Trabajo Práctico Matemática y Programación:

Semana de Integración II

Alumnos

Betina Sanabria, Esteban Rivarola, Agustín Rivarola, Daniela Romero, Juan Romero

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Matemática

Docente Titular

Eduardo Mónaco

Docente Tutor

Mónica Leguiza

13 de Junio de 2025

Índice

1. Introducción

El siguiente es un trabajo grupal enfocado en la aplicación práctica de los conceptos matemáticos de conjuntos y lógica mediante la creación de un programa en Python, integrando así los contenidos de la materia con los de Programación I. El objetivo es combinar los saberes de ambas materias para resolver problemas que involucran la manipulación de datos (DNIs y años de nacimiento), implementando operaciones y proposiciones lógicas.

---

2. Consignas

El trabajo tiene como requisitos:

- Parte 1: Desarrollo matemático
  Se deben crear conjuntos de dígitos únicos a través de DNIs, y luego realizar operaciones con ellos. A su vez, es necesario incluir diagramas de Venn y formulación de expresiones lógicas y sus correspondientes comprobaciones.

- Parte 2: Desarrollo en Python
  Se debe implementar un programa que automatice la creación y operaciones con conjuntos de DNIs y años de nacimiento. Las operaciones incluyen: conteo, sumas, evaluación de condiciones lógicas, cálculo de producto cartesiano y determinación de si el año es bisiesto.

- Parte 3: Video de presentación: Exposición grupal del programa, incluyendo explicaciones individuales de todos los integrantes y una reflexión general de lo aprendido en la integración matemática-programación.

3. Desarrollo Matemático: Conjuntos y Lógica

Para el desarrollo del presente trabajo se utilizaron los siguientes datos personales correspondientes a los integrantes:

ESTUDIANTE	DNI	FECHA DE NAC.
Agustín Rivarola	29.714.455	07/11/1982
Esteban Rivarola	33.816.092	21/06/1988
Daniela Romero	38.440.237	27/05/1994
Juan Romero	38.090.791	17/05/1995
Betina Sanabria	33.466.642	18/05/1987

3.1 Operaciones con DNIs

3.1.1 Creación de los conjuntos

Con los datos de los DNI creamos conjuntos con dígitos únicos (ya que los conjuntos no tienen elementos repetidos), y los ordenamos.

Esteban Rivarola: 33.816.092 → Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Agustín Rivarola: 29.714.455 → Conjunto A = {1, 2, 4, 5, 7, 9}
Daniela Romero: 38.440.237 → Conjunto D = {0, 2, 3, 4, 7, 8}
Juan Romero: 38.090.791 → Conjunto J = {0, 1, 3, 7, 8, 9}
Betina Sanabria: 33.466.642 → Conjunto B = {2, 3, 4, 6}

3.1.2 Operaciones entre los conjuntos

A continuación se detallan los resultados de las operaciones realizadas entre los 10 pares posibles formados por los 5 conjuntos resultantes de los DNIs.

Par 1: Conjunto E y conjunto A

Conjunto E	{0, 1, 2, 3, 6, 8, 9}
Conjunto A	{1, 2, 4, 5, 7, 9}
Unión (E ∪ A)	{0,1, 2,3, 4, 5,6, 7,8, 9}
Intersección (E ∩ A)	{1,2,9}
Diferencia (E - A)	{*0,3,6,8*}
Diferencia (A - E)	{*4,5,7*}
Diferencia Simétrica (E Δ A)	{0,3,4,5,6,7,8}

Par 2: Conjunto E y conjunto D

Conjunto E	{0, 1, 2, 3, 6, 8, 9}
Conjunto D	{0, 2, 3, 4, 7, 8}
Unión (E ∪ D)	{0,1,2,3,4,6,7,8,9}
Intersección (E ∩ D)	{0,2,3,8}
Diferencia (E - D)	{1,6,9}
Diferencia (D - E)	{4,7}

Diferencia Simétrica (E Δ D)	{1,4,6,7,9}

Par 3: Conjunto E y conjunto J

Conjunto E	{0, 1, 2, 3, 6, 8, 9}
Conjunto J	{0, 1, 3, 7, 8, 9}
Unión (E ∪ J)	{0, 1, 2, 3, 6, 7, 8, 9}
Intersección (E ∩ J)	{0, 1, 3, 8, 9}
Diferencia (E - J)	{2, 6}
Diferencia (J - E)	{7}
Diferencia Simétrica (E Δ J)	{2, 6, 7}

