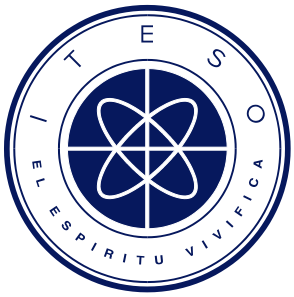
INSTITUTO TECNOLÓGICO Y DE ESTUDIOS SUPERIORES DE OCCIDENTE

FUNDAMENTOS DE CIENCIAS COMPUTACIONALES



CALCULADORA DE TABLAS DE VERDAD

Presentan:

Ana Carolina Arellano Valdez

Luis Raúl Acosta Mendoza

Arlyn Linette Medina García

Profesora: Silvia Piña Romero

Fecha 1/03/2022

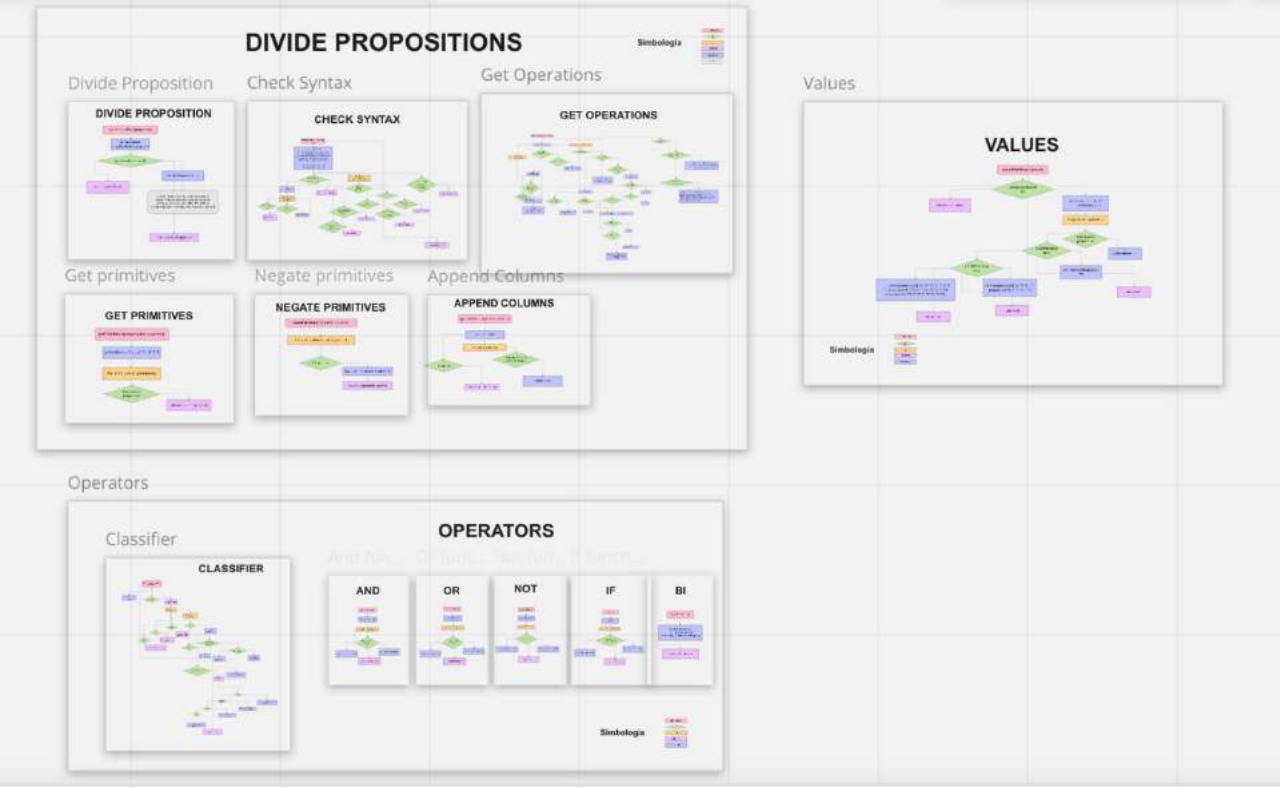
Contexto

Tablas de verdad: Significado, función y aplicaciones

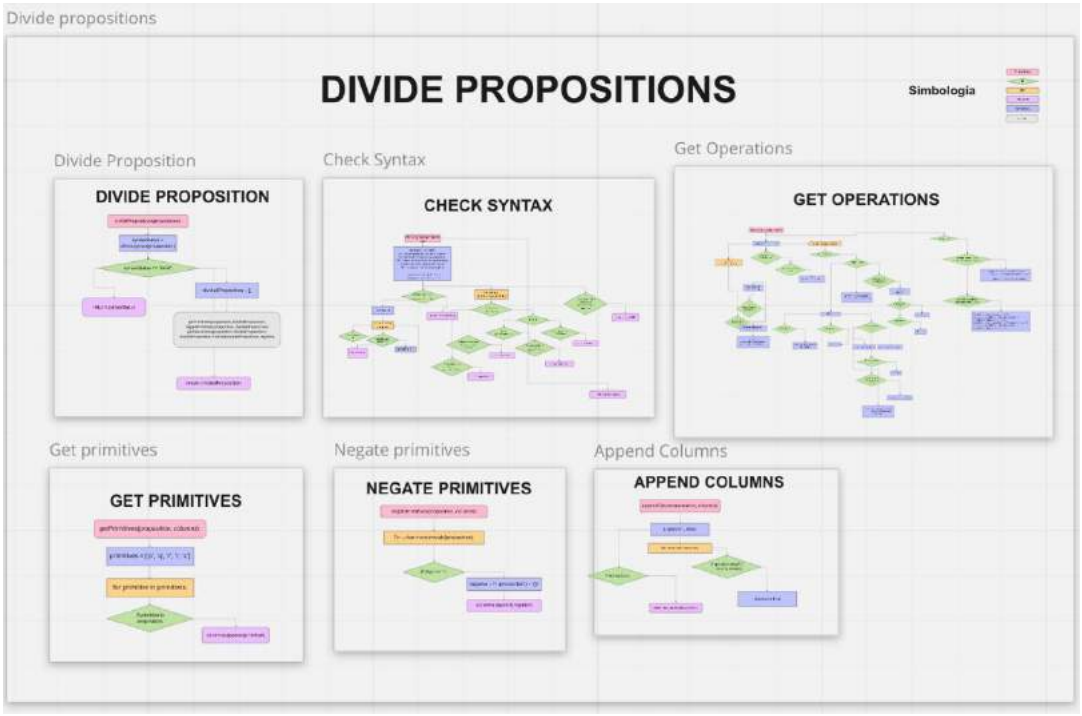
Las tablas de verdad son una estrategia usada en la lógica simple con la que se puede determinar la validez de uno o varias proposiciones dentro de ciertas condiciones llamados operadores lógicos. Estas mismas tienen el funcionamiento de permitirnos analizar los valores de verdad y falso de cualquier enunciado de manera sencilla y visual. Las aplicaciones que puede llegar a tener las tablas de verdad puede ser tan simple como encontrar la solución de un problema matemático hasta encontrar cuál es una de las soluciones más optimas para que un programa funcione de manera efectiva y eficaz. Algunos ejemplos de en que se puede usar las tablas de verdad en la vida laboral son en el cálculo lógico, en la construcción de aparatos computacionales incluyendo circuitos y en el desarrollo de algoritmos de programación.

