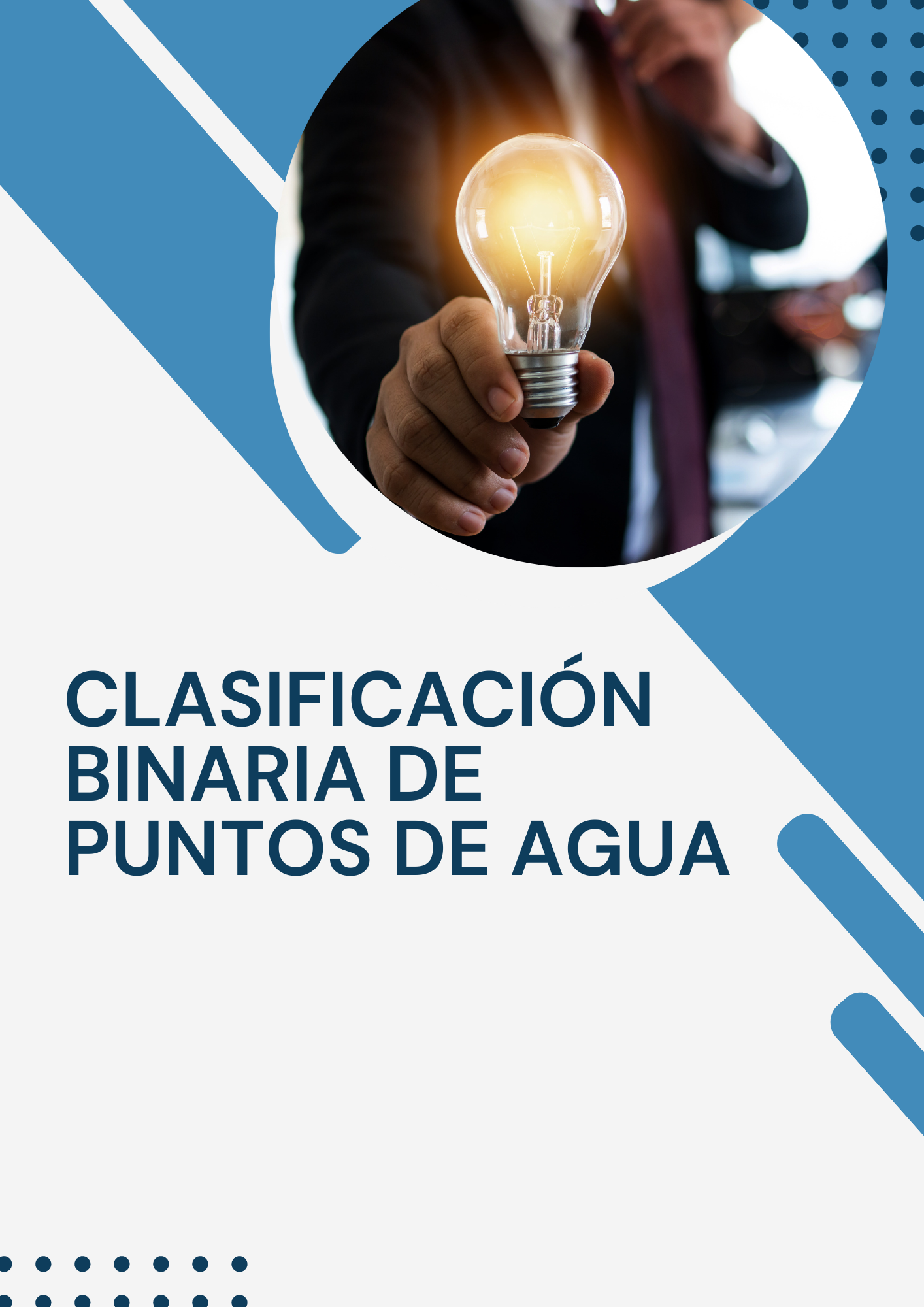
**~**

****

**CONTEXTO DEL CASO**

En este caso de estudio analizaremos el funcionamiento de las bombas de aguas de Tanzania gracias a datos entregados por el Ministerio de Agua de este país. El objetivo es reforzar sus sistemas de agua y saneamiento para mejorar su impacto en la población.

