# Mara Bayurk

# Metodología

Para el desarrollo de la parte de del TP separamos la consigna en 2 partes:

- Funciones en común a los ejercicios:
    - Ingreso de DNIs
    - Generar conjuntos únicos
- Mara:
    - Cálculo y visualización de: unión, intersección.
    - Conteo de frecuencia de cada dígito en cada DNI utilizando estructuras repetitivas.
- Lina:
    - Cálculo y visualización de:diferencias y diferencia simétrica.
    - Suma total de los dígitos de cada DNI.

# Explicación

## Funciones en común:  ingresarDni, generarConjuntosUnicos

La primera función es **ingresarDni()** que solicita al usuario los números de DNI separados por comas y los convierte en una lista para almacenarlos en la lista dnis.

```
def ingresarDni():
    dnis = input("Ingrese los DNI, separados por ',':")
    dnis = dnis.split(',')
    return dnis
```

Después con **generarConjuntosUnicos(dnis)** se toma cada DNI de la lista dnis y se los convierte en un conjunto de dígitos únicos, eliminando duplicados.

```
def generarConjuntosUnicos(dnis):
    conjuntos = []
    for dni in dnis:
            conjunto_unico = set(dni) # set lo que hace es crear un
conjunto de dígitos únicos y desordenados -> {3,2,4,1}
        conjuntos.append(conjunto_unico)
    return conjuntos
```

# Mara

Desarrollo de ejercicios:

## Unión

Para comenzar creé la función **calcularUnion** donde el objetivo es obtener la unión de los conjuntos sin repetición, generados a partir de los DNIs ingresados. Para ello, utilicé el tipo de dato `set` de Python, que permite trabajar con conjuntos, es decir, colecciones no ordenadas de elementos únicos.

Se genera una lista de conjuntos únicos a partir de los DNIs y luego se recorre uno por uno, aplicando el método .union() que devuelve un nuevo conjunto con todos los elementos únicos de los conjuntos involucrados.

```
def calcularUnion(dnis):
    conjuntos = generarConjuntosUnicos(dnis)
    unionConjuntos = set()  # Inicializamos un conjunto vacío para la unión
    for conjunto in conjuntos:
        unionConjuntos = unionConjuntos.union(conjunto)
    return unionConjuntos
```

Luego, optimicé esta función utilizando la función integrada `set.union()` junto con el operador de *unpacking* (*), que permite pasar todos los conjuntos como argumentos y obtener directamente la unión total en una sola línea:

```
def calcularUnion(conjuntos):
    return set.union(*conjuntos)
```

## Intersección

Para continuar con el próximo ejercicio hice la función **calcularInterseccion**, el objetivo es obtener los dígitos que se repiten en todos los DNIs ingresados, es decir, la intersección entre los conjuntos de dígitos únicos generados a partir de cada número.

En este caso, el proceso comienza tomando el primer conjunto como base y luego se recorre el resto, aplicando la intersección uno por uno. El resultado final contiene únicamente los dígitos que están presentes en todos los DNIs.

```
def calcularInterseccion(dnis):
    conjuntos = generarConjuntosUnicos(dnis)
    interseccionConjuntos = conjuntos[0]  # Inicializamos con el primer conjunto
    for conjunto in conjuntos[1:]:
        interseccionConjuntos = interseccionConjuntos.intersection(conjunto)
    return interseccionConjuntos
```

Al igual que en el caso anterior, utilicé el tipo de dato set de Python, que disponibiliza el método específico .intersection(), que devuelve un nuevo conjunto con los elementos que están presentes en todos los conjuntos involucrados.

```
def calcularInterseccion(conjuntos):
   return set.intersection(*conjuntos)
```

## Contar Frecuencia de dígitos

Después desarrollé la función **conteoFrecuencia**, que lo hace es contar cuántas veces aparece cada dígito en la lista de DNIs ingresados.

Para resolverlo, decidí usar un diccionario (dict) de Python, ya que me permite almacenar pares clave–valor. En este caso, la clave es el dígito y el valor es la cantidad de veces que aparece.

```
def conteoFrecuencia(dnis):
   frecuencia = {} # es un diccionario que almacena la frecuencia de cada dígito
clave: digito y el valor: frecuencia
   for dni in dnis:
       for digito in dni:
           if digito in frecuencia: # valida si existe el dígito en el diccionario
               frecuencia[digito] += 1
           else:
               frecuencia[digito] = 1
   return frecuencia
```