**Trabajo Práctico Integrador Nº 2 -Parte B-**

**Matemática y Programación**

**Conectando Conjuntos, Lógica y Código: Desarrollo Colaborativo**

**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**

Matemáticas: Carina Jovanovich

Programación I: AUS Bruselario, Sebastián

**Docente Tutor**

Matemática: Sebastián Marinier

Programación I: Candia, Verónica

XX de [mes] de 2025

**Tabla de contenido**

**Introducción** 3

**Desarrollo** 4

Consigna A 4 **Desarrollo del Programa en Python** 5

Consigna B 14 **Desarrollo del Programa en Python** 15

**Video de Presentación** 22

**Conclusión** 23

**Referencias** 24

**Tareas realizadas por cada Integrante** 25

**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**

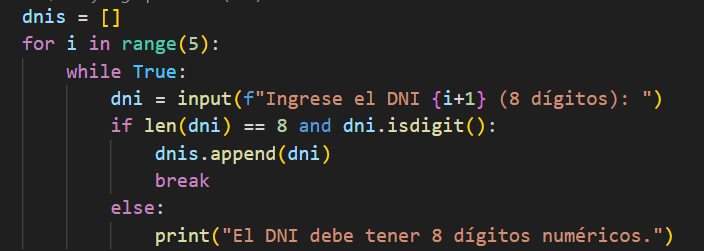
* Ingreso de los DNIs (reales o ficticios).
* Generación automática de los conjuntos de dígitos únicos.
* Cálculo y visualización de: unión, intersección, diferencias y diferencia simétrica.
* Conteo de frecuencia de cada dígito en cada DNI utilizando estructuras repetitivas.
* Suma total de los dígitos 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:

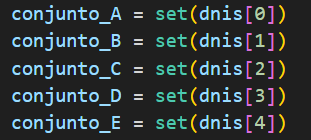
1. **Recolección de los DNIs**

****

**Explicación:**

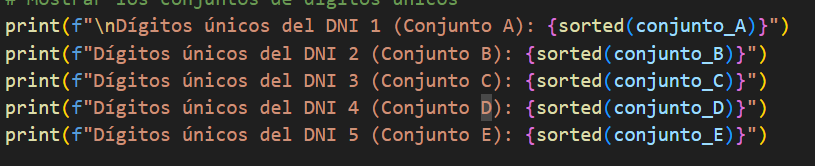
* **Propósito**: Recolecta cinco DNIs válidos de 8 dígitos ingresados por el usuario y los almacena en una lista llamada dnis.
* **Proceso**:
  + Inicializa una lista vacía dnis para almacenar los DNIs.
  + Usa un bucle for que itera cinco veces (para los cinco DNIs).
  + Dentro de un bucle while True, 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.

1. **Creación e impresión de Conjuntos de Dígitos Únicos**

****

**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**

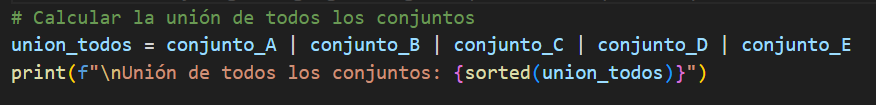
****

Se muestra los conjuntos de dígitos únicos para cada DNI.

**Proceso**:

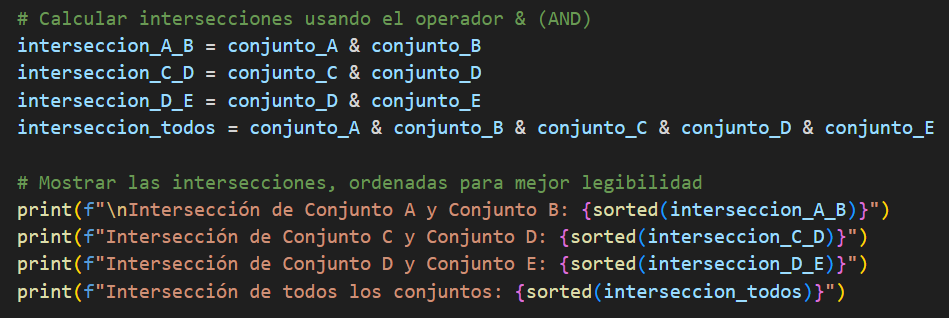
* Imprime cada conjunto (conjunto\_A a conjunto\_E) con un formato claro, indicando a qué DNI corresponde.
* Usa la función sorted() para mostrar los dígitos en orden ascendente, mejorando la legibilidad.