Par 4: Conjunto E y conjunto B

Conjunto E	{0, 1, 2, 3, 6, 8, 9}
Conjunto B	{2, 3, 4, 6}
Unión (E ∪ A)	{0 ,1, 2, 3, 4, 6, 8, 9}
Intersección (E ∩ B)	{2, 3, 6}
Diferencia (E - B)	{0, 1, 8, 9}
Diferencia (B - E)	{4}
Diferencia Simétrica (E Δ B)	{0, 1, 4, 8, 9}

Par 5: Conjunto A y conjunto D

Conjunto A	{1, 2, 4, 5, 7, 9}
Conjunto D	{0, 2, 3, 4, 7, 8}
Unión (A ∪ D)	{0, 1, 2, 3, 4, 5, 7, 8, 9}
Intersección (A ∩ D)	{2, 4, 7}
Diferencia (A - D)	{1, 5, 9}
Diferencia (D - A)	{0, 3, 8}
Diferencia Simétrica (A Δ D)	{0, 1, 3, 5, 8, 9}

Par 6: Conjunto A y conjunto J

Conjunto A	{1, 2, 4, 5, 7, 9}
Conjunto J	{0, 1, 3, 7, 8, 9}
Unión (A ∪ J)	{0, 1, 2, 3, 4, 5, 7, 8, 9}
Intersección (A ∩ J)	{1, 7, 9}
Diferencia (A - J)	{2, 4, 5}
Diferencia (J - A)	{0, 3, 8}
Diferencia Simétrica (A Δ J)	{0, 2, 3, 4, 5, 8}

Par 7: Conjunto A y conjunto B

Conjunto A	{1, 2, 4, 5, 7, 9}
Conjunto B	{2, 3, 4, 6}

Unión (A ∪ B)	{1, 2, 3, 4, 5, 6, 7, 9}
Intersección (A ∩ B)	{2, 4}
Diferencia (A - B)	{1, 5, 7, 9}
Diferencia (B - A)	{3, 6}
Diferencia Simétrica (A Δ B)	{1, 3, 5, 6, 7, 9}

Par 8: Conjunto D y conjunto J

Conjunto D	{0, 2, 3, 4, 7, 8}
Conjunto J	{0, 1, 3, 7, 8, 9}
Unión (D ∪ J)	{0, 1, 2, 3, 4, 7, 8, 9}
Intersección (D ∩ J)	{0, 3, 7, 8}
Diferencia (D- J)	{2, 4}
Diferencia (J - D)	{1, 9}
Diferencia Simétrica (D Δ J)	{1, 2, 4, 9}

Par 9: Conjunto D y conjunto B

Conjunto D	{0, 2, 3, 4, 7, 8}
Conjunto B	{2 ,3 ,4, 6}
Unión (D ∪ B)	{0, 2, 3, 4, 6, 7, 8}
Intersección (D ∩ B)	{2, 3, 4,}
Diferencia (D - B)	{0, 7, 8}
Diferencia (B - D)	{6}

Diferencia Simétrica (D Δ B)	{0, 6, 7, 8}

Par 10: Conjunto B y conjunto J

Conjunto B	{2, 3, 4, 6}
Conjunto J	{0, 1, 3, 7, 8, 9}
Unión (B ∪ J)	{0, 1, 2, 3, 4, 6, 7, 8, 9}
Intersección (B ∩ J)	{3}
Diferencia (B - J)	{2, 4 ,6}
Diferencia (J - B)	{0, 1, 7, 8, 9}
Diferencia Simétrica (B Δ J)	{0, 1, 2, 4, 6, 7, 8, 9}

3.1.3 Diagramas de Venn

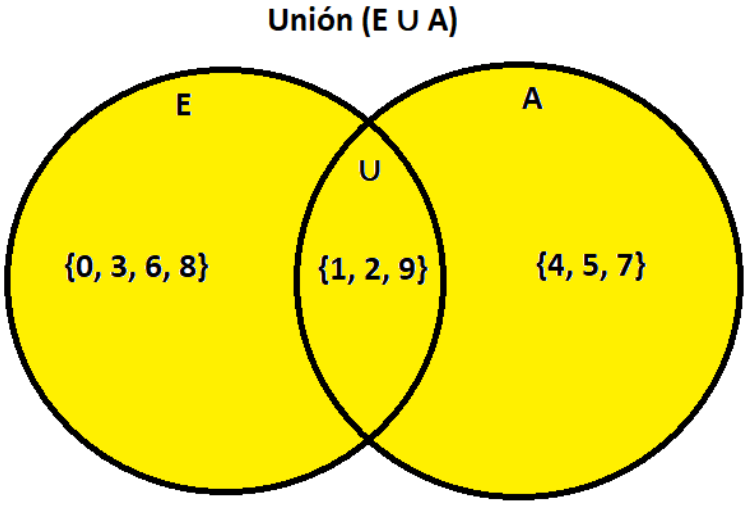
A continuación, se presentan los correspondientes diagramas de Venn de las operaciones entre el Par 1 de conjuntos para dar un ejemplo visual de los resultados de 5 de las 50 operaciones realizadas. Recordemos los conjuntos:

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}

He aquí los resultados de las operaciones graficados en diagramas de Venn:

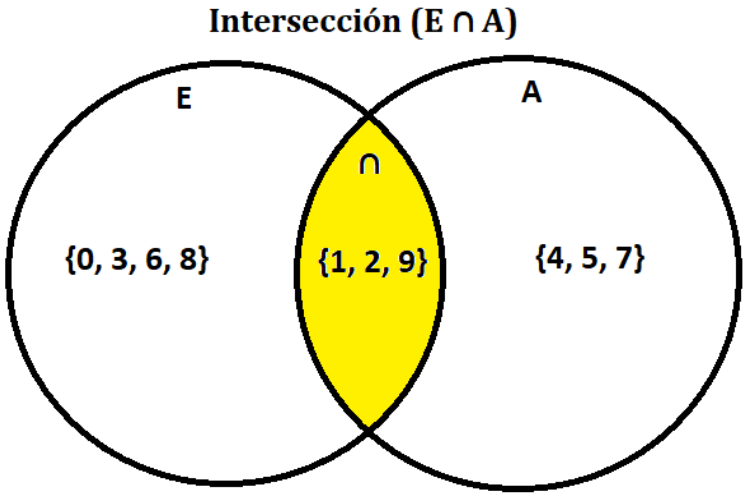
Unión (E ∪ A)

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}



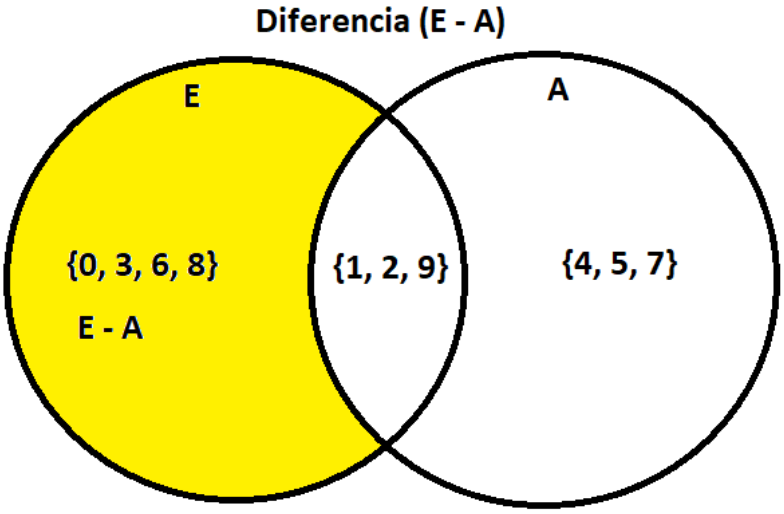
Intersección (E ∩ A)

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}



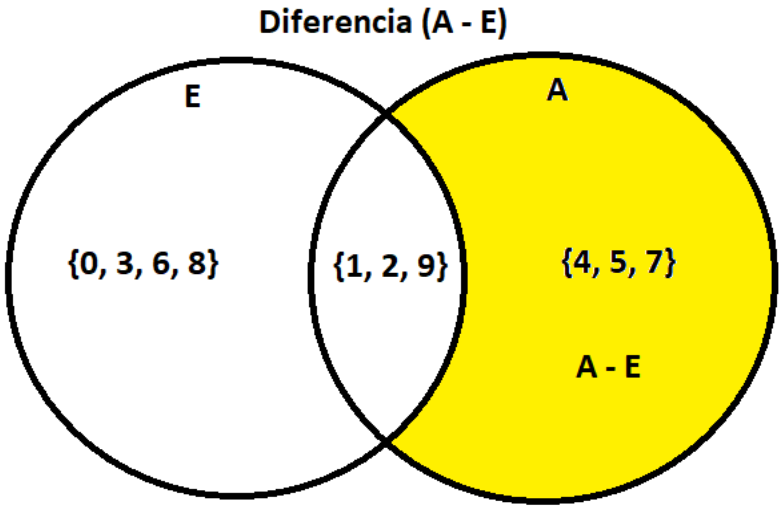
Diferencia (E - A)