Explicación técnica de la programación

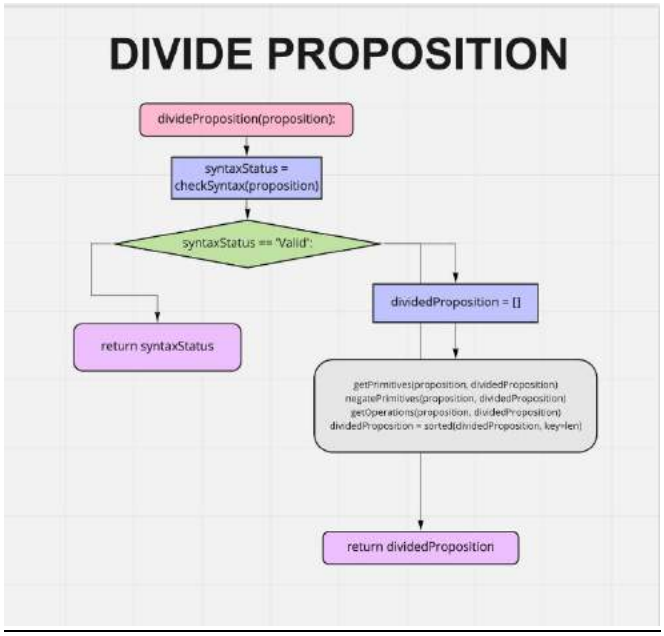
- Nuestra calculadora lógica se basa en tres principales componentes los cuales son la división de proposiciones, los valores y los operadores, más adelante habrá una explicación detallada de cada una de estas.
- Link para ver diagramas de flujo con más claridad: https://miro.com/welcomeonboard/a1E3UGFubkZmbVNvV3VyVTdTR21iMlVBblhNNWFqVjlhdk1TbUNhMmVRTWlqSVRxYWp4OGhFU0c5VVpKV1hXTXwzMDc0NDU3MzY3MTE0NDcwMzUw?invite_link_id=141100425357



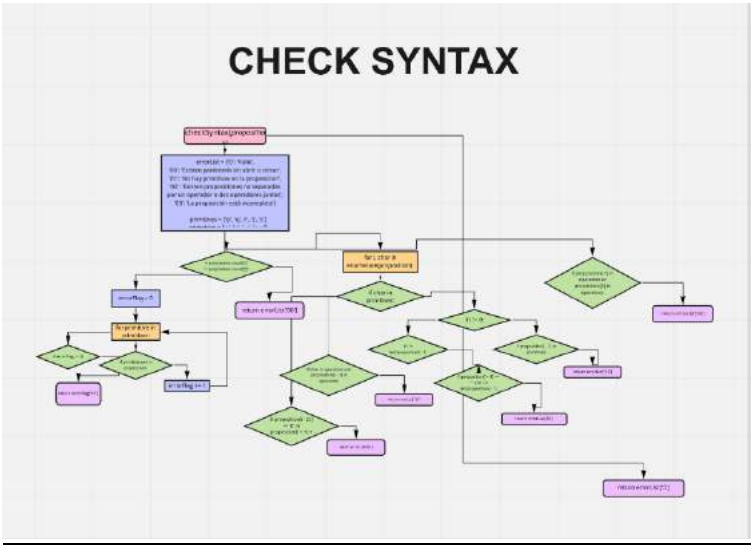
1. Primero se empieza con la **división de proposiciones**. En esta misma se analiza el valor ingresado por el usuario en forma de cadena, para después checar algún error que pueda haber en la sintaxis y si no, dividir sus diferentes combinaciones de proposiciones, siguiendo el orden lógico e irlo agregando a una lista.



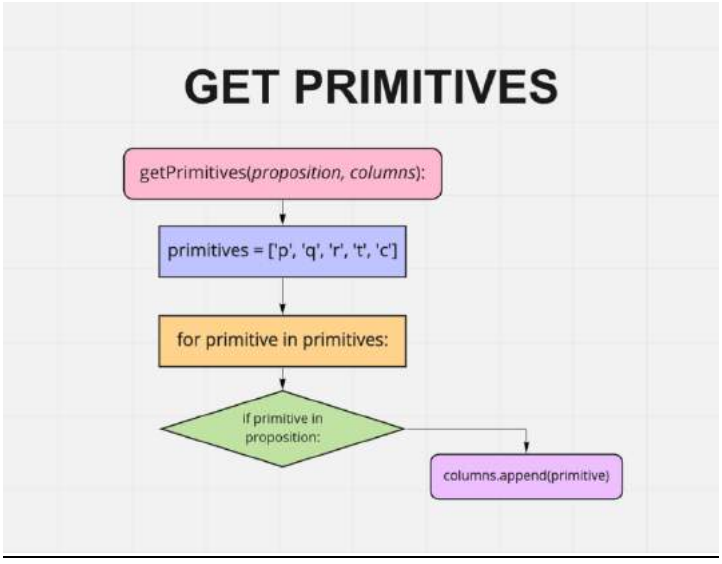
En esta función, se tiene como entrada el valor ingresado por el usuario en forma de cadena. Al comprobar que no haya ningún error de sintaxis por medio de otra función llamada checkSyntax, se regresará que el valor es valido y se permitirá que se haga un análisis de cada operador y proposición para poder dividirlas y agregarlas a una lista con el comando append.

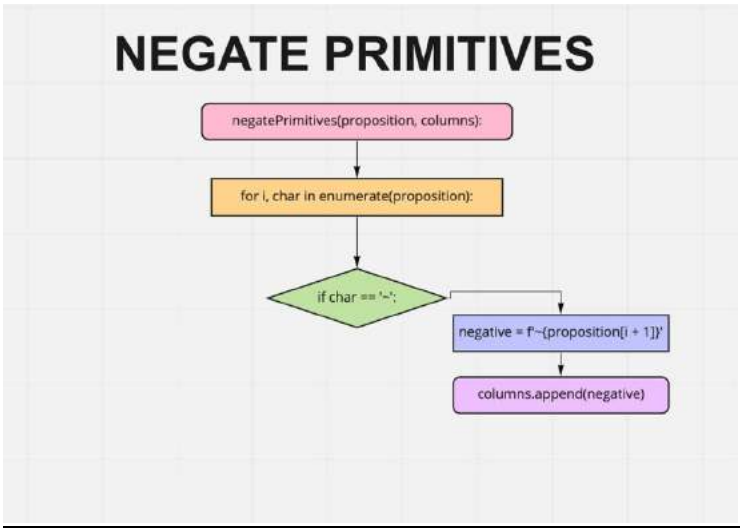


Como lo dice su nombre, esta función se encarga de revisar que el valor ingresado por el usuario no está incorrecto/incompleto. Se concentra en 3 principales errores. El primero es que no haya paréntesis sin cerrar el cual se checa al verificar que haya el mismo número de “(” y “)” con el comando find. El segundo error es que no haya primitivas en la proposición dada, esta fácilmente se puede ubicar con el comando “not in” que checa si no hay primitivos en la proposición. El tercer error se concentra en que no haya operadores lógicos entre proposiciones. Esta se arregla al primero ubicar la posición de cada primitiva y después verificar que la posición anterior y posterior no sea primitivo. De no ser cualquiera de estos casos, el programa devuelve ‘valido’ y permite que se continúe.

