**Trabajo Práctico N°2**

**INTEGRADOR MATEMÁTICA - PROGRAMACIÓN**

**Alumnos - Grupo n° 1 - Comisión 10**

Berrone Lanza Lina Lucia

Bayurk Mara Valentina

Bonetti Kunt Daniela Sofia

Cabrera Dario Ezequiel

Benitez Carolina Anabel

Bustamante Erica

**Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.**

**Matemática**

**Docente Titular**

Doc. Vanina Durrutty

**Docente Tutor**

Doc. Augusto Bertuzzi Gaspari

05 de Junio de 2025

Índice

1.Introducción 3

2.Objetivos 4

3.Consignas del trabajo 5

4. Desarrollo

1. Desarrollo Matemático 6
2. Desarrollo del programa en python
   1. Operaciones con DNIs 24
   2. Operaciones con años de nacimiento 30

6. Conclusiones 34

7. Bibliografía 35

8. Link al video de youtube 36

Introducción

En este trabajo integrador abordamos la relación entre la matemática y la programación, articulando contenidos de conjuntos, lógica y estructuras de datos mediante la práctica concreta en Python. A partir de los números de DNI y los años de nacimiento de cada integrante del grupo, desarrollamos operaciones matemáticas fundamentales como unión, intersección, diferencia y diferencia simétrica, tanto de manera teórica como práctica. Estas se tradujeron en código utilizando estructuras repetitivas, condicionales, funciones y manipulación de listas y conjuntos, permitiendo reforzar tanto el razonamiento lógico como la capacidad de abstracción. Asimismo, implementamos cálculos vinculados al análisis de años pares, impares y bisiestos, integrando funciones matemáticas con estructuras algorítmicas eficientes.

El enfoque colaborativo y la división de tareas favorecieron el desarrollo de habilidades técnicas y blandas. Esta experiencia permitió consolidar la comprensión de cómo los principios abstractos de la matemática se convierten en herramientas prácticas a través de la programación.

Objetivos

Profundizar la integración entre los contenidos de Matemática (conjuntos y lógica) y Programación (estructuras condicionales, repetitivas y funciones), fortaleciendo también el trabajo en equipo, la comunicación clara y la responsabilidad individual en proyectos colaborativos.

Consignas del trabajo

**Parte 1 – Desarrollo Matemático (Conjuntos y Lógica)**

1. Cada integrante debe anotar su número de DNI.

2. A partir de los DNIs, se deben formar tantos conjuntos de dígitos únicos como integrantes tenga el grupo.

3. Realizar entre esos conjuntos las siguientes operaciones: unión, intersección, diferencia (entre pares) y diferencia simétrica.

4. Para cada una de estas operaciones, se debe realizar un diagrama de Venn (a mano o digital), que debe incluirse en la entrega.

5. Redactar al menos dos expresiones lógicas en lenguaje natural, que puedan luego implementarse en Python y escribir en la documentación que van a presentar cuál sería el resultado con los conjuntos que tienen.

#### **Parte 2 – Desarrollo del Programa en Python**

#### El programa debe implementar varias de las ideas trabajadas en papel. Debe incluir:

#### **A. Operaciones con DNIs**

#### · Ingreso de los DNIs (reales o ficticios).

#### · Generación automática de los conjuntos de dígitos únicos.

#### · Cálculo y visualización de: unión, intersección, diferencias y diferencia simétrica.

#### · Conteo de frecuencia de cada dígito en cada DNI utilizando estructuras repetitivas.

#### · Suma total de los dígitos de cada DNI.

#### · Evaluación de condiciones lógicas (condicionales), vinculadas con las expresiones escritas.

#### 

#### **B. Operaciones con años de nacimiento**

#### · Ingreso de los años de nacimiento (Si dos o más integrantes del grupo tienen el mismo año, ingresar algún dato ficticio, según el caso).

#### · Contar cuántos nacieron en años pares e impares utilizando estructuras repetitivas.

#### · Si todos nacieron después del 2000, mostrar "Grupo Z".

#### · Si alguno nació en año bisiesto, mostrar "Tenemos un año especial".

#### · Implementar una función para determinar si un año es bisiesto.

#### · Calcular el producto cartesiano entre el conjunto de años y el conjunto de edades actuales.

#### **Parte 3 – Video de Presentación**

#### Duración estimada entre 5 y 10 minutos. Todos los integrantes deben presentarse en cámara, mostrar el programa funcionando y explicar la parte que realizaron. También deben comentar brevemente qué aprendieron al combinar matemática y programación.

PARTE 1

**Desarrollo Matemático (Conjuntos y Lógica)**

**DESARROLLADO POR Benitez Carolina Anabel y Bustamante Erica**

**PUNTO 1**

Lista de DNI

DARIO 41375492

Erica 32146784

⁠Lina 37155245

⁠Mara 41496608

Daniela 41017034

Caro 44420333

**PUNTO 2**

Si todos los Conjuntos:

A={1,2,3,4,5,7,9}

B={1,2,3,5,7}

C={0,1,4,6,8,9}

D={0,1,3,4,7}

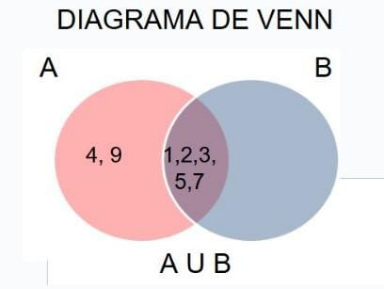
E={0,2,3,4}

F={1,2,3,4,6,7,8}

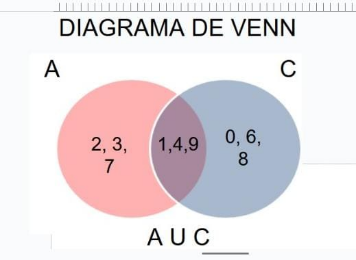
**PUNTOS 3 Y 4**

**Unión de conjuntos**: La unión de dos conjuntos, en este caso A y B, es un nuevo conjunto que contiene todos los elementos que están en A, o en B, o en ambos, sin repetir ninguno. Esto es juntar todo y eliminar los duplicados.

- A ∪ B = {1, 2, 3, 4, 5, 7, 9}



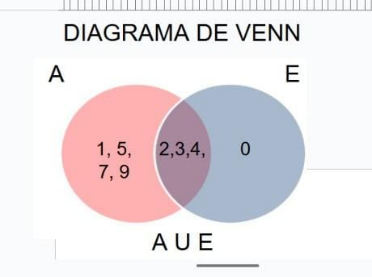
- A ∪ C = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



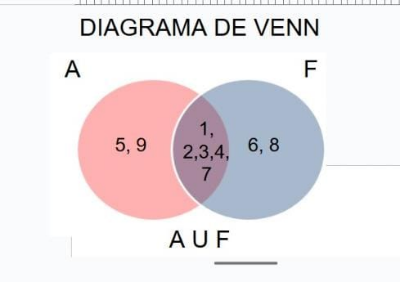
- A ∪ D = {0, 1, 2, 3, 4, 5, 7, 9}



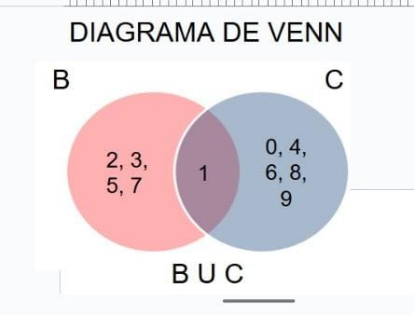
- A ∪ E = {0, 1, 2, 3, 4, 5, 7, 9}



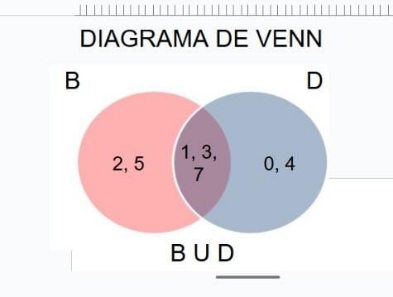
- A ∪ F = {1, 2, 3, 4, 5, 6, 7, 8, 9}



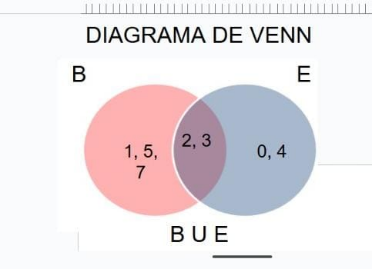
- B ∪ C = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}