1. **Cálculo y Visualización de la Unión de Todos los Conjuntos**

****

* **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.

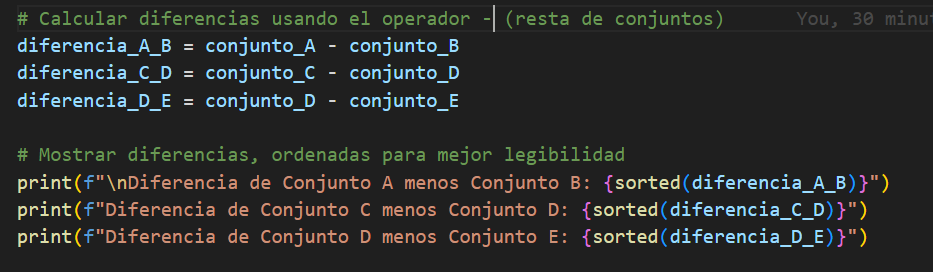
1. **Cálculo y Visualización de Intersecciones**

****

**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.

1. **Cálculo y Visualización de Diferencias**

****

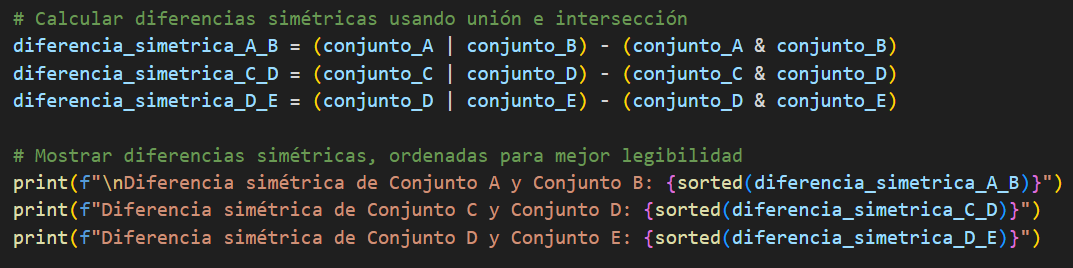
**Explicación**

* **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.

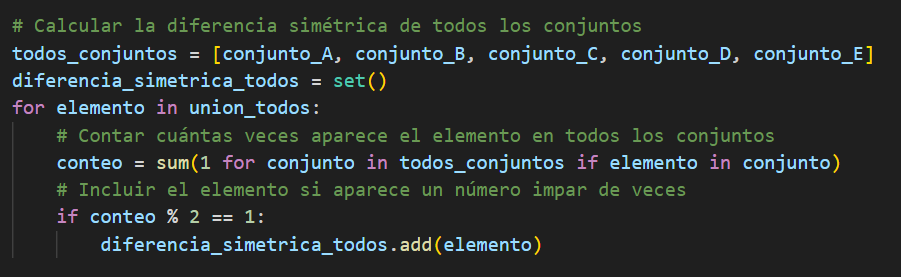
1. **Cálculo y Visualización de Diferencias Simétricas**

****

**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.

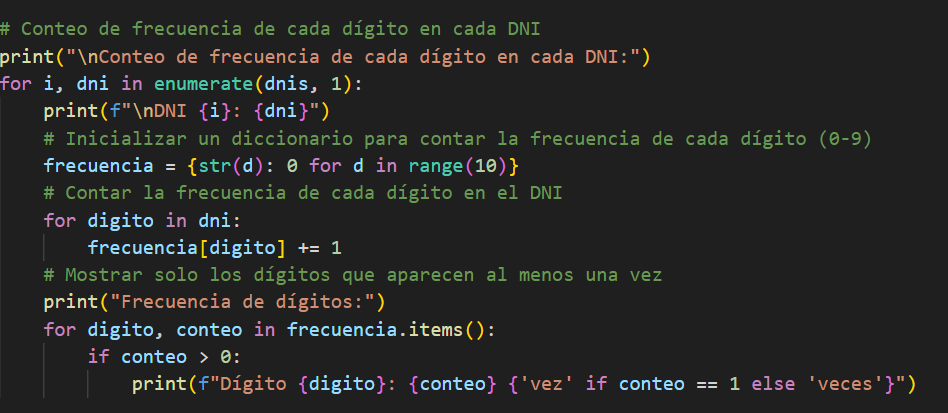
1. **Cálculo de la Diferencia Simétrica de Todos los Conjuntos**