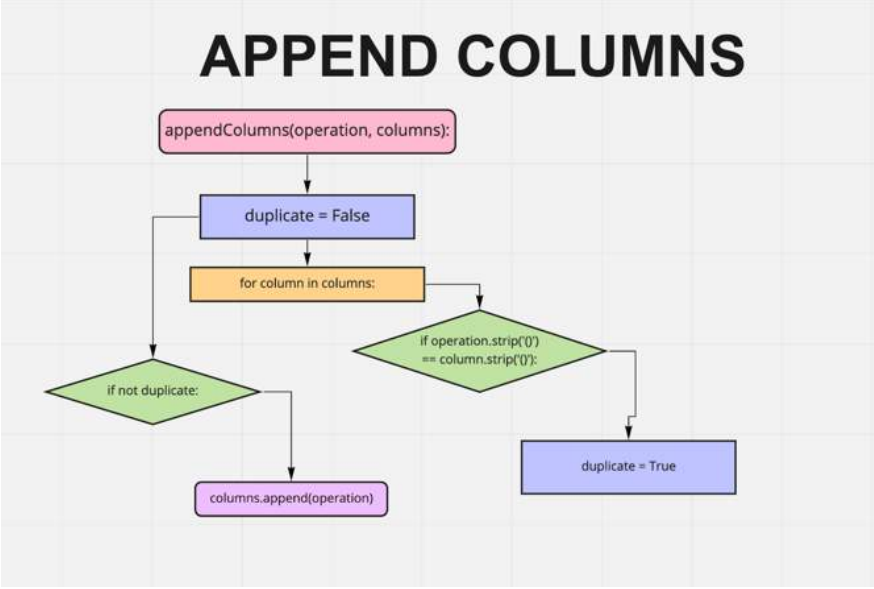


Estas tres funciones se concentran en analizar y obtener los primitivos tanto positivos como negativos, así como los operadores lógicos establecidos por el usuario. En este caso primero se analizan los valores primitivos en la cadena para después agregarla en la lista. Después se analiza si hay algún primitivo negativo para después agregarlo a la misma lista. Por último, está el de los operadores que primero checa si hay algún paréntesis para dar prioridad a esos valores y después va examinando por el orden lógico de los operadores para irlo agregando a la lista con la función appendColumns.

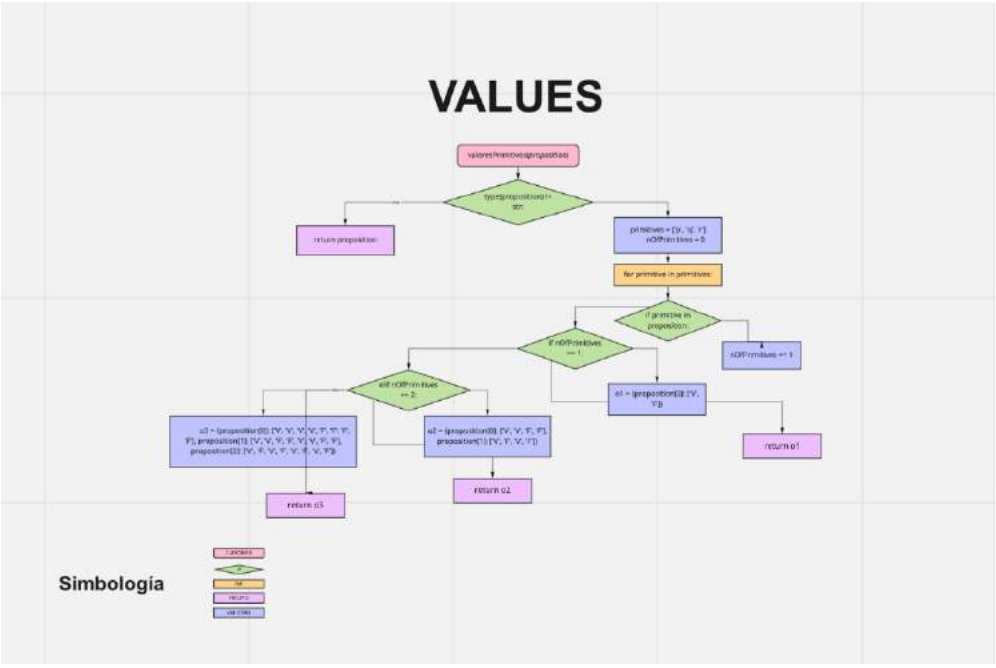




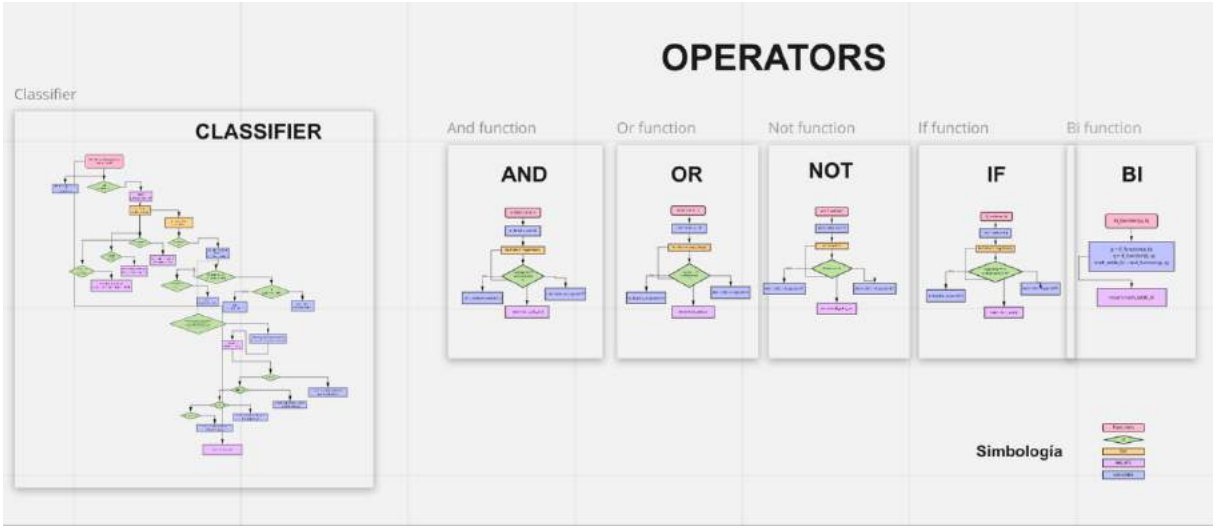
Por último, esta función se encarga de agregar cada valor identificado por los anteriores a la lista que posteriormente se va a utilizar para empezar con las tablas de verdad. Esta función se asegura de que no haya ningún valor repetido dentro de la misma lista.



