**Ejercicio 1:(2.5 puntos)**

Hacer un programa, utilizando funciones, que solicite al usuario un número e indique si es un número ***perfectamente deficiente****.* Un número perfectamente deficiente es un natural que es deficiente y que la suma de sus divisores propios da como resultado un número perfecto.

**Número deficiente:** todo número natural que cumple que la suma de sus divisores propios es menor que el propio número. Por ejemplo, 10 es un número deficiente ya que sus divisores propios son 1, 2 y 5 y se cumple que 1+2+5=8, que es menor que 10.

**Número perfecto:** todo número natural que es igual a la suma de sus divisores propios. Por ejemplo, el 6 es perfecto dado que la suma de sus divisores da dicho número: 1+2+3 = 6.

Los números 1, 25 y 652 son los primeros perfectamente deficientes.

**Ejercicio 2: (2.5puntos)**

Desarrollar las funciones y el programa principal para obtener la intersección entre tres listas (sin repetidos), a su vez, de la intersección se tendrá que obtener el complemento de dichas listas.

El complemento relativo a los tres listados son aquellos elementos que están en las tres listas excepto en la intersección.

Se pide: escribir el código de tres funciones y el programa principal

**esta(elem, lista):** recibe un número y una lista. Devuelve True o False.

**interseccion(lista1, lista2, lista3):** recibe tres listas y retorna un listado.

**complemento(lista1, lista2, lista3):** recibe tres listas y retorna un listado.

La funcion complemento(lista1, lista2, lista3) deberá llamar a interseccion(lista1, lista2, lista3) y a esta(elem, lista). La función interseccion(lista1, lista2, lista3) deberá llamar a la función esta(elem, lista).

**Ejercicio 3: (2.5 puntos)**

Una empresa tiene almacenada la siguiente información en 3 listas *que se corresponden*:

***Lista de DNI*** donde se almacenan los Documentos de sus Clientes.

***Lista de Nombre de Usuarios*** donde se almacenan los nombres de Usuarios de sus Clientes.

***Lista de Passwords*** donde se guardan las claves correspondientes a los Nombres de Usuarios.

Se solicita desarrollar un Programa que dado un DNI ingresado por consola, ***informe el Nombre de Usuario y la Password de ese Cliente y si el nivel de seguridad de la misma es Bueno o Malo***, **se considera Bueno si la longitud es mayor que 4 y Mala si es menor o igual a 4.**

**Implementar una función para realizar la búsqueda, como estudiamos en la práctica de listas.**

**Ejercicio 4: (2.5 puntos)**

En una región se está por lanzar una campaña de vacunación contra la gripe A.

Se piensa en utilizar la existencia de 5 salas aptas en donde personal de enfermería de la región pueda concurrir a vacunar en esos lugares.

Para cada una de esas salas se convocará a lxs enfermerxs cuyos domicilios de residencia estén a una distancia menor que 50 km de esa sala. Se supone que las salas están geográficamente separadas de modo tal que no hay posibilidad alguna de que una enfermera o enfermero sea convocado para más de una sala. (Puede ocurrir que algún enfermerx no sea convocado a ninguna sala).

Se disponen de una cierta cantidad de vacunas que posiblemente sea insuficientes para vacunar a todos los habitantes que decidan vacunarse. No todos los habitantes están dispuestos a vacunarse y la estimación de la cantidad de vacunas se basó en una encuesta previa a cada habitante en la que se les preguntó a cada uno si se va a vacunar o no.

**Se deberán construir:**

**1.-** Una función **enfermParaSala(nombreSala)** que devuelva una lista de enfermerxs a convocar para la sala de nombre nombreSala.

**2**.- Una función **suficientesVacunas() :** que devuelva **True:** Si alcanzaron las vacunas y **False** en otro caso.

**3.-** Una función **EnfermSinConvocar():** que devuelva **True:** Si No se convocaron a todas o todos lxs enfermerxs y **False** en otro caso.