****

**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.

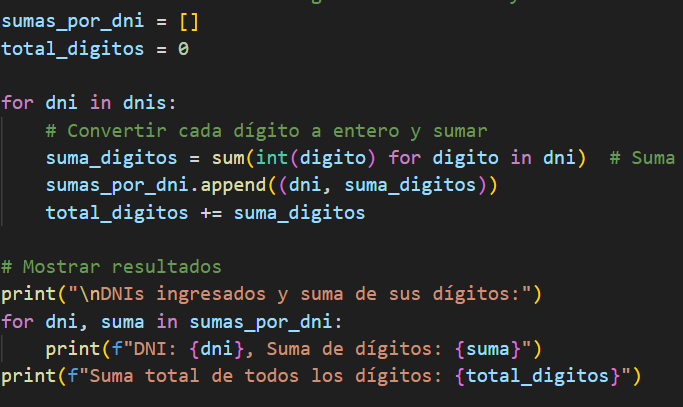
1. **Conteo de Frecuencia de Dígitos**

****

**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 solo los dígitos con frecuencia mayor a 0, usando una expresión condicional para elegir entre "vez" (singular) o "veces" (plural).

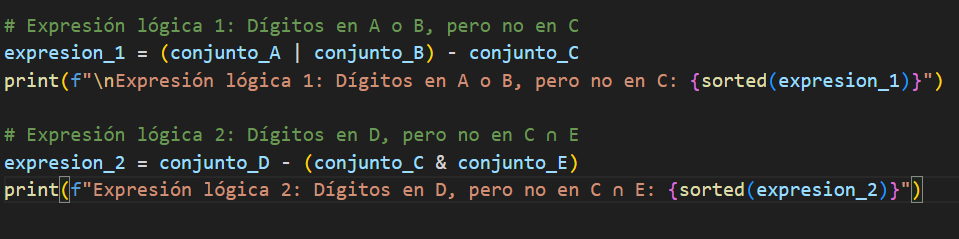
1. **Suma de Dígitos y Suma Total**

****

**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.

1. **Evaluación de Expresiones Lógicas**

****

**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:

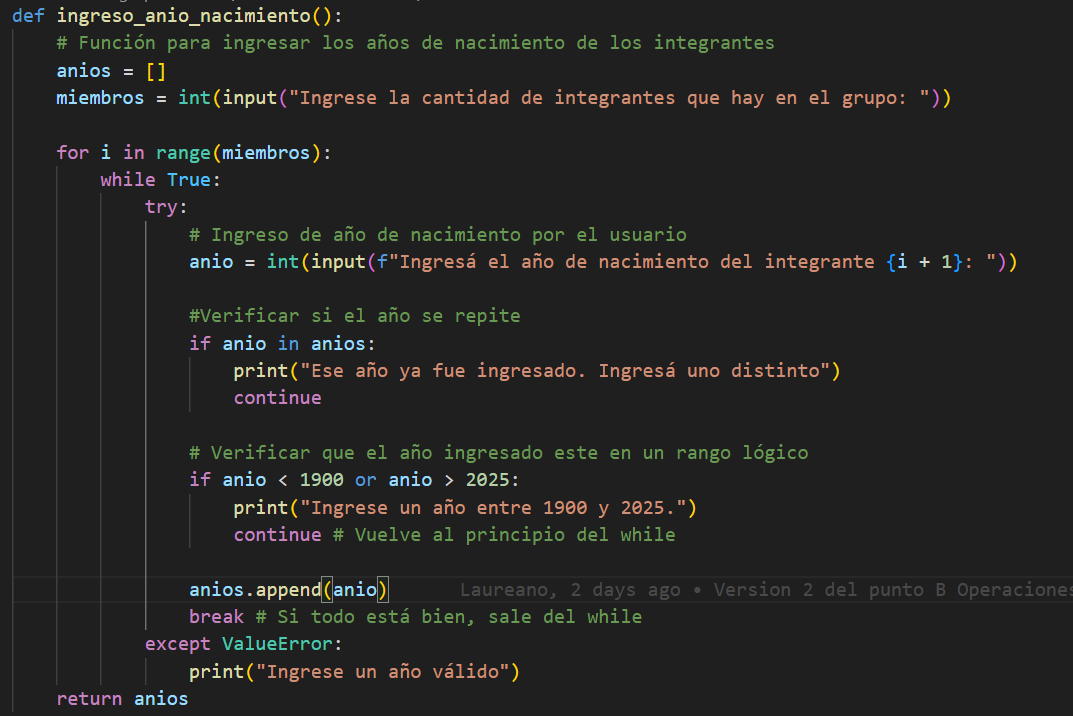
* Que permita el ingreso de los años de nacimiento de los integrantes del grupo. Si dos o más integrantes comparten el mismo año de nacimiento, se debe ingresar un dato ficticio para diferenciarlos, según corresponda.
* Utilice estructuras repetitivas para contar cuántos integrantes nacieron en años 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.



