Ej. 1: Desarrollar la función **`estaEnLista`** qué recibe por parámetro un número entero **`num`** y una lista de enteros **`lst`**. La función debe retornar **`True`** si **`num`** se encuentra dentro de **`lst`**, caso contrario deberá retornar **`False.`**
`estaEnLista` deberá ser desarrollada (y probada) de tres formas distintas, según el siguiente criterio:

1.1. **`estaEnLista1`** *Utilizando sólo el operador IN para verificar si se encuentra un elemento en la lista.*

1.2. **`estaEnLista2`** *Recorriendo la lista con un ciclo* ***for*** *y realizando comparaciones elemento a elemento para verificar si se encuentra un elemento en la lista.*

1.3. **`estaEnLista3`** *Recorriendo la lista con un ciclo* ***while*** *y realizando comparaciones elemento a elemento para verificar si se encuentra un elemento en la lista.*

Nota: Se recomienda realizar pruebas directas (en intérprete en modo interactivo) como se indica a continuación:

Ejemplos:

```
>>> estaEnLista(23,[1,2,3,4])
False

>>> estaEnLista(23,[1,23,3,4])
True

>>> estaEnLista(23,[])
False
```

Ej. 2: Desarrollar la función **`cargarLista`**, que retorna una lista la cual será cargada desde el teclado con números **enteros positivos** (*lo cual deberá ser validado*) hasta que el usuario ingrese 0 (cero). Además, no se permitirá al usuario cargar dos veces el mismo valor. En caso de verificar un error de validación (*por repetido o por NO positivo*) se deberá mostrar un mensaje con el error que corresponda, permitiendo luego continuar con la carga de los siguientes números. Para determinar si un número se encuentra dentro de la lista se deberá verificar invocando a la función **`estaEnLista`** (*alguna de las realizadas del ejercicio anterior*).
Los elementos dentro de la lista deben estar almacenados **como números enteros** y deberán ser tratados como tal.

Desde el programa principal invocar a la función **cargarLista**, luego imprimir directamente el contenido de la lista en consola; como se indica en el ejemplo.

Ejemplo de salida:

```
Ingresar numeros,o 0 (cero ) para terminar
1
3
4
-9
Error, numero NO positivo.
5
5
Error, número repetido.
7
0

La lista contiene:
[1, 3, 4, 5, 7]
```

Ej. 3: Desarrollar un programa que permita realizar las operaciones de unión, intersección y diferencia entre conjuntos. Los conjuntos estarán representados por listas. El programa deberá presentar un menú como se detalla a continuación y al ingresar el valor, se ejecutará la opción elegida.

Ej: de presentación de menú en pantalla

```
  1. CARGAR CONJUNTOS
  2. UNION
  3. INTERSECCION
  4. DIFERENCIA (A-B)
  5. DIFERENCIA SIMÉTRICA
  6. SALIR
Ingrese el valor de la opción:
```

Para resolver la opción elegida se deberá invocar, por cada opción, a una función que la resuelva:

cargarConjuntos: Función que recibe por parámetro dos listas vacías y realiza la carga de cada una. Esta función llamará a la función **cargarLista** realizada en el ejercicio

anterior. (*aclaración: las listas cargadas no deben tener valores repetidos*).

`union`: Función que retorna una nueva lista con la unión de las dos listas pasada por parámetro (*aclaración: en la nueva lista no debe haber valores repetidos*).

`interseccion`: Función que retorna una nueva lista con la intersección de las dos listas pasada por parámetro. (*aclaración: en la nueva lista no debe haber valores repetidos*).

`diferencia`: Función que retorna una nueva lista con la diferencia de conjuntos de las dos listas pasada por parámetro. Se realizará la diferencia eliminado sólo aquellos valores de la primera lista que se encuentren el la segunda lista. (*aclaración: en la nueva lista no debe haber valores repetidos*).

`diferenciaSimetrica`: Función que retorna una nueva lista con la diferencia simétrica de conjuntos de las dos listas pasada por parámetro. Se realizará la diferencia simétrica cargando en la nueva lista sólo aquellos elementos que no se encuentran en ambas listas (*aclaración: en la nueva lista no debe haber valores repetidos*).

Link sobre operaciones de conjuntos:
https://es.wikipedia.org/wiki/Operaciones_con_conjuntos

Ejemplo: Para caso de prueba.

