

INSTITUTO TECNOLÓGICO DE ESTUDIOS SUPERIORES DE OCCIDENTE

Departamento de Electrónica, Sistemas e Informática.



ITESO

Universidad Jesuita
de Guadalajara

Proyecto Integrador

“Tool Kit FCC: Entrega final”

Materia:	Fundamentos de ciencias computacionales
Equipo:	Chicken Bone (Huesito de pollo)
Integrantes:	Lamego Soriano Mariana López Zúñiga Juan Pablo
Profesor:	Fernando Velasco Loera
Fecha:	13 de mayo de 2021
Periodo:	Primavera 2021

Tool Kit FCC

El programa fue escrito en el lenguaje Python, se trabajó de manera síncrona colaborativa gracias al entorno de programación en línea Repl.it. La librería tkinter del ya mencionado idioma Python fue utilizada para la codificación de la interfaz gráfica.

Interfaz Gráfica del Tool Kit

Instrucciones

Durante la ejecución de la caja de herramientas la primera interacción con el usuario es la interfaz mostrada. Dentro de la interfaz se muestran las 4 opciones de herramientas que un usuario puede elegir usar. Al seleccionar alguna de las 4 herramientas, se despliega una nueva ventana con la herramienta a usar. El programa termina cuando se cierran todas las ventanas de la interfaz gráfica.



Generador de tablas de verdad

Instrucciones y restricciones

Al seleccionar la opción de Tablas de Verdad se despliega en pantalla una calculadora destinada a recibir las sentencias lógicas del usuario, al hacer clic en cada botón se imprime en el cuadro de entrada su valor. Se recomienda el uso apropiado de paréntesis para separar las premisas, los botones de borrado eliminan la entrada y el botón de igual despliega la tabla de verdad.

Se hace mención de que no es posible evaluar la negación si esta se encuentra afuera del paréntesis, en caso de que esto suceda, también lo hará un error fatal en el código.



Otra limitación relevante es que la evaluación de una segunda premisa dentro de un paréntesis culmina en otro error fatal para el programa. La última restricción es que se debe reiniciar por completo el programa para que este funcione óptimamente, debido a que los resultados se quedan guardados y esto afecta el desempeño del programa.

Los valores de la tabla de verdad se representan en valores binarios donde 1 es Verdadero y 0 es Falso.

Calculadora de conjuntos (Set Calculator)

Instrucciones y restricciones

La primera interacción del usuario con la opción de Calculadora de conjuntos es la siguiente ventana:

Calculadora de conjuntos

Instrucciones: Separar por comas sin espacios los elementos ingresados.

Ingresar máximo 10 elementos.

A = { }

B = { }

C = { }

Seleccionar operaciones

Unión

☐ $A \cup B$ ☐ $A \cup C$ ☐ $B \cup C$

☐ $A \cup (B \cap C)$

Intersección

☐ $A \cap B$ ☐ $B \cap C$ ☐ $A \cap C$

☐ $A \cap (B \cap C)$

Diferencia

☐ $A - B$ ☐ $B - A$

☐ $A - C$ ☐ $C - A$

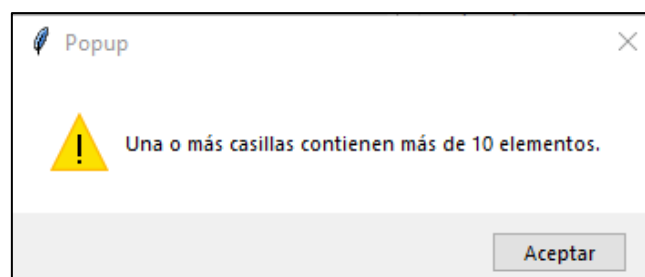
☐ $B - C$ ☐ $C - B$

Diferencia Simétrica

☐ $A \Delta B$ ☐ $A \Delta C$ ☐ $B \Delta C$

Hacer Conjuntos

Del lado izquierdo se explican las instrucciones primordiales para correr el programa de forma efectiva. Se le pide al usuario que ingrese los elementos separados por comas para respetar la sintaxis de los conjuntos, y porque de esta manera el programa detecta que son varios elementos los que se han ingresado. Además, se advierte que no se pongan espacios y que el máximo número de elementos para ingresar es de 10, mostrando el siguiente mensaje en caso de exceder esta cardinalidad.



Los elementos válidos para ingresar a los cuadros de texto son alfanuméricos, esto quiere decir que se aceptan tanto letras como números. Cabe mencionar que el programa no presentará errores si uno de los conjuntos se queda en blanco.

Una vez que los elementos fueron ingresados por el usuario, se debe elegir la o las operaciones que se desean realizar. Las operaciones disponibles para elegir se muestran del lado derecho de la pantalla principal: Unión, Intersección, Diferencia o Diferencia Simétrica. En el siguiente ejemplo se seleccionaron todas las operaciones posibles entre conjuntos.

Calculadora de conjuntos

Seleccionar operaciones

Unión

☒ $A \cup B$ ☒ $A \cup C$ ☒ $B \cup C$
☒ $A \cup (B \cap C)$

Intersección

☒ $A \cap B$ ☒ $B \cap C$ ☒ $A \cap C$
☒ $A \cap (B \cap C)$

Diferencia

☒ $A - B$ ☒ $B - A$
☒ $A - C$ ☒ $C - A$
☒ $B - C$ ☒ $C - B$

Diferencia Simétrica

☒ $A \Delta B$ ☒ $A \Delta C$ ☒ $B \Delta C$

Hacer Conjuntos

Instrucciones: Separar por comas sin espacios los elementos ingresados.

Ingresar máximo 10 elementos.

A = { 1,2,3,4,5 }

B = { 2,3,4,5,5,6,7,8 }

C = { 1,2,3,6 }

Ya que se han elegido las operaciones que se quieren visualizar, solamente queda presionar el botón 'Hacer Conjuntos' del lado inferior derecho de la pantalla. Al presionar dicho botón se desplegará una nueva ventana mostrando los resultados de las operaciones solicitadas.

Resultados

Unión

$A \cup B$: { 1 2 3 4 5 6 7 8 }

$A \cup C$: { 1 2 3 4 5 6 }

$B \cup C$: { 2 3 4 5 6 7 8 1 }

$A \cup (B \cap C)$: { 1 2 3 4 5 6 7 8 }

Intersección

$A \cap B$: { 2 3 4 5 }

$A \cap C$: { 1 2 3 }

$B \cap C$: { 2 3 6 }

$A \cap (B \cap C)$: { 2 3 }

Diferencia

$A - B$: { 1 }

$B - A$: { 6 7 8 }

$A - C$: { 4 5 }

$C - A$: { 6 }

$B - C$: { 4 5 7 8 }

$C - B$: { 1 }

Diferencia Simétrica

$A \Delta B$: { 1 6 7 8 }

$A \Delta C$: { 4 5 6 }

$B \Delta C$: { 4 5 7 8 1 }

Cadinalidad A: 5

Cadinalidad B: 7

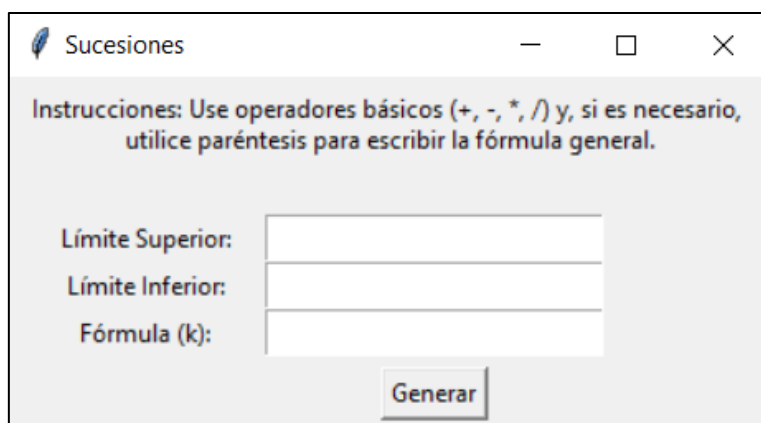
Cadinalidad C: 4

Como se puede ver en la imagen anterior, la cardinalidad de cada conjunto se presenta de forma automática. Por otro lado, los resultados se muestran sin comas, lo cual se toma como una restricción del programa.

En dado caso que se quiera mostrar otras respuestas se debe cerrar la ventana de resultados y modificar los conjuntos y/o las operaciones esperadas.

Instrucciones y restricciones

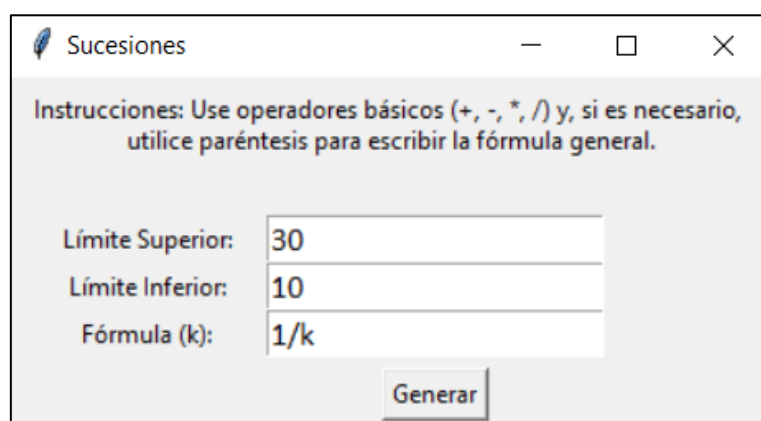
Al seleccionarse la opción de sucesiones, el usuario interactúa con la herramienta mediante la siguiente ventana:



The screenshot shows a window titled 'Sucesiones' with a feather icon. Inside, there is an instruction: 'Instrucciones: Use operadores básicos (+, -, *, /) y, si es necesario, utilice paréntesis para escribir la fórmula general.' Below this, there are three input fields: 'Límite Superior:', 'Límite Inferior:', and 'Fórmula (k):'. A 'Generar' button is located at the bottom right of the input area.