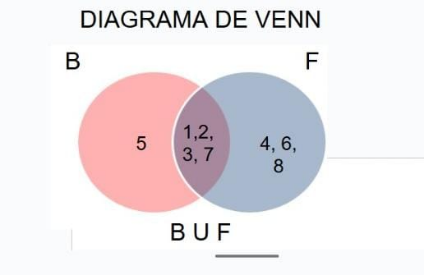
- B ∪ D = {0, 1, 2, 3, 4, 5, 7}



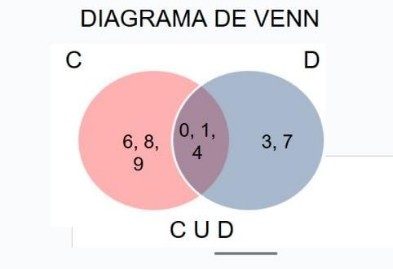
- B ∪ E = {0, 1, 2, 3, 4, 5, 7}



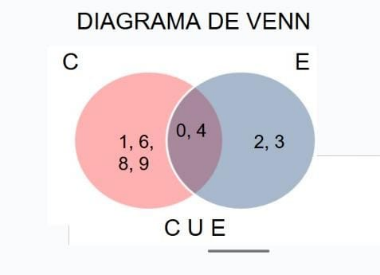
- B ∪ F = {1, 2, 3, 4, 5, 6, 7, 8}



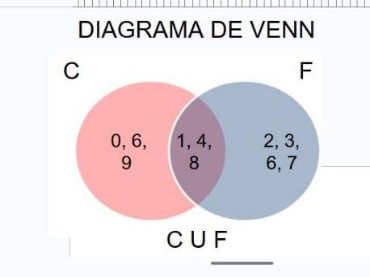
- C ∪ D = {0, 1, 3, 4, 6, 7, 8, 9}



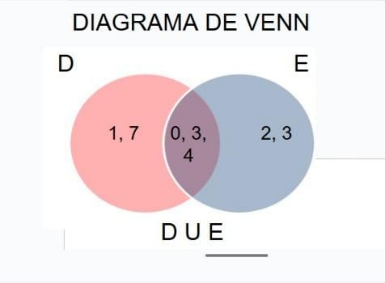
- C ∪ E = {0, 1, 2, 3, 4, 6, 8, 9}



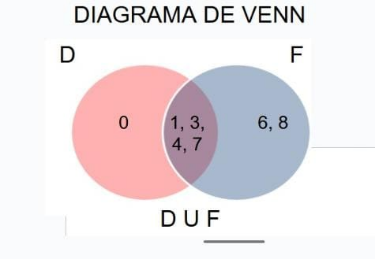
- C ∪ F = {0, 1, 2, 3, 4, 6, 7, 8, 9}



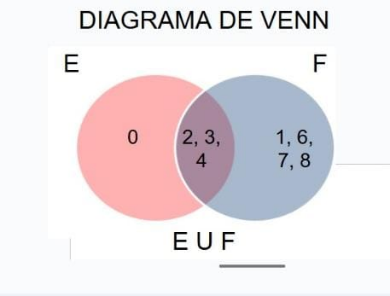
- D ∪ E = {0, 1, 2, 3, 4, 7}



- D ∪ F = {0, 1, 2, 3, 4, 6, 7, 8}

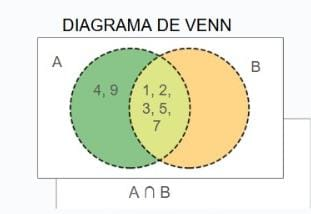


- E ∪ F = {0, 1, 2, 3, 4, 6, 7, 8}

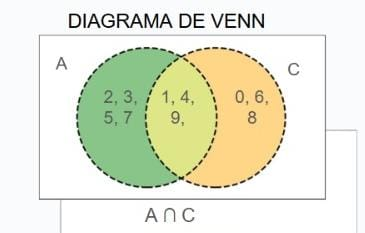


**Intersecciones de conjuntos:** La intersección de A y B son los elementos que tienen en común, los que se encuentran en ambos conjuntos al mismo tiempo.

