Trabajo Integrador 2: Matemática y Programación

Integrantes:
- Roqué, Gabriel Osvaldo
- Airalde, Milagros Abril

Comisión:
- Gabriel Osvaldo Roque: **Comisión 5.**
- Milagros Abril Airalde: **Comisión 1.**

Link al video: https://youtu.be/xpUfarlD2zI

Objetivo:
El objetivo de este trabajo es integrar los contenidos de Matemática (conjuntos y lógica) con Programación (estructuras condicionales, repetitivas y funciones), fortaleciendo el trabajo en equipo, la comunicación clara y la responsabilidad individual en un proyecto colaborativo.

Parte 1 – Desarrollo Matemático

1. DNIs :
Se utilizaron DNIs de ejemplo para realizar los cálculos:
- **DNI 1:** 38945345
- **DNI 2:** 41355780

2. Conjuntos de números únicos
A partir de los DNIs, se generaron los siguientes conjuntos de números únicos:
- **Conjunto 1:** [3, 4, 5, 8, 9]
- **Conjunto 2:** [0, 1, 3, 4, 5, 7, 8]

3. Operaciones con conjuntos
Se realizaron las siguientes operaciones entre los conjuntos, con resultados ordenados de menor a mayor:
- **Unión:** [0, 1, 3, 4, 5, 7, 8, 9]
  - Representa todos los números presentes en el Conjunto 1, Conjunto 2, o ambos, sin repeticiones.

- **Intersección:** [3, 4, 5, 8]
  - Representa los números comunes a ambos conjuntos.

- **Diferencia (Conjunto 1 - Conjunto 2):** [9]
  - Representa los números que están en el Conjunto 1 pero no en el Conjunto 2.

- **Diferencia (Conjunto 2 - Conjunto 1):** [0, 1, 7]
  - Representa los números que están en el Conjunto 2 pero no en el Conjunto 1.

- Diferencia simétrica: [0, 1, 7, 9]

- Representa los números que están en el Conjunto 1 o en el Conjunto 2, pero no en ambos.

4. Diagramas de Venn

Se elaboraron diagramas de Venn para cada operación, dibujados a mano en un cuaderno y fotografiados para su inclusión en este documento. Los diagramas representan:

- Unión: Ambos círculos con todos los números [0, 1, 3, 4, 5, 7, 8, 9], resaltando la región completa.

- Intersección: La región central de los círculos con los números comunes [3, 4, 5, 8].

- Diferencia (Conjunto 1 - Conjunto 2): La parte del círculo del Conjunto 1 con el número [9].

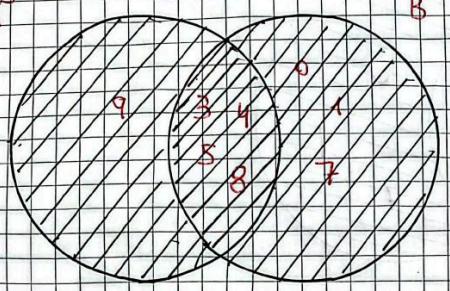
- Diferencia (Conjunto 2 - Conjunto 1): La parte del círculo del Conjunto 2 con los números [0, 1, 7].

- Diferencia simétrica: Las partes no superpuestas del Conjunto 1 ([9]) y Conjunto 2 ([0, 1, 7]).

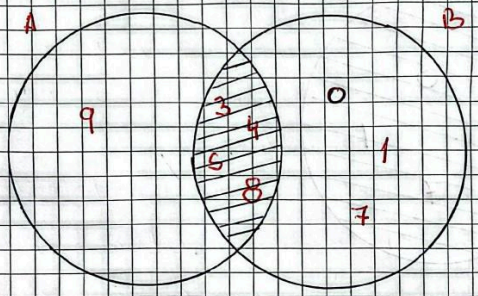
[Imágenes de los diagramas de Venn insertadas en el documento]

