UNIVERSIDAD TECNOLOGICA NACIONAL

Trabajo Práctico Integrador Nº 2 - Matemática y Programación

Conectando Conjuntos, Lógica y Código: Desarrollo Colaborativo

Parte 2

Alumnos - Grupo N° 1

Alex Pedro Dauria, Silvia Giardini, Marina Giselle Cordero

Luciano de la Rubia, Laureano Escoca.

Tecnicatura Universitaria en Programación - Universidad Tecnológica Nacional.

Docente Titular

Matemática: Carina Jovanovich

Programación I: AUS Bruselario, Sebastián

Docente Tutor

Matemática: Sebastián Marinier

Programación I: Candia, Verónica

10 de Junio de 2025

Tabla de contenido

Introducción

El presente trabajo integrador representa una síntesis entre los fundamentos de matemática y programación, específicamente la teoría de conjuntos y la lógica proposicional, con las estructuras esenciales de Python (condicionales, repetitivas y funciones).

Nuestro principal objetivo es profundizar la integración entre estos contenidos, buscando desarrollar a través del código lo aprendido en matemática y poniendo en práctica lo visto en Programación I. Además, este proyecto está diseñado para fortalecer el trabajo en equipo, la comunicación clara y la responsabilidad individual en el contexto de un proyecto colaborativo.

La conformación de grupos para este trabajo no solo busca fomentar la colaboración entre pares, una habilidad fundamental para cualquier programador, sino también asegurar que cada integrante asuma responsabilidades específicas, contribuya activamente al proyecto y pueda explicar su aporte tanto en el video final como en la descripción escrita de sus tareas.

En el ámbito de la programación, el manejo de datos y estructuras repetitivas permite resolver problemas prácticos de manera eficiente. A continuación, se desarrollará un ejercicio práctico que consiste en procesar los años de nacimiento de un grupo de personas, determinar cuántos nacieron en años pares e impares, identificar si todos nacieron después del año 2000 para clasificarlos como "Grupo Z" y detectar si alguno nació en un año bisiesto para mostrar un mensaje especial. Este ejercicio será explicado y desarrollado en detalle en las secciones siguientes.

Desarrollo

Consigna 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.

Desarrollo del Programa en Python

El código está diseñado para procesar cinco DNIs de 8 dígitos, realizar operaciones con conjuntos, calcular la frecuencia de los dígitos en cada DNI, sumar los dígitos y evaluar expresiones lógicas.

A continuación, se desglosa el código en sus componentes principales, con explicaciones detalladas para cada sección:

1. **Recolección de los DNIs**

```python
dnis = []
for i in range(5):
    while True:
        dni = input(f"Ingrese el DNI {i+1} (8 dígitos): ")
        if len(dni) == 8 and dni.isdigit():
            dnis.append(dni)
            break
        else:
            print("El DNI debe tener 8 dígitos numéricos.")
```

Explicación:

- **Propósito**: Recolecta cinco DNIs válidos de 8 dígitos ingresados por el usuario y los almacena en una lista llamada dnis.
- **Proceso**:
  - Inicializa una lista vacía “dnis” para almacenar los DNIs.
  - Usa un bucle for que itera cinco veces (para los cinco DNIs).
  - Dentro de un bucle while True, solicita al usuario que ingrese un DNI para la iteración actual (i+1 para una numeración amigable).
  - Valida la entrada:
    - Verifica que el DNI tenga exactamente 8 dígitos usando len(dni) == 8.
    - Confirma que la entrada contenga solo dígitos numéricos con dni.isdigit().
  - Si la entrada es válida, agrega el DNI a la lista dnis y sale del bucle while con break para pasar al siguiente DNI.
  - Si la entrada no es válida, imprime un mensaje de error y el bucle while continúa solicitando una nueva entrada.
- **Manejo de errores**: Asegura que solo se acepten DNIs válidos de 8 dígitos numéricos, evitando entradas incorrectas o no numéricas.