Sean los conjuntos:
A = {1, 7, 3, 5}
B = {2, 8, 5, 9, 7}

Las operaciones deberían arrojar los siguientes resultados:
Unión = {1, 2, 3, 5, 7, 8, 9}
Intersección = {5, 7}
Diferencia (A-B) = {1, 3}
Diferencia Simétrica = {1, 2, 3, 8, 9}

Ej. 4: Desarrollar un programa que solicite al usuario la carga de tres números entero. Los dos primeros representan los extremos de un intervalo de números, el tercero representa una cantidad. Luego deberá invocar a la función **`cargarListaAleat`** donde se le pasará por parámetro el intervalo (los dos números) y la cantidad (el tercer número). La función deberá retornar un lista de largo “cantidad”, cargada con números enteros positivos generados de forma aleatoria y pertenecientes al intervalo.
Luego se deberá calcular el máximo y el mínimo (*ver más abajo la Aclaración*) de los valores que contiene la lista y mostrar a ambos por pantalla.

Aclaración: Se solicita que el mínimo y el máximo sea determinado de dos formas distintas:

1. Invocando a la función ***min*** y ***max*** que brinda el lenguaje. Considerar resolver el caso de que la lista sea vacía.
2. Invocando a dos funciones (**minVal** y **maxVal**) que deberán ser desarrolladas. El mínimo o el máximo se deberán obtener recorriendo la lista con un ciclo. Considerar resolver el caso de que la lista sea vacía.
    a. **minVal**: Función que recibe por parámetro una lista y retorna el mínimo valor de la lista. Retorna `-1` si la lista está vacía.
    b. **maxVal**: Función que recibe por parámetro una lista y retorna el máximo valor de la lista. Retorna `-1` si la lista está vacía.

cargarListaAleat: Función que recibe recibe por parámetro tres números (**a**, **b**, **can**) y retorna una lista con **can** cantidades de números aleatorios que estén dentro del intervalo cerrado *[a,b]*

a, b: Son valores enteros positivo que determina los límites de un intervalo cerrado. No viene en orden (es decir no se sabe si a es menor que b).

can: Es un número entero positivo que determina la cantidad de números aleatorios que debe contener la lista.

Ej. 5: Desarrollar una función **cambiaLista** que reciba por parámetro una lista **lst**, y dos valores enteros **a** y **b** que representan los extremos de un intervalo numérico. La función deberá reemplazar dentro de la lista aquellos valores que se encuentran fuera del intervalo *[a,b]* por valores aleatorios que se encuentren dentro del intervalo *[a,b].*

Desde el programa principal realizar la carga de una lista (*puede utilizar alguna función de carga de los ejercicios anteriores*), y luego imprimir la lista en pantalla. A continuación solicitar el ingreso de dos números enteros **a** y **b** e invocar a la función **cambiaLista** pasándolo como parámetro los números y la lista. Luego volver a imprimir la lista, ya con los valores cambiados.

```
Lista generada:
[2,12,5,31,7]

Ingrese limite a del rango: 5
Ingrese limite b del rango: 7

La lista modificada es:
[5,6,5,6,7]
```

Ej. 6: Desarrollar un programa que solicite al usuario la carga de una lista de números enteros hasta que se ingrese cero. Dentro del mismo programa invocar a una función **`ordenarLst`** (el cual deberá desarrollar), que reciba por parámetro dicha lista de enteros y ordene los elementos en la lista (por algún método de ordenamiento) en forma ascendente. La función deberá retornar la lista ordenada al programa principal donde se mostrará por pantalla.

```
La Lista cargada es:
[2,12,5,31,7]

La lista ordenada es:
[2,5,7,12,31]
```

Ej. 7: Desarrollar una función **`inserOrd`** que reciba dos parámetros (*una lista* ***`lst`*** *y un número* ***`num`***). La lista **`lst`** de números enteros positivos recibida por parámetro se encuentra cargada y ordenada en forma ascendente, y el parámetro **`num`**, recibido por parámetro, es un número entero positivo. La función debe insertar el valor de **`num`** dentro de la lista de forma ordenada.
*Aclaración: **No se puede insertar (en cualquier lugar) y ordenar luego** debido a que el costo de computo a nivel ordenamiento es superior al de insertar ordenado.*

Desde el programa principal, cargar una lista ordenada (*puede utilizar alguna función de ejercicios anteriores para cargar la lista*) e imprimirla en pantalla (*con otra función*). Luego solicitar el ingreso de un valor entero desde el teclado (*para ser insertado en la lista*), y a continuación invocar **`inserOrd`** (*pasandole la lista y el número a insertar*). Luego volver a imprimir en pantalla la lista (*con el valor insertado en forma ordenada*).

```
Lista generada:
[2,5,12,31,70]

Ingrese valor a insertar: 6

La lista con la inserción ordenada es:
[2,5,6,12,31,70]
```

Ej. 8: Desarrollar una función que solicite al usuario el ingreso de números enteros (hasta el ingreso del 0) y los cargue en una lista. A medida que el usuario va ingresando los números, se invocará a la función **`inserOrd`** *(del ejercicio anterior)* para que inserte cada

número en la lista en forma ordenada. Una vez terminada la carga de la lista, desde el programa principal se deberá mostrar la lista por pantalla para verificar su orden.