**INSTALACIÓN E IMPORTACIÓN DE LIBRERÍAS, DEFINICIÓN DE VARIABLES**

Antes de comenzar a resolver los ejercicios, consideramos que era pertinente elaborar un breve resumen de los elementos con los que trabajaríamos; razón por la cual decidimos instalar en nuestras dos primeras líneas de código las librerías HAVERSINE y FOLIUM.

Esta decisión fue tomada gracias a Javier, quien tuvo la idea antes de comenzar, de leer el enunciado completo y descargar cada una de las librerías a utilizar.

Por otro lado, a raíz de una recomendación de Daniela, decidimos a modo de guía del equipo, copiar en formato comentario utilizando el método de las triple comillas dobles, la definición de cada uno de los conceptos y variables propuestos por la consigna, respecto a la temática que vamos a abordar. Todo el equipo estuvo de acuerdo en esto, no solo porque nos permitió familiarizarnos con términos que veremos más adelante, sino porque nos resultó más claro para quien deba leer el código en el futuro, ver estas definiciones y conceptos en el comienzo del código.

Lo más interesante de esta primera parte del proceso, fue descubrir cómo a raíz de dividirnos la tarea de buscar las librerías correspondientes a los diversos ejercicios, se logró una gran optimización del tiempo.

Por todo esto, proponemos esta solución en la que estamos de acuerdo los 5 participantes:

%pip install folium

%pip install haversine

import folium

from haversine import haversine, Unit

import datetime

"""

Nombres de las variables!

amount\_tsh = carga estática total (cantidad de agua disponible, para el punto de agua).

date\_recorded = fecha en la que se incluyó el registro en los datos.

funder = quién financió el pozo.

gps\_height = altitud del pozo.

installer = organización que lo instaló.

longitude = coordenada GPS.

latitude = coordenada GPS.

wpt\_name = nombre del punto de agua, si lo tiene.

num\_private =

basin = cuenca hidrográfica.

subvillage = localización geográfica.

region = localización geográfica.

region\_code = código localización geográfica.

district\_code = código localización geográfica.

lga = ubicación geográfica.

ward = ubicación geográfica.

population = población alrededor del pozo.

public\_meeting = True/False si es punto de reunión.

recorded\_by = grupo que introdujo este registro en los datos.

scheme\_management = quién opera el punto de agua.

scheme\_name = quién opera el punto de agua.

permit = si el punto de agua está permitido.

construction\_year = año de construcción.

extraction\_type = el tipo de extracción que utiliza el punto de agua.

extraction\_type\_group = el tipo de extracción que utiliza el punto de agua.

extraction\_type\_class = el tipo de extracción que utiliza el punto de agua.

management = cómo se gestiona el pozo.

management\_group = cómo se gestiona el pozo.

payment = coste del agua.

payment\_type = coste del agua.

water\_quality = calidad del agua.

quality\_group = calidad del agua.

quantity = cantidad de agua que aporta el pozo.

quantity\_group = cantidad de agua que aporta el pozo.

source = la fuente del agua.

source\_type = la fuente del agua.

source\_class = la fuente del agua.

waterpoint\_type = el tipo de punto de agua.

waterpoint\_type\_group = el tipo de punto de agua.

"""

**EJERCICIO 1**

En este primer ejercicio nos propusimos como grupo tomarnos 5 minutos para que cada integrante por separado pudiera evaluar la mejor opción, generar su código, y pasados esos 5 minutos, hacer una breve puesta en común para tomar una decisión conjunta.

Una vez debatido entre todos los miembros, coincidimos en que la versión de Rossi era la mejor opción. Si bien en la propuesta de los 5 figuraba esta idea de convertir la lista a set/conjunto y luego reconvertir a lista para desechar duplicados, elegimos la de Rossi ya que resolvió esta acción en una sola línea de código. Posterior a la definición de Rossi, nos dimos cuenta que como equipo, todos habíamos obtenido el mismo resultado en el código al comparar los tamaños de las listas y luego añadir el elemento que se nos pedía.