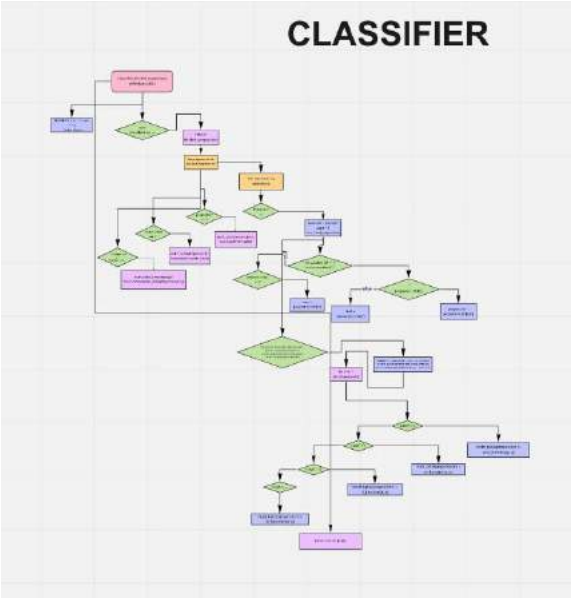
2. En segunda instancia, el programa se enfoca en en empezar a darle más estructura a la lista previamente creada definiendo los primeros **valores de verdad y falso** de cada primitivo ingresado por el usuario. Como podemos ver, la función empieza por encontrar los primitivos individuales en la lista previamente creada. Al haber un solo primitivo, se agrega en un diccionario el valor del primitivo con sus respectivos valores de verdad y falso que son sacados de una lista previamente establecida. Al haber dos primitivos se vuelve a agregar ambos primitivos en el diccionario como keys, y sus respectivos valores de verdadero y falso como sus values. (Se realiza lo mismo al haber tres primitivos)



3. La ultima parte de nuestro programa, se enfoca en identificar y **clasificar los operadores** que existen en cada proposición, incluyendo los parentesis, para poder dar prioridad en la tabla de verdad y poder sacar los correspondientes valores de verdad o falso. Al final de esta parte se termina con un diccionario que contiene todos los valores de verdadero y falso de la proposición ingresada por el usuario. Ya solamente este diccionario se imprime con formato de tabla en la pantalla del usuario y se obtiene el resultado esperado.

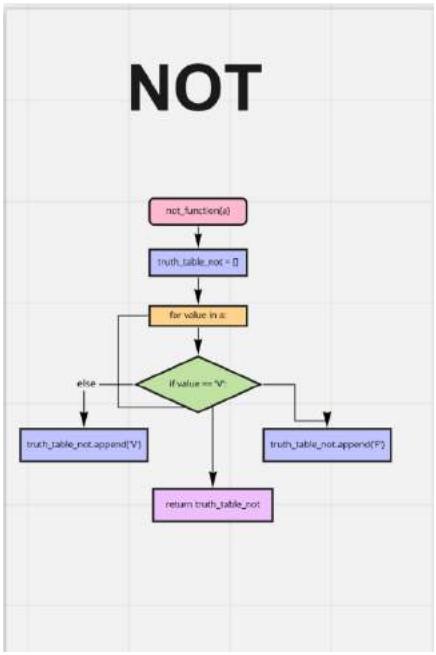


En esta función, primeramente se busca si hay alguna tautología, contradicción o negación por separado para poner sus valores de verdad primero, dependiendo completamente de los valores primitivos que se tenían desde la lista anterior. Después se empieza a enfocar en los operadores lógicos en los que primero se revisa si hay un paréntesis para dar prioridad. Después se revisa los operadores '∧', '∨', '→', '↔' en ese respectivo orden. Cuando identifica cada uno de estos operadores, se va a las diferentes funciones especificas que posteriormente estarán explicadas para definir sus valores de verdad y agregarlos al diccionario para despues ser impresos en la pantalla del usuario en formato de tabla.



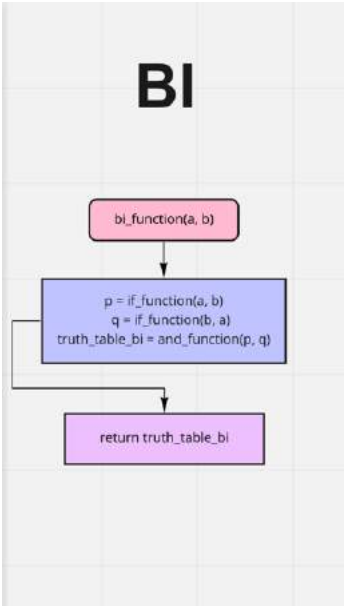
Estas funciones están principalmente basadas en un ciclo que recorre los valores primitivos que están asignados en las proposiciones que cuentan con un signo 'Λ' o 'V'. En ambos casos se recorre los valores de verdadero y falso de los primitivos de la proposición del diccionario previamente hecho. En el caso de and, se usa una condicional en donde solamente el valor resultante será verdadero si ambos parámetros son verdaderos. En el caso de or, solamente será falso si ambos parámetros son falsos. Cada valor de verdadero y falso se va agregando a una lista que después es enviada a la función de clasificación para agregarlo al diccionario con la proposición como el key y la lista de verdaderos y falsos como el value.

La función not, sirve para cuando hay una primitiva negativa. En esta se identifica la primitiva de la lista anteriormente creada y con un ciclo for, se recorre los valores de verdadero y falso de cada una de esta. Lo único que hace es cambiar los valores de verdadero por falso y viceversa para irlos agregando a una lista que posteriormente es enviada al diccionario final de la tabla de verdad.



Esta función se centra en el operador lógico de implicación o condicional. También está compuesto por un ciclo en el que se revisa los valores de verdadero y falso de los primitivos incluidos en la proposición. Se tiene una condición en donde en todas se regresa un valor de verdadero a menos que el primer valor del primitivo sea verdadero y el segundo falso. Como en los anteriores casos, todos los resultados de verdaderos y falsos se van agregando a una lista para después ser enviada al diccionario final en la que esta basada la tabla de verdad.

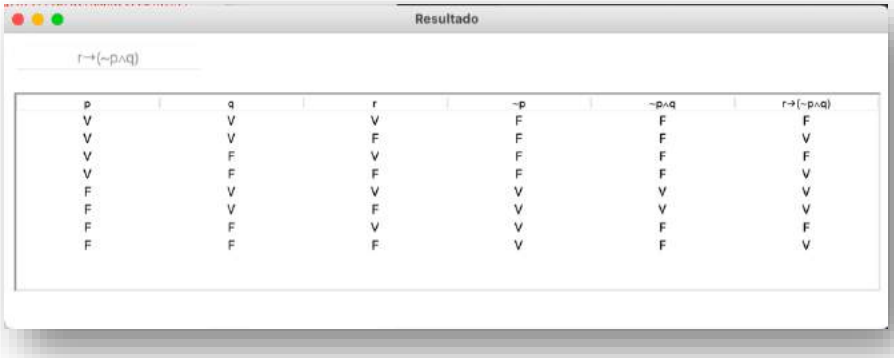
Por último, está la función que se encarga del operador lógico bicondicional. Como se puede observar, esta función utiliza la función if que se concentra en el operador lógico de la implicación y la función de and. Primero se saca los valores de verdadero y falso de los elementos incluidos en la proposición, esto se hace dos veces ya que se invierten los valores de los elementos. Por último se realiza un and de los resultados de verdad y falso de ambos casos para después agregarlos a la lista y posteriormente al diccionario final. Estas funciones se utilizan ya que el resultado solo será verdadero cuando ambos casos son iguales.