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}



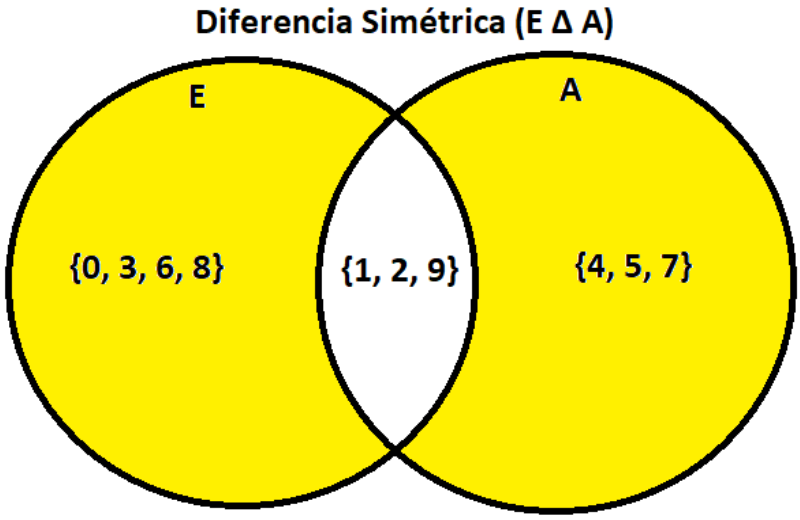
Diferencia (A - E)

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}



Diferencia Simétrica (E Δ A)

Conjunto E = {0, 1, 2, 3, 6, 8, 9}
Conjunto A = {1, 2, 4, 5, 7, 9}



3.1.4 Expresiones Lógicas

Expresión 1: "Grupo con tendencia par"

"Si todos los conjuntos de dígitos únicos de los DNIs contienen al menos un número par (0, 2, 4, 6 u 8), entonces el grupo tiene una tendencia par."

Resultado con nuestros conjuntos: Verdadero.

Análisis:

Conjunto E: contiene 0, 2, 6, 8 ⟶ V.

Conjunto A: contiene 2, 4 ⟶ V.

Conjunto D: contiene 0, 2, 4, 8 ⟶ V.

Conjunto J: contiene 0, 8 ⟶ V.

Conjunto B: contiene 2, 4, 6 ⟶ V.

Todos los conjuntos contienen al menos un número par, por lo tanto, la expresión es Verdadera.

Expresión 2: "Unión completa"

“Si al menos un conjunto individual, o la unión de dos conjuntos, contiene todos los dígitos del 0 al 9, entonces se considera que hay una unión completa.

Resultado con nuestros conjuntos: Verdadero.

Análisis:

Conjunto A (Agustín): {1, 2, 4, 5, 7, 9}.

Conjunto E (Esteban): {0, 1, 2, 3, 6, 8, 9}.

Unión A ∪ **E =** {0, 1, 2, 3, 4, 5, 6, 7, 8, 9} ⟶ contiene todos los dígitos del 0 al 9 ⟶ V.

En conclusión, la expresión es Verdadera.

Expresión 3: "Grupo con elementos extremos"

“Si al menos un conjunto contiene el dígito 0 y otro distinto contiene el dígito 9, entonces el grupo contiene elementos extremos.”

Resultado con nuestros conjuntos: Verdadero.

Análisis:

Conjunto E: contiene 0 y 9 ⟶ V.

Conjunto A: contiene 9 ⟶ V.

Conjunto D: contiene 0 ⟶ V.

Conjunto J: contiene 0 y 9 ⟶ V.

Conjunto B: no contiene 0 ni 9 ⟶ F.

Hay al menos un conjunto con 0 (por ejemplo, D) y otro distinto con 9 (por ejemplo, A), por lo tanto, la expresión es Verdadera.