Carlos por su parte, propuso la idea de agregar con almohadillas (#) comentarios que den cuenta del motivo de cada elección en nuestro código. Esta idea fue muy bien recibida por todo el grupo ya que creemos que facilita la comprensión del mismo. De hecho esta idea fue replicada a lo largo de toda la actividad.

Por todo esto, proponemos esta solución en la que estamos de acuerdo los 5 participantes:

lista\_id\_bombas = [8776,34310,67743,19728,9944,19816,54551,53934,46144,49056,50409,36957,50495,53752,61848,48451,58155,18274,48375,6091,37862,51058,55012,9944,20145,19685,69124,46804,6696,12402,41583,57355,67359,60048, 16583,25,70238,12796,52019,19282]

*#Utilizamos la función set ya que no admite duplicados y luego lo volvemos a transformar en lista*

lista\_id\_bombas\_sd=list(set(lista\_id\_bombas))

*#Corroboramos el largo de la lista inicial y la sin duplicados y notamos que solo existía un elemento duplicado*

print(len(lista\_id\_bombas),len(lista\_id\_bombas\_sd))

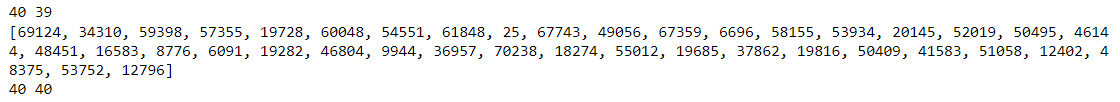
*#Añadimos el elemento "59398" en la posición 3 que seria el index 2 de la lista*

lista\_id\_bombas\_sd.insert(2,59398)

*#corroboramos la posición y largo de las lista final*

print(lista\_id\_bombas\_sd)

print(len(lista\_id\_bombas),len(lista\_id\_bombas\_sd))



**EJERCICIO 2**

En este ejercicio repetimos la modalidad de trabajo del ejercicio anterior; cuando comparamos las propuestas de todos, tuvimos algunas diferencias tanto en el desarrollo del código como en la finalidad del mismo. Quien había presentado la propuesta más simple fue Dante, ya que su versión era una función que simplemente solicitaba al usuario ingresar el ID de una bomba; si ese número de ID era mayor que el primer defectuoso y menor que el último defectuoso, sería indicador de estar incluido entre los defectuosos, por ende pusimos un print que lo indica. Por el contrario si alguna de estas dos condiciones no fuera cierta, el print señalaría que la bomba no es defectuosa.

Decidimos crear la función y probarla aparte (funcionamiento correcto), pero no ejecutarla dentro de este código ya que depende de un input del usuario.

En este ejercicio rescatamos el enriquecedor e interesante debate que compartimos respecto de la interpretación de la consigna y de cuál sería la mejor forma de resolverlo. Destacamos el excelente clima de trabajo y nuestra capacidad de expresión y de escucha.

Por todo esto, proponemos esta solución en la que estamos de acuerdo los 5 participantes:

def defectuosos():

X=int(input("Introduzca el ID de la bomba que desea verificar: "))

if X>=73890 and X<=74890 :

print(f"la bomba con ID {X} es defectuosa")

else:

print(f"la bomba con ID {X} no es defectuosa")

**EJERCICIO 3**

Durante la resolución de esta actividad descubrimos las verdaderas ventajas del trabajo en equipo en la programación. En primera instancia implementaríamos el mismo método que en los ejercicios anteriores. Pero debido a la dificultad que este ejercicio presentaba, surgieron dudas y problemas para avanzar en todos nosotros. Razón por la cual el proceso tomó otro rumbo, mucho más colaborativo en su concepción.