Ej. 9: Desarrollar la función **`cargarLstAlu`** que retorne una lista de los datos de alumnos (*lista de lista*). Por cada alumno se cargará una lista con DNI, nombre y edad (*una lista por cada alumno*). El listado general con los datos de alumnos se almacenará en una única lista. Por tanto se tendrá una lista conteniendo un grupo de listas (*lista de lista*). Ejemplo:

```
[[2698705, 'James Howlett', 18], [38698705, 'Jakie chan', 22],
 [35698705, 'Jean Grey', 22]]
```

Desarrollar la función **`ordenarAluXDNI`** que reciba por parámetro la lista generada en **`cargarAlu`** y la ordene en forma descendente por DNI.

Desde el programa principal invocar a **`cargarAlu`** donde se cargará la lista de datos de los alumnos. Una vez terminada la carga imprimir la lista por pantalla. Luego invocar a **`ordenarAluxDNI`**, que ordenará la lista por DNI. A continuación volver a imprimir la lista para verificar que su orden sea el correcto.

Ej. 10: Crear una función **`atributoTriple`** que recibe por parámetro una lista de números enteros y retorne un mensaje (*Texto*) de acuerdo a las características de la lista que figuran en la tabla 1 (más abajo).

Tabla 1 - Mensajes según característica de una lista:

MENSAJE	Descripción del atributo de la lista
”Un Triple”	Si la lista contiene solamente **un** conjunto de exactamente tres valores iguales consecutivos.
”Dos Triples”	Si la lista contiene exactamente **dos** conjuntos de exactamente tres valores iguales consecutivos.
“+ Triples”:	Si la lista contiene **tres o más** conjuntos de exactamente tres valores iguales consecutivos.
“NADA”:	Si la lista no cumple con ninguna de las tres especificaciones anteriores.

Desde el programa principal, invocar a la función **`cargarListaAleat`** (desarrollada en ejercicio anterior), crear una lista con **`can`** cantidades valores generados en forma

aleatoria entre un intervalo **a** y **b**, inclusive. Los valores can, a y b deberán ser previamente ingresados por el usuario. Luego, una vez cargada la lista, se deberá pasar por parámetro la lista cargada a la función **atributoTriple** y se deberá imprimir el mensaje que retorna.

*Aclaración: Se recomienda realizar pruebas directas (en intérprete en modo interactivo) con la función **atributoTriple** para verificar su correcto funcionamiento. A continuación algunos ejemplos de pruebas directas:*

```
>>> atributoTriple ([1,2,2,2,4,4,1])
Un Triple

>>> atributoTriple ([1,2,2,2,4,4,1,3,3,3])
Dos Triple

>>> atributoTriple ([1,2,2,2,4,4,1,3,3,3,5,5,5])
+ Triple

>>> atributoTriple ([1,2,2,1,2,])
NADA
```

Ej. 11: Desarrollar un función **ruleta** que solicite al usuario el ingreso de un número entero positivo. Ese número representará la cantidad de números al azar que deberán ser cargados en una lista. Generar los valores aleatorios entre el valor 0 y el valor 36 (inclusive). La función deberá retornar la lista cargada con números aleatorios.

Desarrollar un función **porcentual** que recibe por parámetro la lista generada en la función **ruleta.** La función **porcentual** deberá imprimir en pantalla la cantidad de veces que salió (aparece en lista) cada valor y el porcentaje que representa (con respecto al total de la muestra). No mostrar aquellos números que no hayan salido.

Desde el programa principal invocar a `ruleta` e imprimir la lista que retorna la función. Luego invocar a `porcentual` pasándole por parámetro la lista generada en `ruleta`.