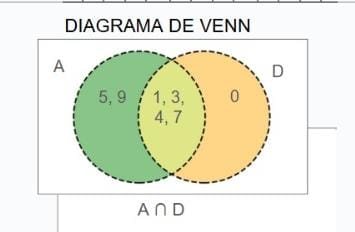
- A ∩ B = {1, 2, 3, 5, 7}



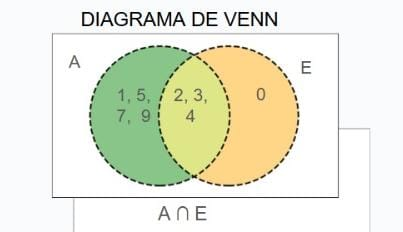
- A ∩ C = {1, 4, 9}



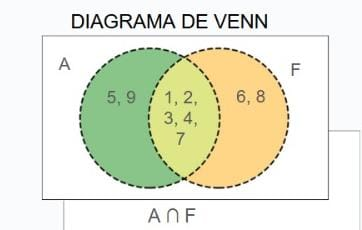
- A ∩ D = {1, 3, 4, 7}



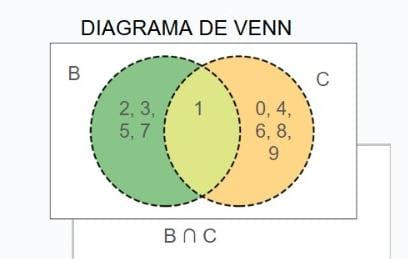
- A ∩ E = {2, 3, 4}



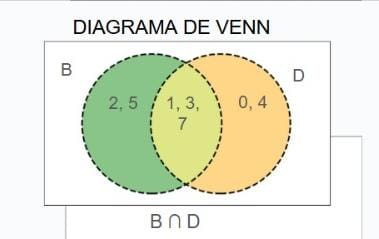
- A ∩ F = {1, 2, 3, 4, 7}



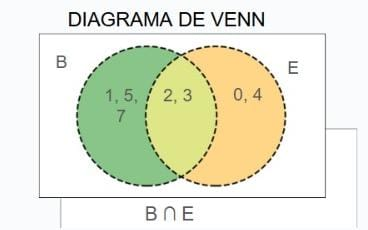
- B ∩ C = {1}



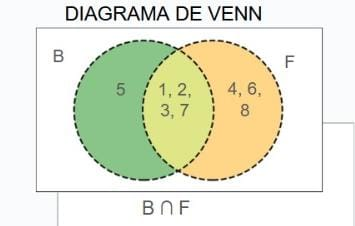
- B ∩ D = {1, 3, 7}



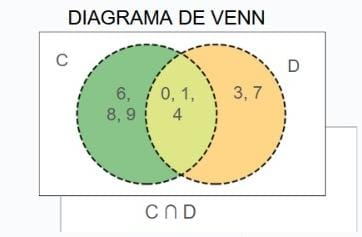
- B ∩ E = {2, 3}



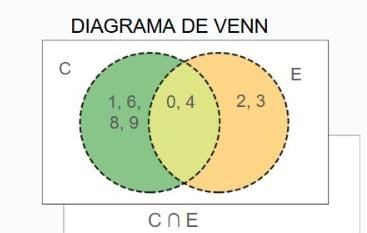
- B ∩ F = {1, 2, 3, 7}



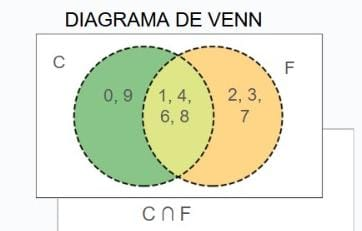
- C ∩ D = {0, 1, 4}



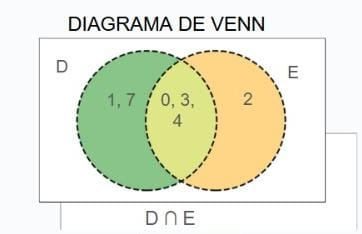
- C ∩ E = {0, 4}



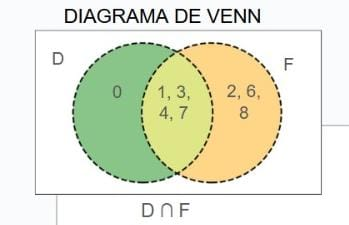
- C ∩ F = {1, 4, 6, 8}



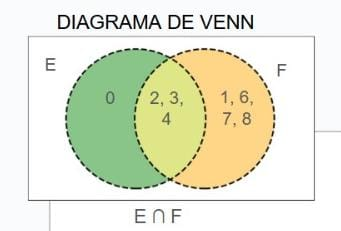
- D ∩ E = {0, 3, 4}



- D ∩ F = {1, 3, 4, 7}

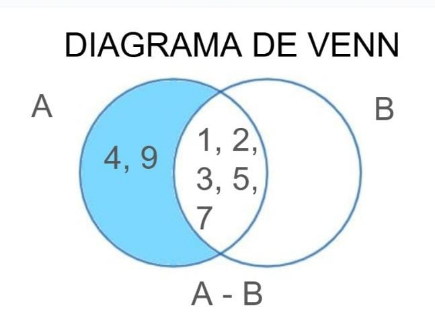


- E ∩ F = {2, 3, 4}

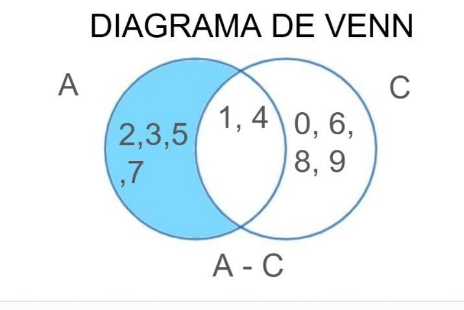


**Diferencias de conjuntos:** La Diferencia de Conjuntos, simbolizada por el signo menos, nos da los elementos que están en el primer conjunto, pero no en el segundo. Es como "quitarle" al primer conjunto lo que comparte con el segundo.