En la interfaz se le muestran al usuario 3 recuadros para que pueda introducir la información necesaria para crear una sucesión. Primero se le pide al usuario el límite superior del número de términos que formará la sucesión. Después, se le pregunta al usuario desde que posición desea iniciar la sucesión, es decir el límite inferior. Por último, se le pregunta al usuario cuál es la fórmula en la que estará basada la sucesión y se presenta el botón “Generar” para los resultados.

Dentro de la GUI se le especifica al usuario que debe de escribir la fórmula con operadores aritméticos básicos y que puede usar paréntesis para priorizar una operación sobre otra. Aunque no está escrito de manera explícita en las instrucciones, el usuario debe de crear la fórmula en función de k como se muestra en el siguiente ejemplo:



This screenshot shows the same 'Sucesiones' window as before, but with example values entered: 'Límite Superior:' is 30, 'Límite Inferior:' is 10, and 'Fórmula (k):' is $1/k$. The 'Generar' button remains at the bottom right.

Finalmente, al presionar el botón *Generar*, se abre otra ventana con los términos de la sucesión, así como la sumatoria y la multiplicación de estos. Los términos empiezan en el límite inferior y van hasta el superior. En la siguiente página se muestra la pantalla de resultados para el ejemplo anterior.

Para mostrar otras respuestas se debe cerrar la ventana de resultados y modificar los parámetros que se deseen.

Resultados	
Sumatoria = 1.1660188769521374	
Multiplicación = 1.3680531107447158e-27	
Término 10:	0.1
Término 11:	0.09090909090909091
Término 12:	0.08333333333333333
Término 13:	0.07692307692307693
Término 14:	0.07142857142857142
Término 15:	0.06666666666666667
Término 16:	0.0625
Término 17:	0.058823529411764705
Término 18:	0.05555555555555555
Término 19:	0.05263157894736842
Término 20:	0.05
Término 21:	0.047619047619047616
Término 22:	0.045454545454545456
Término 23:	0.043478260869565216
Término 24:	0.041666666666666664
Término 25:	0.04
Término 26:	0.038461538461538464
Término 27:	0.037037037037037035
Término 28:	0.03571428571428571
Término 29:	0.034482758620689655
Término 30:	0.03333333333333333

Calculadora de Relaciones

Instrucciones y restricciones

Al ejecutarse la opción de Relaciones y funciones, la primera interacción del usuario con la herramienta es la siguiente ventana:

Relaciones y funciones

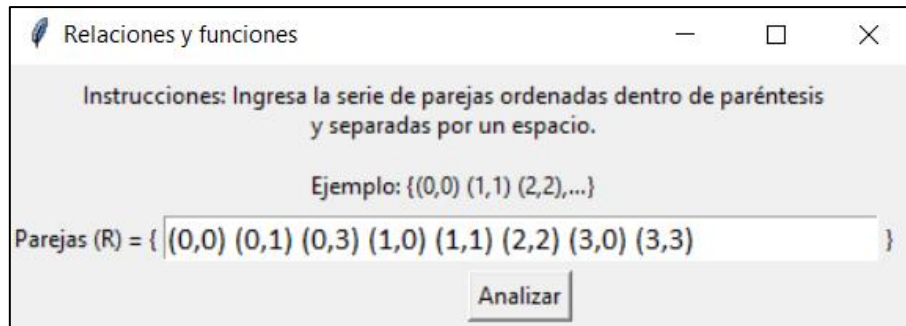
Instrucciones: Ingresa la serie de parejas ordenadas dentro de paréntesis y separadas por un espacio.

Ejemplo: {(0,0) (1,1) (2,2),...}

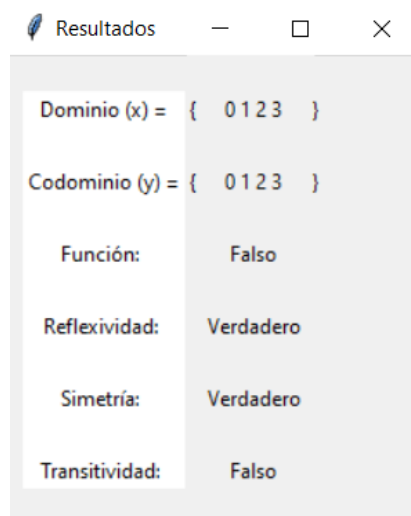
Parejas (R) = {
}

Analizar

En la interfaz se encuentra un recuadro en el que se debe introducir la relación, expresada como una serie de parejas ordenadas. Se especifica que el usuario debe de ingresar la serie de parejas dentro de paréntesis, además de que se deben separar por un espacio y se muestra un ejemplo. Este formato se pide por especificaciones en el código.



Al presionar el botón *Analizar*, se abre una nueva ventana mostrando los resultados del análisis. Se muestra el dominio y codominio, así como si se trata de una función o no. Del mismo modo, se muestra si la relación cuenta con las propiedades de reflexividad, simetría y transitividad. En la imagen de abajo se muestran los resultados del ejemplo anterior.



Nota: Para utilizar esta herramienta varias veces es necesario cerrar la ventana y volver a seleccionar la opción.

Código fuente

```
# Interfaz gráfica para caja de herramientas
from tkinter import * # librería
from tkinter import messagebox # librería

# Formato de la ventana
ventana = Tk()
ventana.title("Tool-Kit FCC")

D = 0 # Variable global para recorrer entrada
```

```

# Funciones para cada herramienta
def Tablas_de_Verdad():
    ventana_1 = Toplevel()
    ventana_1.title("Tablas de verdad")
    ventana_1.geometry("600x300")

    frame_tabla = Frame(ventana_1)
    frame_calculadora = Frame(ventana_1)
    frame_tabla.grid(row=0, column=1)
    frame_calculadora.grid(row=0, column=0)

# Listas y diccionarios globales donde se guardan los argumentos ingresados
lista_de_premisas = ['p', 'q', 'r', 's', 't']
valores = {}
premisas_perdidas = {}
num_premisa = []
lista_de_simbolos = ['^', 'v', '→', '↔', '(', ')']
todas_premisas = []
lista_negacion = []
premisa_negada = []
lista_de_simbolo = []
lista_parentesis = []
parentesis_cadena = []

# Cuadro de entrada de la calculadora
entradat = Entry(frame_calculadora, font=("Calibri 18"))
entradat.grid(row=0, column=0, columnspan=6, padx=5, pady=5)

# Funciones (de los botones de la calculadora)

```



```

# Para que al dar click en un botón, su valor se imprima en la pantalla
def click_boton(valor):
    global D
    entradat.insert(D, valor)

    D += 1 # De esta forma se desplaza un lugar para insertar el próximo
valor
    return entradat

# Para borrar todo el cuadro de entrada
def borrar():
    entradat.delete(0, END)

    D = 0

    for widget in frame_tabla.winfo_children():
        widget.destroy()

# Borrar sólo un valor
def dele():
    # Se toma todo el "texto" que está en el cuadro de entrada menos el último
dígito
    txt = entradat.get()[:-1]

    # Se borra todo el contenido del cuadro
    entradat.delete(0, END)

    # Se reinserta el enunciado, sin el último dígito
    entradat.insert(0, txt)

# Funciones para generar la tabla

# Guardar la entrada del usuario en listas diferentes
def generador_tabla():
    argumento = entradat.get()

    x = 0

    for premisa in argumento:
        if premisa in lista_de_premisas:

```

```

    todas_premisas.append(premisa)
    if premisa not in num_premisa:
        num_premisa.append(premisa)
if premisa in lista_de_simbolos and x == 0:
    lista_de_simbolo.append(premisa)
if premisa == '~':
    lista_negacion.append(premisa)
if (premisa != '~') and (premisa not in lista_de_simbolos):
    lista_negacion.append('')
if premisa == '(':
    x = 1
if x == 0:
    lista_parentesis.append('')
if x == 1:
    lista_parentesis.append(premisa)
if premisa == ')':
    x = 0

```

Otorgar valores de verdadero o falso a cada proposición de acuerdo con la cantidad de premisas que tenga

```

def valor_premisa(cantidad_premisas):
    if cantidad_premisas == 1:
        valores[num_premisa[0]] = [True, False]
    if cantidad_premisas == 2:
        valores[num_premisa[0]] = [True, True, False, False]
        valores[num_premisa[1]] = [True, False, True, False]
        premisas_perdidas[num_premisa[1]] = [True, False, True, False]
    if cantidad_premisas == 3:
        valores[num_premisa[0]] = [True, True, True, True, False, False,
False, False]
        valores[num_premisa[1]] = [True, True, False, False, True, True,
False, False]
        valores[num_premisa[2]] = [True, False, True, False, True, False,
True, False]
        premisas_perdidas[num_premisa[1]] = [True, True, False, False,
True, True, False, False]

```

```

        premisas_perdidas[num_premisa[2]] = [True, False, True, False,
True, False, True, False]

        if cantidad_premisas == 4:

            valores[num_premisa[0]] = [True, True, True, True, True, True,
True, True, False, False, False, False,

False, False, False, False]

            valores[num_premisa[1]] = [True, True, True, True, False, False,
False, False, True, True, True, True,

False, False, False, False]

            valores[num_premisa[2]] = [True, True, False, False, True, True,
False, False, True, True, False, False,

True, True, False, False]

            valores[num_premisa[3]] = [True, False, True, False, True, False,
True, False, True, False, True, False,

True, False, True, False]

            premisas_perdidas[num_premisa[1]] = [True, True, True, True,
False, False, False, False, True, True, True,

True,

False, False, False, False]

            premisas_perdidas[num_premisa[2]] = [True, True, False, False,
True, True, False, False, True, True, False,

False,

True, True, False, False]

            premisas_perdidas[num_premisa[3]] = [True, False, True, False,
True, False, True, False, True, False, True,

False,

True, False, True, False]

        if cantidad_premisas == 5:

            valores[num_premisa[0]] = [True, True, True, True, True, True,
True, True, True, True, True, True,

True, True, True, False, False, False,

False, False, False, False, False, False,

False, False]

            valores[num_premisa[1]] = [True, True, True, True, True, True,
True, True, False, False, False, False,

False, False, False, False, True, True,

True, True, True, True, True, True,

