriental)
print(andalucia)
```

Ejercicio 22. Crea una lista con las asignaturas del primer curso DAW y otra con las del segundo curso. Usando .extend crea una tercera lista que contenga todas las asignaturas de DAW.

.remove(x)

Elimina la PRIMERA aparición del elemento x en una lista.

```
# Creamos una lista con las provincias de Andalucía
andalucia = ["Almería", "Cádiz", "Córdoba", "Granada",
"Huelva", "Jaén", "Málaga", "Sevilla"]
# Imaginemos un futuro distópico en el que Portugal invade
Huelva
andalucia.remove("Huelva")
print(andalucia)
```

Ejercicio 23. Crea un programa que permita insertar números naturales* en una lista. Cada vez que insertes un número deberá mostrar la lista entera. Si se inserta un número negativo deberá eliminar la primera aparición de su análogo positivo de la lista, es decir si inserto -3 significa "elimina el primer tres insertado"

* Números enteros entre 1 e infinito

Ejemplo de ejecución:

```
Inserta un número: 4
Números introducidos: [4]
Inserta un número: 7
Números introducidos: [4, 7]
Inserta un número: 8
Números introducidos: [4, 7, 8]
Inserta un número: -7
Números introducidos: [4, 8]
Inserta un número: 3
Números introducidos: [4, 8, 3]
Inserta un número: 11
Números introducidos: [4, 8, 3, 11]
Inserta un número: -8
Números introducidos: [4, 3, 11]
```

.pop([i])

Elimina (y devuelve) el elemento en la posición i. Si no se especifica una posición elimina el último elemento de la lista.

Ejercicio 24. Crea un programa que inserte números en una lista hasta introducir el número 0. A continuación deberá mostrar en pantalla todos los números introducidos en orden inverso, es decir primero el último número introducido.

Ejercicio 25. Amplía el programa del ejercicio 20 para que se puedan completar tareas y que al hacerlo desaparezcan de la lista. En concreto cuando se muestren las tareas pendientes haz que si el usuario inserta un número de tarea válido (0 para salir al menú principal) se complete dicha tarea:

```
Bienvenido a la app de listas to-do más avanzada del universo
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
1
--------------------
LISTA TAREAS PENDIENTES:
1. Comprar fruta
2. Estudiar programación
3. Desinstalar el LOL
--------------------
¿Has completado alguna tarea?2
Eres una crack!!! sigue así
--------------------
LISTA TAREAS PENDIENTES:
1. Comprar fruta
2. Desinstalar el LOL
--------------------
¿Has completado alguna tarea?9
La tarea no existe
--------------------
LISTA TAREAS PENDIENTES:
1. Comprar fruta
2. Desinstalar el LOL
--------------------
¿Has completado alguna tarea?0
1. Ver tareas pendientes
2. Agregar tarea pendiente
0. Salir
```

.count(x)

Cuenta el número de apariciones del elemento x en la lista.

Ejercicio 26. Crea un programa que inserte números en una lista. Si el número ya existe en la lista NO debe volver a introducirlo, de tal forma que la lista no contenga número repetidos:

Ejemplo de ejecución:

```
Inserta un número: 1
Contenido de la lista: [1]
Inserta un número: 3
Contenido de la lista: [1, 3]
Inserta un número: 4
Contenido de la lista: [1, 3, 4]
Inserta un número: 7
Contenido de la lista: [1, 3, 4, 7]
Inserta un número: 1
Contenido de la lista: [1, 3, 4, 7]
Inserta un número: 4
Contenido de la lista: [1, 3, 4, 7]
Inserta un número: 5
Contenido de la lista: [1, 3, 4, 7, 5]
Inserta un número: 7
Contenido de la lista: [1, 3, 4, 7, 5]
```

Ejercicio 27. Crea una lista con 1000 números aleatorios entre 1 y 100. Encuentra el número que MÁS veces se repite.

.index(x)

Devuelve la posición del elemento x. Si el elemento no existe se produce un error del tipo ValueError. Si el elemento está varias veces devuelve la posición de su primera aparición.

Ejercicio 28. Repite el ejercicio 11 pero utilizando el método .index(x)

Ejercicio 29. Crea un programa que tenga una lista con los equipos de la liga de fútbol ordenados según su posición en la liga. Haz que el programa lea el nombre de equipos por teclado, si el equipo existe devolverá su posición en la clasificación si no existe se informará que dicho equipo no existe pero el programa NO debe fallar.

Ejercicio 30. Crea un programa que contenga 3 listas: una con el nombre de los 8 planetas del Sistema Solar, otra con sus respectivas masas y otra con sus respectivos radios. Cuando insertes por teclado el nombre de un planeta

el programa deberá dar la siguiente información: su posición en el Sistema Solar, su masa y su radio.
Ejercicio 31. A partir del código del ejercicio 25 haz que se puedan insertar nuevas tareas en una determinada posición.

.sort()

Ordena los elementos de una lista. Para poder ordenar una lista TODOS los elementos de la lista deben ser de un mismo tipo y existir una relación de orden entre ambos. Los números, el texto y los booleanos tienen una relación de orden establecida. En los próximos temas aprenderemos a crear relaciones de órden de objetos que nosotros creemos.

Ejercicio 32. Crea un programa que lea números hasta que insertes un cero. Una vez se inserte el cero se mostrarán los números introducidos de menor a mayor (el cero no debe contar).
Ejercicio 33. Crea un programa que lea palabras hasta que insertes la palabra 'fin'. Una vez se inserte 'fin' se mostrarán las palabras introducidas en orden alfabético ('fin' no debe contar).

.reverse()

Invierte el orden de los elementos de una lista.
Ejercicio 34. Crea un programa que lea números hasta que insertes un cero. Una vez se inserte el cero se mostrarán los números introducidos en orden inverso, es decir primero el último número introducido y al final el primero.
Ejercicio 35. Crea un programa que lea números hasta que insertes un cero. Una vez se inserte el cero se mostrarán los números introducidos de mayor a menor (el cero no debe contar).
Ejercicio 36. Crea un programa que lea números hasta que insertes un cero. A continuación te volverá a pedir un número n y mostrará los últimos n número introducidos.

.copy()

Realiza una copia por valor de la lista. Si simplemente hacemos l2 = l1 entonces se realiza una copia por referencia. Explicación

.clear()

Elimina todos los elementos de una lista. Puede parecer un método algo 'inútil' pues para eliminar todos los elementos podríamos simplemente igualar la lista a una lista vacía pero… Lee este artículo y aprende.

Otras funciones útiles con listas

Además de los métodos de las listas existen una serie de funciones que te serán útiles a la hora de trabajar con las listas:

- max(l) → devuelve el elemento mayor de la lista l. Todos los elementos de la lista deben ser del mismo tipo y tener una relación de orden definida.
- min(l) → devuelve el elemento menor de la lista l. Todos los elementos de la lista deben ser del mismo tipo y tener una relación de orden definida.
- sum(l) → devuelve la sumatoria (suma acumulada) de todos los elementos de la lista. Todos los elementos de la lista deben ser del mismo tipo y tener el operador + definido (es decir tiene que tener sentido sumarlos).
- statistics.mean(l) → devuelve el valor medio de la lista. Todos los elementos de la lista deben ser del mismo tipo y tener el operador + definido (es decir tiene que tener sentido sumarlos).

Ejercicio 37. Crea un programa que lea números hasta que insertes un cero. Al finalizar el programa deberá mostrar:

- El mayor de los números introducidos
- El menor de los números introducidos
- La media de los números introducidos

Utiliza las funciones que acabamos de estudiar.

Ejercicio 38. Repite el ejercicio 15 pero utilizando las funciones que acabas de estudiar.

El método split para crear listas

Son dos métodos de las cadenas de texto utilizados para crear listas. El método split se usa para dividir una cadena (string) en una lista de subcadenas, utilizando un delimitador específico (por defecto, el espacio en blanco).
Ejemplo:

```
nombre = "Borja Molina Zea"
# Creo una lista con tres elementos: ["Borja", "Molina", "Zea"]
lista_nombre = nombre.split()
print(f"Mi primer apellido es {lista_nombre[1]}")
# Cadena que contiene los planetas separados por comas
planetas_txt = "Mercurio,Venus,La
Tierra,Marte,Júpiter,Saturno,Urano,Neptuno"
planetas = planetas_txt.split(",")
print(f"El primer planeta es {planetas[0]} y el último es
{planetas[-1]}")
```

Ejercicio 39. Crea un programa en el que insertes palabras separados por comas:

```
Inserta palabras separadas por coma: perro,oca,gato,rinoceronte
```

El programa deberá mostrar la palabras más corta y larga:

```
Inserta palabras separadas por coma: perro,oca,gato,rinoceronte
La palabra con menos letras es: oca
La palabra con más letras es: rinoceronte
```

Ejercicio 40. Crea un programa que lea números separados por un punto y coma. El programa deberá mostrar:

- El número mayor
- El número menor
- La suma de todos los números
- La media

```
Inserta números separados por ; --> 3;5;0;9;1;2;7
El número mayor es: 9
El número menor es: 0
La suma es: 27
La media es: 3.857142857142857
```

Slicing

Técnica que nos permite seleccionar un rango de elementos de la lista. El resultado de aplicar slicing a una lista es otra lista de igual o menor tamaño que la lista original. El operador slicing es ':' y se utiliza dentro de los corchetes.

Ejemplos:

```
liga = ["Real Madrid", "Atlético de Madrid", "FC Barcelona",
"Athletic Club","Villarreal CF","Rayo Vallecano","Girona FC","CA
Osasuna","RCD Mallorca","Real Betis Balompié","Real
Sociedad","Celta de Vigo","Sevilla FC","Getafe CF","UD Las
Palmas","RCD Espanyol","CD Leganés","Valencia CF","Deportivo
Alavés","Real Valladolid"]

# Seleccionamos los tres primeros equipos
print(liga[0:3])
# También podemos hacer
print(liga[:3])

# Seleccionamos los equipos entre la posición 5 y 10
print(liga[4:10])

# Seleccionamos los 4 últimos equipos
print(liga[-4:])

# Mostramos los equipos clasificados de la mitad hacía arriba
mitad = len(liga) // 2
print(liga[:mitad])
# Mostramos los equipos clasificados de la mitad hacia abajo
print(liga[mitad:])
```

Ejercicio 41. A partir de la lista del ejemplo muestra (utilizando slicing) los equipos que irán a la Champions League (los cuatro primeros), los que irán a la UEFA (quinto y sexto) y los que descenderán (los tres últimos).

```
A la champions van: ['Real Madrid', 'Atlético de Madrid', 'FC Barcelona', 'Athletic Club']
A la UEFA van: ['Villarreal CF', 'Rayo Vallecano']
A segunda van: ['Valencia CF', 'Deportivo Alavés', 'Real Valladolid']
```

List comprehensions

Hay dos tipos de acciones con listas que se repiten muchas veces:

- Filtrar los elementos de una lista según una condición
- Modificar los elementos de una lista

Ejemplos:

```
# Ejemplo de filtrado
# Dada una lista de números quédate solo con los números pares
nums = [3, 4, 5, 6, 3, 11, 33, 44, 64]
nums_pares = []
for n in nums:
    if n % 2 == 0:
        nums_pares.append(n)
print(nums_pares)

# Otro ejemplo de filtrado
# Dada una lista de palabras quédate con aquellas cuya longitud sea mayor
que 5
palabras = ["perro", "gato", "rinoceronte", "pez", "canario"]
palabras_largas = []
for palabra in palabras:
    if len(palabra) > 5:
        palabras_largas.append(palabra)
print(palabras_largas)