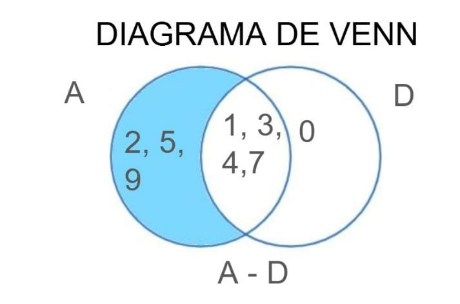
- A - B = {4, 9}



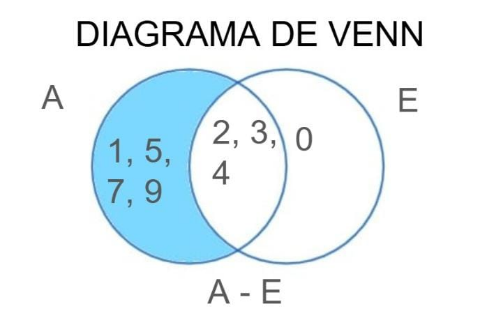
- A - C = {2, 3, 5, 7}



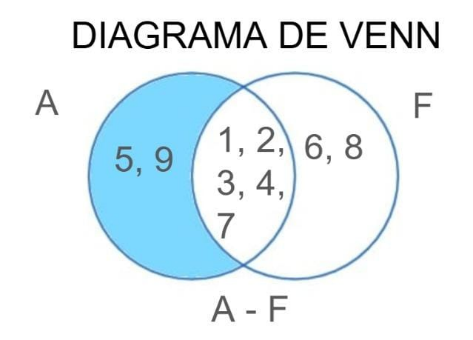
- A - D = {2, 5, 9}



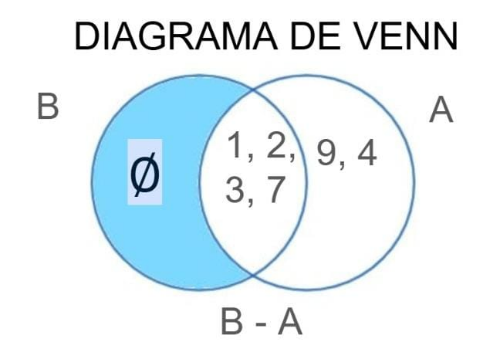
- A - E = {1, 5, 7, 9}



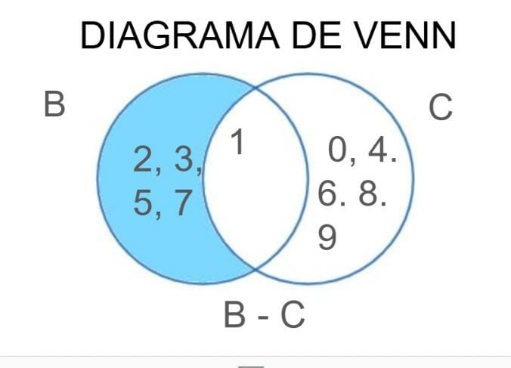
- A - F = {5, 9}



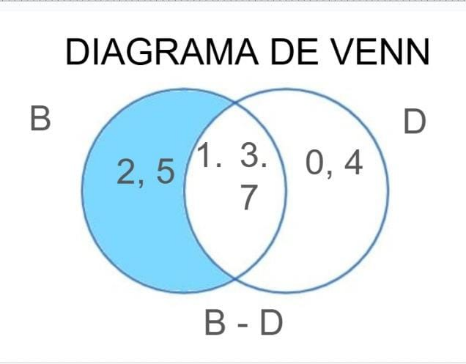
- B - A = ∅



- B - C = {2, 3, 5, 7}



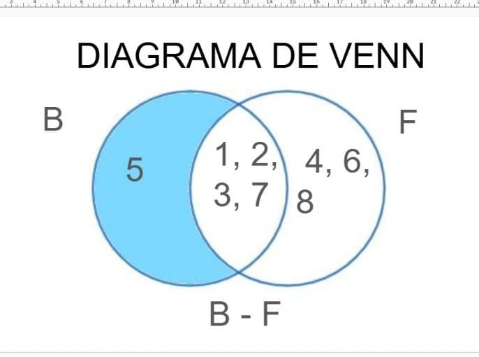
- B - D = {2, 5}



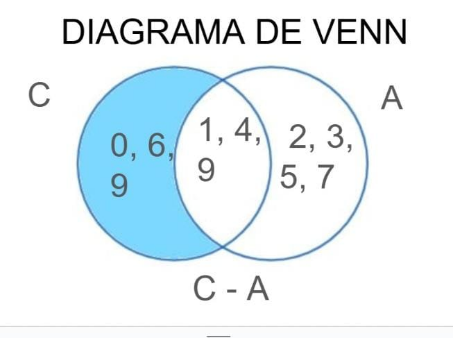
- B - E = {1, 5, 7}



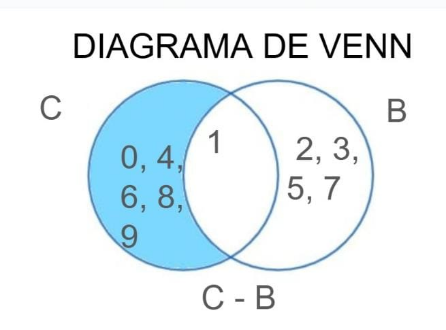
- B - F = {5}



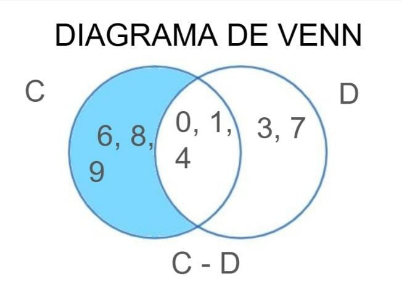
- C - A = {0, 6, 8}



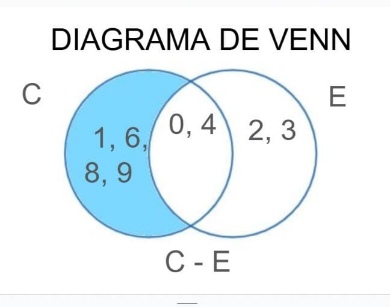
- C - B = {0, 4, 6, 8, 9}



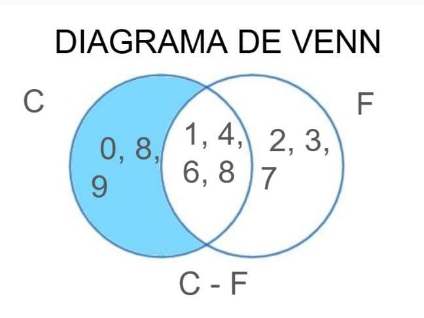
- C - D = {6, 8, 9}



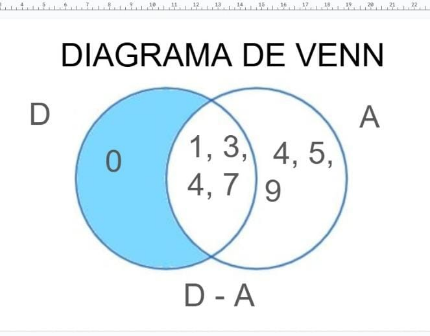
- C - E = {1, 6, 8, 9}



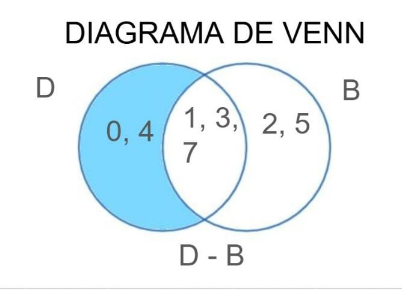
- C - F = {0, 8, 9}



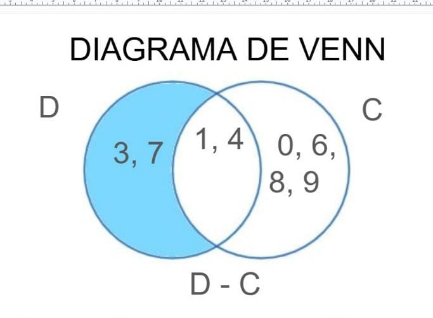
- D - A = {0}



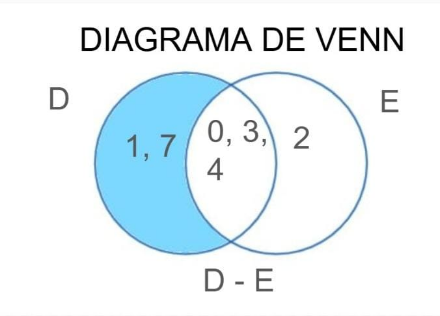
- D - B = {0, 4}



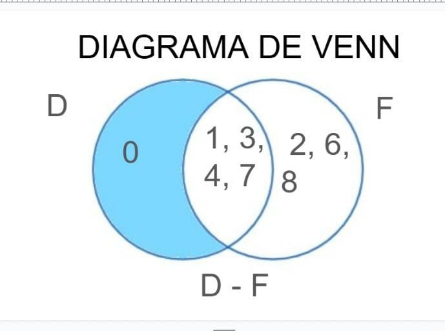
- D - C = {3, 7}



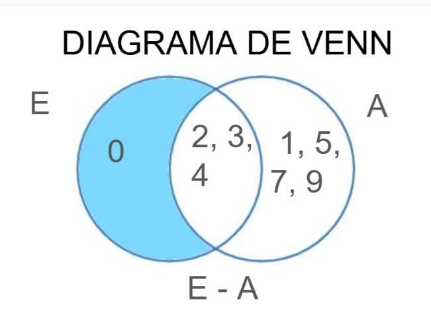
- D - E = {1, 7}



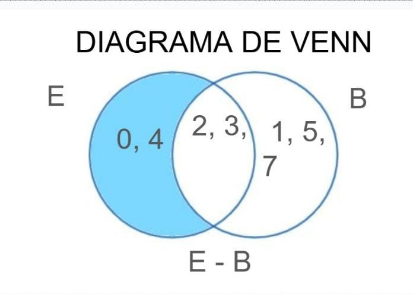
- D - F = {0}



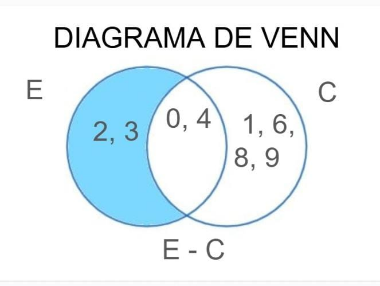
- E - A = {0}



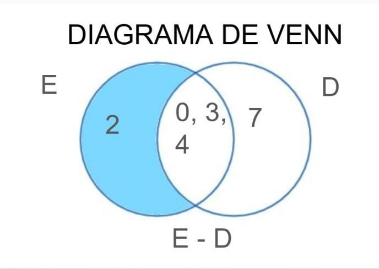
- E - B = {0, 4}



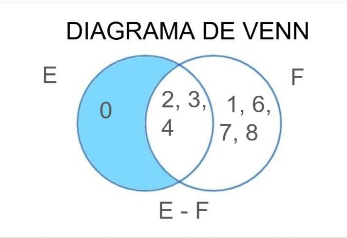
- E - C = {2, 3}



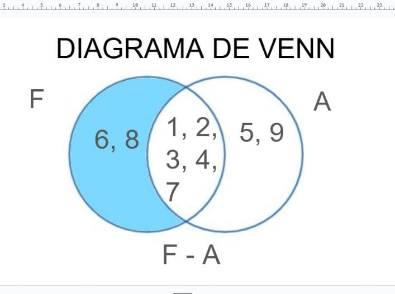
- E - D = {2}



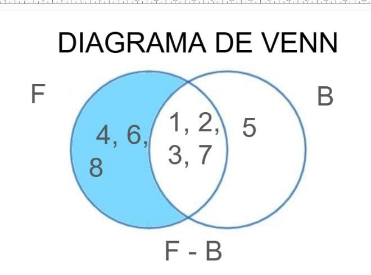
- E - F = {0}



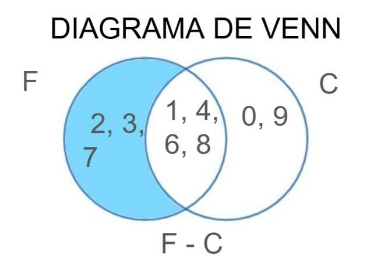
- F - A = {6, 8}



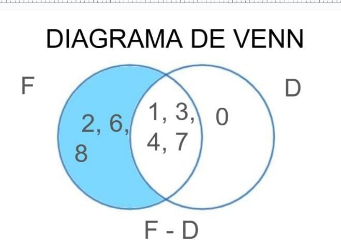
- F - B = {4, 6, 8}



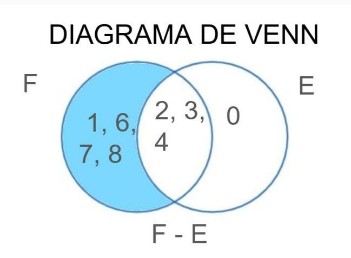
- F - C = {2, 3, 7}



- F - D = {2, 6, 8}

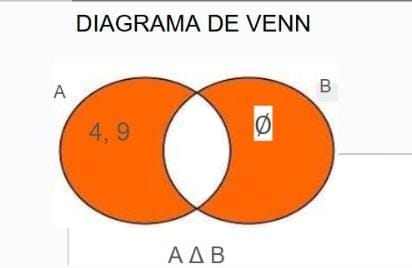


- F - E = {1, 6, 7, 8}

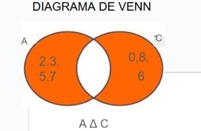


**Diferencias simétricas de conjuntos:** la Diferencia Simétrica, marcada con un triángulo, es el conjunto de elementos que están en A o en B, pero no en ambos. Se podría pensar que es como la unión menos la intersección. Es muy útil cuando queremos saber qué elementos son únicos de cada conjunto.