```

```

False, False, False]
False, False, False, False, False,
False, False, False]
valores[num_premisa[2]] = [True, True, True, True, False, False,
False, False, True, True, True, True,
False, False, False, False, True, True,
True, True, False, False, False, False,
True, True, True, True, False, False,
False, False]
valores[num_premisa[3]] = [True, True, False, False, True, True,
False, False, True, True, False, False,
True, True, False, False, True, True,
False, False, True, True, False, False,
True, True, False, False, True, True,
False, False]
valores[num_premisa[4]] = [True, False, True, False, True, False,
True, False, True, False, True, False,
True, False, True, False, True, False,
True, False, True, False, True, False,
True, False, True, False, True, False,
True, False]
premisas_perdidas[num_premisa[1]] = [True, True, True, True, True,
True, True, True, False, False, False,
False,
False, False, False, False,
True, True, True, True, True, True, True,
True,
False, False, False, False,
False, False, False, False]
premisas_perdidas[num_premisa[2]] = [True, True, True, True,
False, False, False, False, True, True, True,
True,
False, False, False, False,
True, True, True, True, False, False,
False, False,
True, True, True, True, False,
False, False, False]
premisas_perdidas[num_premisa[3]] = [True, True, False, False,
True, True, False, False, True, True, False,
False,
True, True, False, False,
True, True, False, False, True, True, False,
True, True, False, False, True, True, False,

```

```

False,
True, True, False, False,
True, True, False, False]

premisas_perdidas[num_premisa[4]] = [True, False, True, False,
True, False, True, False, True, False, True,
False,
True, False, True, False,
True, False, True, False, True, False, True,
False,
True, False, True, False,
True, False, True, False]

```

```

valor_premisa(len(num_premisa))

```

Agregar las proposiciones que se encuentran negadas por el símbolo '~',
se agrega con la letra que lo acompaña

```

def negacion():
    contador1 = 0
    contador2 = 0
    lista_temporalTF = []
    lista_temporaln = []
    lista_temporal_igual_a_p_negada = []
    for simbolo_n in lista_negacion:
        if contador1 < contador2:
            contador1 += 1
        if simbolo_n == '~':
            clave = f"{simbolo_n}{todas_premisas[contador1]}"
            premisa_negada.append(clave)
            todas_premisas.append(clave)
            contador1 += 1
        if simbolo_n == '':
            contador2 += 1
    for n in premisa_negada:
        letra = n[1:2]
        lista_temporaln.append(letra)
    for i in premisa_negada:

```

```

        lista_temporal_igual_a_p_negada.append(i)
    for premisa in lista_temporaln:
        llave = valores.get(premisa)
        for TF in llave:
            TF = not TF
            lista_temporalTF.append(TF)
        for x in lista_temporal_igual_a_p_negada:
            valores[x] = lista_temporalTF
            lista_temporalTF = []
            lista_temporal_igual_a_p_negada.remove(x)
            break

```

negacion()

Guarda las expresiones dentro de una pareja de paréntesis en una lista y en el diccionario

```

def parentesis():
    lista_temporal = []
    lista_temporal_letras = []
    lista_TF = []
    otroTF = []
    premisa_final = []
    acumulador = ''
    letranegada = ''
    x = 0
    contador1 = 0
    contador2 = 0
    letra_a_quitar = ''
    for premisa in lista_parentesis:
        if premisa != '':
            lista_temporal.append(premisa)
    for premisa in lista_temporal:
        if premisa == '~':
            x = 1

```

```

        acumulador += premisa
    if x == 1:
        letranegada += premisa
        if premisa in todas_premisas:
            letra_a_quitar += premisa
    if premisa in todas_premisas:
        acumulador += premisa
        lista_temporal_letras.append(premisa)
        x = 0
    if premisa in letranegada and premisa != '~':
        lista_temporal_letras.remove(letra_a_quitar)
        lista_temporal_letras.append(letranegada)
        contador2 += 1
        letranegada = ''
        letra_a_quitar = ''
    if premisa in lista_de_simbolos:
        acumulador += premisa
    parentesis_cadena.append(acumulador)
    if parentesis_cadena[0] != '':
        todas_premisas.append(acumulador)
    if '^' in acumulador:
        for premisa in lista_temporal_letras:
            llave = valores.get(premisa)
            lista_TF.append(llave)
        otroTF.append(lista_TF[1])
        lista_TF.pop(1)
        for valores_TF1 in lista_TF:
            for valores_TF2 in otroTF:
                for TF1 in valores_TF1:
                    for TF2 in valores_TF2:
                        TF = TF1 and TF2
                        premisa_final.append(TF)
                        valores_TF2.pop(contador1)
                        break

```

```

    valores[acumulador] = premisa_final
if 'V' in acumulador:
    for premisa in lista_temporal_letras:
        llave = valores.get(premisa)
        lista_TF.append(llave)
    otroTF.append(lista_TF[1])
    lista_TF.pop(1)
    for valores_TF1 in lista_TF:
        for valores_TF2 in otroTF:
            for TF1 in valores_TF1:
                for TF2 in valores_TF2:
                    TF = TF1 or TF2
                    premisa_final.append(TF)
                    valores_TF2.pop(contador1)
                    break
    valores[acumulador] = premisa_final
if '→' in acumulador:
    for premisa in lista_temporal_letras:
        llave = valores.get(premisa)
        lista_TF.append(llave)
    otroTF.append(lista_TF[1])
    lista_TF.pop(1)
    for valores_TF1 in lista_TF:
        for valores_TF2 in otroTF:
            for TF1 in valores_TF1:
                for TF2 in valores_TF2:
                    TF = not TF1 or TF2
                    premisa_final.append(TF)
                    valores_TF2.pop(contador1)
                    break
    valores[acumulador] = premisa_final
if '↔' in acumulador:
    for premisa in lista_temporal_letras:
        llave = valores.get(premisa)

```



```

        lista_TF.append(llave)
    otroTF.append(lista_TF[1])
    lista_TF.pop(1)
    for valores_TF1 in lista_TF:
        for valores_TF2 in otroTF:
            for TF1 in valores_TF1:
                for TF2 in valores_TF2:
                    TF = (not TF1 or TF2) and (not TF2 or TF1)
                    premisa_final.append(TF)
                    valores_TF2.pop(contador1)
                    break
    valores[acumulador] = premisa_final

```

```

parentesis()

```

Evalua los argumentos dando prioridad a los parentesis y la jerarquía de operaciones lógicas

```

def operacion():
    contador1 = 0
    contador2 = 1
    contador3 = 0
    contador4 = 0
    lista_TF = []
    otroTF = []
    lista_temporal = []
    lista_temporaln = []
    lista_temporal_parentesis = []
    premisa_final = []
    acumulador = ''
    acumulador2 = ''
    primero_parentesis = 0
    primero_letra = 0
    for cadenas in lista_parentesis:
        lista_temporal_parentesis.append(cadenas)

```

```

for n in lista_negacion:
    lista_temporaln.append(n)
for premisa in lista_de_simbolo:
    if premisa == '^':
        if parentesis_cadena[0] == '':
            for p in lista_temporaln:
                if p == '~':
                    lista_temporal.append(premisa_negada[contador1])
                    lista_temporaln.pop(contador2)
                    contador1 += 1
                    contador2 += 1
                    contador4 += 1
                if p == '':
                    lista_temporal.append(todas_premisas[contador4])
                    contador2 += 1
                    contador4 += 1
            todas_premisas.append(argumento)
            for conjuncion in lista_temporal:
                llave = valores.get(conjuncion)
                lista_TF.append(llave)
            otroTF.append(lista_TF[1])
            lista_TF.pop(1)
            for valores_TF1 in lista_TF:
                for valores_TF2 in otroTF:
                    for TF1 in valores_TF1:
                        for TF2 in valores_TF2:
                            TF = TF1 and TF2
                            premisa_final.append(TF)
                            valores_TF2.pop(0)
                        break
            valores[argumento] = premisa_final
        if parentesis_cadena[0] != '':
            for f in lista_temporal_parentesis:
                if f == '(':

```

```

        primero_parentesis = 50
        break
    if f == '':
        primero_letra = 50
        break
if primero_parentesis > primero_letra:
    lista_temporal.append(parentesis_cadena[contador1])
    acumulador = argumento[6:]
    for o in acumulador:
        if o == '~':
            for h in parentesis_cadena:
                acumulador2 += h
                while premisa_negada[contador3] in
acumulador2:
                    contador3 += 1
                    if premisa_negada[contador3] not in
acumulador2:
lista_temporal.append(premisa_negada[contador3])
        if o in lista_de_premisas:
            for h in parentesis_cadena:
                acumulador2 += h
                while num_premisa[contador3] in
acumulador2:
                    contador3 += 1
                    if num_premisa[contador3] not in
acumulador2:
lista_temporal.append(num_premisa[contador3])
    if primero_parentesis < primero_letra:
        acumulador = argumento[0:1]
        for o in acumulador:
            if o == '~':
                lista_temporal.append(premisa_negada[0])
            if o in lista_de_premisas:
                lista_temporal.append(num_premisa[0])