2. Creación e impresión de Conjuntos de Dígitos Únicos

```
conjunto_A = set(dnis[0])
conjunto_B = set(dnis[1])
conjunto_C = set(dnis[2])
conjunto_D = set(dnis[3])
conjunto_E = set(dnis[4])
```

Explicación:

- **Propósito**: Convierte cada DNI en un conjunto de sus dígitos únicos.
- **Proceso**:
    - Cada DNI (una cadena de 8 dígitos) se convierte en un set para extraer los dígitos únicos, eliminando duplicados.
    - Los conjuntos se nombran conjunto_A, conjunto_B, conjunto_C, conjunto_D y conjunto_E, correspondientes al primer, segundo, tercer, cuarto y quinto DNI, respectivamente.
    - Por ejemplo, si dnis[0] = "12345678", entonces conjunto_A = {'1', '2', '3', '4', '5', '6', '7', '8'}. Si un DNI tiene dígitos repetidos, como "11223344", el conjunto resultante sería {'1', '2', '3', '4'}.
- **Salida:**

```
print(f"\nDígitos únicos del DNI 1 (Conjunto A): {sorted(conjunto_A)}")
print(f"Dígitos únicos del DNI 2 (Conjunto B): {sorted(conjunto_B)}")
print(f"Dígitos únicos del DNI 3 (Conjunto C): {sorted(conjunto_C)}")
print(f"Dígitos únicos del DNI 4 (Conjunto D): {sorted(conjunto_D)}")
print(f"Dígitos únicos del DNI 5 (Conjunto E): {sorted(conjunto_E)}")
```

Se muestran los conjuntos de dígitos únicos para cada DNI.

Proceso:

- Imprime cada conjunto (conjunto_A a conjunto_E) con un formato claro, indicando a qué DNI corresponde.
- Usa la función sorted() para mostrar los dígitos en orden ascendente, mejorando la legibilidad.

3. Cálculo y Visualización de la Unión de Todos los Conjuntos

```
# Calcular la unión de todos los conjuntos
union_todos = conjunto_A | conjunto_B | conjunto_C | conjunto_D | conjunto_E
print(f"\nUnión de todos los conjuntos: {sorted(union_todos)}")
```

- **Propósito**: Calcula la unión de los cinco conjuntos de dígitos únicos.
- **Proceso**:
    - Se usa el operador | para combinar todos los dígitos únicos de conjunto_A, conjunto_B, conjunto_C, conjunto_D y conjunto_E en un solo conjunto, union_todos.
    - La unión incluye todos los dígitos que aparecen al menos en uno de los conjuntos, sin duplicados, ya que el operador | elimina automáticamente cualquier repetición.
    - Muestra el resultado ordenado con sorted(union_todos) para una presentación clara y consistente.

4. Cálculo y Visualización de Intersecciones

```
# Calcular intersecciones usando el operador & (AND)
interseccion_A_B = conjunto_A & conjunto_B
interseccion_C_D = conjunto_C & conjunto_D
interseccion_D_E = conjunto_D & conjunto_E
interseccion_todos = conjunto_A & conjunto_B & conjunto_C & conjunto_D & conjunto_E

# Mostrar las intersecciones, ordenadas para mejor legibilidad
print(f"\nIntersección de Conjunto A y Conjunto B: {sorted(interseccion_A_B)}")
print(f"Intersección de Conjunto C y Conjunto D: {sorted(interseccion_C_D)}")
print(f"Intersección de Conjunto D y Conjunto E: {sorted(interseccion_D_E)}")
print(f"Intersección de todos los conjuntos: {sorted(interseccion_todos)}")
```

Explicación

- **Propósito**: Calcula las intersecciones entre los conjuntos de dígitos únicos de los DNIs.

- **Proceso**:
    - Se usa el operador & para combinar los conjuntos conjunto_A, conjunto_B, conjunto_C, conjunto_D y conjunto_E en diferentes combinaciones, generando nuevos conjuntos que contienen solo los dígitos comunes a los conjuntos involucrados.
    - La intersección con "&" incluye únicamente los dígitos que aparecen en todos los conjuntos especificados en la operación, eliminando cualquier elemento no común.
    - Muestra los resultados ordenados con sorted() para una presentación clara y consistente.