*Se cuenta con las siguientes funciones ya implementadas:*

**NombresDeSalas() :** provee una lista con los nombres de las 5 salas. Cada nombre identifica unívocamente a cada sala.

**Enfermerxs():** provee una lista con los DNI de cada enfermera(o) de la región.

**Residencia(DNI):** función que devuelv e la localidad de cada persona de la región cuyo dni es DNI.

**SeVacuna(DNI) :** func ión que devuelve True si la persona cuyo dni es DNI se va a vacunar, False en otro caso.

**Localidad(nombreSala):** función que provee la localidad de la sala cuyo nombre es nombreSala.

**Distancia(loc1, loc2) :** función que devuelve la distancia en km entre las localidades loc1 y loc2.

**CantidadVacunas():** indica la cantidad de vacunas disponibles para la campaña.

**Habitantes():** función que devuelve una lista con los DNI de los habitantes de la región.

**Posibles soluciones**

**Ejercicio 1**

#Número perfectamente deficiente: número tal que la suma de sus divisores es menor a sí mismo y además, dicha suma da como resultado un número perfecto.

def esPerfectamenteDeficiente(n):

suma = 0

perfectoDeficiente = False

for i in range(1,n):

if(n%i == 0):

suma += i

if(suma < n): #Si suma < n entonces es Deficiente, ahora verificamos si la suma de sus divisores propios da un número Perfecto

perfectoDeficiente = esPerfecto(suma)

return perfectoDeficiente

def esPerfecto(n):

suma=0

for i in range(1,n):

if(n%i==0):

suma += i

return(suma==n)

#Código principal

numero = int(input("Ingrese un número natural"))

if(esPerfectamenteDeficiente(numero)):

print(numero, "Es perfectamente deficiente")

else:

print(numero, "No es perfectamente deficiente")

**Ejercicio 2**

def esta(elem, lista1):

for i in lista1:

if(i==elem):

return True

return False

def interseccion(lista1, lista2, lista3):

nueva=[]

for i in lista1:

if(esta(i, lista2) and esta(i, lista3)):

nueva.append(i)

return nueva

def complemento(lista1, lista2, lista3):

todos=[]

complement=[]

for i in range(len(lista1)):

todos.append(lista1[i])

todos.append(lista2[i])

todos.append(lista3[i])

for i in todos:

if(not esta(i, interseccion(lista1, lista2, lista3))):

complement.append(i)

return complement

**Ejercicio 3**

def RetornaIndice (entero,lista):

for i in range (len(lista)):

if entero==lista[i]:

return i

return -1

def RetornaNivelSeguridad (cadena):

nivel=""

if len(cadena)>=4:

nivel="bueno"

else:

nivel="malo"

return nivel

Dni=[12345655, 34564998, 17234087,24655998]

NombreDeUsuario=["pelufo25","alacranxx","zanahoria1222","grajeaesc"]

Passwords=["canival11","2345toto","vege","pastilladelabuelo"]

ingresodni=int(input("ingresa DNI:"))

j=RetornaIndice (ingresodni,Dni)

if j!=-1:

print ("dni",ingresodni,"corresponde al Usuario",NombreDeUsuario[j],"su password es",Passwords[j],"nivel de seguridad ",RetornaNivelSeguridad (Passwords[j]))

else:

print ("DNI NO ENCONTRADO")

**Ejercicio 4**

def enfermParaSala(nombreSala):

enfermeras = Enfermerxs()

listaEnf = []

loc2 = Localidad(nombreSala)

for dni in enfermeras: # recorremos la lista de enfermerxs

loc1 = Residencia(dni) # se obtiene la localidad donde reside enfermerx

if Distancia(loc1, loc2) < 50:

listaEnf.append(dni)

return listaEnf

def suficientesVacunas() :

cuantosSeVacunaron = 0

for dni in Habitantes():

if SeVacuna(dni):

cuantosSeVacunaron = cuantosSeVacunaron + 1

resto = CantidadVacunas() – cuantosSeVacunaron

return (resto >= 0)

def EnfermSinConvocar():

convocados = 0

for nomSala in NombresDeSalas() :

convocados = convocados + len(enfermParaSala(nomSala))

return convocados < len(Enfermerxs())