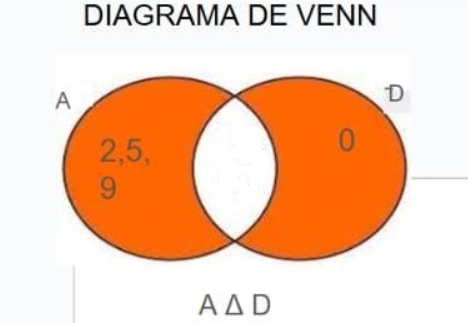
- A Δ B = {4, 9}



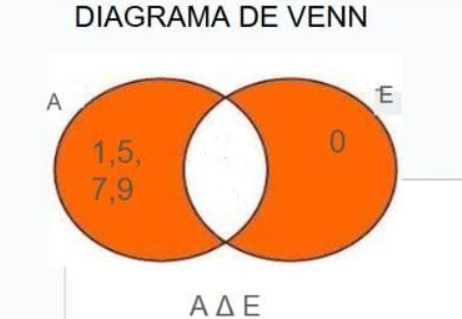
- A Δ C = {0,2, 3, 5, 6, 7, 8}



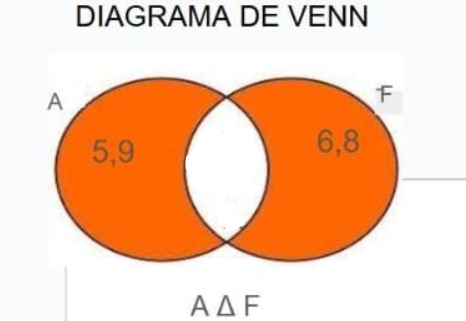
- A Δ D = {2, 5, 9, 0}



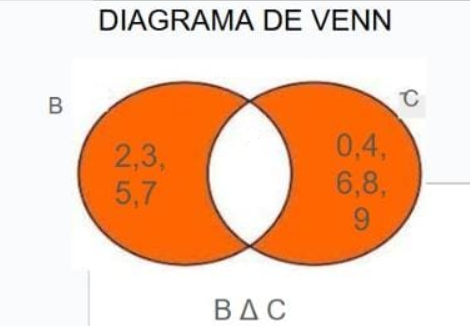
- A Δ E = {0, 1, 5, 7, 9}



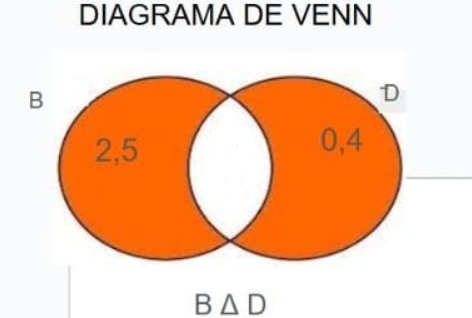
- A Δ F = {5, 6, 8, 9}



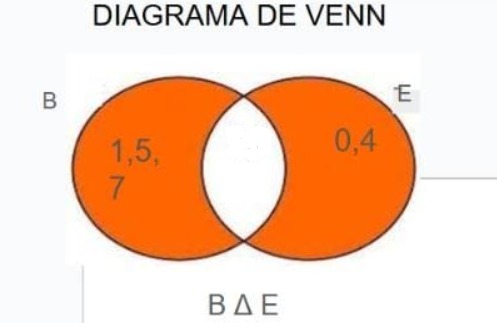
- B Δ C = {0, 2, 3, 4, 5, 6, 7, 8, 9}



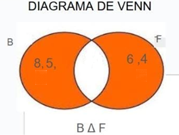
- B Δ D = {0, 2, 4, 5}



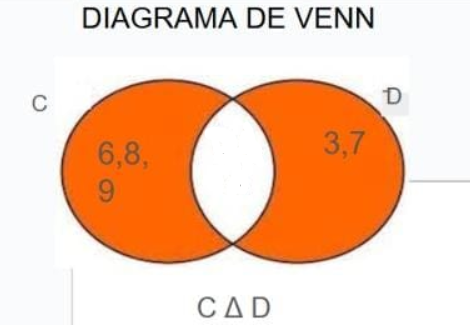
- B Δ E = {1, 5, 7, 0, 4}



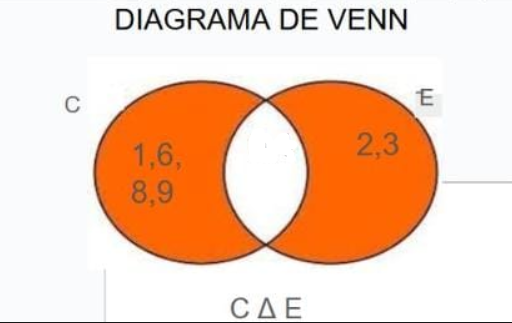
- B Δ F = {5, 4, 6, 8}



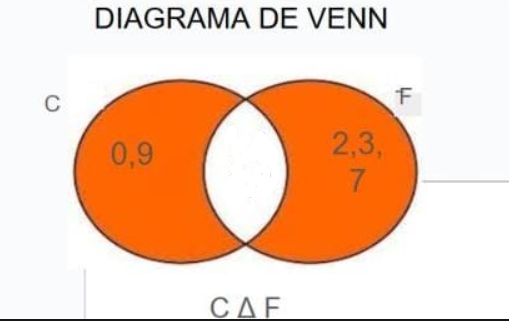
- C Δ D = {3, 7, 6, 8, 9}



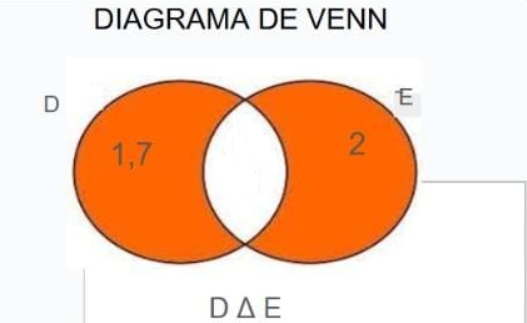
- C Δ E = {1, 6, 8, 9, 2, 3}



- C Δ F = {0, 2, 3, 7, 9}



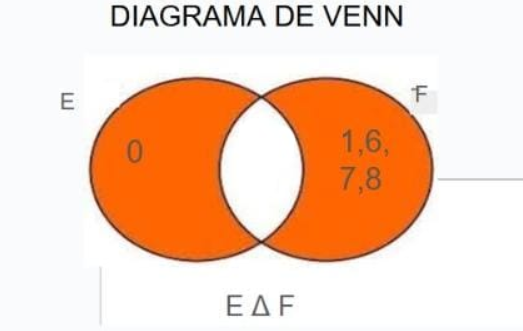
- D Δ E = {1, 7, 2}



- D Δ F = {0, 2, 6, 8}



- E Δ F = {0, 1, 6, 7, 8}

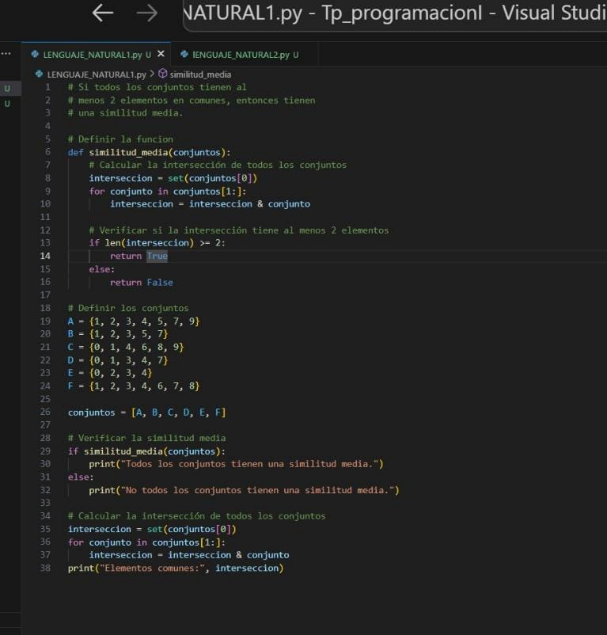


**PUNTO 5**

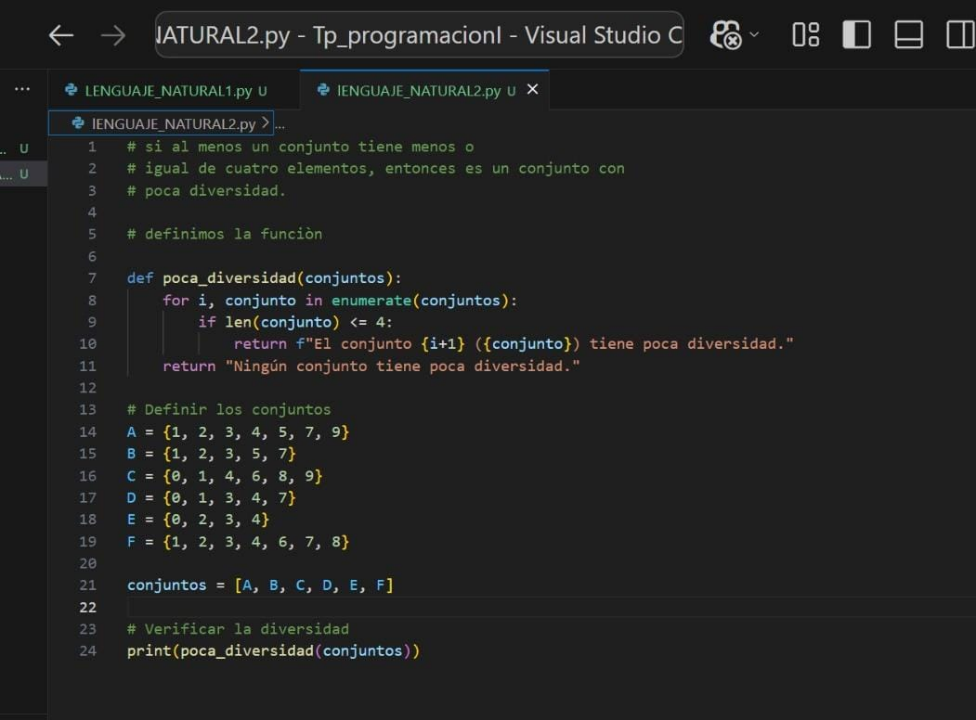
**En este punto se combinó el ejercicio 5 de expresiones matemáticas y una parte del ejercicio 1 de implementación en python.**

IMPLEMENTACIÓN EN PYTHON ejercicio 2.A.6: este ejercicio pertenece a la parte 2, implementación en python, pero fue movido a esta parte del trabajo así todos los integrantes del grupo realizamos operaciones con código.

A)Si todos los conjuntos tienen al menos 2 elementos en comunes, entonces tienen una similitud media.



B) Si al menos un conjunto tiene menos o igual de cuatro elementos, entonces es un conjunto con poca diversidad.



PARTE 2.A

**Desarrollo del programa en python - Operaciones con DNIs**

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.

# 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

## **DESARROLLADO POR Mara Valentina Bayurk**

### **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

## **DESARROLLADO POR Lina Lucia Berrone Lanza**

La parte de la consigna que trabaje me pedía :

· Ingreso de los DNIs (reales o ficticios).

· Generación automática de los conjuntos de dígitos únicos.

· Cálculo y visualización de:diferencias y diferencia simétrica.

· Suma total de los dígitos de cada DNI.

Siguiendo esta consigna elabore código y funciones para cumplimentar la consigna:

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

Luego se comienza con las funciones relacionadas a matemática:

**diferencia\_listas(lista1, lista2)** calcula la diferencia entre dos listas, retornando los elementos que están en la primera lista pero no en la segunda mediante el bucle for que itera sobre cada elemento en la lista y si no lo encuentra en la lista 2 ni en el resultado mediante el método .append lo agrega al final de la lista llamada resultado

def diferencia\_listas(lista1, lista2):