5. Cálculo y Visualización de Diferencias

```
# Calcular diferencias usando el operador - (resta de conjuntos)      You, 30 minu
diferencia_A_B = conjunto_A - conjunto_B
diferencia_C_D = conjunto_C - conjunto_D
diferencia_D_E = conjunto_D - conjunto_E

# Mostrar diferencias, ordenadas para mejor legibilidad
print(f"\nDiferencia de Conjunto A menos Conjunto B: {sorted(diferencia_A_B)}")
print(f"Diferencia de Conjunto C menos Conjunto D: {sorted(diferencia_C_D)}")
print(f"Diferencia de Conjunto D menos Conjunto E: {sorted(diferencia_D_E)}")
```

Explicación

- **Propósito**: Calcula las diferencias entre pares de conjuntos de dígitos únicos de los DNIs.
- **Proceso**:
    - Se usa el operador - para obtener los elementos que están en un conjunto pero no en otro. Por ejemplo:

        conjunto_A - conjunto_B devuelve los dígitos que están en conjunto_A pero no en conjunto_B.
    - La diferencia elimina cualquier elemento que esté presente en el segundo conjunto, dejando solo los elementos exclusivos del primer conjunto.
    - Muestra los resultados ordenados con sorted() para una presentación clara y consistente.

6. Cálculo y Visualización de Diferencias Simétricas

```python
# Calcular diferencias simétricas usando unión e intersección
diferencia_simetrica_A_B = (conjunto_A | conjunto_B) - (conjunto_A & conjunto_B)
diferencia_simetrica_C_D = (conjunto_C | conjunto_D) - (conjunto_C & conjunto_D)
diferencia_simetrica_D_E = (conjunto_D | conjunto_E) - (conjunto_D & conjunto_E)

# Mostrar diferencias simétricas, ordenadas para mejor legibilidad
print(f"\nDiferencia simétrica de Conjunto A y Conjunto B: {sorted(diferencia_simetrica_A_B)}")
print(f"Diferencia simétrica de Conjunto C y Conjunto D: {sorted(diferencia_simetrica_C_D)}")
print(f"Diferencia simétrica de Conjunto D y Conjunto E: {sorted(diferencia_simetrica_D_E)}")
```

Explicación

- **Propósito**: Calcula las diferencias simétricas entre pares de conjuntos de dígitos únicos de los DNIs utilizando operadores lógicos de conjuntos.
- **Proceso**:
  - Para cada par de conjuntos (por ejemplo, conjunto_A y conjunto_B):
    - Calcula la unión (conjunto_A | conjunto_B) para obtener todos los elementos presentes en al menos uno de los conjuntos.
    - Calcula la intersección (conjunto_A & conjunto_B) para obtener los elementos comunes.
    - Resta la intersección de la unión ((conjunto_A | conjunto_B) - (conjunto_A & conjunto_B)) para obtener los elementos que están en uno u otro conjunto, pero no en ambos.
  - Muestra los resultados ordenados con sorted () para una presentación clara.

7. Cálculo de la Diferencia Simétrica de Todos los Conjuntos

```python
# Calcular la diferencia simétrica de todos los conjuntos
todos_conjuntos = [conjunto_A, conjunto_B, conjunto_C, conjunto_D, conjunto_E]
diferencia_simetrica_todos = set()
for elemento in union_todos:
    # Contar cuántas veces aparece el elemento en todos los conjuntos
    conteo = sum(1 for conjunto in todos_conjuntos if elemento in conjunto)
    # Incluir el elemento si aparece un número impar de veces
    if conteo % 2 == 1:
        diferencia_simetrica_todos.add(elemento)
```

Explicación:

- **Propósito**: Calcula la diferencia simétrica de los cinco conjuntos, que incluye los dígitos que aparecen un número impar de veces en los conjuntos.
- **Proceso**:
  - Crea una lista todos_conjuntos con los cinco conjuntos.
  - Inicializa un conjunto vacío diferencia_simetrica_todos.
  - Itera sobre cada dígito en union_todos (la unión de todos los conjuntos).
  - Para cada dígito, cuenta cuántas veces aparece en los conjuntos usando una suma de comprensión de lista.
  - Si el conteo es impar (conteo % 2 == 1), agrega el dígito a diferencia_simetrica_todos.
  - Muestra el resultado ordenado con sorted().
- **Ejemplo**: Si un dígito como '1' aparece en conjunto_A y conjunto_C (2 veces, par), no se incluye. Si aparece en conjunto_B solo (1 vez, impar), se incluye.

