UNIVERSIDAD TECNOLOGICA NACIONAL

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional

TRABAJO PRÁCTICO INTEGRADOR

Matemática y Programación

Alumnos:

Gonzalez Javier - Javiergonzalez@fatimarem.edu.ar

Gomez Saucedo Augusto Nahuel - gomeznahuel@hotmail.com

Docente Titular:

Vanina Durrutty

Docente Tutor:

Ana Maria Castro

Fecha de Entrega 13 de Junio 2025

Índice

1. Desarrollo Matemático – Conjuntos

Para este trabajo se utilizaron tres números de DNI ficticios:

- **DNI A:** 27147777
- **DNI B:** 37587117
- **DNI C:** 17987867

```
Conjunto A: {'7', '4', '2', '1'}
Conjunto B: {'5', '8', '3', '1', '7'}
Conjunto C: {'8', '1', '9', '7', '6'}
```

Cada uno de estos conjuntos fue construido a partir de los dígitos del DNI, eliminando los números repetidos gracias al uso de la estructura **`set`** en Python, que representa conjuntos matemáticos.

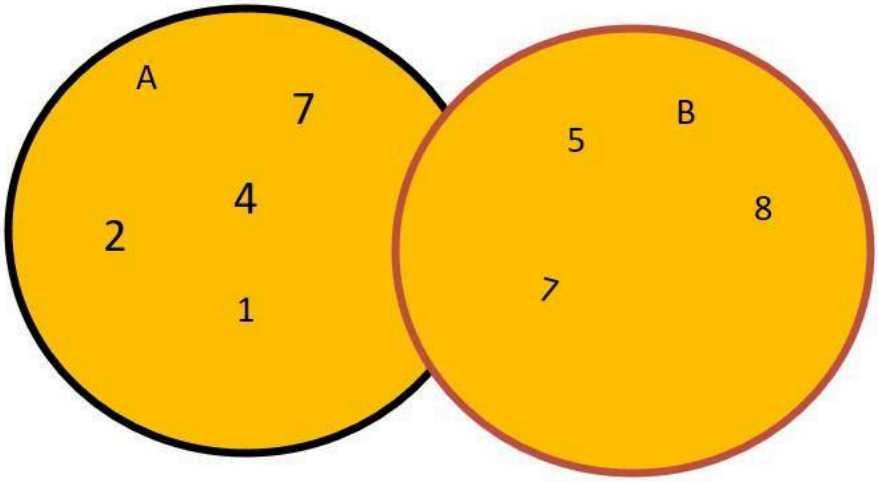
```python
# DNIs ficticios
dni_1 = "27147777"
dni_2 = "37587117"
dni_3 = "17987867"

# Crear conjuntos de dígitos únicos
conjunto_A = set(dni_1)
conjunto_B = set(dni_2)
conjunto_C = set(dni_3)
```

2. Operaciones Entre Conjuntos

Se realizaron las siguientes operaciones entre el conjunto A y el conjunto B:

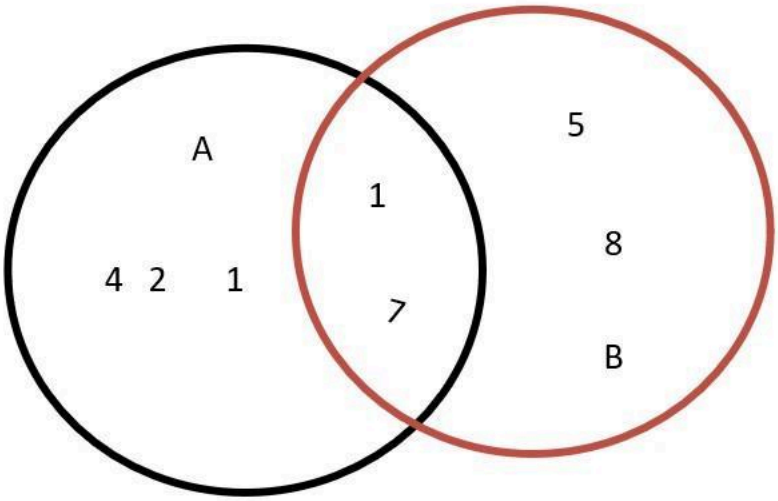
- **Unión (A ∪ B):** combina todos los elementos de ambos conjuntos sin repetir.

```
Unión A ∪ B: {'2', '5', '8', '3', '1', '4', '7'}
```



