UTN
UNIVERSIDAD TECNOLÓGICA NACIONAL

Trabajo Integrador 2: Matemática y Programación

Materia: Matemática
Profesor/a: Vanina Durrutty
Alumnos/as: Chiappone Michael
Campana Jonatan
Fecha de entrega: 13/06/2025

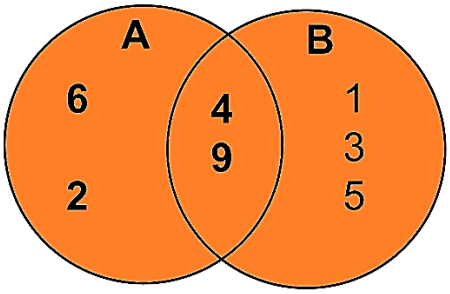
Contenido

Parte 1:

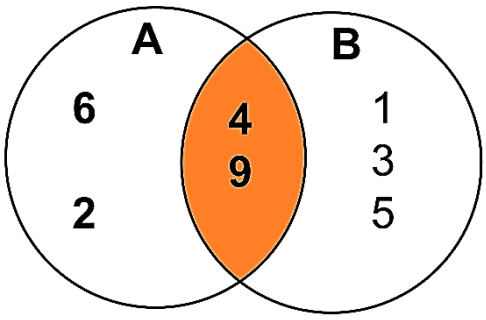
1. Dígitos únicos de los DNIs
Integrante A – DNI 42692226
→ Dígitos: 4, 2, 6, 9, 2, 2, 2, 6
→ Conjunto B = {2, 4, 6, 9}

Integrante B – DNI 39151194
→ Dígitos: 3, 9, 1, 5, 1, 1, 9, 4
→ Conjunto A = {1, 3, 4, 5, 9}

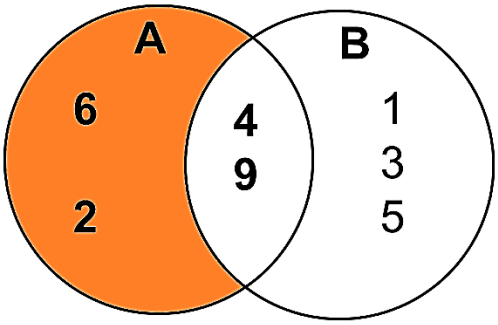
2. Operaciones entre conjuntos
Unión (A ∪ B):
Todos los elementos de A y B →
→ {1, 2, 3, 4, 5, 6, 9}



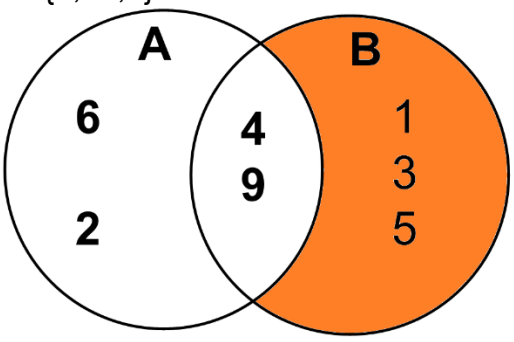
Intersección (A ∩ B):
Elementos en común entre A y B →
→ {4, 9}



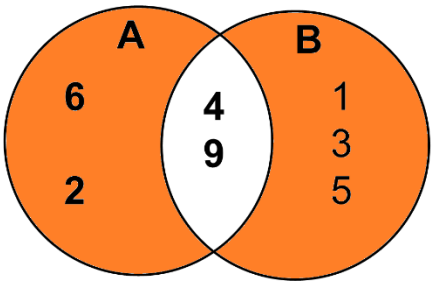
Diferencia (A - B):
Elementos en A que no están en B →
→ {6, 2}



Diferencia (B - A):
Elementos en B que no están en A →
→ {1, 3 ,5}



Diferencia simétrica (A Δ B):
Elementos que están en A o en B, pero no en ambos →
→ {1, 2, 3, 5, 6}



Expresión 1 (lenguaje natural):

“Los dígitos que están en A y no están en B”

Traducción a conjunto:

A – B

→ {1, 3, 5}

Resultado con tus conjuntos:

Conjunto A = {1, 3, 4, 5, 9}

Conjunto B = {2, 4, 6, 9}→ Resultado: **{1, 3, 5}**

Expresión 2 (lenguaje natural, complejidad similar):

“Los dígitos que están en A o en B, pero no en ambos”

Esta es la definición de **diferencia simétrica**.