4. Desarrollo en Python

4.1 Operaciones con DNIs

A continuación se adjunta el código de nuestro programa que realiza operaciones con DNIs. Este módulo es el corazón de nuestro proyecto, donde los principios de conjuntos y lógica encuentran aplicación práctica. Acá se puede ver cómo manejamos el ingreso de DNIs, los transformamos en conjuntos y realizamos diversas operaciones con ellos.

```
#FUNCIONES PARA OPERAR SOLICITAR Y VERIFICAR DNIs
#Generador de conjuntos a partir de un número
def verificador_digito_numerico(dni):
    for caracter in dni:
        if caracter not in "0123456789":
            return False
    return True

def solicitar_dni():
    # Solicita un DNI al usuario hasta que se introduce uno válido
    while True:
        dni = input("Introduce un DNI: ")
        if verificador_digito_numerico(dni) and len(str(dni)) == 8:
            return dni
        else:
            print("El número solo puede tener caracteres numéricos y debe
tener 8 dígitos. Por favor, introduce un DNI válido.")

#FUNCIONES PARA OPERAR CON CONJUNTOS

# Función para generar un conjunto a partir de un DNI numérico
def generar_conjunto(dni):
```

```
    # Genera un conjunto de dígitos únicos a partir de un DNI numérico.
    # Devuelve False si el DNI no es numérico.
    conjunto = []
    for i in dni:
        if int(i) not in conjunto:
            conjunto.append(int(i))
    return sorted(conjunto)

#operaciones con conjuntos
def union_conjuntos(conjunto1, conjunto2):
    union = conjunto1
    for elemento in conjunto2:
        if elemento not in union:
            union.append(elemento)
    return sorted(union)

def interseccion_conjuntos(conjunto1, conjunto2):
    interseccion = []
    for e in conjunto1:
         if e in conjunto2:
            interseccion.append(e)
    return sorted(interseccion)

def diferencia_conjuntos(conjunto1, conjunto2):
    diferencia = []
    for e in conjunto1:
        if e not in conjunto2:
            diferencia.append(e)
    return sorted(diferencia)

def diferencia_simetrica_conjuntos(conjunto1, conjunto2):
    diferencia1 = diferencia_conjuntos(conjunto1, conjunto2)
    diferencia2 = diferencia_conjuntos(conjunto2, conjunto1)
    return sorted(diferencia1 + diferencia2)

def operar_con_conjuntos(conjunto1, conjunto2, operacion):
```

```
    if operacion == "union":
        return union_conjuntos(conjunto1, conjunto2)
    elif operacion == "interseccion":
        return interseccion_conjuntos(conjunto1, conjunto2)
    elif operacion == "diferencia":
        return diferencia_conjuntos(conjunto1, conjunto2)
    elif operacion == "diferencia_simetrica":
        return diferencia_simetrica_conjuntos(conjunto1, conjunto2)
    else:
        print("Operación no válida.")

#Programa para conjuntos de DNIs
def conjuntos_dnis():
    # Función principal el la que usuario genera los conjuntos
correspondientes a los DNIs y realiza operaciones entre ellos.
    print("Bienvenido al generador de conjuntos a partir de DNIs.")
    print("Por favor, introducí dos DNIs para generar los conjuntos y
realizar operaciones entre ellos.")
    conjunto1 = generar_conjunto(solicitar_dni())
    conjunto2 = generar_conjunto(solicitar_dni())
    operacion = input("Introduce la operación (union, interseccion,
diferencia, diferencia_simetrica): ").strip().lower()
    resultado = operar_con_conjuntos(conjunto1, conjunto2, operacion)
    if resultado is not None:
        print(f"El resultado de la operación {operacion} entre los
conjuntos es:\nConjunto 1: {conjunto1} \nConjunto
2:{conjunto2}\n{operacion.capitalize()}: {resultado}")
    else:
        print("No se pudo realizar la operación.")


#FUNCION PARA SUMAR LOS DÍGITOS DE UN DNI
def sumar_digitos():
    dni= input("Introduce un DNI: ")
    if verificador_digito_numerico(dni) and len(str(dni)) == 8:
        suma = 0
```

```
        for caracter in dni:
            if caracter in "0123456789":
                suma += int(caracter)
        print(f"La suma de los dígitos del DNI {dni} es: {suma}")

#FUNCIÓN PARA CONTAR LA FRECUENCIA DE LOS DÍGITOS EN UN DNI
def contar_frecuencia_digitos():
    dni = solicitar_dni()
    frecuencia = {}
    for caracter in dni:
        if caracter in frecuencia:
            frecuencia[caracter] += 1
        else:
            frecuencia[caracter] = 1
    print(f"La frecuencia de los dígitos en el DNI {dni} es:
{frecuencia}")


#FUNCIONES PARA VERIFICAR EXPRESIONES LÓGICAS

#EXPRESIÓN LÓGICA 1: Función para determinar tendencia par en el grupo de
DNIs de integrantes
def tendencia_par():
    integrantes = int(input("Introduce el número de integrantes del grupo:
"))
    tendencias = []
    for i in range(integrantes):
        dni = input(f"Introduce el DNI del integrante {i + 1}: ")
        tendencias.append(verificar_tendencia_par(dni))
    if tendencias.count(True) == integrantes:
        print("El grupo de DNIs de los integrantes tiene tendencia par.")
    else:
        print("El grupo de DNIs de los integrantes no tiene tendencia
par.")

def verificar_tendencia_par(dni):
```

```
    if verificador_digito_numerico(dni) and len(dni) == 8:
        pares = 0

        for caracter in dni:
            if int(caracter) % 2 == 0:
                pares += 1
        if pares > 0:
            return True
        else:
            return False
    else:
        print("El número solo puede tener caracteres numéricos y debe
tener 8 dígitos.")

#EXPRESIÓN LÓGICA 2: Función para determinar si existe una unión completa
entre dos conjuntos de DNIs
def union_completa():
    integrantes = int(input("Introduce el número de integrantes del grupo:
"))
    union = []
    for i in range(integrantes):
       conjunto = generar_conjunto(solicitar_dni())
       for e in conjunto:
           if e not in union:
               union.append(e)
    if len(union) == 10:  # Si la unión tiene todos los dígitos del 0 al 9
        print("Existe una unión completa entre los conjuntos de DNIs, ya
que contiene todos los dígitos del 0 al 9.")
    else:
        print("No existe una unión completa entre los conjuntos de DNIs,
ya que falta alguno de los dígitos del 0 al 9.")


#EXPRESIÓN LÓGICA 3: Función para determinar si existen elementos extremos
(0 o 9) en al menos uno de los conjuntos de DNIs
```

```
def extremos_en_conjuntos():
    integrantes = int(input("Introduce el número de integrantes del grupo: "))
    union = []
    for i in range(integrantes):
        conjunto = generar_conjunto(solicitar_dni())
        for e in conjunto:
            if e not in union:
                union.append(e)
    if "0" in union or "9" in union:
        print("Existen elementos extremos (0 o 9) en al menos uno de los conjuntos de DNIs.")
    else:
        print("No existen elementos extremos (0 o 9) en los conjuntos de DNIs.")

# FUNCIÓN PARA EJECUTAR EL PROGRAMA PRINCIPAL
def main():
    while True:
        print("\nOpciones:")
        print("1. Operar con conjuntos a partir de DNIs")
        print("2. Sumar dígitos de un DNI")
        print("3. Contar frecuencia de dígitos en un DNI")
        print("4. Verificar tendencia par en un grupo de DNIs")
        print("5. Verificar unión completa entre conjuntos de DNIs")
        print("6. Verificar existencia de elementos extremos (0 o 9) en conjuntos de DNIs")
        print("7. Salir")

        opcion = input("Selecciona una opción: ")

        if opcion == "1":
            conjuntos_dnis()
        elif opcion == "2":
            sumar_digitos()
        elif opcion == "3":
```

```
            contar_frecuencia_digitos()
        elif opcion == "4":
            tendencia_par()
        elif opcion == "5":
            union_completa()
        elif opcion == "6":
            extremos_en_conjuntos()
        elif opcion == "7":
            print("Saliendo del programa.")
            break
        else:
            print("Opción no válida, por favor intenta nuevamente.")

if __name__ == "__main__":
    main()
```

