Detalle de integrantes, tareas realizadas y relación entre expresiones lógicas y código

Integrantes del grupo

- José Gabriel Torres – DNI : 42475956
- Gabriel Valdez Arce – DNI: 42642106
- Daiana Judith Velasquez Torrez – DNI: 39646139

Tareas realizadas por cada integrante

José Gabriel Torres :
Realizó la parte práctica relacionada con los DNIs. Programó el ingreso de los números, generó automáticamente los conjuntos de dígitos únicos, y desarrolló las operaciones de conjuntos (unión, intersección, deferencia y deferencia simétrica) en código. También implementó estructuras repetitivas para contar la frecuencia de aparición de cada dígito y calcular la suma total de los dígitos de cada DNI.

Relación con expresiones Lógicas:

Se utilizaron varias condiciones y estructuras repetitivas en el bloque relacionado con los DNIs. Por ejemplo:

1. **Validación del DNI ingresado:**
- Expresión lógica:
  "El DNI debe tener entre 7 y 9 dígitos y contener sólo números"
- Código:

```
8           if not dni.isdigit():
9               print("Por favor ingresa solo numeros.");
10          elif len(dni) < 7 or len(dni) > 9:
11              print("El DNI debe tener entre 7 y 9 digitos.");
12          else:
13              dnis.append(dni);
14              valido = True;
```

2. **Evaluación con estructuras repetitivas y condiciones:**
- Expresión lógica: "Contar cuántas veces aparece cada dígito en el DNI."

- Código:

```
# bucle for para recorrer los dni
for dni in dnis:
    suma = 0
    frecuencia = [0] * 10  ## creamos una lista vacia con 10, 0.
    for digito in dni:      ## para registrar la frecuencia.
        frecuencia[int(digito)] += 1
        suma += int(digito)
```

3. **Construcción y comparación ente conjuntos:**
   - Expresión lógica: “Mostrar los elementos únicos de cada DNI y aplicar operaciones entre ellos (unión, intersección, diferencia, etc.)”
   - Código:

```
# funcion con operaciones entre conjuntos de digitos
def operaciones_conjuntos(conjuntos):
    c1, c2, c3 = conjuntos

    union = c1 | c2 | c3
    interseccion = c1 & c2 & c3
    diferencia_1_2 = c1 - c2
    diferencia_1_3 = c1 - c3
    diferencia_2_3 = c2 - c3
    diferencia_simetrica = c1 ^ c2 ^ c3
```

Gabriel Valdez Arce:
Trabajó con los años de nacimiento. Programó el ingreso de los años, contó cuántos eran pares o impares y agregó condiciones como: *"Si todos nacieron después del 2000, mostrar 'Grupo Z'"* o *"Si alguno nació en año bisiesto, mostrar 'Tenemos un año especial'"*. Además, implementó una función para determinar si un año es bisiesto y calculó el producto cartesiano entre los años y las edades actuales.

Relación con expresiones lógicas:

Se aplicaron varias expresiones lógicas sobre los años de nacimiento:

1. **Validación del año ingresado:**
   - Expresión lógica: “El año debe estar entre 1900 y 2025.”
   - Código:

```
            if año < 1900 or año > 2025:
                print("Por favor ingrese un año válido entre 1900 y 2025.")
            else:
                años.append(año)
                valido = True
        except ValueError:
            print("Por favor ingrese un número válido.")
```

2. **Clasificación pares e impares:**
   - Expresión lógica: “Contar cuántos nacieron en años pares e impares.”
   - Código:

```
# Contar pares e impares
pares = sum(1 for a in años if a % 2 == 0)
impares = len(años) - pares
print("////////////////////////////")
print(f"Años pares: {pares} - Años impares: {impares}")
```

3. **Condición para grupo generacional:**
   - Expresión lógica: “Si todos nacieron después del 2000, mostrar Grupo Z.”
   - Código:

```
# Grupo Z
if all(a > 2000 for a in años):
    print("Grupo Z")
```

4. **Detección de año bisiesto:**
   - Expresión lógica: "Si alguno nació en un año bisiesto, mostrar 'Tenemos un año especial'."
   - Código:

```
# Año especial (bisiesto)
if any(es_bisiesto(a) for a in años):
    print("Tenemos un año especial")
```

5. **Producto cartesiano:**
   - Expresión lógica: "Relacionar cada año de nacimiento con cada edad actual para formar pares (año, edad)."
   - Código:

```python

# Producto cartesiano (año x edad)
print("Producto cartesiano (año x edad):")
for a in años:
    for e in edades:
        print((a, e))

```

Daiana Velásquez:
Se encargó de redactar el informe final del trabajo, relacionando los contenidos teóricos con las actividades prácticas. Realizó las operaciones entre conjuntos a partir de los dígitos de los DNIs (unión, intersección, diferencia y diferencia simétrica). También redactó expresiones lógicas en lenguaje natural y las implementó en código utilizando estructuras condicionales en Python.

1. **Diversidad numérica:**
   - Expresión lógica: "Si todos los conjuntos tienen al menos 5 elementos, mostrar 'Alta diversidad numérica'"

   - Código:

```python

# Expresión lógica 1: Alta diversidad
if len(A) >= 5 and len(B) >= 5 and len(C) >= 5:
    print("Alta diversidad numérica")
else:
    print("Diversidad numérica no suficiente")

```

2. **Diferencia entre conjuntos:**
   - Expresión lógica: “Si hay dígitos en A que no estén en B, mostrarlos.”
   - Código:

```python
# Expresión lógica 2: Diferencias
diferencia_A_B = A - B

if diferencia_A_B:
    print("Dígitos en A y no en B:", sorted(diferencia_A_B))
else:
    print("No hay dígitos en A que no estén en B")

```