resultado = []

for elemento in lista1:

if elemento not in lista2 and elemento not in resultado:

resultado.append(elemento)

return resultado

En **diferencia\_simetrica\_listas(lista1, lista2)** calcula la diferencia simétrica entre dos listas, retornando los elementos que están en una lista o en la otra, pero no en ambas. Tiene la misma funcionalidad que diferencia\_listas mediante bucles for solamente que esta vez compara el elemento con ambas listas y con el resultado.

def diferencia\_simetrica\_listas(lista1, lista2):

resultado = []

for elemento in lista1:

if elemento not in lista2 and elemento not in resultado:

resultado.append(elemento)

for elemento in lista2:

if elemento not in lista1 and elemento not in resultado:

resultado.append(elemento)

return resultado

**diferencia(conjuntos)** aplica la función de diferencia de manera secuencial entre todos los conjuntos generados; cuando se pasan más de dos dni y se tienen más de dos conjuntos esta función compara el primero con el segundo y a ese resultado lo compara con el tercero, por esta razón usualmente devuelve una lista vacía, debido a que si se pasan más de 2 dni usualmente los números terminan repitiendose

def diferencia(conjuntos):

diferencia = conjuntos[0]

for conj in conjuntos[1:]:

diferencia = diferencia\_listas(diferencia, conj)

return diferencia

**diferenciaSimetrica(conjuntos)** aplica la función de diferencia simétrica de manera secuencial entre todos los conjuntos generados, cuando se pasan más de dos conjuntos se realiza el primero con el segundo y a ese resultado con el tercero, por eso a veces también se presta a confusión ya que para realizar la comparación siempre se toman dos conjuntos parámetros.

def diferenciaSimetrica(conjuntos):

diferencia\_simetrica = conjuntos[0]

for conj in conjuntos[1:]:

diferencia\_simetrica = diferencia\_simetrica\_listas(diferencia\_simetrica, conj)

return diferencia\_simetrica

Antes de continuar se deben agregar dos opciones que fueron contempladas y descartadas pensando en mostrar la mayor cantidad de variables disponibles.  
**1. OPCIÓN CON FUNCIONES INTEGRADAS DE PYTHON:** Esta opción fue investigada e implementada en la parte de mi compañera cuando realiza las opciones de unión e intersección por ese motivo yo solo procedo a listarla. Se comprende que esta forma es más concisa y eficiente, sin embargo la implementación larga resulta más simple de leer y entender las operaciones de diferencia y diferencia simétrica aun sabiendo que se adicionan muchas más líneas de código.