4.2 Operaciones con años de nacimiento

En esta sección, se utilizaron los años de nacimiento de los integrantes del grupo para realizar operaciones que permitieran aplicar conceptos matemáticos y lógicos mediante programación en Python. A partir de esos años, se llevaron a cabo las siguientes acciones:

- 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

Estas operaciones permiten ilustrar cómo la lógica matemática puede aplicarse no solo a los DNIs, sino también a otros tipos de datos numéricos como fechas, integrando así el

razonamiento lógico con el uso práctico de programación. El desarrollo en Python automatiza estos cálculos, permitiendo obtener resultados precisos de manera eficiente.

```
# Importa función de verificación numérica desde el módulo
'dnis.py'
from dnis import verificador_digito_numerico


def es_bisiesto(year):
    return (year % 4 == 0 and year % 100 != 0) or (year % 400 ==
0)


#Funcion para solicitar y verificar años de nacimiento
def solicitar_anios():
    integrantes = int(input("Introduce el número de integrantes
del grupo: "))
    anios = []
    for i in range(integrantes):
        anio = input(f"Introduce el año de nacimiento del
integrante {i + 1}: ")
        if verificador_digito_numerico(anio) and len(anio) == 4:
            anios.append(int(anio))
        else:
            print("El año debe tener 4 dígitos numéricos.")
    return anios


# Funciones para realizar operaciones con los años de nacimiento
```

```
## Contar años bisiestos, verificar si todos nacieron después del 2000, calcular edades y producto cartesiano entre años y edades

def contar_bisiestos(anios):

    bisiestos = 0

    for anio in anios:

        if es_bisiesto(anio):

            bisiestos += 1

            print(f"Tenemos un año especial: el año {anio} es bisiesto.")

    print(f"El número de años bisiestos en el grupo es {bisiestos}")


def contar_posterior_2000(anios):

    posterior_2000 = []

    for anio in anios:

        if anio > 2000:

            posterior_2000.append(anio)

    if posterior_2000 == anios:

        print("Grupo Z: todos los integrantes del grupo nacieron después del 2000.")

    else:

        print("No todos los integrantes del grupo nacieron después del 2000.")


def calcular_edades(anios):

    edades = []
```

```
    for anio in anios:
        edad = 2025 - anio
        edades.append(edad)
    print(f"Las edades de los integrantes del grupo son: {edades}")
    return edades


def producto_cartesiano(anios):
    edades = calcular_edades(anios)
    producto = [(a, e) for a in anios for e in edades]
    print(f"El producto cartesiano entre los años y las edades es: {producto}")



# Función principal para ejecutar el programa
def main():
    print("Bienvenido al programa de análisis de años de nacimiento.")
    print("A continuación, se solicitarán los años de nacimiento con los que se trabajará.")
    anios = solicitar_anios()
    print("\nOperaciones disponibles:")
    print("1. Contar años bisiestos")
    print("2. ¿Todos nacieron después del 2000?")
    print("3. Calcular edades")
```

```
    print("4. Producto cartesiano entre años y edades")
    print("5. Salir")

    while True:
        print("\n----------------------")
        opcion = input("Selecciona una opción (1-5): ")
        if opcion == '1':
            contar_bisiestos(anios)
        elif opcion == '2':
            contar_posterior_2000(anios)
        elif opcion == '3':
            calcular_edades(anios)
        elif opcion == '4':
            producto_cartesiano(anios)
        elif opcion == '5':
            print("Saliendo del programa. ¡Nos vemos!")
            break
        else:
            print("Opción no válida. Por favor, selecciona una
opción del 1 al 5.")

# ejecuta la función principal si el script se ejecuta
directamente
if __name__ == "__main__":
    main()
```