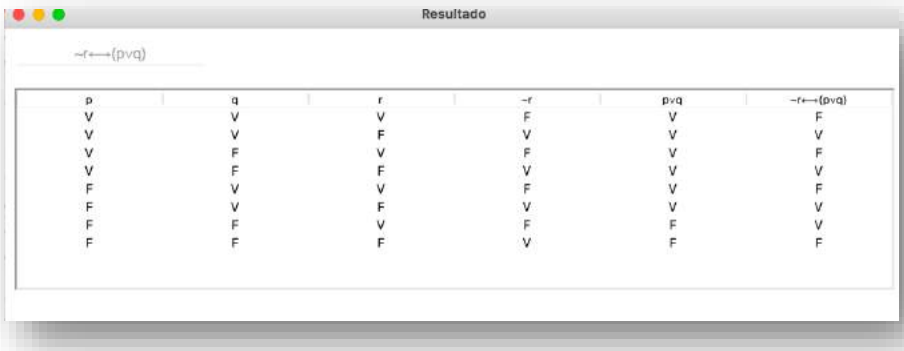
Capturas de pantallas del programa:

- Página Principal

- Crear Tablas de verdad



p	q	r	~p	~p∧q	r→(~p∧q)
V	V	V	F	F	F
V	V	F	F	F	V
V	F	V	F	F	F
V	F	F	F	F	V
F	V	V	V	V	V
F	V	F	V	V	V
F	F	V	V	F	F
F	F	F	V	F	V

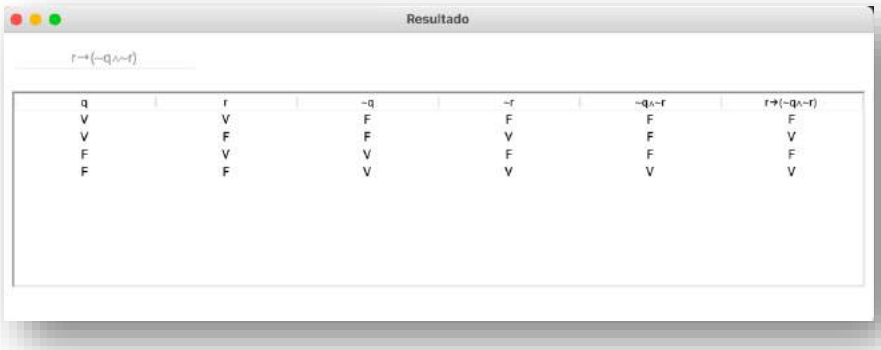


p	q	r	~r	p∨q	~r⟷(p∨q)
V	V	V	F	V	F
V	V	F	V	V	V
V	F	V	F	V	F
V	F	F	V	V	V
F	V	V	F	V	F
F	V	F	V	V	V
F	F	V	F	F	V
F	F	F	V	F	F

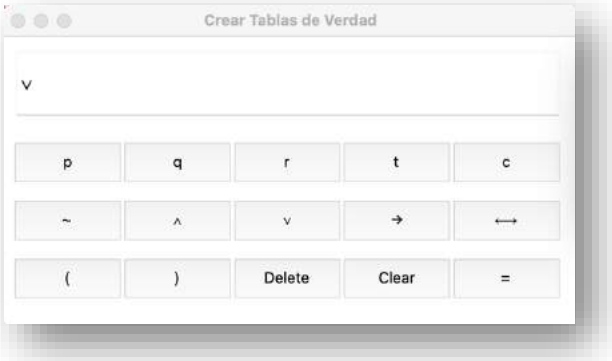

Crear Tablas de Verdad
t∧~c
p
q
r
t
c
~
∧
∨
→
⟷
(
)
Delete
Clear
=


Resultado
t∧~c
t c ~c t∧~c
V F V V
V F V V
V F V V
V F V V
V F V V
V F V V
V F V V
V F V V

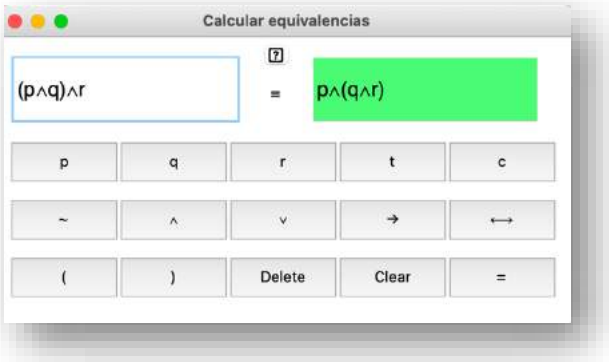

Crear Tablas de Verdad
r→(~q∧~r)
p
q
r
t
c
~
∧
∨
→
⟷
(
)
Delete
Clear
=

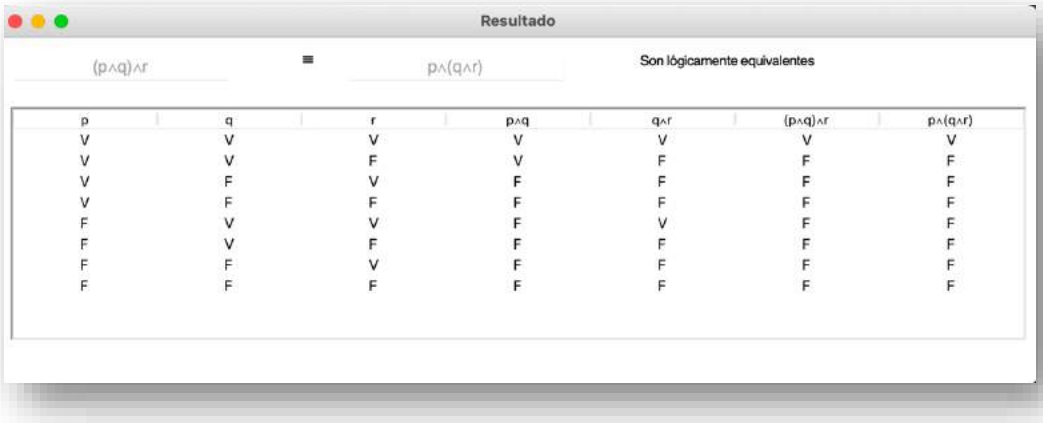

Resultado
r→(~q∧~r)
q r ~q ~r ~q∧~r r→(~q∧~r)
V V F F F F
V F F V F V
F V V F F F
F F V V V V

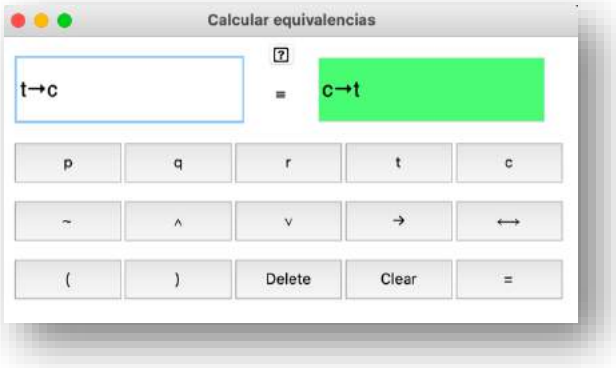
- Crear tablas de verdad (Errores posibles)