8. Conteo de Frecuencia de Dígitos

```python
# Conteo de frecuencia de cada dígito en cada DNI
print("\nConteo de frecuencia de cada dígito en cada DNI:")
for i, dni in enumerate(dnis, 1):
    print(f"\nDNI {i}: {dni}")
    # Inicializar un diccionario para contar la frecuencia de cada dígito (0-9)
    frecuencia = {str(d): 0 for d in range(10)}
    # Contar la frecuencia de cada dígito en el DNI
    for digito in dni:
        frecuencia[digito] += 1
    # Mostrar solo los dígitos que aparecen al menos una vez
    print("Frecuencia de dígitos:")
    for digito, conteo in frecuencia.items():
        if conteo > 0:
            print(f"Dígito {digito}: {conteo} {'vez' if conteo == 1 else 'veces'}")
```

Explicación:

- **Propósito**: Calcula y muestra la frecuencia de cada dígito (0-9) en cada DNI.
- **Proceso**:
  - Itera sobre cada DNI en la lista dnis usando enumerate para numerar los DNIs (1 a 5).
  - Para cada DNI:
    - Inicializa un diccionario “frecuencia” con claves del '0' al '9' y valores iniciales de 0.
    - Itera sobre cada dígito del DNI y actualiza el conteo en frecuencia.
    - Imprime sólo los dígitos con frecuencia mayor a 0, usando una expresión condicional para elegir entre "vez" (singular) o "veces" (plural).

9. Suma de Dígitos y Suma Total

```
sumas_por_dni = []
total_digitos = 0

for dni in dnis:
    # Convertir cada dígito a entero y sumar
    suma_digitos = sum(int(digito) for digito in dni)  # Suma
    sumas_por_dni.append((dni, suma_digitos))
    total_digitos += suma_digitos

# Mostrar resultados
print("\nDNIs ingresados y suma de sus dígitos:")
for dni, suma in sumas_por_dni:
    print(f"DNI: {dni}, Suma de dígitos: {suma}")
print(f"Suma total de todos los dígitos: {total_digitos}")
```

Explicación:

- **Propósito**: Calcula la suma de los dígitos de cada DNI y la suma total de todos los dígitos.
- **Proceso**:
    - Inicializa una lista vacía sumas_por_dni para almacenar tuplas (DNI, suma) y una variable total_digitos en 0.
    - Para cada DNI:
        - Calcula la suma de sus dígitos convirtiendo cada dígito a entero (int(digito)) y usando sum().
        - Agrega una tupla (dni, suma_digitos) a sumas_por_dni.
        - Suma: suma_digitos a total_digitos.
    - Imprime cada DNI con su suma de dígitos y la suma total de todos los dígitos.

10. Evaluación de Expresiones Lógicas

```
# Expresión lógica 1: Dígitos en A o B, pero no en C
expresion_1 = (conjunto_A | conjunto_B) - conjunto_C
print(f"\nExpresión lógica 1: Dígitos en A o B, pero no en C: {sorted(expresion_1)}")

# Expresión lógica 2: Dígitos en D, pero no en C ∩ E
expresion_2 = conjunto_D - (conjunto_C & conjunto_E)
print(f"Expresión lógica 2: Dígitos en D, pero no en C ∩ E: {sorted(expresion_2)}")
```

Explicación de cada expresión

Expresión 1:

- **Propósito**: Calcula los dígitos que están en conjunto_A o en conjunto_B (o en ambos), pero no en conjunto_C.
- **Proceso**:
  - Usa el operador | para calcular la unión de conjunto_A y conjunto_B, que incluye todos los dígitos presentes en al menos uno de estos conjuntos.
  - Usa el operador - para restar conjunto_C de la unión, eliminando los dígitos que están en conjunto_C.
  - Muestra el resultado ordenado con sorted () para una presentación clara.

Expresión 2:

Propósito: Calcula los dígitos que están en conjunto_D, pero no en la intersección de conjunto_C y conjunto_E.

- **Proceso**:
  - Usa el operador “&” para calcular la intersección de conjunto_C y conjunto_E, que incluye los dígitos comunes a ambos conjuntos.
  - Usa el operador - para restar la intersección de conjunto_D, eliminando los dígitos que están en ambos conjunto_C y conjunto_E.
  - Muestra el resultado ordenado con sorted () para una presentación clara.