Javier, que era quien más avanzado estaba en su intento de resolverlo, compartió pantalla y eso fue un disparador, ya que todos nosotros al verlo sentimos que ese era el camino y que podíamos aportar a ese desarrollo. Daniela estaba investigando la mejor utilización de Haversine y le pasó a Javier un código por el chat para que lo probara en el programa. A su vez, Carlos, Rossi y Dante daban ideas y opiniones para optimizar el código mientras Javier trabajaba en función de eso. Todos veíamos su pantalla.

Dante redactó los inputs, Carlos consiguió un listado de las funciones de “folium”. En conjunto analizamos las diferentes opciones y cuando encontramos la indicada, Rossi le envió a Javier el código que debía incluir.   
Luego Dante tuvo una idea respecto de cómo ubicar los makers en el mapa, la ejecutamos y funcionó.

Realmente los 5 sentimos una sensación muy gratificante al ver que las 7 bombas figuraban en el mapa de Tanzania con sus respectivos nombres; no solo por la labor terminada sino porque fue un proceso realmente colectivo. Entendimos que era mucho más productivo trabajar con una sola pantalla generando código, mientras todos aportamos recursos e ideas a ese desarrollo.

Por todo esto, proponemos esta solución en la que estamos de acuerdo los 5 participantes:

poz1=(35.861315, -6.378573)

poz2=(38.104048, -6.747464)

H=haversine(poz1,poz2) *#en KM*

def distancia():

lat1=float(input("introduzca la latitud de la primer bomba: "))

lon1=float(input("introduzca la longitud de la primer bomba: "))

lat2=float(input("introduzca latitud de la segunda bomba: "))

lon2=float(input("introduzca longitud de la segunda bomba: "))

id1=(lat1,lon1)

id2=(lat2,lon2)

print('la distancia es: ', haversine(id1,id2), 'km')

return haversine(id1,id2)

dic\_bombas={593998:[-6.378573,35.861315],

593999:[-6.747464,38.104048],

69572:[-9.856322,34.938093],

8776:[-2.147466,34.698766],

34310:[-3.821329,37.460664],

67743:[-11.155298,38.486161],

19728:[-1.825359,31.130847]}

print(dic\_bombas)

*#crea un mapa centrado el la locación designada*

m=folium.Map(location=dic\_bombas[593998], zoom\_start=5)

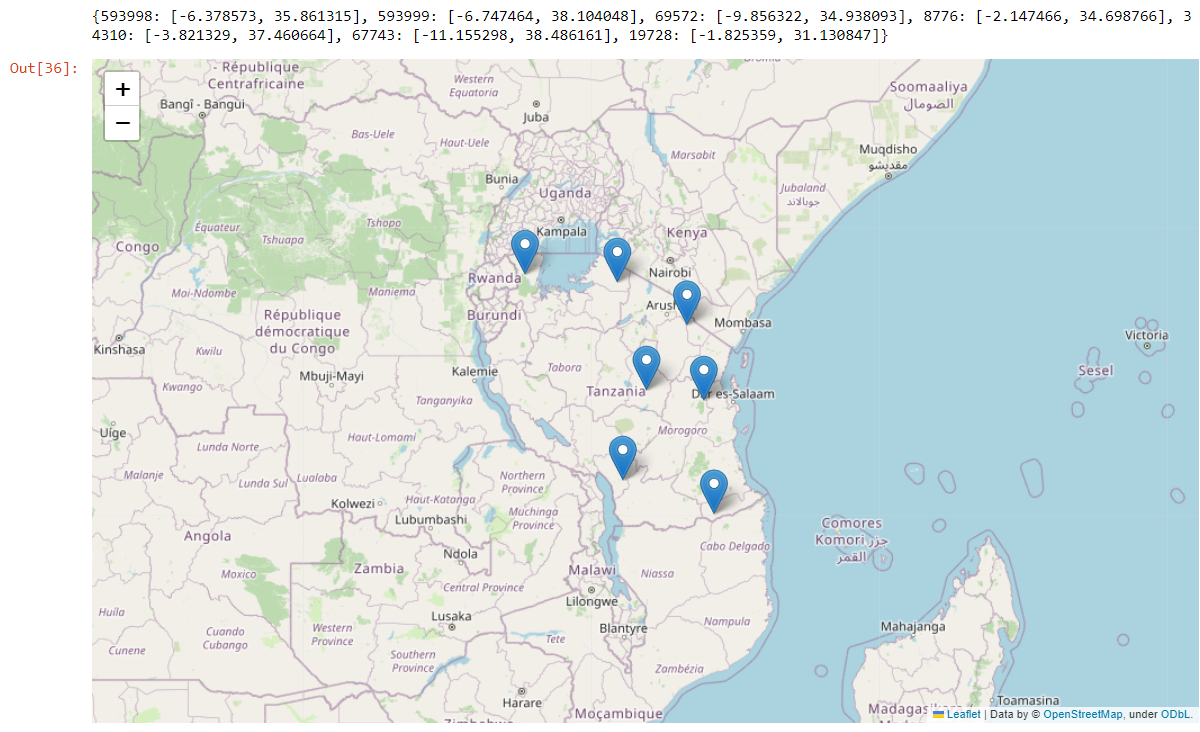
*#Crea demarcación para cada una de las ubicaciones de las bombas en el diccionario propuesto*