Ejemplo de salida generado por **porcentual** :

```
El número 8 salió 1 vez (10%).
El número 14 salió 2 veces (20%).
El número 20 salió 4 veces (40%).
El número 23 salió 1 veces (10%).
El número 30 salió 2 veces (20%).
```

Ej. 12: Desarrollar la función **operaciones** que reciba por parámetro dos números enteros **a** y **b**, y retorne una tupla con el contenido correspondiente a los resultados de cada una de las operaciones fundamentales entre **a** y **b, (***es decir suma, resta, producto, division)*

```
>>> print(operaciones (18,9) )
(27, 9, 162, 2.0)

>>> print(operaciones (54,0) )
(54, 54, 0, None)
```

Experimentar: Verificar que en la tupla no puede modificarse el contenido de un elemento en particular (Inmutabilidad). Luego, convertir la tupla a una lista, y verificar que en una lista, puede modificarse el contenido de un elemento en particular.

Ej. 13: Desarrollar la función **agregarDicEle** que agregue elementos desde el teclado a un diccionario pasado por parámetro. En la clave introducir un número entero y al valor asignar un string que representa el número clave en letras. Ejemplo:

```
{3: 'Tres',5: 'Cinco', 1: 'Uno'}
```

Realizar un programa que realice la carga de datos en un diccionario, invocando a **agregarDicEle,** luego solicitar al usuario que ingrese un numero entero, el programa deberá imprimir en pantalla el número ingresado en letras. Para ello utilizar el diccionario donde se encuentra almacenado la traducción “de número a letra”.

Ejemplo. Una vez cargado el diccionario se debe mostrar:

```
Ingresar número a traducir o cero para salir: 3
3 -> Tres

Ingresar número a traducir o cero para salir: 24
ERROR: Ingresar número a traducir o cero para salir: 1
1 -> Uno

Ingresar número a traducir o cero para salir: 0
```

Ej. 14: Desarrollar la función **agregarDicEle2** que agregue elementos desde el teclado a un diccionario pasado por parámetro. En la clave introducir un número entero y al valor asignar una tupla de tres elementos donde cada elementos es un string que representa el número clave en letras, un posición de la tupla en idioma Español (sp), otra en Inglés(en) y la tercera en Alemán (de) . Ejemplo:

```
{3:('Tres', 'Three', 'Drei'), 5:('Cinco', 'Five', 'Fünf'),
1:('Uno', 'One', 'ein')}
```

Realizar un programa que realice la carga de datos en un diccionario, invocando a **agregarDicEle2,** luego solicitar al usuario que ingrese un número a traducir y el idioma. El programa mostrará en pantalla el número en letras en el idioma especificado. Volver a repetir la solicitud de entrada de un número o el ingreso de cero para salir del programa. Si el número no está en el diccionario o la abreviatura no existe, hay que mostrar un error y continuar con la solicitud de datos al usuario.

Ejemplo. Una vez cargado el diccionario se debe mostrar:

```
Ingresar número a traducir o cero para salir: 3
Ingresar idioma [‘en’|’sp’|’de’] : de

3 en Alemán: Drei

Ingresar número a traducir o cero para salir: 3
Ingresar idioma [‘en’|’sp’|’de’] : xx
ERROR: Ingresar idioma [‘en’|’sp’|’de’] : sp

3 en Español: Tres

Ingresar número a traducir o cero para salir: 170
ERROR: Ingresar número a traducir o cero para salir: 1
Ingresar idioma [‘en’|’sp’|’de’] : en

1 en Inglés: One

Ingresar número a traducir o cero para salir: 0
```

Ej. 15: Desarrollar la función **agregarDicEle3** que retorne un diccionario con los datos de personas cargados desde el teclado. Por cada alumno se cargará como clave el DNI. Para cada DNI clave se le asignará una lista con: una sublista con el apellido, nombre y otra sublista con las notas obtenidas. Por tanto se tendrá una diccionario como el siguiente ejemplo:

```
{2698705: [['James', 'Howlett'], [2,9,7]], 38698705: [['Jakie', '
chan'], [2,9,7]], 35698705: [['Jean', 'Grey'], [2,9,7]]}
```

Desarrollar la función **`imprimirDic`** que recibe por parámetro un diccionario, la función deberá imprimir el diccionario en el orden que fue cargado.

Desarrollar la función **`impirmirDicOrd`** que recibe por parámetro un diccionario, la función deberá imprimir el diccionario ordenado por su clave.

Desde el programa principal invocar **`agregarDicEle3`** para la carga del diccionario, luego imprimir el diccionario tal cual fue cargado invocando a **`imprimirDic`** y por último volver a imprimir el diccionario pero esta vez en forma ordenada invocando a **`impirmirDicOrd.`**