def diferenciaSimetrica(conjuntos):

return set.symmetric\_difference(\*conjuntos)

def diferencia(conjuntos):

return set.difference(\*conjuntos)

**2. OPCIÓN REFACTORIZADA:** Esta vez se observó una repetición en el código, en los bucles FOR utilizados en las funciones de diferencia y diferencia simétrica. Se logró refactorizar pero de nuevo se observó un detrimento en la lectura del código, por lo que se decidió continuar con la primera forma, entendiendo que la refactorización contempla una mejor práctica.

def agregar\_diferencia(lista1, lista2, resultado):

for elemento in lista1:

if elemento not in lista2 and elemento not in resultado:

resultado.append(elemento)

def diferencia\_listas(lista1, lista2):

resultado = []

agregar\_diferencia(lista1, lista2, resultado)

return resultado

def diferencia\_simetrica\_listas(lista1, lista2):

resultado = []

agregar\_diferencia(lista1, lista2, resultado)

agregar\_diferencia(lista2, lista1, resultado)

return resultado

Continuando con el trabajo la parte de matemática se solicita la suma de todos los dígitos de los dni. Con la función **sumaTotal(dnis)** se calcula la suma total de todos los dígitos numéricos presentes en los DNIs ingresados. Se ingresa al bucle for y por cada dni se ejecuta la variable suma\_dni en donde se itera sobre los números del dni y si efectivamente son digitos numericos, retorna true , se convierte en un entero y se suma. Agregando a la variable suma\_total. Al final se retorna el valor de la misma

def sumaTotal(dnis):

suma\_total = 0

for dni in dnis:

suma\_dni = sum(int(d) for d in dni if d.isdigit())

suma\_total += suma\_dni

return suma\_total

Por último se ejecuta esta parte del código, donde pasan en una función todos los print que necesitamos, coordinando la ejecución de las funciones anteriores y mostrando los resultados al usuario.

def ejecutarOperacionesAdicionales():

print('\_\_\_Operaciones Adicionales con DNIs\_\_\_')

dnis = ingresarDni()

conjuntos= generarConjuntosUnicos(dnis)

print('1. Conjuntos unicos:', conjuntos)

print('2. Diferencia entre todos los conjuntos:', diferencia(conjuntos))

print('3. Diferencia simetrica entre conjuntos:', diferenciaSimetrica(conjuntos))

print('4. Suma total', sumaTotal(dnis))

ejecutarOperacionesAdicionales()

En este ejercicio, se abordaron operaciones fundamentales de la teoría de conjuntos, como la diferencia y la diferencia simétrica, y se implementaron mediante estructuras básicas como los bucles, las condicionales y las funciones.

Estas operaciones matemáticas se tradujeron al código utilizando bucles y condicionales. Los bucles for permitieron iterar sobre los elementos de las listas, mientras que las estructuras if se utilizaron para evaluar condiciones específicas, como la pertenencia de un elemento a otra lista. Además, el uso de funciones facilitó la organización del código en bloques reutilizables, mejorando su legibilidad y mantenimiento.

Esta integración de conceptos matemáticos y estructuras de programación me permitió una comprensión más profunda de ambas disciplinas, demostrando cómo los principios abstractos de la teoría de conjuntos pueden aplicarse de manera práctica en el desarrollo de algoritmos y soluciones computacionales.

PARTE 2.B

**Desarrollo del programa en python - Operaciones con años de nacimiento**

**DESARROLLADO POR Dario Ezequiel Cabrera**

Nos dividimos los distintos bloques de código de esta sección. En mi caso lo que hice fueron los siguientes puntos:

**1. Importación de Módulos Esenciales**

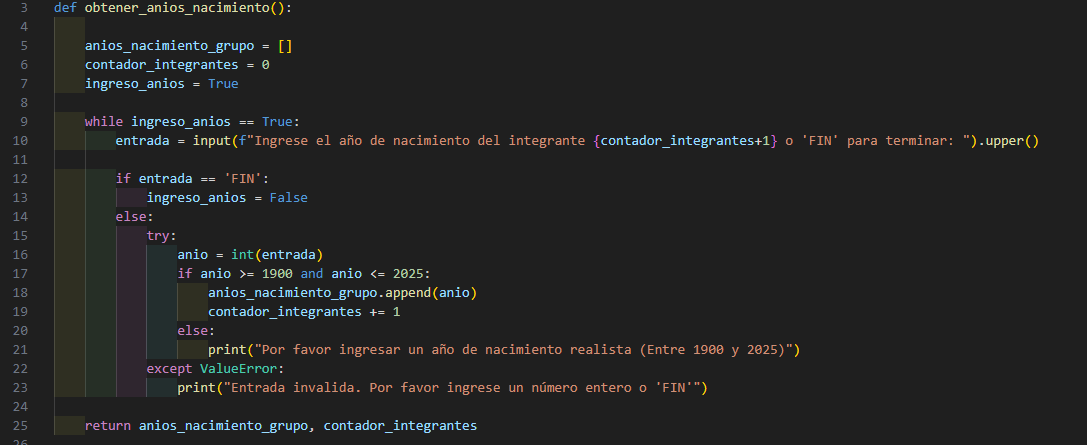
Mi primera tarea fue asegurar la correcta funcionalidad del programa al importar el módulo itertools. Esta importación es fundamental, ya que itertools nos proporciona la función product, una herramienta altamente eficiente y optimizada para el cálculo del producto cartesiano entre dos o más colecciones de datos, lo cual es esencial para una de las operaciones finales del proyecto.



**2. Función para la Recopilación de Años de Nacimiento**

Desarrollé la función principal obtener\_anios\_nacimiento(). Esta función está diseñada para interactuar con el usuario y recopilar los años de nacimiento de los integrantes del grupo de forma dinámica.

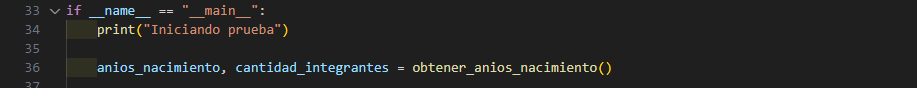
* Solicita al usuario que ingrese los años uno por uno, validando que sean valores numéricos y estén dentro de un rango realista (entre 1900 y el año actual, que se ajusta automáticamente para mayor precisión).
* Maneja posibles errores de entrada (por ejemplo, si el usuario ingresa texto en lugar de números).
* Permite ingresar años ficticios si hay duplicados, asegurando que cada "integrante" tenga un año asociado, aunque dos personas reales compartan el mismo año de nacimiento.
* Acumula los años válidos en una lista y lleva un conteo del número total de integrantes ingresados.
* El proceso de entrada finaliza cuando el usuario escribe 'FIN'.
* Finalmente, la función devuelve tanto la lista de años de nacimiento recopilados como la cantidad total de integrantes, lo cual es crucial para las fases posteriores del análisis.



**3. Ejecución Principal del Script**

Me responsabilicé de la estructura del bloque de ejecución principal (if \_\_name\_\_ == "\_\_main\_\_":). Este bloque es el punto de entrada de nuestro programa.

Cuando el script se ejecuta directamente, es lo primero que se activa. Su función inicial es imprimir un mensaje de bienvenida. Acto seguido, realiza la llamada a la función obtener\_anios\_nacimiento(), almacenando los datos de los años y la cantidad de integrantes que el usuario ha proporcionado. Esto asegura que tengamos la información base necesaria para proceder con todos los análisis subsiguientes.

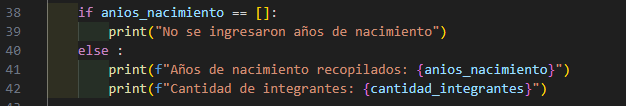


**4. Verificación y Resumen de Datos Ingresados**

Implementé el segmento de código encargado de la verificación inicial de los datos recopilados.

Este bloque evalúa si la lista de anios\_nacimiento contiene algún elemento. Si la lista está vacía (es decir, no se ingresaron años), se notifica al usuario.