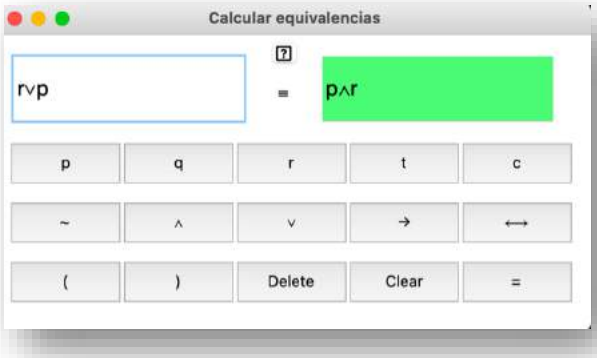
- Calcular equivalencias lógicas



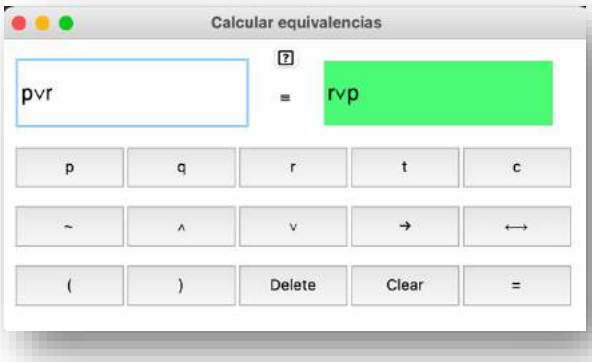


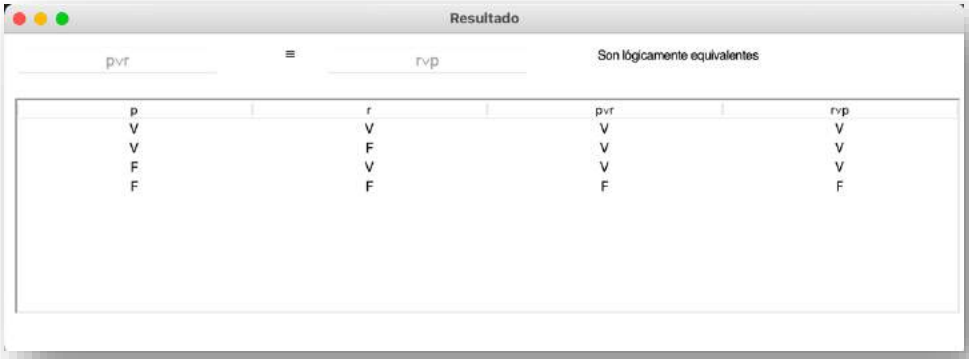



Resultado
t→c
≠
c→t
No son lógicamente equivalentes
t c t→c c→t
V F F V
V F F V
V F F V
V F F V
V F F V
V F F V
V F F V
V F F V


Calcular equivalencias
rvp
=
p∧r
p q r t c
~ ∧ v → ⟷
( ) Delete Clear =


Resultado
rvp
≠
p∧r
No son lógicamente equivalentes
p r rvp p∧r
V V V V
V F V F
F V V F
F F F F


Calcular equivalencias
pvr
≡
rvp
p
q
r
t
c
~
∧
∨
→
⟷
(
)
Delete
Clear
=

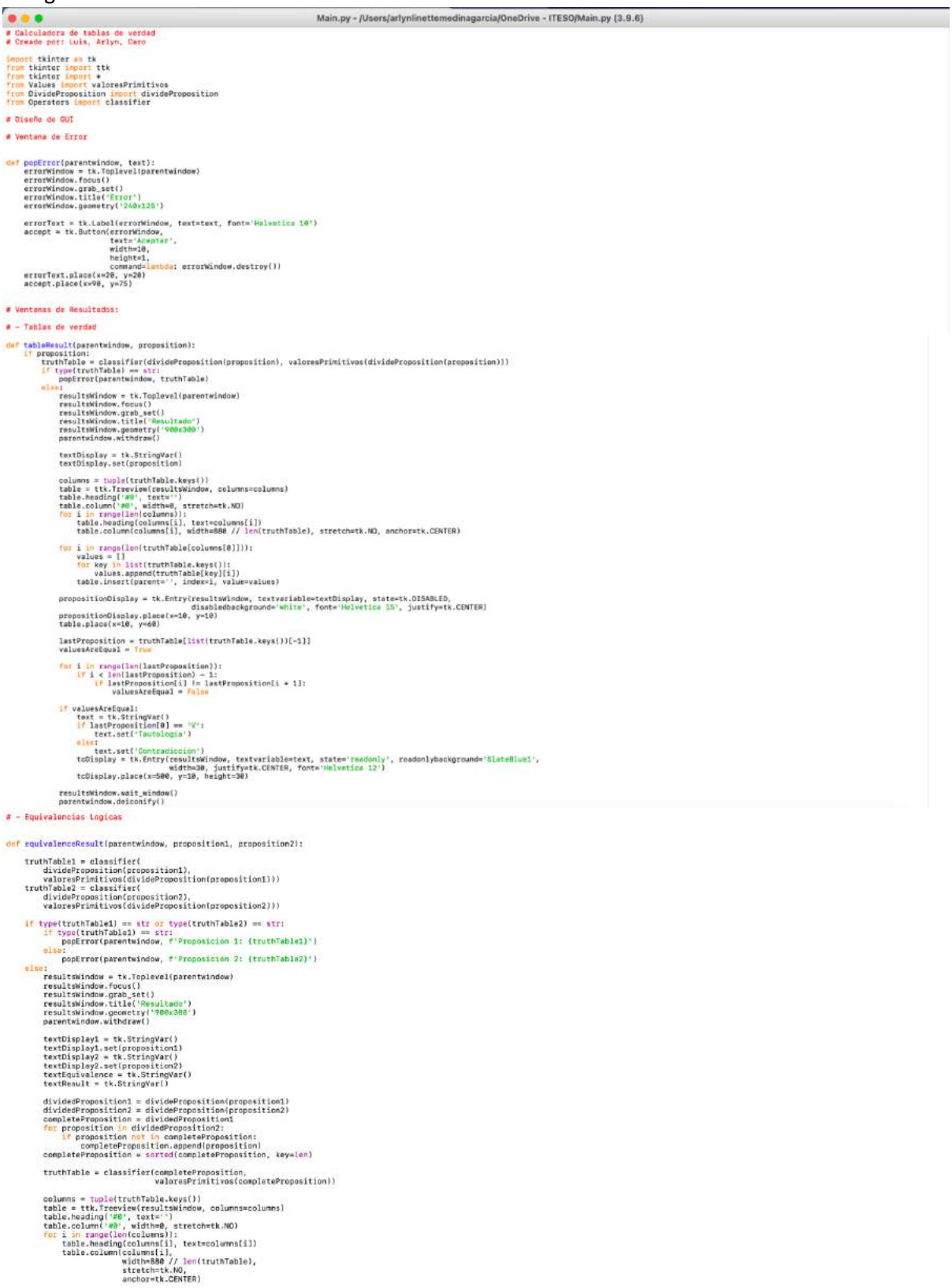

Resultado
pvr
≡
rvp
Son lógicamente equivalentes
p
r
pvr
rvp
V V V V
V F V V
F V V V
F F F F