Consigna B:

Desarrolle un programa que cumpla con los siguientes requisitos:

- Que permita el ingreso de los años de nacimiento de los integrantes del grupo. Si dos o más integrantes comparten el mismo año de nacimiento, se debe ingresar un dato ficticio para diferenciarlos, según corresponda.
- Utilice estructuras repetitivas para contar cuántos integrantes nacieron en años pares y cuántos en años impares.
- Determine si todos los integrantes nacieron después del año 2000. En caso afirmativo, muestre el mensaje "Grupo Z".
- Identifique si alguno de los años de nacimiento corresponde a un año bisiesto y, de ser así, muestre el mensaje "Tenemos un año especial".
- Implemente una función que determine si un año es bisiesto (un año es bisiesto si es divisible por 4, pero no por 100, salvo que también sea divisible por 400).
- Calcule el producto cartesiano entre el conjunto de años de nacimiento y el conjunto de edades actuales de los integrantes (calculadas restando cada año de nacimiento al año actual, 2025).

Desarrollo del Programa en Python

Estructura del programa desarrollado:

El programa consta de seis funciones y un programa principal:

1. **def ingreso_anio_nacimiento(): Recolecta los años de nacimiento únicos ingresados por el usuario.**

```python
def ingreso_anio_nacimiento():
    # Función para ingresar los años de nacimiento de los integrantes
    anios = []
    miembros = int(input("Ingrese la cantidad de integrantes que hay en el grupo: "))

    for i in range(miembros):
        while True:
            try:
                # Ingreso de año de nacimiento por el usuario
                anio = int(input(f"Ingresá el año de nacimiento del integrante {i + 1}: "))

                #Verificar si el año se repite
                if anio in anios:
                    print("Ese año ya fue ingresado. Ingresá uno distinto")
                    continue

                # Verificar que el año ingresado este en un rango lógico
                if anio < 1900 or anio > 2025:
                    print("Ingrese un año entre 1900 y 2025.")
                    continue # Vuelve al principio del while

                anios.append(anio)      Laureano, 2 days ago • Version 2 del punto B Operaciones
                break # Si todo está bien, sale del while
            except ValueError:
                print("Ingrese un año válido")
    return anios
```

Solicita al usuario la cantidad de integrantes del grupo y sus años de nacimiento, asegurando que los años sean válidos (entre 1900 y 2025) y únicos (sin repeticiones).

Esta función se encarga de obtener los años de nacimiento de los integrantes del grupo por parte del usuario.

Explicación:

- Se inicialista una lista vacía anios donde se almacenarán los años de nacimiento válidos.
- Se pide al usuario que ingrese la cantidad de integrantes en el grupo y lo almacena como un número entero en la variable miembros.
- Continúa un ciclo for que se repite la misma cantidad de integrantes almacenados.
- Un bucle While interno que continúa indefinidamente hasta que se introduce un año válido para el integrante actual.
- Este bloque maneja posibles errores.
    - Pide al usuario que ingrese el año de nacimiento del primer integrante e intenta convertir la entrada a un número entero.
    - Comprueba si el año ingresado ya existe en la lista anios. Si es así, imprime un mensaje pidiendo que se ingrese un número que no sea repetido y vuelve a pedir que ingrese nuevamente un año.
    - Valida si el año ingresado está dentro de un rango lógico (entre 1900 y 2025). Si no lo está, imprime un mensaje de error y continúe, solicitando un año válido.
    - Si el año es válido y no está duplicado, se añade a la lista anios.
    - Una vez que un año válido se añade con éxito, esta instrucción sale del bucle while para pasar al siguiente integrante.
    - Si el usuario introduce algo que no se puede convertir a un número entero (por ejemplo, texto), se produce un ValueError. Este bloque “except” lo captura e imprime "Ingrese un año válido", luego el bucle while continúa pidiendo la entrada de nuevo.

- Después de obtener todos los años de nacimiento, la función devuelve la lista anios.

2. **def cont_pares_impares(anios): Cuenta cuántos años de nacimiento son pares e impares.**

```python
def cont_pares_impares(anios):
    # Cuenta cuántos años son pares y cuántos impares en la lista recibida
    pares = 0
    impares = 0
    for anio in anios:
        if anio % 2 == 0:
            pares += 1
        else:
            impares += 1
    return pares, impares
```

Explicación:

- Inicializa contadores para años pares e impares.
- Con un ciclo “for” Itera a través de cada anio en la lista anios.
- Comprueba si el año es divisible por 2 (resto %)(lo que significa que es un año par).
    - Si es par, incrementa el contador pares.
    - Si no es par, es un año impar.
- Incrementa el contador impares.
- Devuelve tanto el recuento de años pares como el de años impares.