```

```

        lista_temporal.append(parentesis_cadena[0])
    todas_premisas.append(argumento)
    for conjuncion in lista_temporal:
        llave = valores.get(conjuncion)
        lista_TF.append(llave)
    otroTF.append(lista_TF[1])
    lista_TF.pop(1)
    for valores_TF1 in lista_TF:
        for valores_TF2 in otroTF:
            for TF1 in valores_TF1:
                for TF2 in valores_TF2:
                    TF = TF1 and TF2
                    premisa_final.append(TF)
                    valores_TF2.pop(0)
                    break
        valores[argumento] = premisa_final
if premisa == 'V':
    if parentesis_cadena[0] == '':
        for p in lista_temporaln:
            if p == '~':
                lista_temporal.append(premisa_negada[contador1])
                lista_temporaln.pop(contador2)
                contador1 += 1
                contador2 += 1
                contador4 += 1
            if p == '':
                lista_temporal.append(todas_premisas[contador4])
                contador2 += 1
                contador4 += 1
    todas_premisas.append(argumento)
    for conjuncion in lista_temporal:
        llave = valores.get(conjuncion)
        lista_TF.append(llave)
    otroTF.append(lista_TF[1])

```

```

lista_TF.pop(1)
for valores_TF1 in lista_TF:
    for valores_TF2 in otroTF:
        for TF1 in valores_TF1:
            for TF2 in valores_TF2:
                TF = TF1 or TF2
                premisa_final.append(TF)
                valores_TF2.pop(0)
            break
    valores[argumento] = premisa_final
if parentesis_cadena[0] != '':
    for f in lista_temporal_parentesis:
        if f == '(':
            primero_parentesis = 50
            break
        if f == '':
            primero_letra = 50
            break
    if primero_parentesis > primero_letra:
        lista_temporal.append(parentesis_cadena[contador1])
        acumulador = argumento[6:]
        for o in acumulador:
            if o == '~':
                for h in parentesis_cadena:
                    acumulador2 += h
                    while premisa_negada[contador3] in
acumulador2:
                        contador3 += 1
                        if premisa_negada[contador3] not in
acumulador2:
lista_temporal.append(premisa_negada[contador3])
    if o in lista_de_premisas:
        for h in parentesis_cadena:
            acumulador2 += h

```

```

                                while num_premisa[contador3] in
acumulador2:
                                contador3 += 1
                                if num_premisa[contador3] not in
acumulador2:

lista_temporal.append(num_premisa[contador3])
    if primero_parentesis < primero_letra:
        acumulador = argumento[0:1]
        for o in acumulador:
            if o == '~':
                lista_temporal.append(premisa_negada[0])
            if o in lista_de_premisas:
                lista_temporal.append(num_premisa[0])
            lista_temporal.append(parentesis_cadena[0])
        todas_premisas.append(argumento)
        for conjuncion in lista_temporal:
            llave = valores.get(conjuncion)
            lista_TF.append(llave)
        otroTF.append(lista_TF[1])
        lista_TF.pop(1)
        for valores_TF1 in lista_TF:
            for valores_TF2 in otroTF:
                for TF1 in valores_TF1:
                    for TF2 in valores_TF2:
                        TF = TF1 or TF2
                        premisa_final.append(TF)
                        valores_TF2.pop(0)
                    break
                valores[argumento] = premisa_final
if premisa == '→':
    if parentesis_cadena[0] == '':
        for p in lista_temporal:
            if p == '~':
                lista_temporal.append(premisa_negada[contador1])

```

```

        lista_temporaln.pop(contador2)
        contador1 += 1
        contador2 += 1
        contador4 += 1
    if p == '':
        lista_temporal.append(todas_premisas[contador4])
        contador2 += 1
        contador4 += 1
todas_premisas.append(argumento)
for conjuncion in lista_temporal:
    llave = valores.get(conjuncion)
    lista_TF.append(llave)
otroTF.append(lista_TF[1])
lista_TF.pop(1)
for valores_TF1 in lista_TF:
    for valores_TF2 in otroTF:
        for TF1 in valores_TF1:
            for TF2 in valores_TF2:
                TF = not TF1 or TF2
                premisa_final.append(TF)
                valores_TF2.pop(0)
            break
    valores[argumento] = premisa_final
if parentesis_cadena[0] != '':
    for f in lista_temporal_parentesis:
        if f == '(':
            primero_parentesis = 50
            break
        if f == '':
            primero_letra = 50
            break
if primero_parentesis > primero_letra:
    lista_temporal.append(parentesis_cadena[contador1])
    acumulador = argumento[6:]

```

```

        for o in acumulador:
            if o == '~':
                for h in parentesis_cadena:
                    acumulador2 += h
                    while premisa_negada[contador3] in
acumulador2:
                        contador3 += 1
                        if premisa_negada[contador3] not in
acumulador2:

lista_temporal.append(premisa_negada[contador3])

        if o in lista_de_premisas:
            for h in parentesis_cadena:
                acumulador2 += h
                while num_premisa[contador3] in
acumulador2:
                    contador3 += 1
                    if num_premisa[contador3] not in
acumulador2:

lista_temporal.append(num_premisa[contador3])

        if primero_parentesis < primero_letra:
            acumulador = argumento[0:1]
            for o in acumulador:
                if o == '~':
                    lista_temporal.append(premisa_negada[0])
                if o in lista_de_premisas:
                    lista_temporal.append(num_premisa[0])
                lista_temporal.append(parentesis_cadena[0])
            todas_premisas.append(argumento)
            for conjuncion in lista_temporal:
                llave = valores.get(conjuncion)
                lista_TF.append(llave)
            otroTF.append(lista_TF[1])
            lista_TF.pop(1)
            for valores_TF1 in lista_TF:

```



```

        for valores_TF2 in otroTF:
            for TF1 in valores_TF1:
                for TF2 in valores_TF2:
                    TF = not TF1 or TF2
                    premisa_final.append(TF)
                    valores_TF2.pop(0)
                    break
        valores[argumento] = premisa_final
    if premisa == '↔':
        if parentesis_cadena[0] == '':
            for p in lista_temporaln:
                if p == '~':
                    lista_temporal.append(premisa_negada[contador1])
                    lista_temporaln.pop(contador2)
                    contador1 += 1
                    contador2 += 1
                    contador4 += 1
                if p == '':
                    lista_temporal.append(todas_premisas[contador4])
                    contador2 += 1
                    contador4 += 1
        todas_premisas.append(argumento)
        for conjuncion in lista_temporal:
            llave = valores.get(conjuncion)
            lista_TF.append(llave)
        otroTF.append(lista_TF[1])
        lista_TF.pop(1)
        for valores_TF1 in lista_TF:
            for valores_TF2 in otroTF:
                for TF1 in valores_TF1:
                    for TF2 in valores_TF2:
                        TF = (not TF1 or TF2) and (not TF2 or TF1)
                        premisa_final.append(TF)
                        valores_TF2.pop(0)

```

```

break
valores[argumento] = premisa_final
if parentesis_cadena[0] != '':
    for f in lista_temporal_parentesis:
        if f == '(':
            primero_parentesis = 50
            break
        if f == '':
            primero_letra = 50
            break
    if primero_parentesis > primero_letra:
        lista_temporal.append(parentesis_cadena[contador1])
        acumulador = argumento[6:]
        for o in acumulador:
            if o == '~':
                for h in parentesis_cadena:
                    acumulador2 += h
                    while premisa_negada[contador3] in
acumulador2:
                        contador3 += 1
                        if premisa_negada[contador3] not in
acumulador2:
lista_temporal.append(premisa_negada[contador3])
        if o in lista_de_premisas:
            for h in parentesis_cadena:
                acumulador2 += h
                while num_premisa[contador3] in
acumulador2:
                    contador3 += 1
                    if num_premisa[contador3] not in
acumulador2:
lista_temporal.append(num_premisa[contador3])
        if primero_parentesis < primero_letra:
            acumulador = argumento[0:1]

```

```

        for o in acumulador:
            if o == '~':
                lista_temporal.append(premisa_negada[0])
            if o in lista_de_premisas:
                lista_temporal.append(num_premisa[0])
            lista_temporal.append(parentesis_cadena[0])
        todas_premisas.append(argumento)
        for conjuncion in lista_temporal:
            llave = valores.get(conjuncion)
            lista_TF.append(llave)
        otroTF.append(lista_TF[1])
        lista_TF.pop(1)
        for valores_TF1 in lista_TF:
            for valores_TF2 in otroTF:
                for TF1 in valores_TF1:
                    for TF2 in valores_TF2:
                        TF = (not TF1 or TF2) and (not TF2 or TF1)
                        premisa_final.append(TF)
                        valores_TF2.pop(0)
                    break
        valores[argumento] = premisa_final

```

operacion()

Se despliega en pantalla la tabla de verdad

```

def etiqueta():
    columna = 0
    fila = 1
    for d in todas_premisas:
        etiqueta_p = Label(frame_tabla, text=d, font="Calibri 16")
        etiqueta_p.grid(row=fila, column=columna)
        columna += 1
    columna = 0
    for d in todas_premisas:

```

```

        z = valores.get(d)
        if z == []:
            z = premisas_perdidas.get(d)
        columna += 1
        fila = 1
        for y in z:
            etiqueta_v1 = Label(frame_tabla, text=y, font="Calibri 16")
            etiqueta_v1.grid(row=fila, column=columna - 1)
            fila += 1

    etiqueta()

# Botones de la calculadora

    boton_borrar = Button(frame_calculadora, text="AC", width=5, height=2,
command=lambda: borrar())

    boton_del = Button(frame_calculadora, text="⌫", width=5, height=2,
command=lambda: dele())

    boton_igual = Button(frame_calculadora, text="=", width=15, height=2,
                        command=lambda: generador_tabla()) # Y aquí se hace la
tabla

    boton_parent1 = Button(frame_calculadora, text="(", width=5, height=2,
command=lambda: click_boton("("))

    boton_parent2 = Button(frame_calculadora, text=")", width=5, height=2,
command=lambda: click_boton("))"))

    botonP = Button(frame_calculadora, text="p", width=5, height=2,
command=lambda: click_boton("p"))

    botonQ = Button(frame_calculadora, text="q", width=5, height=2,
command=lambda: click_boton("q"))

    botonR = Button(frame_calculadora, text="r", width=5, height=2,
command=lambda: click_boton("r"))

    botonS = Button(frame_calculadora, text="s", width=5, height=2,
command=lambda: click_boton("s"))

    botonT = Button(frame_calculadora, text="t", width=5, height=2,
command=lambda: click_boton("t"))

```

```

    boton_no = Button(frame_calculadora, text="~", width=5, height=2,
command=lambda: click_boton("~"))

    boton_y = Button(frame_calculadora, text="^", width=5, height=2,
command=lambda: click_boton("^"))

    boton_o = Button(frame_calculadora, text="v", width=5, height=2,
command=lambda: click_boton("v"))

    boton_implicacion = Button(frame_calculadora, text="→", width=5, height=2,
command=lambda: click_boton("→"))

    boton_dobleimpl = Button(frame_calculadora, text="↔", width=5, height=2,
command=lambda: click_boton("↔"))

```

```

# Posición botones en pantalla

```

```

boton_parent1.grid(row=1, column=0, padx=5, pady=5)
boton_parent2.grid(row=1, column=1, padx=5, pady=5)
boton_del.grid(row=1, column=2, padx=5, pady=5)
boton_borrar.grid(row=1, column=3, padx=5, pady=5)

```

```

botonP.grid(row=2, column=0, padx=5, pady=5)
botonQ.grid(row=2, column=1, padx=5, pady=5)
botonR.grid(row=2, column=2, padx=5, pady=5)
botonS.grid(row=2, column=3, padx=5, pady=5)

```

```

botonT.grid(row=3, column=0, padx=5, pady=5)
boton_no.grid(row=3, column=1, padx=5, pady=5)
boton_y.grid(row=3, column=2, padx=5, pady=5)
boton_o.grid(row=3, column=3, padx=5, pady=5)

```

```

boton_implicacion.grid(row=4, column=0, padx=5, pady=5)
boton_dobleimpl.grid(row=4, column=1, padx=5, pady=5)
boton_igual.grid(row=4, column=2, colspan=2, padx=5, pady=5)

```

```

#Fin de la herramienta de Tablas de Verdad

```

```

ventana_1.wait_window()

```

```

def Calculadora_de_Conjuntos():
    ventana_2 = Toplevel()

```

```
ventana_2.title("Calculadora de Conjuntos")
```

```
frame_c1 = Frame(ventana_2)
frame_c1.grid(row=0, column=0)
frame_c2 = Frame(ventana_2)
frame_c2.grid(row=0, column=2)
```

```
# Listas globales union
```

```
AuniB = []
AuniC = []
BuniC = []
AuniBuniC = []
```

```
# Listas globales intersección
```

```
AintB = []
AintC = []
BintC = []
AintBintC = []
```

```
# Listas globales diferencia
```

```
AdifB = []
BdifA = []
AdifC = []
CdifA = []
BdifC = []
CdifB = []
```

```
# Listas globales diferenciaS
```

```
AdifSB = []
AdifSC = []
BdifSC = []
```

```
# Función para operación unión
```

```

def union():
    conjunto_A = entradatA.get() # Se obtiene la entrada de texto
    conjunto_B = entradatB.get()
    conjunto_C = entradatC.get()

    lista_A = conjunto_A.split(",") # Se hace una lista con los elementos
separados por la coma
    lista_B = conjunto_B.split(",")
    lista_C = conjunto_C.split(",")

    # A unión B
    for elemento in lista_A: # Se recorre la lista del primer conjunto y se
agregan sus elementos a la de unión global
        AuniB.append(elemento)

    for elemento in lista_B: # Se recorre la lista del segundo conjunto y, si
no estaban ya, se agregan a la global
        if elemento not in AuniB:
            AuniB.append(elemento)

    # A unión C
    for elemento in lista_A:
        AuniC.append(elemento)
    for elemento in lista_C:
        if elemento not in AuniC:
            AuniC.append(elemento)

    # B unión C
    for elemento in lista_B:
        BuniC.append(elemento)
    for elemento in lista_C:
        if elemento not in BuniC:
            BuniC.append(elemento)

    # A unión (B unión C)
    for elemento in lista_A:

```

```

        AuniBuniC.append(elemento)
    for elemento in BuniC:
        if elemento not in AuniBuniC:
            AuniBuniC.append(elemento)

# Función para operación intersección
def interseccion():
    conjunto_A = entradatA.get() # Se obtiene la entrada de texto
    conjunto_B = entradatB.get()
    conjunto_C = entradatC.get()

    lista_A = conjunto_A.split(",") # Se hace una lista con los elementos
    # separados por la coma
    lista_B = conjunto_B.split(",")
    lista_C = conjunto_C.split(",")

    # A int B
    for elemento in lista_A: # Se recorre la lista del primer conjunto
        if elemento in lista_B: # y si el elemento también está en el segundo
            AintB.append(elemento) # Se agrega a la lista de intersección
global

    # B int C
    for elemento in lista_B:
        if elemento in lista_C:
            BintC.append(elemento)

    # A int C
    for elemento in lista_A:
        if elemento in lista_C:
            AintC.append(elemento)

    # A int (B int C)
    for elemento in BintC:

```



```

        if elemento in lista_A:
            AintBintC.append(elemento)

# Función para operación diferencia
def diferencia():
    conjunto_A = entradatA.get() # Se obtiene la entrada de texto
    conjunto_B = entradatB.get()
    conjunto_C = entradatC.get()

    lista_A = conjunto_A.split(",") # Se hace una lista con los elementos
separados por la coma
    lista_B = conjunto_B.split(",")
    lista_C = conjunto_C.split(",")

    # A - B
    for elemento in lista_A: # Se recorre la lista del primer conjunto
        if elemento not in lista_B: # Si el elemento no está en la lista del
segundo conjunto
            AdifB.append(elemento) # Se agrega a la lista global de
diferencia

    # B - A
    for elemento in lista_B:
        if elemento not in lista_A:
            BdifA.append(elemento)

    # A - C
    for elemento in lista_A:
        if elemento not in lista_C:
            AdifC.append(elemento)

    # C - A
    for elemento in lista_C:
        if elemento not in lista_A:

```

```

        CdifA.append(elemento)

# B - C
for elemento in lista_B:
    if elemento not in lista_C:
        BdifC.append(elemento)

# C - B
for elemento in lista_C:
    if elemento not in lista_B:
        CdifB.append(elemento)

# Función para operación diferencia Simétrica
def diferenciaS():
    conjunto_A = entradatA.get() # Se obtiene la entrada de texto
    conjunto_B = entradatB.get()
    conjunto_C = entradatC.get()

    lista_A = conjunto_A.split(",") # Se hace una lista con los elementos
    separados por la coma
    lista_B = conjunto_B.split(",")
    lista_C = conjunto_C.split(",")

# A Δ B
for elemento in lista_A: # Se recorre la lista del primer conjunto
    if elemento not in lista_B: # Si el elemento no se encuentra en la
    lista del segundo conjunto
        AdifSB.append(elemento) # Se agrega a la lista global de
    diferencia simétrica

for elemento in lista_B: # Se recorre la lista del segundo conjunto
    if elemento not in lista_A: # Si el elemento no se encuentra en la
    lista del primer conjunto
        AdifSB.append(elemento) # Se agrega a la lista global de
    diferencia simétrica

```

```

# A Δ C
for elemento in lista_A:
    if elemento not in lista_C:
        AdifSC.append(elemento)
for elemento in lista_C:
    if elemento not in lista_A:
        AdifSC.append(elemento)

# B Δ C
for elemento in lista_B:
    if elemento not in lista_C:
        BdifSC.append(elemento)
for elemento in lista_C:
    if elemento not in lista_B:
        BdifSC.append(elemento)

# Función para imprimir valores
def hacerConjuntos():
    conjunto_A = entradatA.get() # Se obtiene la entrada de texto
    conjunto_B = entradatB.get()
    conjunto_C = entradatC.get()

    lista_A = conjunto_A.split(",") # Se hace una lista con los elementos
    separados por la coma
    lista_B = conjunto_B.split(",")
    lista_C = conjunto_C.split(",")

    # Restricción de los 10 elementos.
    if (len(lista_A) >= 10) or (len(lista_B) >= 10) or (len(lista_C) >= 10):
        messagebox.showwarning("Popup", "Una o más casillas contienen más de
10 elementos.") # Mensaje de alerta
    else:
        # Llamar a las funciones de las operaciones
        union()

```

```

interseccion()
diferencia()
diferenciaS()
# Los resultados se imprimen en una nueva ventana
top_c = Toplevel()
top_c.title("Resultados")

# Depeniendo del Checkbutton marcado, se imprimen los valores

# Unión
Label(top_c, text="Unión", bg="white").grid(column=0, row=0)
if Un_AB.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=0)
    Label(top_c, text="}").grid(column=4, row=0)
    U_AB = Label(top_c, text=AuniB)
    U_AB.grid(column=3, row=0)
    Label(top_c, text="AUB:", bg="white").grid(column=1, row=0,
sticky=W)

if Un_AC.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=1)
    Label(top_c, text="}").grid(column=4, row=1)
    U_AC = Label(top_c, text=AuniC)
    U_AC.grid(column=3, row=1)
    Label(top_c, text="AUC:", bg="white").grid(column=1, row=1,
sticky=W)

if Un_BC.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=2)
    Label(top_c, text="}").grid(column=4, row=2)
    U_BC = Label(top_c, text=BuniC)
    U_BC.grid(column=3, row=2)
    Label(top_c, text="BUC:", bg="white").grid(column=1, row=2,
sticky=W)