- Acerca de:


Información del programa
Creado por
Luis Raúl Acosta Mendoza
Ana Carolina Arellano Valdez
Arlyn Linette Medina García
GitHub
@carolinarellano Logic-Calculator

Código Fuente comentado

- Código de interfaz tkinter



```
Main.py - /Users/arlynlinettemedinagarcia/OneDrive - ITESO/Main.py (3.9.6)
# Calculadora de tablas de verdad
# Creada por: Luis, Arlyn, Dano

import tkinter as tk
from tkinter import ttk
from tkinter import *
from Values import valoresPrimitivos
from DivideProposition import divideProposition
from Operators import classifier

# Diseño de GUI

# Ventana de Error


def popError(parentwindow, text):
    errorWindow = tk.Toplevel(parentwindow)
    errorWindow.focus()
    errorWindow.grab_set()
    errorWindow.title('Error')
    errorWindow.geometry('240x125')

    errorText = tk.Label(errorWindow, text=text, font='Helvetica 10')
    accept = tk.Button(errorWindow,
                       text='Aceptar',
                       width=10,
                       height=1,
                       command=lambda: errorWindow.destroy())
    errorText.place(x=20, y=20)
    accept.place(x=90, y=75)


# Ventanas de Resultados:

# - Tablas de verdad

def tableResult(parentwindow, proposition):
    if proposition:
        truthTable = classifier(divideProposition(proposition), valoresPrimitivos(divideProposition(proposition)))
        if type(truthTable) == str:
            popError(parentwindow, truthTable)
        else:
            resultsWindow = tk.Toplevel(parentwindow)
            resultsWindow.focus()
            resultsWindow.grab_set()
            resultsWindow.title('Resultado')
            resultsWindow.geometry('900x300')
            parentwindow.withdraw()

            textDisplay = tk.StringVar()
            textDisplay.set(proposition)

            columns = tuple(truthTable.keys())
            table = ttk.Treeview(resultsWindow, columns=columns)
            table.heading('#0', text='')
            table.column('#0', width=0, stretch=tk.NO)
            for i in range(len(columns)):
                table.heading(columns[i], text=columns[i])
                table.column(columns[i], width=880 // len(truthTable), stretch=tk.NO, anchor=tk.CENTER)

            for i in range(len(truthTable[columns[0]])):
                values = []
                for key in list(truthTable.keys()):
                    values.append(truthTable[key][i])
                table.insert(parent='', index=i, value=values)

            propositionDisplay = tk.Entry(resultsWindow, textvariable=textDisplay, state=tk.DISABLED,
                                          disabledbackground='white', font='Helvetica 15', justify=tk.CENTER)
            propositionDisplay.place(x=10, y=10)
            table.place(x=10, y=60)

            lastProposition = truthTable[list(truthTable.keys())[-1]]
            valuesAreEqual = True

            for i in range(len(lastProposition)):
                if i < len(lastProposition) - 1:
                    if lastProposition[i] != lastProposition[i + 1]:
                        valuesAreEqual = False

            if valuesAreEqual:
                text = tk.StringVar()
                if lastProposition[0] == 'V':
                    text.set('Tautología')
                else:
                    text.set('Contradicción')
                tcDisplay = tk.Entry(resultsWindow, textvariable=text, state='readonly', readonlybackground='SlateBlue1',
                                     width=30, justify=tk.CENTER, font='Helvetica 12')
                tcDisplay.place(x=500, y=10, height=30)

            resultsWindow.wait_window()
            parentwindow.deiconify()
# - Equivalencias Logicas


def equivalenceResult(parentwindow, proposition1, proposition2):

    truthTable1 = classifier(
        divideProposition(proposition1),
        valoresPrimitivos(divideProposition(proposition1)))
    truthTable2 = classifier(
        divideProposition(proposition2),
        valoresPrimitivos(divideProposition(proposition2)))

    if type(truthTable1) == str or type(truthTable2) == str:
        if type(truthTable1) == str:
            popError(parentwindow, f'Proposición 1: {truthTable1}')
        else:
            popError(parentwindow, f'Proposición 2: {truthTable2}')
    else:
        resultsWindow = tk.Toplevel(parentwindow)
        resultsWindow.focus()
        resultsWindow.grab_set()
        resultsWindow.title('Resultado')
        resultsWindow.geometry('900x300')
        parentwindow.withdraw()

        textDisplay1 = tk.StringVar()
        textDisplay1.set(proposition1)
        textDisplay2 = tk.StringVar()
        textDisplay2.set(proposition2)
        textEquivalence = tk.StringVar()
        textResult = tk.StringVar()

        dividedProposition1 = divideProposition(proposition1)
        dividedProposition2 = divideProposition(proposition2)
        completeProposition = dividedProposition1
        for proposition in dividedProposition2:
            if proposition not in completeProposition:
                completeProposition.append(proposition)
        completeProposition = sorted(completeProposition, key=len)

        truthTable = classifier(completeProposition,
                                valoresPrimitivos(completeProposition))

        columns = tuple(truthTable.keys())
        table = ttk.Treeview(resultsWindow, columns=columns)
        table.heading('#0', text='')
        table.column('#0', width=0, stretch=tk.NO)
        for i in range(len(columns)):
            table.heading(columns[i], text=columns[i])
            table.column(columns[i],
                         width=880 // len(truthTable),
                         stretch=tk.NO,
                         anchor=tk.CENTER)
```

```
        if truthTable[proposition1] == truthTable[proposition2]:
            textEquivalence.set('*')
            textResult.set('Son logicamente equivalentes')
        else:
            textEquivalence.set('#')
            textResult.set('No son logicamente equivalentes')

        for i in range(len(truthTable[columns[0]])):
            values = []
            for key in list(truthTable.keys()):
                values.append(truthTable[key][i])
            table.insert(parent='', index=i, value=values)

        propositionDisplay1 = tk.Entry(resultsWindow,
                                       textvariable=textDisplay1,
                                       state=tk.DISABLED,
                                       disabledbackground='white',
                                       font='Helvetica 15',
                                       justify=tk.CENTER)
        propositionDisplay2 = tk.Entry(resultsWindow,
                                       textvariable=textDisplay2,
                                       state=tk.DISABLED,
                                       disabledbackground='white',
                                       font='Helvetica 15',
                                       justify=tk.CENTER)
        equivalenceSign = tk.Label(resultsWindow,
                                   textvariable=textEquivalence,
                                   font='Helvetica 20',
                                   justify=tk.CENTER)
        textResultLabel = tk.Label(resultsWindow,
                                   textvariable=textResult,
                                   font='Helvetica 12')

        propositionDisplay1.place(x=10, y=10)
        propositionDisplay2.place(x=300, y=10)
        equivalenceSign.place(x=267, y=3)
        textResultLabel.place(x=550, y=10)
        table.place(x=10, y=60)

        resultsWindow.wait_window()
        parentwindow.deiconify()


# Ventana de la calculadora
#   mode:
#       0: Tablas de verdad
#       1: Equivalencias logicas

focusedDisplay = ''  # Ugh, variable global (indica el display seleccionado)