5. Conclusiones

Este trabajo integrador permitió vincular los contenidos de Matemática I con los de Programación I, aplicando conceptos abstractos como conjuntos y lógica en contextos prácticos mediante el uso de Python. A través del análisis de datos reales como los DNIs y los años de nacimiento, pudimos realizar operaciones matemáticas formales y luego reflexionar sobre cómo trasladarlas al lenguaje de programación.

Uno de los aprendizajes más valiosos fue comprender que detrás de cada estructura de código hay una lógica matemática que la respalda, y que desarrollar esa conexión es clave para pensar como programadores. También se destacó el valor del trabajo en equipo, donde cada integrante aportó desde sus fortalezas.

El proceso permitió avanzar significativamente en la comprensión conceptual y en la traducción de ideas lógicas al lenguaje computacional, sentando bases sólidas para futuras integraciones más complejas.

6. Bibliografía

Downey, A. B. (2015). *Think Python: How to Think Like a Computer Scientist* (2nd ed.). Green Tea Press. https://greenteapress.com/wp/think-python-2e/

Universidad Tecnológica Nacional. (2025). *Material de cátedra de Matemática I y Programación I* [apuntes digitales].

7. Anexos

7.1 Enlace a vídeo de YouTube

A continuación se adjunta el enlace al video explicativo del programa desarrollado: ver video.

7.2 Enlace a repositorio del código en GitHub

En el siguiente enlace se puede acceder al repositorio de GitHub con el código desarrollado en este trabajo: ver aquí.

7.3 División de trabajo

Agustín Rivarola: Cálculo de operaciones entre conjuntos (parte matemática), generación y análisis de combinaciones de pares, verificación de resultados, colaboración en los diagramas de Venn.

Betina Sanabria: Redacción y análisis de expresiones lógicas en lenguaje natural (sección 3.1.4). Formulación de condiciones, evaluación lógica y justificación con los conjuntos. Redacción explicativa de la sección 4.2 sobre operaciones con años de nacimiento. Revisión y sugerencias generales para la redacción del documento.

Daniela Romero: Programación en Python de las funciones para generar conjuntos de dígitos a partir de DNIs y realizar operaciones entre ellos (unión, intersección, diferencia, diferencia simétrica). Participación en el diseño de expresiones lógicas. Propuesta de criterios de verificación y estructura del código.

Esteban Rivarola: Redacción del desarrollo matemático (sección 3.1.1 y 3.1.2), sistematización de datos de los DNIs, participación en la elaboración del índice, organización y formato general del documento.

Juan Romero: Desarrollo de funciones adicionales para la interacción del programa en Python. Estructura del flujo principal del programa. Carga y validación de datos. Soporte técnico general

del código.

7.4 Relación entre expresiones lógicas y el código

A continuación se presentan las expresiones lógicas presentadas en este trabajo en lenguaje natural y su implementación en código Python, con una breve explicación de cómo se vinculan.

Expresión lógica: "Grupo con tendencia par"

Esta expresión se encuentra en la sección 3.1.4 del trabajo y establece que "Si todos los conjuntos de dígitos únicos de los DNIs contienen al menos un número par (0, 2, 4, 6 u 8), entonces el grupo tiene una tendencia par."