1) $A \cup B$

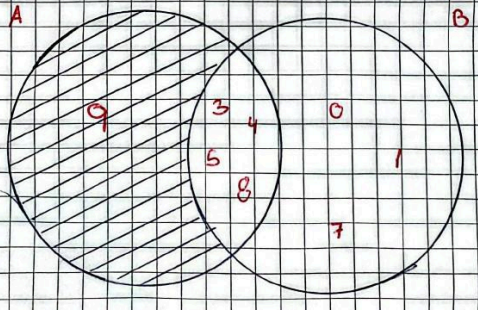
$A \cup B = \{0, 1, 3, 4, 5, 7, 8, 9\}$

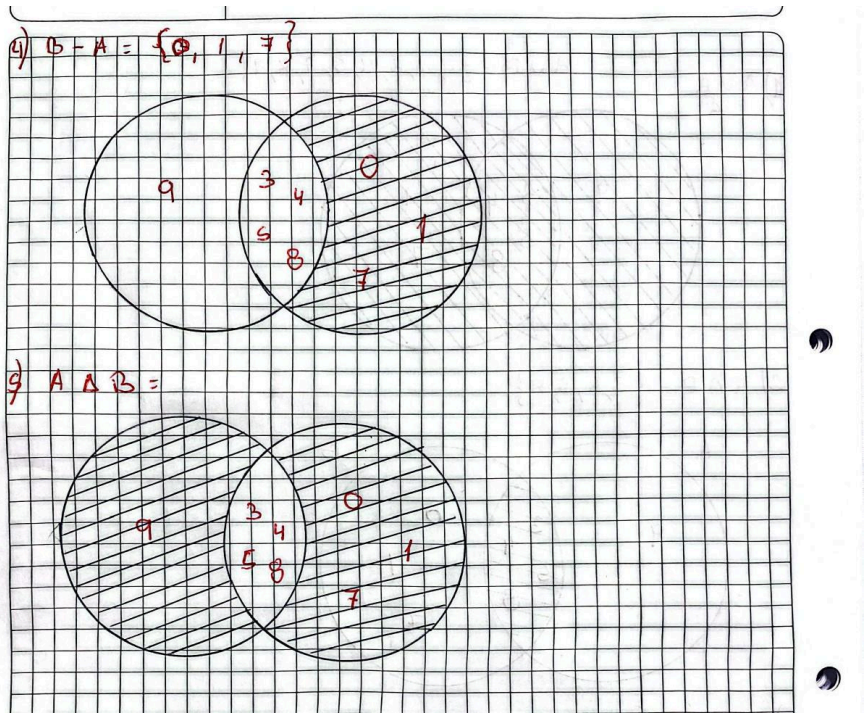


2) $A \cap B = \{3, 4, 5, 8\}$



3) $A - B = \{9\}$





5. Expresiones lógicas en lenguaje natural

Se definieron cinco expresiones lógicas basadas en los conjuntos, con sus resultados:

- Expresión 1: "Si ambos conjuntos tienen al menos 5 números, entonces se considera alta diversidad numérica."

- Resultado: Conjunto 1 tiene 5 números, Conjunto 2 tiene 7 números. Se cumple porque ambos tienen al menos 5 números.

- Expresión 2: "Si la intersección entre los conjuntos tiene más de 3 números, entonces hay un número común significativo."

- Resultado: Intersección = [3, 4, 5, 8] (4 números). Se cumple porque 4 > 3.

- Expresión 3: "Si algún conjunto no contiene el número 0, entonces se considera un grupo sin ceros."

- Resultado: Conjunto 1 no contiene el número 0, Conjunto 2 sí lo contiene. Se cumple porque al menos un conjunto no tiene el 0.

- Expresión 4: "Si la diferencia simétrica tiene más números que la intersección, entonces hay mayor diversidad exclusiva."

- Resultado: Diferencia simétrica = [0, 1, 7, 9] (4 números), Intersección = [3, 4, 5, 8] (4 números). No se cumple porque 4 no es mayor que 4.

- Expresión 5: "Si la unión de los conjuntos contiene todos los números del 0 al 9, entonces se considera un grupo completo."

