**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**

Consigna 4

Parte 1: **Desarrollo del Programa en Python** 5

**B)** Operaciones con años de nacimiento

Parte 2: **Desarrollo Matemático (Conjuntos y Lógica)** 12

Parte 3: **Video de Presentación** 14

**Conclusión** 15

**Referencias** 16

**Tareas realizadas por cada Integrante** 17

**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:**

Desarrolle un programa que cumpla con los siguientes requisitos:

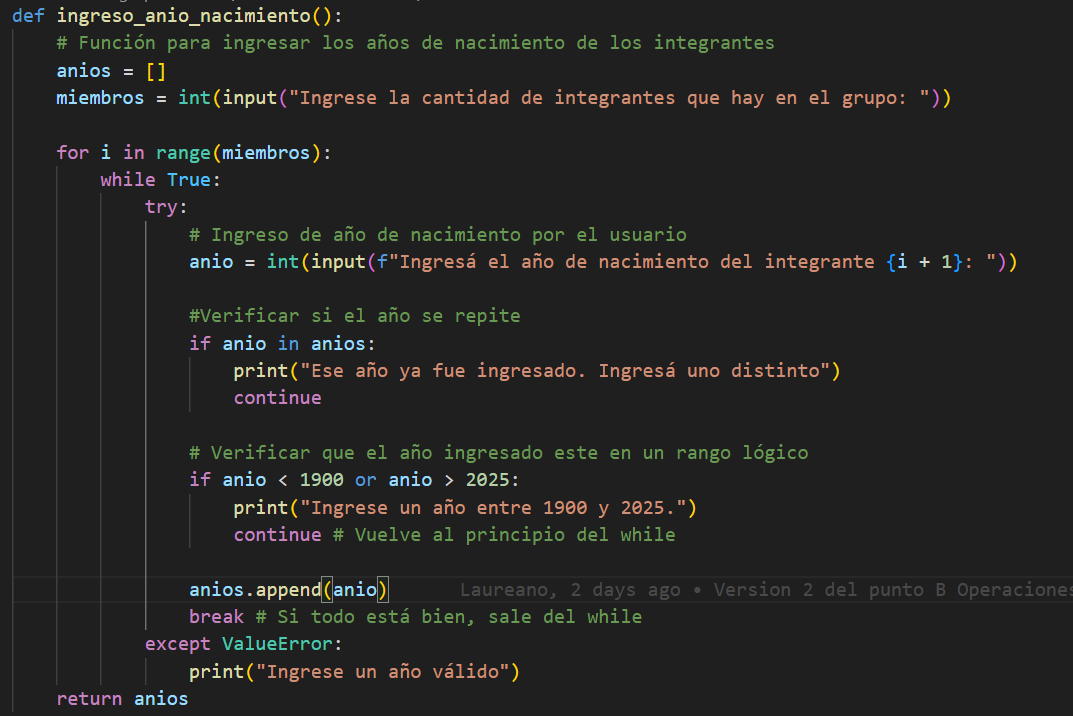
* Que permita el ingreso de los años de nacimiento de los integrantes del grupo. Si dos o más integrantes comparten el mismo año de nacimiento, se debe ingresar un dato ficticio para diferenciarlos, según corresponda.
* Utilice estructuras repetitivas para contar cuántos integrantes nacieron en años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

Parte 1: **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.



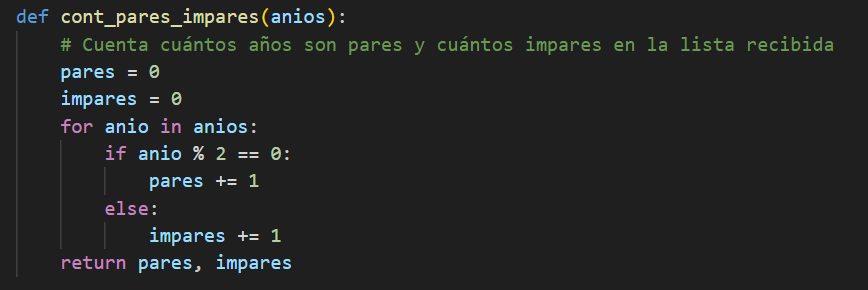
Solicita al usuario la cantidad de integrantes del grupo y sus años de nacimiento, asegurando que los años sean válidos (entre 1900 y 2025) y únicos (sin repeticiones).