3. **def verificar_grupo_z(anios): Verifica si todos los miembros nacieron después del 2000 (Generación Z).**

```python
def verificar_grupo_z(anios):
    # Verifica si todos los integrantes nacieron después del año 2000
    for anio in anios:
        if anio <= 2000:
            return False # No es Grupo Z
    return True # Todos nacieron despues del 2000 -> Grupo Z
```

Explicación:

- Con un ciclo "for" Itera a través de cada anio en la lista anios.
- Si se encuentra algún año que sea 2000 o anterior, significa que no todos los integrantes nacieron después del 2000.
- Devuelve inmediatamente False porque no se cumple la condición para el "Grupo Z".
- Si el bucle se completa sin encontrar ningún año menor o igual a 2000, significa que todos los integrantes nacieron después del 2000, por lo que devuelve True.

4. **def es_bisiesto(anio): Determina si un año es bisiesto.**

```python
def es_bisiesto(anio):
    #Es bisiesto si es divisible por 4 y NO es divisible por 100 o es divisible por 400
    if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):
        return True
    else:
        return False
```

Explicación:

Se usa la regla estándar para determinar un año bisiesto**: if (anio % 4 == 0 and anio % 100 != 0) or (anio % 400 == 0):**

  - Un año es bisiesto si es divisible por 4 Y no es divisible por 100.
  - O, si es divisible por 400.

- Si la condición se cumple, el año es bisiesto.

- De lo contrario, no es un año bisiesto.

5. **def verificar_bisiesto(anios): Busca años bisiestos en la lista e imprime un mensaje si encuentra alguno.**

```
def verificar_bisiesto(anios):
    # Recorre los años para verificar si alguno es bisiesto
    # Imprime un mensaje si encuentra al menos uno
    for anio in anios:
        if es_bisiesto(anio):
            print("\nTenemos un año especial")
            break
```

- Con un ciclo “for” Itera a través de cada anio en la lista anios.
- Llama a la función def es_bisiesto() para comprobar si el año actual es un año bisiesto.
- Si se encuentra un año bisiesto, imprime este mensaje: **Tenemos un año especial**
- Una vez que se encuentra un año bisiesto y se imprime el mensaje, no es necesario comprobar los años restantes, por lo que el bucle se termina con break.

6. **def calcular_edades(anios, anio_actual=2025):** Calcula las edades restando los años de nacimiento del año actual.

```
def calcular_edades(anios, anio_actual = 2025):
    # Verificamos las edades a partir de los años ingresados y el año actual
    edades = [anio_actual - anio for anio in anios]
    return edades
```

Explicación:

- Establece un valor predeterminado para el año actual en 2025.
- Crea una nueva lista llamada edades restando cada anio de la lista anios a anio_actual.
- Devuelve la lista de edades calculadas.

7. **def producto_cartesiano(anios, edades): Calcula el producto cartesiano entre los años de nacimiento y las edades.**

```python
def producto_cartesiano(anios, edades):
    # Calcula el producto cartesiano entre la lista de años y la lista de edades
    # Devuelve una lista de tuplas (año, edad) con todas las combinaciones posibles
    producto_car = [(anio, edad) for anio in anios for edad in edades]
    return producto_car
```

Explicación:

- Hace una comprensión de lista anidada, genera todos los pares posibles (tuplas) donde el primer elemento es un año de nacimiento de anios y el segundo elemento es una edad de edades.
- Devuelve la lista de estas tuplas (año, edad).

8. **main(): Coordina el flujo del programa, llamando a las funciones anteriores y mostrando los resultados.**

```python
# Programa Principal
def main():
    print("\nOperaciones con años de nacimiento")
        Laureano, 2 days ago • Version 2 del punto B Operaciones con Años de n…
    #Ingreso de los años de nacimiento
    anios_nacimiento = ingreso_anio_nacimiento()
    print(f"\nAños ingresados: {anios_nacimiento}")

    # Contador de años pares impares
    pares, impares = cont_pares_impares(anios_nacimiento)
    print(f"\nCantidad de miembros que nacieron en años pares: {pares}")
    print(f"Cantidad de miembros que nacieron en años impares: {impares}")

    # Verificación Grupo Z
    if verificar_grupo_z(anios_nacimiento):
        print("\nGrupo Z")

    # Imprimir años bisiestos
    verificar_bisiesto(anios_nacimiento)

    # Calculo de edades
    edades = calcular_edades(anios_nacimiento)
    print(f"\nEdades actuales en 2025: {edades}")

    # Calculo de producto cartesiano entre años y edades
    product_car = producto_cartesiano(anios_nacimiento, edades)
    print("\nProducto cartesiano de edades y años de nacimiento")
    for anio, edad in product_car:
        print(f"({anio}, {edad})")
```

Esta es la función principal que da orden a la ejecución del programa.