- Resultado: Unión = [0, 1, 3, 4, 5, 7, 8, 9] (8 números). No se cumple porque faltan los números 2 y 6.

Parte 2 – Desarrollo del Programa en Python

Descripción del programa
Se desarrolló un programa en Python compuesto por dos archivos:
- **IntegradorMatematica.py:** Contiene la lógica principal, incluyendo ingreso de datos, validaciones, operaciones con conjuntos, evaluación de expresiones lógicas y procesamiento de años de nacimiento.
- **funciones.py:** Contiene las funciones auxiliares para contar frecuencias, sumar números y verificar años bisiestos.

El programa permite al usuario ingresar dos nombres, dos DNIs de 8 números y dos años de nacimiento. Sus funcionalidades son:
- Genera conjuntos de números únicos a partir de los DNIs, mostrándolos ordenados de menor a mayor.
- Calcula y muestra la unión, intersección, diferencia (1-2), diferencia (2-1) y diferencia simétrica, todas ordenadas.
- Calcula la frecuencia de cada número en cada DNI, mostrando los resultados ordenados por número.
- Calcula la suma total de los números de cada DNI.
- Evalúa las cinco expresiones lógicas definidas.
- Para los años de nacimiento, cuenta años pares e impares, verifica si ambos son posteriores al 2000 ("Grupo Z"), identifica años bisiestos y calcula el producto cartesiano de años y edades.

Código fuente
Los códigos completos se encuentran en los archivos IntegradorMatematica.py y funciones.py, incluidos en la entrega. A continuación, se muestran ambos:

IntegradorMatematica.py

```python
# Trabajo Integrador 2: Matemática y Programación
# Integrantes: Gabriel y Milagros
# Este programa permite ingresar nombres, DNIs y años de nacimiento,
# genera conjuntos de números únicos, realiza operaciones de conjuntos,
# calcula frecuencias (ordenadas), sumas, evalúa expresiones lógicas,
# y realiza operaciones con años de nacimiento.
# Los conjuntos y frecuencias se muestran ordenados de menor a mayor.
# Las funciones auxiliares están en funciones.py.

from datetime import datetime
from funciones import contar_frecuencias, sumar_numeros, es_bisiesto

# --- Parte 2A: Validación y procesamiento de DNIs ---
```

```
# Pedir al usuario los nombres para los DNIs
nombre1 = input("Ingrese el nombre para el primer DNI: ")
nombre2 = input("Ingrese el nombre para el segundo DNI: ")

# Pedir al usuario que ingrese los DNIs
dni1 = input(f"Ingrese el DNI de {nombre1} (8 números): ")
dni2 = input(f"Ingrese el DNI de {nombre2} (8 números): ")

# Validar que el primer DNI tenga 8 números y sea solo numérico
while not (dni1.isdigit() and len(dni1) == 8):
    print(f"Error: El DNI de {nombre1} debe tener exactamente 8
números.")
    dni1 = input(f"Ingrese el DNI de {nombre1} (8 números): ")

# Validar que el segundo DNI tenga 8 números y sea solo numérico
while not (dni2.isdigit() and len(dni2) == 8):
    print(f"Error: El DNI de {nombre2} debe tener exactamente 8
números.")
    dni2 = input(f"Ingrese el DNI de {nombre2} (8 números): ")

# Crear conjuntos con los números únicos de cada DNI
conjunto_1 = set(dni1)  # Conjunto del primer DNI
conjunto_2 = set(dni2)  # Conjunto del segundo DNI

# Realizar operaciones con conjuntos
union = conjunto_1 | conjunto_2  # Unión: todos los números de
ambos conjuntos
interseccion = conjunto_1 & conjunto_2  # Intersección: números
comunes
diferencia_1_2 = conjunto_1 - conjunto_2  # Diferencia 1-2
diferencia_2_1 = conjunto_2 - conjunto_1  # Diferencia 2-1
diferencia_simetrica = conjunto_1 ^ conjunto_2  # Diferencia
simétrica

# Mostrar los conjuntos y resultados, ordenados de menor a mayor
print(f"\nConjunto de {nombre1}: {sorted(conjunto_1)}")
print(f"Conjunto de {nombre2}: {sorted(conjunto_2)}")
print(f"Unión: {sorted(union)}")
print(f"Intersección: {sorted(interseccion)}")
print(f"Diferencia ({nombre1} - {nombre2}):
{sorted(diferencia_1_2)}")
print(f"Diferencia ({nombre2} - {nombre1}):
{sorted(diferencia_2_1)}")
print(f"Diferencia simétrica: {sorted(diferencia_simetrica)}")

# Calcular frecuencias (ordenadas) y sumas para cada DNI
frec_1 = contar_frecuencias(dni1)  # Frecuencias del primer DNI
frec_2 = contar_frecuencias(dni2)  # Frecuencias del segundo DNI
```

```
suma_1 = sumar_numeros(dni1)  # Suma del primer DNI
suma_2 = sumar_numeros(dni2)  # Suma del segundo DNI

# Mostrar frecuencias y sumas
print(f"\nFrecuencias de {nombre1}: {frec_1}")
print(f"Frecuencias de {nombre2}: {frec_2}")
print(f"Suma de números de {nombre1}: {suma_1}")
print(f"Suma de números de {nombre2}: {suma_2}")

# Evaluación de las expresiones lógicas
print("\nEvaluación de expresiones lógicas:")
# Expresión 1: Ambos conjuntos tienen al menos 5 números
if len(conjunto_1) >= 5 and len(conjunto_2) >= 5:
    print("Expresión 1: Se considera alta diversidad numérica.")
else:
    print("Expresión 1: No se considera alta diversidad
numérica.")

# Expresión 2: La intersección tiene más de 3 números
if len(interseccion) > 3:
    print("Expresión 2: Hay un número común significativo.")
else:
    print("Expresión 2: No hay un número común significativo.")

# Expresión 3: Algún conjunto no contiene el número 0
if '0' not in conjunto_1 or '0' not in conjunto_2:
    print("Expresión 3: Se considera un grupo sin ceros.")
else:
    print("Expresión 3: No se considera un grupo sin ceros.")

# Expresión 4: La diferencia simétrica tiene más números que la
intersección
if len(diferencia_simetrica) > len(interseccion):
    print("Expresión 4: Hay mayor diversidad exclusiva.")
else:
    print("Expresión 4: Vaya, no hay mayor diversidad exclusiva.")

# Expresión 5: La unión contiene todos los números del 0 al 9
if len(union) == 10:
    print("Expresión 5: Se considera un grupo completo.")
else:
    print("Expresión 5: No se considera un grupo completo, faltan
números.")

# --- Parte 2B: Validación y procesamiento de años de nacimiento
---
# Pedir al usuario los nombres para los años de nacimiento
print("\nAhora vamos con los años de nacimiento.")
```

```
nombre_anio1 = input("Ingrese el nombre para el primer año de
nacimiento: ")
nombre_anio2 = input("Ingrese el nombre para el segundo año de
nacimiento: ")

# Pedir y validar el primer año de nacimiento
anio1 = input(f"Ingrese el año de nacimiento de {nombre_anio1}
(por ejemplo, 2000): ")
while not (anio1.isdigit() and 1900 <= int(anio1) <=
datetime.now().year):
    print(f"Error: El año de {nombre_anio1} debe ser un número
entre 1900 y {datetime.now().year}.")
    anio1 = input(f"Ingrese el año de nacimiento de {nombre_anio1}
(por ejemplo, 2000): ")
anio1 = int(anio1)

# Pedir y validar el segundo año de nacimiento
anio2 = input(f"Ingrese el año de nacimiento de {nombre_anio2}
(por ejemplo, 2002): ")
while not (anio2.isdigit() and 1900 <= int(anio2) <=
datetime.now().year):
    print(f"Error: El año de {nombre_anio2} debe ser un número
entre 1900 y {datetime.now().year}.")
    anio2 = input(f"Ingrese el año de nacimiento de {nombre_anio2}
(por ejemplo, 2002): ")
anio2 = int(anio2)

# Crear una lista con los años de nacimiento
anios = [anio1, anio2]

# Contar años pares e impares
pares = 0  # Contador para años pares
impares = 0  # Contador para años impares
for anio in anios:  # Iterar por cada año
    if anio % 2 == 0:  # Si el año es divisible por 2, es par
        pares += 1
    else:  # Si no, es impar
        impares += 1
print(f"\nAños pares: {pares}")
print(f"Años impares: {impares}")

# Verificar si ambos nacieron después del 2000 ("Grupo Z")
es_grupo_z = True  # Suponer que todos son después del 2000
for anio in anios:  # Verificar cada año
    if anio <= 2000:  # Si algún año es 2000 o anterior
        es_grupo_z = False
        break
if es_grupo_z:
```

```
    print("Grupo Z: Todos nacieron después del 2000.")
else:
    print("No todos nacieron después del 2000.")

# Verificar años bisiestos
for anio, nombre in zip(anios, [nombre_anio1, nombre_anio2]):  #
Iterar por años y nombres
    if es_bisiesto(anio):
        print(f"Año {anio} de {nombre} es bisiesto: ¡Tenemos un
año especial!")
    else:
        print(f"Año {anio} de {nombre} no es bisiesto.")

# Calcular edades y producto cartesiano
anio_actual = datetime.now().year  # Año actual (2025)
edades = [anio_actual - anio for anio in anios]  # Calcular edades
producto_cartesiano = [(anio, edad) for anio in anios for edad in
edades]  # Producto cartesiano
print(f"Producto cartesiano (años, edades):
{producto_cartesiano}")
```