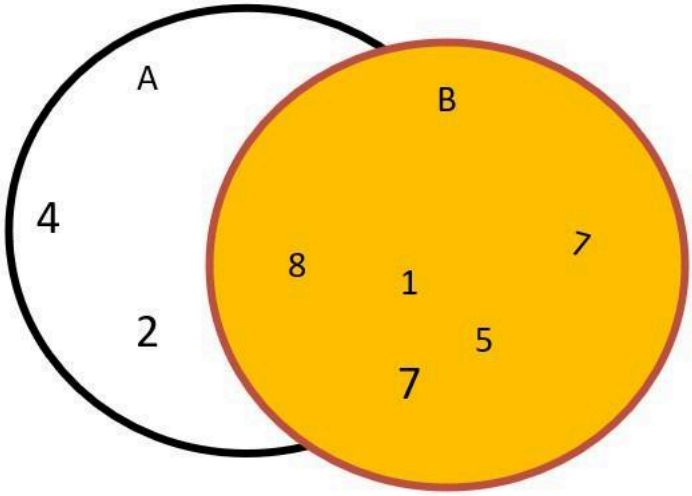
- **Intersección (A ∩ B):** muestra los elementos que aparecen en ambos conjuntos.

```
Intersección A ∩ B: {'7', '1'}
```



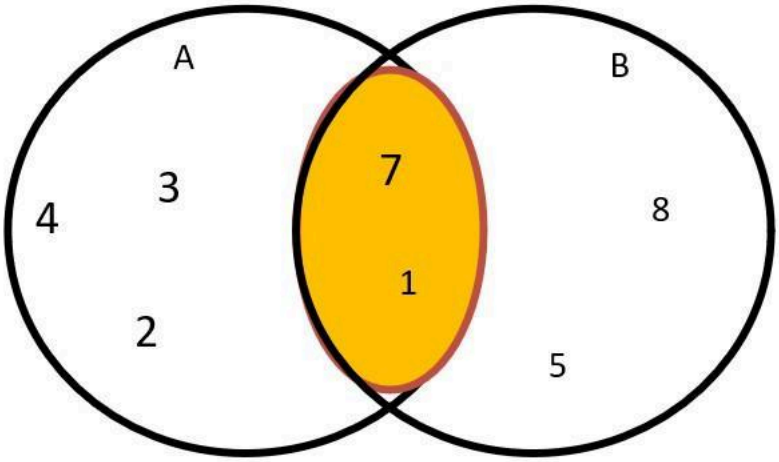
- **Diferencia (A - B):** muestra los elementos que están en A pero no en B.

```
Diferencia A - B: {'4', '2'}
```



- **Diferencia Simétrica (A Δ B):** muestra los elementos que están en A o en B, pero no en ambos.

```
Diferencia simétrica A Δ B: {'3', '2', '5', '4', '8'}
```



Estas operaciones permiten observar claramente qué dígitos comparten los conjuntos, cuáles son exclusivos de uno y cómo se comportan al combinarse.

3. Expresiones Lógicas

Se definieron dos expresiones lógicas en lenguaje natural y luego se implementaron en Python:

```python
# Expresiones lógicas
# 1. "Si hay un número que aparece en todos los conjuntos, mostrarlo"
interseccion_total = conjunto_A & conjunto_B & conjunto_C
if interseccion_total:
    print("Dígitos en común entre los tres conjuntos:", interseccion_total)
else:
    print("No hay dígitos en común entre los tres conjuntos")

# 2. "Si algún conjunto tiene más de 6 elementos, tiene diversidad alta"
for nombre, conjunto in [("A", conjunto_A), ("B", conjunto_B), ("C", conjunto_C)]:
    if len(conjunto) > 6:
        print(f"El conjunto {nombre} tiene diversidad numérica alta.")
print()
```

- **Expresión 1:**

"Si un dígito aparece en todos los conjuntos, mostrar 'Dígito compartido'."