Traducción a conjunto:

A Δ B

→ (A - B) ∪ (B - A)

→ {1, 3, 5, 2, 6}

Resultado con tus conjuntos:

→ **{1, 2, 3, 5, 6}**

Parte 2: Código en Python:

Parte A:

parte_2-A.py > ...

```python
def obtener_conjunto_digitos(dni):
    return set(dni)

def contar_frecuencias(dni):
    frecuencias = {}
    for digito in dni:
        if digito in frecuencias:
            frecuencias[digito] += 1
        else:
            frecuencias[digito] = 1
    return frecuencias

def suma_digitos(dni):
    return sum(int(d) for d in dni)

# Ingreso de DNIs (reales o ficticios)
dni1 = input("Ingrese el DNI del primer integrante: ")  # ejemplo: 39151194
dni2 = input("Ingrese el DNI del segundo integrante: ")  # ejemplo: 42692226

# Generación de conjuntos de dígitos únicos
conjunto_A = obtener_conjunto_digitos(dni1)
conjunto_B = obtener_conjunto_digitos(dni2)

print(f"\nConjunto A = {conjunto_A}")
print(f"Conjunto B = {conjunto_B}")

# Operaciones entre conjuntos
union = conjunto_A.union(conjunto_B)
interseccion = conjunto_A.intersection(conjunto_B)
diferencia_AB = conjunto_A.difference(conjunto_B)
diferencia_BA = conjunto_B.difference(conjunto_A)
diferencia_simetrica = conjunto_A.symmetric_difference(conjunto_B)
```

```python
print(f"\nUnión (A ∪ B) = {union}")
print(f"Intersección (A ∩ B) = {interseccion}")
print(f"Diferencia (A - B) = {diferencia_AB}")
print(f"Diferencia (B - A) = {diferencia_BA}")
print(f"Diferencia simétrica (A Δ B) = {diferencia_simetrica}")

# Frecuencia de cada dígito
frecuencias_A = contar_frecuencias(dni1)
frecuencias_B = contar_frecuencias(dni2)

print("\nFrecuencias en DNI 1:")
for digito, cantidad in frecuencias_A.items():
    print(f"Dígito {digito}: {cantidad} vez/veces")

print("\nFrecuencias en DNI 2:")
for digito, cantidad in frecuencias_B.items():
    print(f"Dígito {digito}: {cantidad} vez/veces")

# Suma total de dígitos
suma_A = suma_digitos(dni1)
suma_B = suma_digitos(dni2)

print(f"\nSuma de los dígitos del DNI 1: {suma_A}")
print(f"Suma de los dígitos del DNI 2: {suma_B}")

# Evaluación de condiciones lógicas

# Condición: Dígito compartido
digito_comun = conjunto_A.intersection(conjunto_B)
if digito_comun:
    print(f"\nDígito compartido entre todos los conjuntos: {digito_comun}")

# Condición: Diversidad numérica alta
if len(conjunto_A) > 6:
    print("Conjunto A tiene alta diversidad numérica.")
if len(conjunto_B) > 6:
    print("Conjunto B tiene alta diversidad numérica.")

# Condición: Resultado de expresiones lógicas del papel
print(f"\nDígitos en A y no en B: {diferencia_AB}")
print(f"Dígitos en A o en B, pero no en ambos: {diferencia_simetrica}")
```

Parte B:

parte_2-B.py > ...

```python
from datetime import datetime
from itertools import product

# Función para determinar si un año es bisiesto
def es_bisiesto(anio):
    return (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0)

# Ingreso de años de nacimiento (ejemplo ficticio si se repiten)
anios_nacimiento = []
cantidad_integrantes = int(input("Ingrese la cantidad de integrantes del grupo: "))

for i in range(cantidad_integrantes):
    anio = int(input(f"Ingrese el año de nacimiento del integrante {i + 1}: "))
    anios_nacimiento.append(anio)

# Contar pares e impares
pares = 0
impares = 0
for anio in anios_nacimiento:
    if anio % 2 == 0:
        pares += 1
    else:
        impares += 1

print(f"\nCantidad que nacieron en años pares: {pares}")
print(f"Cantidad que nacieron en años impares: {impares}")

# Verificar si todos nacieron después del 2000
if all(anio > 2000 for anio in anios_nacimiento):
    print("Grupo Z")

# Verificar si alguno nació en año bisiesto
if any(es_bisiesto(anio) for anio in anios_nacimiento):
    print("Tenemos un año especial")

# Calcular edades actuales
anio_actual = datetime.now().year
edades = [anio_actual - anio for anio in anios_nacimiento]

# Producto cartesiano entre años y edades
producto_cartesiano = list(product(anios_nacimiento, edades))

print("\nProducto cartesiano entre años de nacimiento y edades actuales:")
for par in producto_cartesiano:
    print(par)
```

Parte 3:

Enlace a video en YouTube:

https://youtu.be/yS02kUJAbhY

Conclusión:

Este trabajo integrador nos permitió aplicar de forma conjunta conceptos fundamentales de Matemática y Programación. A través del análisis de los DNIs y los años de nacimiento de los integrantes del grupo, logramos trabajar con conjuntos, expresiones lógicas, y estructuras condicionales y repetitivas en Python.

En la primera parte, exploramos operaciones de conjuntos como unión, intersección y diferencia, visualizando sus resultados y analizando condiciones lógicas que pudimos traducir a código. En la segunda parte, trabajamos con datos reales y desarrollamos funciones que nos permitieron automatizar el cálculo de edades, detectar años bisiestos y clasificar información relevante usando programación estructurada.

Este proyecto reforzó nuestra comprensión de cómo la lógica matemática puede implementarse eficientemente en un programa para resolver problemas concretos. Además, fortalecimos el trabajo en equipo, dividiendo tareas y aprendiendo de la integración entre teoría y práctica.