Fragmento de código relacionado:

```
#EXPRESIÓN LÓGICA 1: Función para determinar tendencia par en el grupo de
DNIs de integrantes
def tendencia_par():
    integrantes = int(input("Introduce el número de integrantes del grupo:
"))
    tendencias = []
    for i in range(integrantes):
        dni = input(f"Introduce el DNI del integrante {i + 1}: ")
        tendencias.append(verificar_tendencia_par(dni))
    if tendencias.count(True) == integrantes:
        print("El grupo de DNIs de los integrantes tiene tendencia par.")
    else:
        print("El grupo de DNIs de los integrantes no tiene tendencia
par.")

def verificar_tendencia_par(dni):
    if verificador_digito_numerico(dni) and len(dni) == 8:
        pares = 0

        for caracter in dni:
            if int(caracter) % 2 == 0:
```

```
                pares += 1
        if pares > 0:
            return True
        else:
            return False
    else:
        print("El número solo puede tener caracteres numéricos y debe
tener 8 dígitos.")
```

Explicación:

La función tendencia_par() orquesta la verificación. Solicita los DNI de cada integrante y, para cada uno, llama a verificar_tendencia_par(). Esta función recorre los dígitos del DNI para determinar si existe al menos un dígito par. Finalmente, tendencia_par() compara el número de DNIs con tendencia par (donde verificar_tendencia_par devolvió True) con el total de integrantes. Si todos cumplen la condición, la expresión lógica se evalúa como verdadera.

Expresión lógica: "Unión completa"

Esta expresión se encuentra en la sección 3.1.4 y plantea la siguiente condición:

"Si al menos un conjunto individual, o la unión de dos conjuntos, contiene todos los dígitos del 0 al 9, entonces se considera que hay una unión completa."

Fragmento de código relacionado:

```
#EXPRESIÓN LÓGICA 2: Función para determinar si existe una unión completa
entre dos conjuntos de DNIs
def union_completa():
    integrantes = int(input("Introduce el número de integrantes del grupo:
"))
    union = []
    for i in range(integrantes):
       conjunto = generar_conjunto(solicitar_dni())
       for e in conjunto:
           if e not in union:
```

```
                union.append(e)
    if len(union) == 10:  # Si la unión tiene todos los dígitos del 0 al 9
        print("Existe una unión completa entre los conjuntos de DNIs, ya
que contiene todos los dígitos del 0 al 9.")
    else:
        print("No existe una unión completa entre los conjuntos de DNIs,
ya que falta alguno de los dígitos del 0 al 9.")
```

Explicación:

La función union_completa() implementa la siguiente lógica. Primero, solicita los DNI de todos los integrantes y construye una unión acumulativa de todos los dígitos únicos presentes en todos los conjuntos. Aunque la expresión menciona "un conjunto individual o la unión de dos conjuntos", la implementación simplifica esto al verificar si la unión global de todos los conjuntos alcanza los 10 dígitos (0-9). Si la longitud de esta unión acumulada es 10, la condición se cumple, indicando una "unión completa".

Expresión lógica: "Grupo con elementos extremos"

Esta expresión se encuentra en la sección 3.1.4 del trabajo y establece que “Si al menos un conjunto contiene el dígito 0 y otro distinto contiene el dígito 9, entonces el grupo contiene elementos extremos.”

Fragmento de código relacionado:

```
#EXPRESIÓN LÓGICA 3: Función para determinar si existen elementos extremos
(0 o 9) en al menos uno de los conjuntos de DNIs

def extremos_en_conjuntos():
    integrantes = int(input("Introduce el número de integrantes del grupo:
"))
    union = []
    for i in range(integrantes):
        conjunto = generar_conjunto(solicitar_dni())
```

```
        for e in conjunto:
            if e not in union:
                union.append(e)
    if "0" in union or "9" in union:
        print("Existen elementos extremos (0 o 9) en al menos uno de los conjuntos de DNIs.")
    else:
        print("No existen elementos extremos (0 o 9) en los conjuntos de DNIs.")
```

Explicación:

La función extremos_en_conjuntos() recorre los conjuntos de cada DNI ingresado. Utiliza banderas (contiene_cero, contiene_nueve) para registrar si se encuentra el dígito 0 y el dígito 9, respectivamente, en cualquiera de los conjuntos. La lógica final comprueba si ambas banderas son True. Esto indica que el 0 apareció en al menos un DNI y el 9 apareció en al menos otro DNI, cumpliendo la condición de "elementos extremos". Es crucial notar que si un mismo DNI contuviera tanto el 0 como el 9, la condición también se consideraría cumplida, ya que "otro distinto" no implica necesariamente una exclusión mutua de los dígitos en un mismo conjunto.