funciones.py

```python
# Funciones auxiliares para el Trabajo Integrador 2
# Contiene las funciones para contar frecuencias, sumar números y
verificar años bisiestos

# Función para contar la frecuencia de cada número en un DNI
def contar_frecuencias(dni):
    frecuencias = {}  # Diccionario para almacenar frecuencias
    for numero in dni:  # Iterar por cada número del DNI
        frecuencias[numero] = frecuencias.get(numero, 0) + 1  #
Incrementar la cuenta
    # Devolver un diccionario ordenado por clave (número) de menor
a mayor
    return dict(sorted(frecuencias.items()))

# Función para sumar los números de un DNI
def sumar_numeros(dni):
    total = 0  # Variable para acumular la suma
    for numero in dni:  # Iterar por cada número del DNI
        total += int(numero)  # Convertir a entero y sumar
    return total

# Función para verificar si un año es bisiesto
def es_bisiesto(anio):
```

```
    # Un año es bisiesto si es divisible por 4 y no por 100, o si
es divisible por 400
    if anio % 4 == 0:  # Si es divisible por 4
        if anio % 100 == 0:  # Si también es divisible por 100
            return anio % 400 == 0  # Debe ser divisible por 400
        return True  # Si no es divisible por 100, es bisiesto
    return False  # Si no es divisible por 4, no es bisiesto
```

Resultados con datos de ejemplo
Se probaron los DNIs 38945345 y 41355780, y los años de nacimiento 2000 y 2002, con nombres "Juan" y "Ana". Los resultados son:
- Conjunto de Juan: [3, 4, 5, 8, 9]
- Conjunto de Ana: [0, 1, 3, 4, 5, 7, 8]
- Unión: [0, 1, 3, 4, 5, 7, 8, 9]
- Intersección: [3, 4, 5, 8]
- Diferencia (Juan - Ana): [9]
- Diferencia (Ana - Juan): [0, 1, 7]
- Diferencia simétrica: [0, 1, 7, 9]
- Frecuencias de Juan: {'3': 2, '4': 2, '5': 2, '8': 1, '9': 1}
- Frecuencias de Ana: {'0': 1, '1': 1, '3': 1, '4': 1, '5': 2, '7': 1, '8': 1}
- Suma de números de Juan: 41
- Suma de números de Ana: 33
- Expresión 1: Se considera alta diversidad numérica. (Se cumple)
- Expresión 2: Hay un número común significativo. (Se cumple)
- Expresión 3: Se considera un grupo sin ceros. (Se cumple)
- Expresión 4: No hay mayor diversidad exclusiva. (No se cumple)
- Expresión 5: No se considera un grupo completo, faltan números. (No se cumple)
- Años pares: 2
- Años impares: 0
- No todos nacieron después del 2000.
- Año 2000 de Juan es bisiesto: ¡Tenemos un año especial!
- Año 2002 de Ana no es bisiesto.
- Producto cartesiano (años, edades): [(2000, 25), (2000, 23), (2002, 25), (2002, 23)]