- Imprime un título para el programa.
- Llama a la función ingreso_anio_nacimiento() para obtener los años de nacimiento y almacena el resultado en anios_nacimiento.
- Imprime la lista de años de nacimiento ingresados.
- Llama a cont_pares_impares() para obtener el recuento de años de nacimiento pares e impares.
- Imprime los recuentos de integrantes nacidos en años pares e impares.
- Llama a verificar_grupo_z() para comprobar si el grupo es "Grupo Z".
- Si es un "Grupo Z", imprime este mensaje.
- Llama a verificar_bisiesto() para comprobar e imprimir un mensaje sobre años bisiestos.
- Llama a calcular_edades() para obtener las edades actuales de los integrantes.
- Imprime la lista de edades calculadas.
- Llama a producto_cartesiano() para obtener el producto cartesiano.
- Imprime un encabezado para el producto cartesiano.
- Con un ciclo for Itera a través de la lista product_car e imprime cada tupla (año, edad).
- El programa se ejecuta llamando a main() cuando se corre directamente (a través de if __name__ == "__main__":). Esta es una construcción común de Python, asegura que la función main() se llame solo cuando el script se ejecuta directamente (no cuando se importa como un módulo en otro script). Es una buena práctica y es altamente recomendable utilizarlo

Video de Presentación

Link del video: https://youtu.be/ml6DOmdR-Bg
Repositorio en GitHub: https://github.com/Alex-Dauria/Trabajo-Integrador2

Contribuciones Individuales

Silvia Giardini

- Propuso las expresiones lógicas que fueron tomadas como base para la implementación en Python.

Alex Pedro Dauria

- Propuso y programó la segunda versión del código en Python para representar expresiones lógicas.
- Verificó su funcionamiento con los conjuntos derivados de los DNIs.
- Creó y gestionó el repositorio en GitHub para la organización del trabajo.

Marina Giselle Cordero

- Inició el desarrollo del código Python para trabajar con DNIs.
- Validó el programa final y realizó pruebas con distintos casos.
- Participó activamente en la redacción del PDF del Desarrollo del Programa (ambos ejercicios).

Luciano de la Rubia

- Implementó las funciones para calcular edades a partir de los años de nacimiento.
- Programó la verificación de años bisiestos.
- Calculó el producto cartesiano entre años y edades.
- Integró esta lógica al programa principal.

Laureano Escoca

- Trabajó junto a Luciano en la lógica relacionada a los años.
- Desarrolló estructuras condicionales y repetitivas.
- Conectó esta parte con el resto del programa.
- Fue el encargado de la edición y montaje del video final del proyecto.

Conclusión

Este trabajo integrador ha permitido consolidar los conocimientos de matemática y programación, uniendo la teoría de conjuntos y la lógica proposicional con las herramientas prácticas de Python, como estructuras condicionales, repetitivas y funciones. A través del desarrollo del ejercicio propuesto, se lograron aplicar estos conceptos para resolver problemas concretos, como el procesamiento de años de nacimiento, la clasificación de grupos y la identificación de años bisiestos, demostrando la relevancia de integrar ambas disciplinas para abordar desafíos de manera eficiente y estructurada.

Además, el proyecto fortaleció habilidades esenciales más allá de lo técnico, como el trabajo en equipo, la comunicación clara y la responsabilidad individual. La colaboración entre los integrantes no solo enriqueció el proceso de desarrollo, sino que también destacó la importancia de la distribución equitativa de tareas y la capacidad de cada miembro para explicar su aporte. Este enfoque colaborativo, combinado con la aplicación práctica de los conocimientos adquiridos, refleja el potencial de la programación como una herramienta para resolver problemas reales y fomenta el desarrollo de competencias clave para futuros desafíos académicos y profesionales.