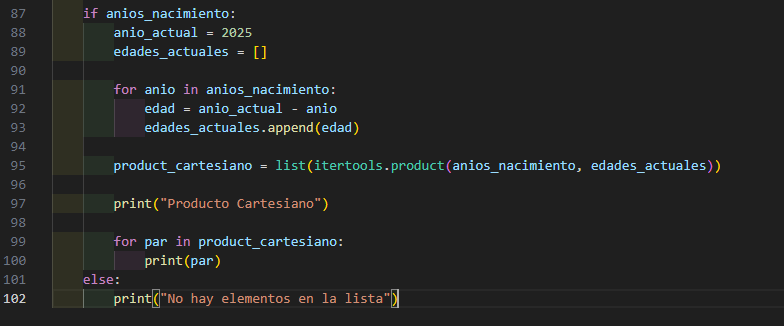
En caso contrario, se muestra un resumen claro: la lista completa de los años de nacimiento ingresados y la cantidad total de integrantes, proporcionando una confirmación inmediata de los datos recibidos.



**5. Cálculo del Producto Cartesiano**

Finalmente, desarrollé el bloque para el cálculo y la presentación del producto cartesiano. Esta es una operación avanzada que permite ver todas las combinaciones posibles entre dos conjuntos de datos:

* Primero, el programa determina el año actual (de forma dinámica).
* Con el año actual, se calcula la edad actual de cada integrante basándose en su año de nacimiento, almacenando estas edades en una nueva lista.
* Luego, se utiliza la función itertools.product() para generar todas las posibles tuplas donde cada tupla contiene un año de nacimiento y una edad actual. Esto crea una relación de "todos con todos" entre ambos conjuntos de datos.
* Para facilitar la visualización, el resultado se convierte a una lista y se imprime de manera ordenada, mostrando cada par (año de nacimiento, edad actual) en una línea separada. Este resultado es valioso para comprender la interrelación de estos dos atributos dentro del grupo.

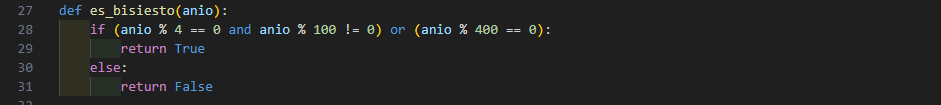


**DESARROLLADO POR Daniela Sofia Bonetti Kunt**

Nos dividimos los distintos bloques de código de esta sección. En mi caso lo que hice fueron los siguientes puntos:

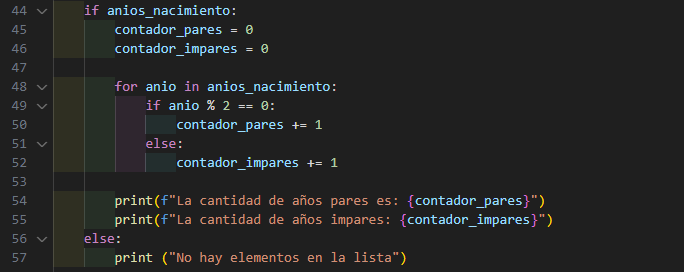
**1. Función para la Determinación de Años Bisiestos**

Desarrollé la función es\_bisiesto(anio), la cual es esencial para identificar "años especiales". Esta función toma un año como entrada y aplica las reglas del calendario gregoriano: un año es bisiesto si es divisible por 4, excepto si también es divisible por 100, a menos que sea divisible por 400. La función devuelve True si el año cumple estas condiciones y es bisiesto, y False en caso contrario.



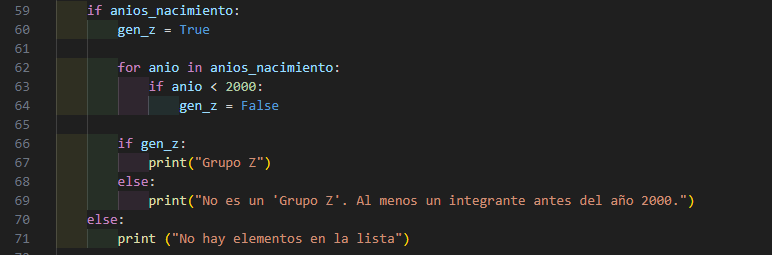
**2. Conteo de Años Pares e Impares**

Implementé el bloque de código encargado de clasificar los años de nacimiento en pares o impares. Si se han ingresado años, el programa recorre la lista completa de años de nacimiento. Por cada año, verifica si es divisible por 2 para determinar si es par o impar y actualiza contadores específicos. Una vez procesados todos los años, se imprime el total de años pares y el total de años impares, ofreciendo una perspectiva inicial sobre la distribución de los años en el grupo.



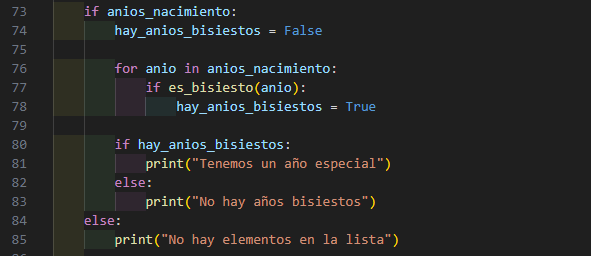
**3. Verificación de la "Generación Z"**

Me encargué de la sección que comprueba si el grupo pertenece a la "Generación Z". Esta lógica se basa en la condición de que todos los integrantes deben haber nacido después del año 2000. Se utiliza una bandera gen\_z, que se inicializa en True (asumiendo que es Generación Z). El código luego itera por cada año de nacimiento; si encuentra un solo año anterior o igual a 2000, la bandera gen\_z se cambia a False. Finalmente, se imprime un mensaje indicando si el grupo cumple o no con la condición de ser "Generación Z".



**4. Detección de Años de Nacimiento Bisiestos**

Implementé el bloque de código que identifica si algún integrante del grupo nació en un año bisiesto. Este proceso recorre la lista de años de nacimiento y, para cada año, invoca la función es\_bisiesto() que definí previamente. Si la función retorna True para alguno de los años, una bandera hay\_anios\_bisiestos se activa. A diferencia de otros enfoques, el bucle continúa hasta el final de la lista incluso después de encontrar un bisiesto. Esto garantiza que todos los años sean revisados si fuera necesario para otras lógicas (aunque para este punto, basta con encontrar uno). Finalmente, el programa informa si "Tenemos un año especial" o si ningún integrante nació en un año bisiesto.



Conclusiones

A lo largo de este trabajo, logramos evidenciar cómo la programación se convierte en una herramienta potente para aplicar y visualizar conceptos matemáticos. La implementación en Python nos permitió no solo automatizar cálculos y operaciones con conjuntos, sino también profundizar en el análisis de datos como los años de nacimiento, mediante funciones que detectan años bisiestos o evalúan condiciones lógicas.

Esta integración favoreció una comprensión más profunda de ambas disciplinas, mostrando que la matemática no es solo teoría, y que la programación, más allá de lo técnico, requiere de pensamiento lógico y estructura. Al trabajar en equipo, también aprendimos a coordinar responsabilidades, compartir conocimientos y construir una solución conjunta, reforzando nuestras competencias profesionales en desarrollo de software y resolución de problemas.

Bibliografia

* Material teórico de las unidades “Conjuntos” y “Lógica”, de la materia Matemática de la Tecnicatura en Programación de la UTN.
* Material audio-visual de las unidades “Conjuntos” y “Lógica”, de la materia Matemática de la Tecnicatura en Programación de la UTN.
* Material audio-visual de las unidades “Funciones”, “Listas", “Est. Condicionales”, “Est. repetitivas” y “Datos complejos”, de la materia Programación de la Tecnicatura en Programación de la UTN.

LINK AL VIDEO DE YOUTUBE:

<https://youtu.be/9Jwo0XRN-M8>