```

```

if Un_ABC.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=3)
    Label(top_c, text=">").grid(column=4, row=3)
    U_ABC = Label(top_c, text=AuniBuniC)
    U_ABC.grid(column=3, row=3)
    Label(top_c, text="AU(BUC):", bg="white").grid(column=1, row=3,
sticky=W)

```

Intersección

```

Label(top_c, text="Intersección", bg="white").grid(column=5, row=0)
if In_AB.get() == "Encendido":
    Label(top_c, text="{").grid(column=7, row=0)
    Label(top_c, text=">").grid(column=9, row=0)
    I_AB = Label(top_c, text=AintB)
    I_AB.grid(column=8, row=0)
    Label(top_c, text="AnB:", bg="white").grid(column=6, row=0,
sticky=W)

```

```

if In_AC.get() == "Encendido":
    Label(top_c, text="{").grid(column=7, row=1)
    Label(top_c, text=">").grid(column=9, row=1)
    I_AC = Label(top_c, text=AintC)
    I_AC.grid(column=8, row=1)
    Label(top_c, text="AnC:", bg="white").grid(column=6, row=1,
sticky=W)

```

```

if In_BC.get() == "Encendido":
    Label(top_c, text="{").grid(column=7, row=2)
    Label(top_c, text=">").grid(column=9, row=2)
    I_BC = Label(top_c, text=BintC)
    I_BC.grid(column=8, row=2)
    Label(top_c, text="BnC:", bg="white").grid(column=6, row=2,
sticky=W)

```

```

if In_ABC.get() == "Encendido":

```

```

Label(top_c, text="{").grid(column=7, row=3)
Label(top_c, text="}").grid(column=9, row=3)
I_ABC = Label(top_c, text=AintBintC)
I_ABC.grid(column=8, row=3)
Label(top_c, text="An(BnC):", bg="white").grid(column=6, row=3,
sticky=W)

```

```

# Diferencia
Label(top_c, text="Diferencia", bg="white").grid(column=0, row=5,
sticky=W)

```

```

if Di_AB.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=6)
    Label(top_c, text="}").grid(column=4, row=6)
    D_AB = Label(top_c, text=AdifB)
    D_AB.grid(column=3, row=6)
    Label(top_c, text="A-B:", bg="white").grid(column=1, row=6,
sticky=W)

```

```

if Di_BA.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=7)
    Label(top_c, text="}").grid(column=4, row=7)
    D_BA = Label(top_c, text=BdifA)
    D_BA.grid(column=3, row=7)
    Label(top_c, text="B-A:", bg="white").grid(column=1, row=7,
sticky=W)

```

```

if Di_AC.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=8)
    Label(top_c, text="}").grid(column=4, row=8)
    D_AC = Label(top_c, text=AdifC)
    D_AC.grid(column=3, row=8)
    Label(top_c, text="A-C:", bg="white").grid(column=1, row=8,
sticky=W)

```

```

if Di_CA.get() == "Encendido":
    Label(top_c, text="{").grid(column=2, row=9)

```

```

        Label(top_c, text=">").grid(column=4, row=9)
        D_CA = Label(top_c, text=CdifA)
        D_CA.grid(column=3, row=9)
        Label(top_c, text="C-A:", bg="white").grid(column=1, row=9,
sticky=W)

    if Di_BC.get() == "Encendido":
        Label(top_c, text="{").grid(column=2, row=10)
        Label(top_c, text=">").grid(column=4, row=10)
        D_BC = Label(top_c, text=BdifC)
        D_BC.grid(column=3, row=10)
        Label(top_c, text="B-C:", bg="white").grid(column=1, row=10,
sticky=W)

    if Di_CB.get() == "Encendido":
        Label(top_c, text="{").grid(column=2, row=11)
        Label(top_c, text=">").grid(column=4, row=11)
        D_CB = Label(top_c, text=CdifB)
        D_CB.grid(column=3, row=11)
        Label(top_c, text="C-B:", bg="white").grid(column=1, row=11,
sticky=W)

    # Diferencia Simétrica
    Label(top_c, text=" ").grid(column=4, row=4)
    Label(top_c, text="Diferencia Simétrica", bg="white").grid(column=5,
row=5, sticky=W)

    if DiS_AB.get() == "Encendido":
        Label(top_c, text="{").grid(column=7, row=6)
        Label(top_c, text=">").grid(column=9, row=6)
        Ds_AB = Label(top_c, text=AdifSB)
        Ds_AB.grid(column=8, row=6)
        Label(top_c, text="AΔB:", bg="white").grid(column=6, row=6,
sticky=W)

    if DiS_AC.get() == "Encendido":
        Label(top_c, text="{").grid(column=7, row=7)

```

```

Label(top_c, text="}").grid(column=9, row=7)
Ds_AC = Label(top_c, text=AdifSC)
Ds_AC.grid(column=8, row=7)
Label(top_c, text="AAC:", bg="white").grid(column=6, row=7,
sticky=W)

```

```

if DiS_BC.get() == "Encendido":
    Label(top_c, text="{").grid(column=7, row=8)
    Label(top_c, text="}").grid(column=9, row=8)
    Ds_BC = Label(top_c, text=BdifSC)
    Ds_BC.grid(column=8, row=8)
    Label(top_c, text="BAC:", bg="white").grid(column=6, row=8,
sticky=W)

```

Cardinalidad

```

Lista_A_limpia = []
Lista_B_limpia = []
Lista_C_limpia = []
for i in lista_A:
    if i not in Lista_A_limpia:
        Lista_A_limpia.append(i)
for i in lista_B:
    if i not in Lista_B_limpia:
        Lista_B_limpia.append(i)
for i in lista_C:
    if i not in Lista_C_limpia:
        Lista_C_limpia.append(i)

```

```

Label(top_c, text="Cardinalidad A:", bg="white").grid(column=6, row=10)
Label(top_c, text=len(Lista_A_limpia)).grid(column=7, row=10)

```

```

Label(top_c, text="Cardinalidad B:", bg="white").grid(column=6, row=11)
Label(top_c, text=len(Lista_B_limpia)).grid(column=7, row=11)

```

```

Label(top_c, text="Cardinalidad C:", bg="white").grid(column=6, row=12)

```



```

Label(top_c, text=len(Lista_C_limpia)).grid(column=7, row=12)

# Vaciar listas (para que no se tenga que reiniciar el programa)
# Unión
AuniB.clear()
AuniC.clear()
BuniC.clear()
AuniBuniC.clear()
# Intersección
AintB.clear()
AintC.clear()
BintC.clear()
AintBintC.clear()
# Diferencia
AdifB.clear()
BdifA.clear()
AdifC.clear()
CdifA.clear()
BdifC.clear()
CdifB.clear()
# Diferencia simétrica
AdifSB.clear()
AdifSC.clear()
BdifSC.clear()
top_c.wait_window()

```

```

# Estado Botones (Encendido o Apagado)
# Unión
Un_AB = StringVar()
Un_AC = StringVar()
Un_BC = StringVar()
Un_ABC = StringVar()
# Intersección

```

```

In_AB = StringVar()
In_BC = StringVar()
In_AC = StringVar()
In_ABC = StringVar()

# Diferencia
Di_AB = StringVar()
Di_BA = StringVar()
Di_AC = StringVar()
Di_CA = StringVar()
Di_BC = StringVar()
Di_CB = StringVar()

# Diferencia Simétrica
DiS_AB = StringVar()
DiS_AC = StringVar()
DiS_BC = StringVar()


# Instrucciones en pantalla
Label(frame_c1, text=" ").grid(pady=5, row=0, column=0)

Label(frame_c1, text="Instrucciones: Separar por comas sin espacios los
elementos ingresados.").grid \
    (pady=5, row=0, column=1, columnspan=10, sticky=N)

Label(frame_c1, text="Ingresar máximo 10 elementos.").grid \
    (pady=5, row=1, column=1, columnspan=10, sticky=N)


# Cuadros de entrada de la calculadora
Label(frame_c1, text="A = {").grid(pady=5, row=2, column=2)
entradatA = Entry(frame_c1, font=("Calibri 12"))
entradatA.grid(row=2, column=3, columnspan=5, padx=5, pady=5)
Label(frame_c1, text="}").grid(pady=5, row=2, column=8)


Label(frame_c1, text="B = {").grid(pady=5, row=3, column=2)
entradatB = Entry(frame_c1, font=("Calibri 12"))
entradatB.grid(row=3, column=3, columnspan=5, padx=5, pady=5)
Label(frame_c1, text="}").grid(pady=5, row=3, column=8)