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.

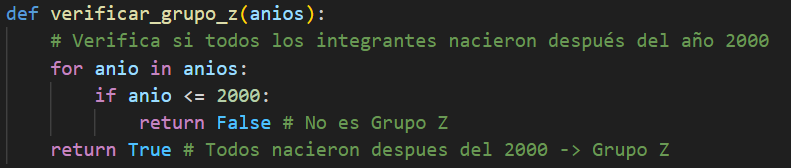
1. def cont\_pares\_impares(anios): Cuenta cuántos años de nacimiento son pares e impares.



**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.

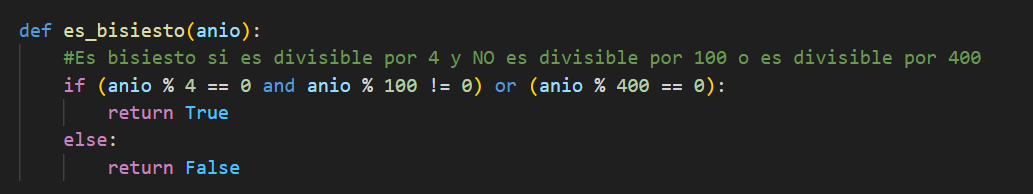
1. def verificar\_grupo\_z(anios): Verifica si todos los miembros nacieron después del 2000 (Generación Z).



**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.

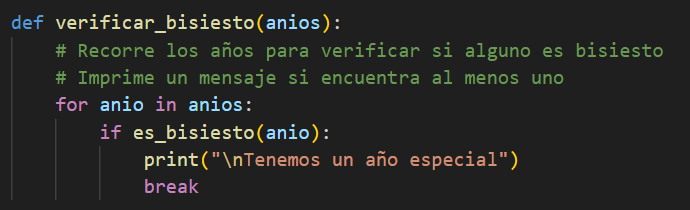


**Explicación:**

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

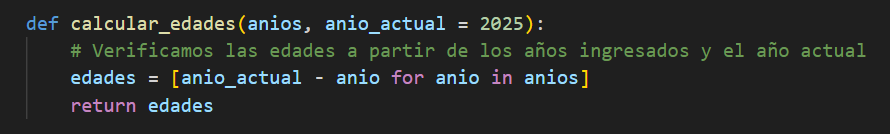
* Un año es bisiesto si es divisible por 4 Y no es divisible por 100.
* O, si es divisible por 400.
* Si la condición se cumple, el año es bisiesto.
* De lo contrario, no es un año bisiesto.

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.

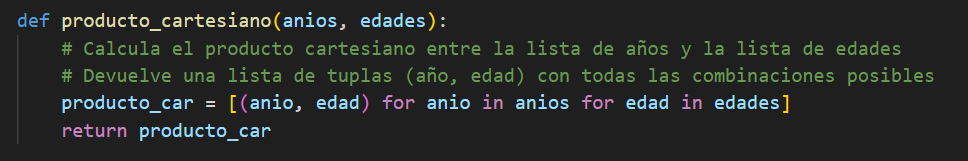
1. def calcular\_edades(anios, anio\_actual=2025): Calcula las edades restando los años de nacimiento del año actual.