for i in dic\_bombas:

tooltip=f'bomba\_ID\_{i}'

folium.Marker(dic\_bombas[i], popup=f"bomba\_ID\_{i}", tooltip=tooltip).add\_to(m)

m



**EJERCICIO 4**

Motivados por el resultado del ejercicio anterior, decidimos repetir esa fórmula nuevamente, para dejar atrás la idea de los 5 desarrollos simultáneos y su posterior comparación.

Investigamos las funciones de la librería “datetime”, generamos un sistema muy dinámico de prueba y error en el que la fluidez fue preponderante. Realmente este ejercicio representó la síntesis de todo el proceso de trabajo en equipo que elaboramos a través del reto grupal. Tanto el desarrollo como la investigación fueron consensuados y compartidos por todos los integrantes del grupo, potenciándonos mutuamente.

Por todo esto, proponemos esta solución en la que estamos de acuerdo los 5 participantes:

*#Transformación de las fechas a formato datetime.date() y suma de 365 días*

fechas\_ultima\_revision = {"69572":"2022-01-10",

"8776":"2022-01-15",

"34310":"2022-02-20",

"67743":"2022-01-02",

"19728":"2022-02-08",

"9944":"2022-04-04",

"19816":"2022-03-07",

"54551":"2022-02-22",

"53934":"2022-01-05",

"46144":"2022-05-10",

"49056":"2022-03-17",

"50409":"2022-02-13",

"36957":"2022-03-02",

"50495":"2022-05-23",

"53752":"2022-04-01"}

fechas\_formato\_date = {} *# Diccionario para guardar las fechas en formato correcto datetime.date*

fechas\_proxima\_revision = {} *# Diccionario donde hemos guardado las fechas sumando los 365 días*

fecha\_actual = datetime.date.today() *# fecha actual en formato 'año-mes-día' y formato datetime.date*

*# Bucle que convierte las fechas en formato datetime.date para luego poder compararlas*

for i in fechas\_ultima\_revision:

fechas\_formato\_date[i] = datetime.datetime.strptime(fechas\_ultima\_revision[i], "%Y-%m-%d").date()

*# Ejemplo de cómo compara la fecha actual con la de la primera bomba*

diferencia = fecha\_actual - fechas\_formato\_date["69572"]