```

```

Label(frame_c1, text="C = {").grid(pady=5, row=4, column=2)
entradatC = Entry(frame_c1, font=("Calibri 12"))
entradatC.grid(row=4, column=3, columnspan=5, padx=5, pady=5)
Label(frame_c1, text="}").grid(pady=5, row=4, column=8)

# Espacio estético
espacio = Label(frame_c1, text=" ").grid(row=2, column=11)

# Operaciones a elegir
operaciones = Label(frame_c2, text="Seleccionar operaciones", bg="white")
operaciones.grid(column=2, row=0, rowspan=2, sticky=N)

# Unión (Check Buttons)
_union = Label(frame_c2, text="Unión", bg="white")
_union.grid(column=1, row=2, sticky=W)

_A_B = Checkbutton(frame_c2, text="AUB", bg="white", variable=Un_AB,
onvalue='Encendido', offvalue='Apagado')
_A_B.deselect()
Label(frame_c2, text=" ", bg="white").grid(column=1, row=4, sticky=NSEW)
_A_B.grid(column=1, row=3, sticky=EW)

_A_C = Checkbutton(frame_c2, text="AUC", bg="white", variable=Un_AC,
onvalue='Encendido', offvalue='Apagado')
_A_C.grid(column=2, row=3, sticky=EW)
_A_C.deselect()

_B_C = Checkbutton(frame_c2, text="BUC", bg="white", variable=Un_BC,
onvalue='Encendido', offvalue='Apagado')
_B_C.deselect()
_B_C.grid(column=3, row=3, sticky=EW)

_A_B_C = Checkbutton(frame_c2, text="AU(BUC)", bg="white", variable=Un_ABC,
onvalue='Encendido', offvalue='Apagado')
_A_B_C.grid(column=2, row=4, sticky=EW)
_A_B_C.deselect()
Label(frame_c2, text=" ", bg="white").grid(column=3, row=4, sticky=NSEW)
Label(frame_c2, text=" ").grid(column=1, row=5)

```

```

# Intersección (Check Buttons)

_interseccion = Label(frame_c2, text="Intersección", bg="white")
_interseccion.grid(column=1, row=6, sticky=W)

_AintB = Checkbutton(frame_c2, text="AnB", bg="white", variable=In_AB,
onvalue='Encendido', offvalue='Apagado')

_AintB.deselect()

Label(frame_c2, text=" ", bg="white").grid(column=1, row=8, sticky=NSEW)

_AintB.grid(column=1, row=7, sticky=EW)

_BintC = Checkbutton(frame_c2, text="BnC", bg="white", variable=In_BC,
onvalue='Encendido', offvalue='Apagado')

_BintC.deselect()

_BintC.grid(column=2, row=7, sticky=EW)

_AintC = Checkbutton(frame_c2, text="AnC", bg="white", variable=In_AC,
onvalue='Encendido', offvalue='Apagado')

_AintC.deselect()

_AintC.grid(column=3, row=7, sticky=EW)

_AintBintC = Checkbutton(frame_c2, text="An(BnC)", variable=In_ABC,
bg="white", onvalue='Encendido', offvalue='Apagado')

_AintBintC.deselect()

_AintBintC.grid(column=2, row=8, sticky=EW)

Label(frame_c2, text=" ", bg="white").grid(column=3, row=8, sticky=NSEW)

Label(frame_c2, text=" ").grid(column=1, row=9)


# Diferencia (Check Buttons)

_diferencia = Label(frame_c2, text="Diferencia", bg="white", )

_diferencia.grid(column=1, row=10, sticky=W)

_AdifB = Checkbutton(frame_c2, text="A-B", bg="white", variable=Di_AB,
onvalue='Encendido', offvalue='Apagado')

_AdifB.deselect()

_AdifB.grid(column=1, row=11, sticky=NSEW)

_BdifA = Checkbutton(frame_c2, text="B-A", bg="white", variable=Di_BA,
onvalue='Encendido', offvalue='Apagado')

_BdifA.deselect()

_BdifA.grid(column=2, row=11, sticky=NSEW)

```

```

        _AdifC = Checkbutton(frame_c2, text="A-C", bg="white", variable=Di_AC,
onvalue='Encendido', offvalue='Apagado')

        _AdifC.deselect()

        _AdifC.grid(column=1, row=12, sticky=NSEW)

        _CdifA = Checkbutton(frame_c2, text="C-A", bg="white", variable=Di_CA,
onvalue='Encendido', offvalue='Apagado')

        _CdifA.deselect()

        _CdifA.grid(column=2, row=12, sticky=NSEW)

        _BdifC = Checkbutton(frame_c2, text="B-C", bg="white", variable=Di_BC,
onvalue='Encendido', offvalue='Apagado')

        _BdifC.deselect()

        _BdifC.grid(column=1, row=13, sticky=NSEW)

        _CdifB = Checkbutton(frame_c2, text="C-B", bg="white", variable=Di_CB,
onvalue='Encendido', offvalue='Apagado')

        _CdifB.deselect()

        _CdifB.grid(column=2, row=13, sticky=NSEW)

        Label(frame_c2, text=" ").grid(column=1, row=14)


# Diferencia Simétrica (Check Buttons)

        _diferenciaS = Label(frame_c2, text="Diferencia Simétrica", bg="white")

        _diferenciaS.grid(column=1, row=15, sticky=W, columnspan=2)

        _AdifsB = Checkbutton(frame_c2, text="AΔB", bg="white", variable=DiS_AB,
onvalue='Encendido', offvalue='Apagado')

        _AdifsB.deselect()

        _AdifsB.grid(column=1, row=16, sticky=EW)

        _AdifsC = Checkbutton(frame_c2, text="AΔC", bg="white", variable=DiS_AC,
onvalue='Encendido', offvalue='Apagado')

        _AdifsC.deselect()

        _AdifsC.grid(column=2, row=16, sticky=EW)

        _BdifsC = Checkbutton(frame_c2, text="BΔC", bg="white", variable=DiS_BC,
onvalue='Encendido', offvalue='Apagado')

        _BdifsC.deselect()

        _BdifsC.grid(column=3, row=16, sticky=W)

        Label(frame_c2, text=" ").grid(column=4, row=14)


# Botón para iniciar

```

```

Label(frame_c2, text=" ").grid(column=1, row=17)

hacer_conjuntos = Button(frame_c2, text="Hacer Conjuntos",
command=hacerConjuntos)

hacer_conjuntos.grid(pady=5, column=2, row=19)

#Fin de la herramienta de Conjuntos
ventana_2.wait_window()

def Sucesiones():
    ventana_3 = Toplevel()
    ventana_3.title("Sucesiones")

    # Lista global para operaciones.
    Operaciones = []

    # Función para hacer sucesiones.
    def Sucesion():
        top_s = Toplevel() # Ventana nueva
        top_s.title("Resultados") # Título de la nueva ventana

        lim_sup_s = entradatA.get() # Obtiene la entrada de texto para el límite
superior
        lim_sup = int(lim_sup_s) # Se transforman a enteros
        lim_inf_s = entradatB.get() # Obtiene la entrada de texto para el límite
inferior
        lim_inf = int(lim_inf_s) # Se transforman a enteros
        formula = entradatC.get() # Obtiene la entrada de texto para la fórmula
general

        # Se hace la sucesión con un clico for
        for k in range(lim_inf, lim_sup + 1): # Va desde el límite inferior al
superior +1
            f = eval(formula) # eval() permite leer un string como fórmula
matemática
            texto = str(f) # Convierte el término en un string

```

```

# Etiquetas en la nueva ventana
Label(top_s, text=f"Término {k}: ").grid(row=k + 3, column=2)
Label(top_s, text=texto).grid(row=k + 3, column=3)

# Agrega los términos de la sucesión a una Lista
Operaciones.append(eval(formula))

# Acumuladores que sirven para hacer las operaciones de suma y
multiplicación.
acumulador_1 = 0
acumulador_2 = 1

for i in Operaciones: # Realiza la sumatoria y multiplicación de los
elementos de la lista Operaciones.
    acumulador_1 += i
    acumulador_2 *= i

# Etiquetas de las respuestas en la nueva ventana
Label(top_s, text=f"Sumatoria = {acumulador_1}", bg="white").grid(row=0,
column=2, columnspan=2)

Label(top_s, text=f"Multiplicación = {acumulador_2}",
bg="white").grid(row=1, column=2, columnspan=2)

# Espacio estético
Label(top_s, text=" ").grid(row=2, column=0)

# Vaciar listas (para que no se tenga que reiniciar el programa)
Operaciones.clear()

# Bluece para la nueva ventana
top_s.wait_window()

# Instrucciones
Label(ventana_3, text="Instrucciones: Use operadores básicos (+, -, *, /) y,
si es necesario, \nutilice paréntesis")

```

```
        " para escribir la fórmula general.").grid(pady=5, row=0,  
column=1, columnspan=10, sticky=N)
```

```
# Espacios estéticos
```

```
Label(ventana_3, text=" ").grid(row=0, column=0)
```

```
Label(ventana_3, text=" ").grid(row=1, column=0)
```

```
Label(ventana_3, text=" ").grid(row=0, column=12)
```

```
# Cuadros de entrada de la calculadora
```

```
Label(ventana_3, text="Límite Superior: ").grid \  
(row=2, column=2)
```

```
Label(ventana_3, text="Límite Inferior: ").grid \  
(row=3, column=2)
```

```
Label(ventana_3, text="Fórmula (k): ").grid \  
(row=4, column=2)
```

```
# Cuadros de entrada de la calculadora
```

```
entradatA = Entry(ventana_3, font=("Calibri 12"))
```

```
entradatA.grid(row=2, column=3)
```

```
entradatB = Entry(ventana_3, font=("Calibri 12"))
```

```
entradatB.grid(row=3, column=3)
```

```
entradatC = Entry(ventana_3, font=("Calibri 12"))
```

```
entradatC.grid(row=4, column=3)
```

```
# Botón para iniciar
```

```
generar = Button(ventana_3, text="Generar", command=Sucesion)
```

```
generar.grid(pady=5, column=3, row=5)
```

```
#Fin de la herramienta de Sucesiones
```

```
ventana_3.wait_window()
```



```

# Función para iniciar la herramienta de relaciones y funciones
def Relaciones_y_Funciones():
    # Nueva
    ventana_4 = Toplevel()
    ventana_4.title("Relaciones y Funciones")

    frame_R_F = Frame(ventana_4)
    frame_R_F.grid(row=0, column=0)

    # Listas globales
    lista_de_dominio = []
    lista_de_codominio = []
    listas_de_relaciones = []
    lista_entrada = []
    lista_salida = []

    # Función para verificar propiedades relaciones
    def Relaciones():
        relacion = parejas.get() # Se obtiene la entrada

        # Se recorre la entrada para obtener solo los valores numéricos
        for i in relacion:
            if i.isnumeric():
                listas_de_relaciones.append(i)

        # Se crean las listas de dominio y codominio, se detecta si es función o
        relación

        contador = 0
        bandera_funcion = "Verdadero" # Bandera valor inicial

        # Se recorre la lista, primero analizando los valores para el dominio
        while contador < len(listas_de_relaciones):
            if contador % 2 == 0:
                # Agrega los valores a la lista de dominio