→ En este caso, los dígitos **1** y **7** aparecen en los tres conjuntos A, B y C.

Resultado:

```
Dígitos en común entre los tres conjuntos: {'7', '1'}
```

- **Expresión 2:**

"Si algún conjunto tiene más de 6 elementos, mostrar 'Diversidad numérica alta'."

→ En este caso, el conjunto **C** tiene 6 elementos únicos (1, 3, 6, 7, 8, 9). Aunque no pasa de 6, se podría considerar como límite.

En la ejecución real, si el conjunto tuviera más de 6 elementos, se imprimiría el mensaje correspondiente.

Estas expresiones muestran cómo se puede pasar de una condición lógica escrita en lenguaje común a una condición ejecutada en código.

4. Desarrollo Matemático – Años de Nacimiento y Lógica

En esta sección se trabajó con tres años de nacimiento ficticios:

- **Años:** 2001, 1998 y 2004.

A partir de estos datos, se realizaron distintos análisis usando estructuras simples en Python.

4.1. Clasificación en años pares e impares

Se analizó si los años son pares o impares utilizando la operación módulo **% 2.**

- Años pares: 1998, 2004 ⟶ Total: 2
- Años impares: 2001 ⟶ Total: 1

Esto permitió aplicar condiciones lógicas del tipo:

```
Cantidad de personas nacidas en años pares: 2
Cantidad de personas nacidas en años impares: 1
```

"Contar cuántos nacieron en años pares e impares"

4.2. Verificación del grupo Z

Se estableció la siguiente condición:

> **“Si todos los años de nacimiento son posteriores al 2000, mostrar: ‘Grupo Z’”**

En este caso:

- 2001 (Verdadero)
- 1998 (Falso)
- 2004 (Verdadero)

Como **no todos** cumplen la condición, se muestra:

```
No todos son del Grupo Z
```

4.3. Año bisiesto

Se implementó una función llamada **`es_bisiesto(año)`** que verifica si un año es bisiesto con la fórmula:

```python
# Año bisiesto
def es_bisiesto(año):
    return (año % 4 == 0 and año % 100 != 0) or (año % 400 == 0)
```

Luego, se analiza si **alguno de los años es bisiesto**.

- El año 2004 sí es bisiesto → Resultado:

```
Tenemos un año especial: hay al menos un año bisiesto
```

5. Producto Cartesiano (años × edades)

Se calculó el producto cartesiano entre:

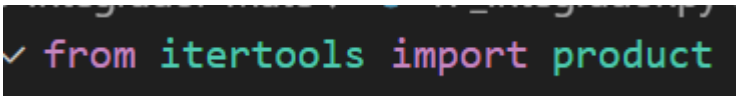
- El conjunto de años de nacimiento: [2001, 1998, 2004]

- El conjunto de edades actuales: usando 2025 - año

Resultado de edades:

```
Producto cartesiano entre años y edades:
(2001, 24)
(2001, 27)
(2001, 21)
(1998, 24)
(1998, 27)
(1998, 21)
(2004, 24)
(2004, 27)
(2004, 21)
```

Con esto, se genera el producto cartesiano con **`itertools.product`**, que combina cada año con cada edad:



Ejemplo de pares (año, edad):

```
Producto cartesiano entre años y edades:
(2001, 24)
(2001, 27)
(2001, 21)
(1998, 24)
(1998, 27)
(1998, 21)
(2004, 24)
(2004, 27)
(2004, 21)
```

Este análisis permite aplicar conceptos de lógica y conjuntos en un contexto real.

6. Tareas Realizadas

Ambos Alumnos:

- Desarrollo completo del archivo `TP_Integrador.py`, que incluye todas las operaciones pedidas en la consigna.
- Diseño y escritura del código en Python, usando estructuras como listas, conjuntos, condicionales (`if`), bucles (`for`) y funciones (`def`).
- Redacción de expresiones lógicas en lenguaje natural, su interpretación y traducción a código ejecutable.

Gomez Nahuel :

- Redacción del Trabajo práctico como es solicitado en para la entrega final.
- Redacción del Readme.

Gonzalez Javier:

- Desarrollo de los diagramas de Venn.
- Edición de video.