print(diferencia) *# De esta manera entrega días y horas*

print(diferencia.days) *# De esta manera solo días*

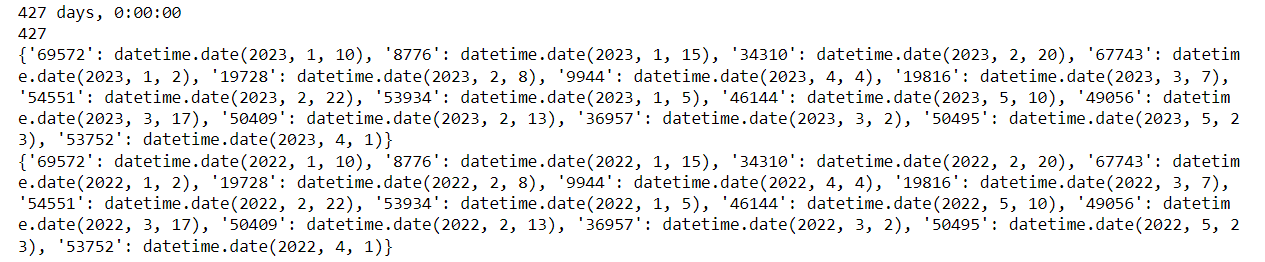
*# Bucle que suma 365 días a la fecha de revisión para saber cuando le toca la próxima revisión*

for n in fechas\_formato\_date:

fechas\_proxima\_revision[n] = fechas\_formato\_date[n] + datetime.timedelta(days=365)

print(fechas\_proxima\_revision)

print(fechas\_formato\_date)



*#Separación entre las bombas que han sido revisadas fuera de plazo y dentro de plazo entregando las fechas correspondientes en*

*#formato string y datetime (el formato string es para mejor visualización de la información)*

en\_plazo\_str={}

en\_plazo={}

fuera\_de\_plazo\_str={}

fuera\_de\_plazo={}

*#Bucle que recorre las fechas de la próxima revisión correspondiente para cada bomba y la compara con la fecha actual*

*#Además entrega los diccionarios con las fechas en formato datetime para poder utilizarlas después*

*#y como string para mejor visualización*

for j in fechas\_proxima\_revision:

if fechas\_proxima\_revision[j] < fecha\_actual:

fuera\_de\_plazo\_str[j]=fechas\_proxima\_revision[j].strftime('%Y-%m-%d')

fuera\_de\_plazo[j]=fechas\_proxima\_revision[j]

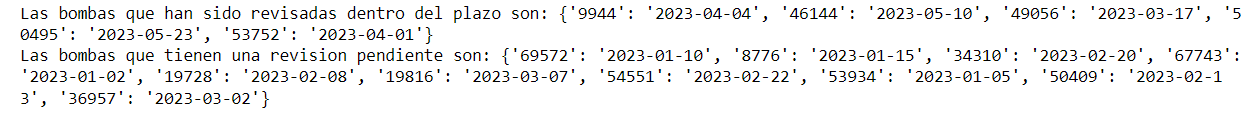
else:

en\_plazo\_str[j]=fechas\_proxima\_revision[j].strftime('%Y-%m-%d')

en\_plazo[j]=fechas\_proxima\_revision[j]

print('Las bombas que han sido revisadas dentro del plazo son:', en\_plazo\_str)

print('Las bombas que tienen una revision pendiente son:',fuera\_de\_plazo\_str)



*#Orden de las bombas con retraso en su revision y muestra la bomba con mayor retraso*

*# Utilizamos la funcion sorted que nos permite ordenar diccionarios, como esta por defecto ordena los diccionarios por claves,*

*# Utilizamos la funcion lambda que nos permite acceder a los valores de las claves, en este caso las fechas.*

ordenado\_por\_fecha = dict(sorted(fuera\_de\_plazo.items(), key=lambda x: x[1]))

*#print(ordenado\_por\_fecha)*

clave\_mas\_retrasado=list(ordenado\_por\_fecha.keys())[0] *#entrega la primera clave sin la necesidad de imprimir el diccionario*

valor\_mas\_retrasado=list(ordenado\_por\_fecha.values())[0] *#entrega el primer valor de la clave sin la necesidad de imprimir el diccionario*

retraso = (fecha\_actual - valor\_mas\_retrasado).days *#calcula los dias de retraso en la revision de la bomba más atrasada*

print(f'El id de la bomba con más días de retraso es: {clave\_mas\_retrasado}, tendría que haber pasado la revisión el {valor\_mas\_retrasado}, por lo que tiene un retraso acumulado de {retraso} días.')