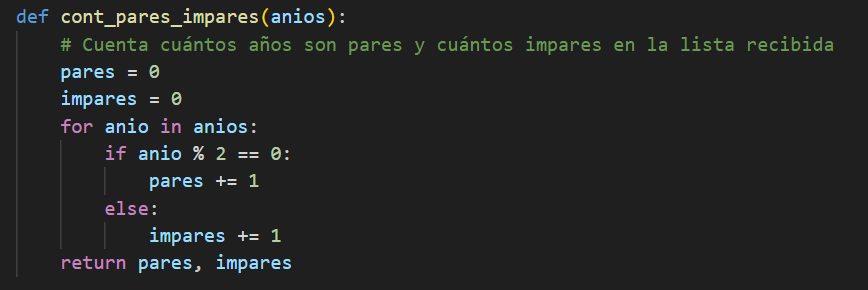
Solicita al usuario la cantidad de integrantes del grupo y sus años de nacimiento, asegurando que los años sean válidos (entre 1900 y 2025) y únicos (sin repeticiones).

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.
* Sepide 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 continue, 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.

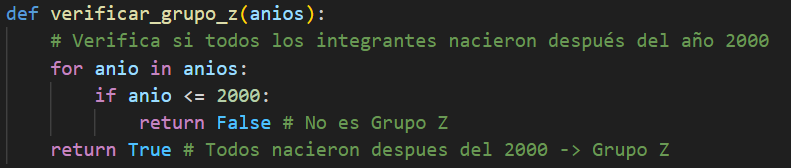
1. def cont\_pares\_impares(anios): Cuenta cuántos años de nacimiento son pares e impares.



**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.

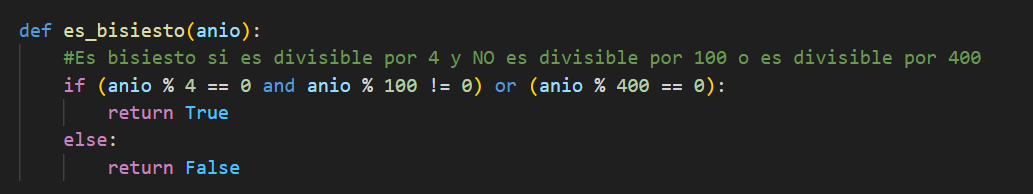
1. def verificar\_grupo\_z(anios): Verifica si todos los miembros nacieron después del 2000 (Generación Z).



**Explicación:**

* Con un ciclo forItera 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.

1. def es\_bisiesto(anio): Determina si un año es bisiesto.

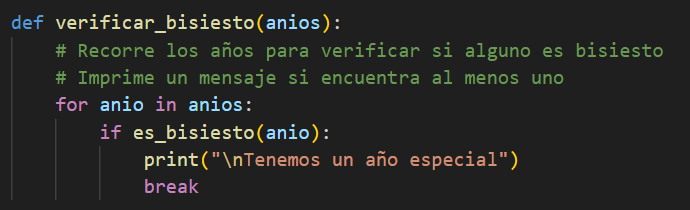


**Explicación:**

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

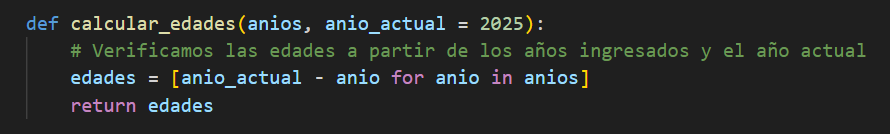
* Un año es bisiesto si es divisible por 4 Y no es divisible por 100.
* O, si es divisible por 400.
* Si la condición se cumple, el año es bisiesto.
* De lo contrario, no es un año bisiesto.