**Explicación:**

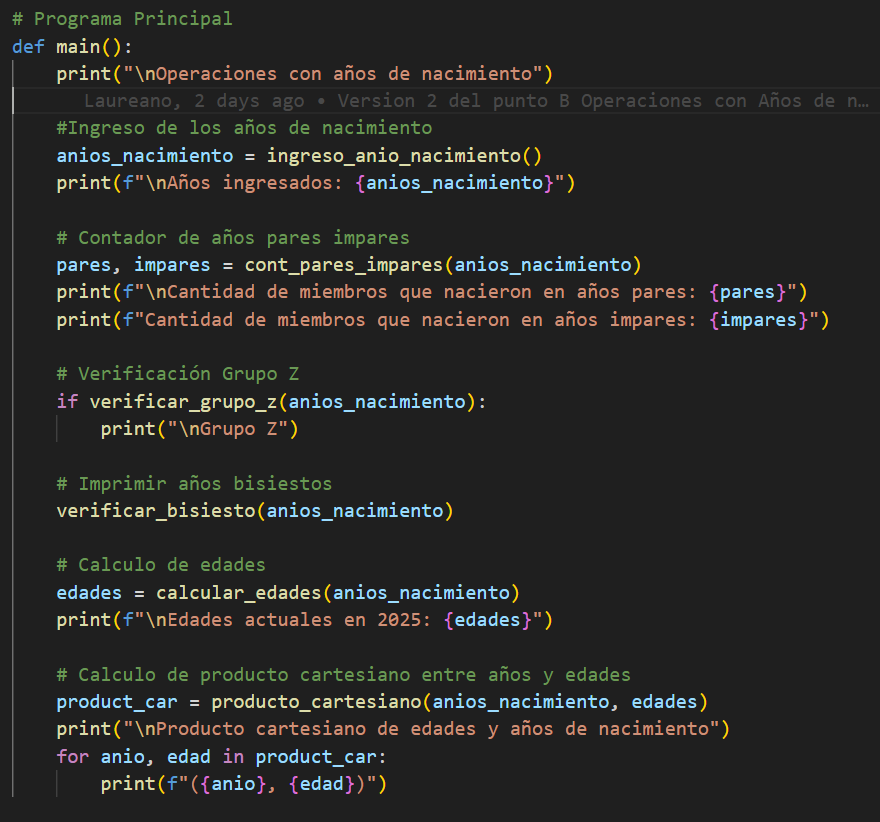
* Establece un valor predeterminado para el año actual en 2025.
* Crea una nueva lista llamada edades restando cada anio de la lista anios a anio\_actual.
* Devuelve la lista de edades calculadas.

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

Parte 2: **Desarrollo Matemático (Conjuntos y Lógica)**

La consigna y posterior desarrollo de código, aplica conceptos de matemáticas, específicamente de teoría de conjuntos y lógica.

1. **Teoría de Conjuntos:**
   * Los años de nacimiento (anios) y edades (edades) se tratan como conjuntos finitos.
   * producto\_cartesiano implementa el producto cartesiano conjunto A x conjunto B, generando todas las combinaciones de años y edades.
   * verificar\_grupo\_z verifica si el conjunto de años es un subconjunto de los años posteriores a 2000.
   * cont\_pares\_impares divide el conjunto de años en subconjuntos disjuntos(no tienen elementos en común) (pares e impares) y calcula sus cardinalidades.
2. **Lógica:**

ingreso\_anio\_nacimiento()

Esta función solicita años de nacimiento y valida las entradas. Las condiciones lógicas son:

**Condición para aceptar un año ingresado**:

* + "El valor ingresado es un número entero Y el año no está en la lista de años previos Y el año es mayor o igual a 1900 Y menor o igual a 2025."

cont\_pares\_impares(anios)

Esta función cuenta años pares e impares. La condición lógica determina si un año es par o impar.

**Condición para clasificar un año**:

* "Si el año es divisible por 2 (es decir, el resto de dividirlo por 2 es 0), entonces es par; de lo contrario, es impar."

verificar\_grupo\_z(anios)

Esta función verifica si todos los años son posteriores al 2000 (Generación Z).

**Condición para ser Generación Z**:

"Todos los años de la lista deben ser mayores a 2000."

Equivalente: "No debe existir ningún año en la lista que sea menor o igual a 2000."

es\_bisiesto(anio)

Esta función determina si un año es bisiesto.

**Condición para que un año sea bisiesto**:

* "El año es bisiesto si es divisible por 4 Y no es divisible por 100, O si es divisible por 400."

verificar\_bisiesto(anios)

Esta función verifica si hay al menos un año bisiesto en la lista.

**Condición para imprimir un mensaje**:

* "Si existe al menos un año en la lista que sea bisiesto, entonces imprime 'Tenemos un año especial'."

**Parte 3 – 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.

**Referencias:**

**Tareas realizadas por cada integrante:**