# Ejemplo de modificación de elementos de una lista
# Dada una lista de números, calcula el cuadrado de todos ellos
nums = [3, 4, 5, 6, 3, 11, 33, 44, 64]
nums_cuadrado = []
for n in nums:
    nums_cuadrado.append(n*n)
print(nums_cuadrado)

# Otro ejemplo de modificación
# Dada una lista de palabras poner en mayúscula la primera letra de todas
palabras = ["perro", "gato", "rinoceronte", "pez", "canario"]
palabras_capitalizadas = []
for palabra in palabras:
    palabras_capitalizadas.append(palabra.capitalize())
print(palabras_capitalizadas)
```

Todo lo anterior puede realizarse con list comprehensions en una pocas líneas:

```
# Ejemplo de filtrado
# Dada una lista de números quédate solo con los númeos pares
nums = [3, 4, 5, 6, 3, 11, 33, 44, 64]
nums_pares = [num for num in nums if num % 2 == 0]
print(nums_pares)

# Otro ejemplo de filtrado
# Dada una lista de palabras quédate con aquellas cuya longitud sea mayor
que 5
palabras = ["perro", "gato", "rinoceronte", "pez", "canario"]
palabras_largas = [palabra for palabra in palabras if len(palabra) > 5]
print(palabras_largas)

# Ejemplo de modificación de elementos de una lista
# Dada una lista de números, calcula el cuadrado de todos ellos
nums = [3, 4, 5, 6, 3, 11, 33, 44, 64]
nums_cuadrado = [n*n for n in nums]
print(nums_cuadrado)

# Otro ejemplo de modificación
# Dada una lista de palabras poner en mayúscula la primera letra de todas
palabras = ["perro", "gato", "rinoceronte", "pez", "canario"]
palabras_capitalizadas = [palabra.capitalize() for palabra in palabras]
print(palabras_capitalizadas)
```

Incluso podemos hacer las dos cosas a la vez:

```
# Dada una lista de números hallar el cuadrado de los números pares

# Sin List Comprehension
nums = [3, 4, 56, 75, 33, 12, 9, 32]
resultado = []
for n in nums:
    if n % 2 == 0:
        resultado.append(n*n)
print(resultado)

# Con List Comprehension
resultado = [n*n for n in nums if n%2==0]
print(resultado)
```

Nota: hacer los siguientes ejercicios con list comprehension obligatoriamente
Ejercicio 42. Crea una lista con palabras. Crea una segunda lista que contenga sólo las palabras que empiezan por a.
Ejercicio 43. Crea una lista con palabras. Crea una segunda lista que contenga las mismas palabras pero con todas sus letras en mayúscula.
Ejercicio 44. Crea una lista con 100 números aleatorios entre 0 y 10.
Ejercicio 45. Crea una lista con 100 números pares aleatorios entre 0 y 10.

Listas de listas

Como vimos la principio del tema las lista puede contener cualquier tipo de elemento, por lo tanto es posible tener una lista dentro de otra lista:

```
lista = [1, 3, [4, 5]]
```

Para acceder a los elementos de una lista dentro de otra lista debemos utilizar dos veces los [].
Para entenderlo comprende por qué se imprime lo que se imprime en el siguiente código:

```
lista = [1, 3, [4, 5]]
print(lista[0])
print(lista[1])
print(lista[2])
print(lista[2][0])
print(lista[2][1])
```

Podemos anidar listas de forma arbitraria:

```
lista = [1, 3, [4, ["hola", 3, 5, ["adios"]], 5]]
```

Matrices

Una matriz es una estructura de números organizados en filas y columnas. Una matriz puede representarse mediante listas anidadas (listas dentro de listas). Cada sublista representa una fila de la matriz.
Ejemplo:

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

print(matriz)  # Muestra la matriz completa
print(matriz[0])  # Muestra la primera fila: [1, 2, 3]
print(matriz[1][2])  # Accede al elemento en la fila 1, columna 2: 6
```