1. def verificar\_bisiesto(anios): Busca años bisiestos en la lista e imprime un mensaje si encuentra alguno.



* 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.

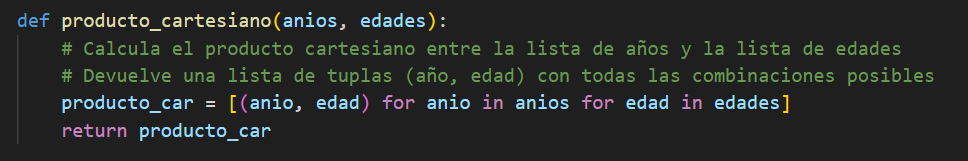
1. def calcular\_edades(anios, anio\_actual=2025): Calcula las edades restando los años de nacimiento del año actual.



**Explicación:**

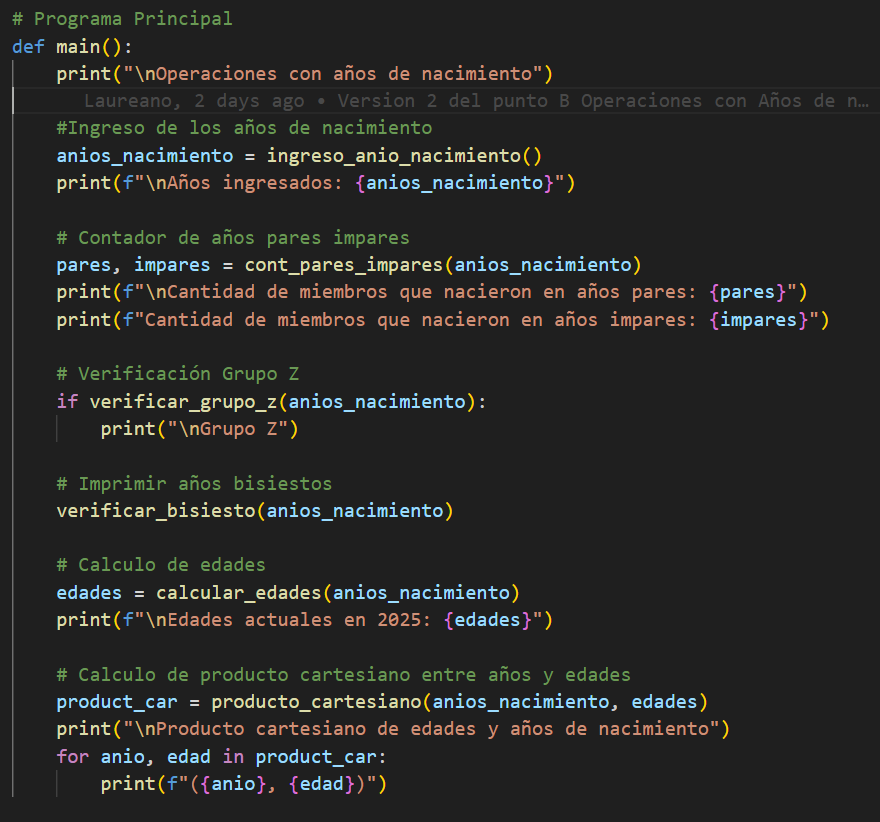
* Establece un valor predeterminado para el año actual en 2025.
* Crea una nueva lista llamada edades restando cada anio de la lista anios a anio\_actual.
* Devuelve la lista de edades calculadas.

1. def producto\_cartesiano(anios, edades): Calcula el producto cartesiano entre los años de nacimiento y las edades.



**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).

1. main(): Coordina el flujo del programa, llamando a las funciones anteriores y mostrando los resultados.
2. 

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 forItera 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**

**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.

**Tareas realizadas por cada integrante:**