```

```

        if listas_de_relaciones[contador] not in lista_de_dominio:
            lista_de_dominio.append(listas_de_relaciones[contador])
            lista_entrada.append(listas_de_relaciones[contador])
        else:
            lista_entrada.append(listas_de_relaciones[contador])

        # Cambia el valor de la bandera si el dominio tiene relacionado
        # más de un valor en el codominio
        bandera_funcion = "Falso"

        # Se recorre la lista, ahora analizando los valores para el codominio
    elif contador % 2 != 0:
        if listas_de_relaciones[contador] not in lista_de_codominio:
            lista_de_codominio.append(listas_de_relaciones[contador])
            lista_salida.append(listas_de_relaciones[contador])
        else:
            lista_salida.append(listas_de_relaciones[contador])

    contador += 1

# Reflexividad:
bandera_reflexividad = "Verdadero" # Bandera valor inicial
lista_de_reflexividad = relacion.split(" ")
contador = 0
lista_temporal = []
# Se guardan valores del dominio en una lista temporal
while contador < len(lista_de_dominio):
    conjunto =
    f"({lista_de_dominio[contador]}, {lista_de_dominio[contador]})"
    lista_temporal.append(conjunto)
    contador += 1

# Se recorre la lista de entrada
for i in lista_temporal:
    # si los valores del dominio no están en la lista la bandera
    if i not in lista_de_reflexividad:
        bandera_reflexividad = "Falso"

```

```

# Simetría:
bandera_simetria = "Verdadero" # Bandera valor inicial
contador = 0
lista_temporal = []
# Se agregan valores de salida y entrada a una lista temporal
while contador < len(lista_entrada):
    conjunto = f"({lista_salida[contador]},{lista_entrada[contador]})"
    lista_temporal.append(conjunto)
    contador += 1
# Se recorren los valores de la lista temporal
for i in lista_temporal:
    # Cambia valor de bandera si no están en la lista de reflexividad
    if i not in lista_de_reflexividad:
        bandera_simetria = "Falso"

# Transitividad:
bandera_transitividad = "Verdadero" # Bandera valor inicial
contador = 0
lista_temporal = []
a = 0
b = 0
c = 0
# Se recorre la lista de entrada
while contador < len(lista_entrada):
    a = lista_entrada[contador]
    b = lista_salida[contador]
    contador1 = 0
    # Se recorre con un nuevo contador
    while contador1 < len(lista_entrada):
        if b == lista_entrada[contador1]:
            c = lista_salida[contador1]
            conjunto = f"({a},{c})"
            lista_temporal.append(conjunto)
        contador1 += 1

```

```

        contador += 1

    # Se cambia el valor de la bandera si no se encuentra en la lista de
    reflexividad

    for i in lista_temporal:

        if i not in lista_de_reflexividad:

            bandera_transitividad = "Falso"

    lista_de_dominio.sort()
    lista_de_codominio.sort()

    # Imprimir resultados en nueva ventana
    top_r_f = Toplevel()
    top_r_f.title("Resultados")

    # Espacio estético
    Label(top_r_f, text=" ").grid(row=0, column=0)

    # Espacio estético
    Label(top_r_f, text=" ").grid(row=0, column=5)

    # Dominio
    Label(top_r_f, text="Dominio (x) =", bg="white").grid(row=1, column=1,
    sticky=EW)
    Label(top_r_f, text="{").grid(row=1, column=2)
    Label(top_r_f, text=lista_de_dominio).grid(row=1, column=3)
    Label(top_r_f, text="}").grid(row=1, column=4)

    # Espacio estético
    Label(top_r_f, text=" ", bg="white").grid(row=2, column=1, sticky=EW)

    # Codominio
    Label(top_r_f, text="Codominio (y) =", bg="white").grid(row=3, column=1,
    sticky=EW)
    Label(top_r_f, text="{").grid(row=3, column=2)
    Label(top_r_f, text=lista_de_codominio).grid(row=3, column=3)

```

```

Label(top_r_f, text="}").grid(row=3, column=4)

# Espacio estético
Label(top_r_f, text=" ", bg="white").grid(row=4, column=1, sticky=EW)

# Función
Label(top_r_f, text="Función: ", bg="white").grid(row=5, column=1,
sticky=EW)
Label(top_r_f, text=bandera_funcion).grid(row=5, column=3)

# Espacio estético
Label(top_r_f, text=" ", bg="white").grid(row=6, column=1, sticky=EW)

# Reflexividad
Label(top_r_f, text="Reflexividad: ", bg="white").grid(row=7, column=1,
sticky=EW)
Label(top_r_f, text=bandera_reflexividad).grid(row=7, column=3)

# Espacio estético
Label(top_r_f, text=" ", bg="white").grid(row=8, column=1, sticky=EW)

# Simetría
Label(top_r_f, text="Simetría: ", bg="white").grid(row=9, column=1,
sticky=EW)
Label(top_r_f, text=bandera_simetria).grid(row=9, column=3)

# Espacio estético
Label(top_r_f, text=" ", bg="white").grid(row=10, column=1, sticky=EW)

# Transitividad
Label(top_r_f, text="Transitividad: ", bg="white").grid(row=11, column=1,
sticky=EW)
Label(top_r_f, text=bandera_transitividad).grid(row=11, column=3)

# Espacio estético

```

```

Label(top_r_f, text=" ").grid(row=12, column=0)

# Vaciar listas (para que no se tenga que reiniciar el programa)
lista_de_dominio.clear()
lista_de_codominio.clear()
listas_de_relaciones.clear()
lista_entrada.clear()
lista_salida.clear()

top_r_f.wait_window() # Bucle de la ventana

# Instrucciones
Label(ventana_4,
      text="Instrucciones: Ingresa la serie de parejas ordenadas dentro de
paréntesis\ny separadas por un espacio.").grid\
      (pady=5, row=0, column=0, columnspan=10, sticky=N)
Label(ventana_4, text="Ejemplo: {(0,0) (1,1) (2,2),...}").grid \
      (pady=5, row=1, column=0, columnspan=10, sticky=N)

# Espacio estético
Label(ventana_4, text=" ").grid(row=1, column=0)

# Cuadros de entrada de la calculadora
Label(ventana_4, text="Parejas (R) = {").grid \
      (row=2, column=0)
parejas = Entry(ventana_4, font=("Calibri 12"), width=45)
parejas.grid(row=2, column=1)
Label(ventana_4, text="}").grid \
      (row=2, column=2)

# Espacio estético
espacio = Label(ventana_4, text=" ").grid(row=2, column=11)

```

```

# Botón para iniciar

relaciones = Button(ventana_4, text="Analizar", command=Relaciones)
relaciones.grid(pady=5, column=1, row=5)


#Fin de la herramienta de Relaciones y funciones
ventana_4.wait_window()


# Interfaz del menú principal
Label(ventana, text="Caja de Herramientas", font=("Calibri 20 bold")).grid(row=0,
column=0, sticky=NW)

# Botón para usar la herramienta de tablas de verdad
Button(ventana, text="Tablas de Verdad", font=("Calibri 11 bold"),
command=Tablas_de_Verdad).grid(row=1, column=0)

# Botón para usar la herramienta de calculadora de conjuntos
Button(ventana, text="Calculadora de Conjuntos", font=("Calibri 11 bold"),
command=Calculadora_de_Conjuntos).grid(row=2, column=0)

# Botón para usar la herramienta de sucesiones
Button(ventana, text="Sucesiones", font=("Calibri 11 bold"),
command=Sucesiones).grid(row=3, column=0)

# Botón para usar la herramienta de relaciones y funciones
Button(ventana, text="Relaciones y Funciones", font=("Calibri 11 bold"),
command=Relaciones_y_Funciones).grid(row=4, column=0)


# Bucle de la aplicación
ventana.mainloop()

```

Relación de la materia con la carrera

Como podemos comprobar, la materia de Fundamentos de Ciencias Computacionales es una que cursan las personas que estudian algo relacionado con TI (Electrónica, Redes y Seguridad, Sistemas, etc.) por una razón justificable. Se suele tomar en los primeros semestres porque da mayor entendimiento a los procesos y toma de decisiones de las máquinas, de modo que, FCC es una asignatura más que introductoria.

La razón de ser de esta materia es presentar los conceptos básicos o 'pilares fundamentales' que se volverán la base de los conocimientos que adquiera el profesionista enfocado al TI. Preguntar sobre en que se utilizan los temas abordados en la materia, es hablar sobre las bases de la tecnología. Por ejemplo, como se explicó en el documento de la herramienta de Sucesiones existen series como la serie de Fourier que describe los comportamientos que llevará a cabo la transmisión de señales por un medio.

Cada tema en particular del curso tiene su uso en la implementación de la tecnología computacional o la creación de nuevos algoritmos para la resolución de problemas específicos. Por ende, quitar estos conocimientos o no conocerlos es lo mismo que no otorgar bases a los ingenieros para el desarrollo y la aplicación de su campo de estudio.

Bibliografía

pildorasinformaticas. (12 de febrero de 2019). *Curso Python. Generar ejecutables. Fin de curso. Vídeo 78 [Video]*. Obtenido de YouTube: <https://www.youtube.com/watch?v=Vr9vI0qlggE&t>