Para recorrer matrices debemos anidar un bucle for dentro de otro bucle for:

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

for fila in matriz:
    for elemento in fila:
        print(elemento, end=" ")
    print()
```

Dada la siguiente matriz:

```
matriz = [
    [4, -2, 3],
    [3, 4, 2],
    [7, 5, 9],
    [1, 0, -1]
]
```

Resuelve los siguientes ejercicios:
Ejercicio 46. Encuentra la suma de todos sus números. Hazlo primero sin usar sum() y luego usándolo
Ejercicio 47. Encuentra la fila y columna en la que se encuentra el número 5. Primero sin usar .index() y luego usándolo
Ejercicio 48. Transforma los números negativos a positivos.

Si estudiaste matemáticas en bachillerato estarás acostumbrado a que los elementos de una matriz sean números, pero en programación los elementos de una matriz también puede ser cadenas de texto, booleanos o incluso objetos (lo estudiaras en la UD 6)

Ejemplo de matriz de cadenas de texto:

```
matriz = [
    ["Perro", "Gato", "Hámster"],
    ["Loro", "Conejo", "Tortuga"],
    ["Pez", "Hurón", "Ardilla"],
    ["Iguana", "Serpiente", "Erizo"]
]
```

A partir de dicha matriz resuelve los siguientes ejercicios:
Ejercicio 49. Encuentra el animal con más letras de la lista
Ejercicio 50. Encuentra el número de animales que empiezan por vocal

Una matriz puede tener elementos de distinto tipo, por ejemplo la siguiente matriz contiene el nombre de 5 alumnos y sus notas en tres asignaturas (Programación, Ecología y Bases de Datos):

```
alumnos = [
    ["Juanillo", 4, 1, 5],
    ["Marta", 9, 10, 9],
    ["Ramoncín", 1, 5, 1],
    ["Gerardo", 5, 5, 5],
    ["Einstein", 10, 10, 10]
]
```

A partir de dicha matriz:
Ejercicio 51. Halla la nota media de cada alumno
Ejercicio 52. Halla el alumno cuya nota media sea la más baja
Ejercicio 53. Halla los alumnos que tengan al menos una asignatura suspensa
Ejercicio 54. Halla la nota media de toda la clase

Y para terminar:
Ejercicio 55. Crea una función que reciba cuatro parámetros:

- n → Número de filas
- m → Número de columnas
- a → Valor inicial
- b → Valor final

La función deberá devolver una matriz de nXm elementos de números aleatorios entre a y b

Tuplas

Una tupla es una lista de tamaño fijo. Se crean de la siguiente forma:

```
tupla = (1, 5, 7)
```

Los elementos se acceden forma que las listas:

```
tupla = (1, 5, 7)
print(tupla[0])
```

Los elementos de una tupla NO pueden ser modificados.
Los métodos de las listas que modifican su tamaño (por ejemplo .append o .remove) NO pueden usarse con tuplas. Los que NO modifican su tamaño (por ejemplo .index o .count) sí pueden utilizarse:

```
tupla = (1, 5, 3, 4, 7, 5)
print(tupla.count(5))
print(tupla.index(7))
```

Si las tuplas son una versión limitada de las listas… ¿qué razón hay para usarlas? El hecho de que usar tuplas es más eficiente, tanto en uso de CPU como en uso de RAM, que las listas.

Por lo tanto: SIEMPRE QUE SEPAS QUE EL NÚMERO DE ELEMENTOS ES FIJO Y QUE NO VAN A SER MODIFICADOS DEBES USAR TUPLAS EN VEZ DE LISTAS.

EN EL EXAMEN Y LA PRÁCTICA SE TENDRÁ EN CUENTA EL USO (O NO) DE TUPLAS. SI USAS LISTAS PUDIENDO USAR TUPLAS SERÁS PENALIZADO.