Relación entre expresiones lógicas y código
- Expresión 1: Implementada con `if len(conjunto_1) >= 5 and len(conjunto_2) >= 5`. Verifica si ambos conjuntos tienen al menos 5 números.
- Expresión 2: Implementada con `if len(interseccion) > 3`. Evalúa si la intersección tiene más de 3 números.
- Expresión 3: Implementada con `if '0' not in conjunto_1 or '0' not in conjunto_2`. Comprueba si al menos un conjunto no contiene el número 0.
- Expresión 4: Implementada con `if len(diferencia_simetrica) > len(interseccion)`. Compara la cantidad de números en la diferencia simétrica y la intersección.
- Expresión 5: Implementada con `if len(union) == 10`. Verifica si la unión contiene todos los números del 0 al 9.

Parte 2B – Años de nacimiento

El programa permite ingresar dos nombres y años de nacimiento. Con los datos de ejemplo (Juan: 2000, Ana: 2002):
- Se cuentan años pares e impares usando un bucle que verifica si cada año es divisible por 2.
- Se verifica si ambos años son posteriores al 2000 usando un bucle que comprueba cada año.
- Se identifica si los años son bisiestos con la función `es_bisiesto` (en funciones.py), que evalúa las condiciones de divisibilidad.
- Se calcula el producto cartesiano de años y edades usando una lista por comprensión.

Responsabilidades de los integrantes

- **Gabriel:**
    - Cálculo de la unión y la intersección de los conjuntos.
    - Elaboración de los diagramas de Venn para unión e intersección (dibujados a mano e insertados como imágenes).
    - Definición y evaluación de las expresiones lógicas 1 ("Alta diversidad numérica") y 3 ("Grupo sin ceros").
    - Creación y prueba del código Python en Visual Studio Code, incluyendo el ingreso dinámico de nombres, DNIs y años de nacimiento, y la modularización en dos archivos.
- **Milagros:**
    - Cálculo de las diferencias (1-2, 2-1) y la diferencia simétrica.
    - Elaboración de los diagramas de Venn para diferencias y diferencia simétrica (dibujados a mano e insertados como imágenes).
    - Implementación y verificación del conteo de frecuencias (ordenadas) y suma de números en el código Python.
    - Definición y evaluación de las expresiones lógicas 2 ("Número común significativo") y 4 ("Mayor diversidad exclusiva").
- **Ambos:**
    - Definición y evaluación de la expresión lógica 5 ("Grupo completo").
    - Inserción de las imágenes de los diagramas de Venn en Google Docs.
    - Colaboración en la redacción del documento, la implementación de la Parte 2B y la preparación del programa Python.

Notas adicionales
- Los diagramas de Venn se encuentran insertados como imágenes en el documento de Google Docs, cumpliendo con los requisitos de la entrega.
- Los archivos IntegradorMatematica.py y funciones.py serán entregados junto con este documento en formato PDF.
- El programa es dinámico, permitiendo ingresar nombres, DNIs y años de nacimiento arbitrarios, con validaciones para garantizar datos válidos.
- Los conjuntos y frecuencias se muestran ordenados de menor a mayor para facilitar la lectura.

- La modularización en dos archivos mejora la organización del código, separando la lógica principal de las funciones auxiliares.