def calcWindow(parentwindow, mode):
    global focusedDisplay
    mainWindow = tk.Toplevel(parentwindow)
    mainWindow.focus()
    mainWindow.grab_set()
    mainWindow.geometry('500x250')
    parentwindow.withdraw()

    if mode == 0:

        # Modo tablas de verdad

        mainWindow.title('Crear Tablas de Verdad')

        text = tk.StringVar()
        focusedDisplay = text
        display = tk.Entry(mainWindow,
                           state='readonly',
                           readonlybackground='white',
                           font='Helvetica 18',
                           textvariable=text)
        calculate = tk.Button(
            mainWindow,
            text='=',
            width=10,
            height=2,
            bg= 'LightSteelBlue3',
            command=lambda: tableResult(mainWindow, text.get()))
        display.place(x=10, y=10, width=475, height=60)

    else:

        # Modo equivalencias logicas

        mainWindow.title('Calcular equivalencias')

        text1 = tk.StringVar()
        text2 = tk.StringVar()
        focusedDisplay = text1  # Display seleccionado es el 1

        display1 = tk.Entry(mainWindow,
                            state='readonly',
                            readonlybackground='white',
                            font='Helvetica 18',
                            textvariable=text1)
        display1.configure(readonlybackground='#83F684')
        display2 = tk.Entry(mainWindow,
                            state='readonly',
                            readonlybackground='white',
                            font='Helvetica 18',
                            textvariable=text2)
```

```
        def switchDisplay():  # Cambiar el display seleccionado
            global focusedDisplay
            if focusedDisplay == text1:
                focusedDisplay = text2
                display2.configure(readonlybackground='#83F684'
                                   )  # Color del display seleccionado
                display1.configure(readonlybackground='white'
                                   )  # Color del display no seleccionado
            else:
                focusedDisplay = text1
                display1.configure(readonlybackground='#83F684')
                display2.configure(readonlybackground='white')

        switch = tk.Button(mainWindow,
                           text='⇄',
                           width=2,
                           height=1,
                           command=lambda: switchDisplay())
        identical = tk.Label(mainWindow, font='Helvetica 18', text='≡')
        calculate = tk.Button(mainWindow,
                              text='=',
                              width=10,
                              height=2,
                              command=lambda: equivalenceResult(
                                  mainWindow, text1.get(), text2.get()))

        display1.place(x=10, y=15, width=200, height=60)
        display2.place(x=270, y=15, width=200, height=60)
        identical.place(x=230, y=35)
        switch.place(x=230, y=5)

    p = tk.Button(mainWindow,
                  text='p',
                  width=10,
                  height=2,
                  bg= 'dodger blue',
                  command=lambda: showInDisplay('p', focusedDisplay))
    q = tk.Button(mainWindow,
                  text='q',
                  width=10,
                  height=2,
                  bg= 'dodger blue',
                  command=lambda: showInDisplay('q', focusedDisplay))
    r = tk.Button(mainWindow,
                  text='r',
                  width=10,
                  height=2,
                  bg= 'dodger blue',
                  command=lambda: showInDisplay('r', focusedDisplay))
    t = tk.Button(mainWindow,
                  text='t',
                  width=10,
                  height=2,
                  bg= 'gold',
                  command=lambda: showInDisplay('t', focusedDisplay))
```

```
    c = tk.Button(mainWindow,
                  text='c',
                  width=10,
                  height=2,
                  bg= 'dark orange',
                  command=lambda: showInDisplay('c', focusedDisplay))
    negation = tk.Button(mainWindow,
                         text='~',
                         width=10,
                         height=2,
                         bg= 'cadet blue',
                         command=lambda: showInDisplay('~', focusedDisplay))
    conjunction = tk.Button(mainWindow,
                            text='^',
                            width=10,
                            height=2,
                            bg= 'cadet blue',
                            command=lambda: showInDisplay('^', focusedDisplay))
    disjunction = tk.Button(mainWindow,
                            text='v',
                            width=10,
                            height=2,
                            bg= 'cadet blue',
                            command=lambda: showInDisplay('v', focusedDisplay))
    implication = tk.Button(mainWindow,
                            text='→',
                            width=10,
                            height=2,
                            bg= 'cadet blue',
                            command=lambda: showInDisplay('→', focusedDisplay))
    bi = tk.Button(mainWindow,
                   text='↔',
                   width=10,
                   height=2,
                   bg= 'cadet blue',
                   command=lambda: showInDisplay('↔', focusedDisplay))
    openP = tk.Button(mainWindow,
                      text='(',
                      width=10,
                      height=2,
                      bg= 'light grey',
                      command=lambda: showInDisplay('(', focusedDisplay))
    closeP = tk.Button(mainWindow,
                       text=')',
                       width=10,
                       height=2,
                       bg= 'light grey',
                       command=lambda: showInDisplay(')', focusedDisplay))

    delete = tk.Button(mainWindow,
                       text='Delete',
                       width=10,
                       height=2,
                       bg= 'LightSteelBlue3',
                       command=lambda: showInDisplay(0, focusedDisplay))
    clear = tk.Button(mainWindow,
                      text='Clear',
                      width=18,
                      height=2,
                      bg= 'LightSteelBlue3',
                      command=lambda: showInDisplay(1, focusedDisplay))

    p.place(x=10, y=90)
    q.place(x=105, y=90)
    r.place(x=200, y=90)
    t.place(x=295, y=90)
    c.place(x=390, y=90)
    negation.place(x=10, y=140)
    conjunction.place(x=105, y=140)
    disjunction.place(x=200, y=140)
    implication.place(x=295, y=140)
    bi.place(x=390, y=140)
    openP.place(x=10, y=190)
    closeP.place(x=105, y=190)
    delete.place(x=200, y=190)
    clear.place(x=295, y=190)
    calculate.place(x=390, y=190)

    mainWindow.wait_window()
    parentwindow.deiconify()


# Modificar la proposicion en display


def showInDisplay(char, text):
    if char == 0:  # Borrar el ultimo caracter
        text.set(text.get()[0:-1])
    elif char == 1:  # Limpiar el display
        text.set('')
    else:
        text.set(f'{text.get()}{char}')


# Ventana Acerca de
def Acercade(parentwindow,mode):
    global focusedDisplay
    mainWindow = tk.Toplevel(parentwindow)
    mainWindow.focus()
    mainWindow.grab_set()
    mainWindow.geometry('500x250')
    parentwindow.withdraw()
    mainWindow.title('Información del programa')
    text2 = tk.StringVar()

    label1 = tk.Label(mainWindow,text="Creado por ", font='Helvetica 16')
    label1.place(x=180,y=20)
    label2 = tk.Label(mainWindow,text="Luis Raul Acosta Mendoza", font='Helvetica 10')
    label2.place(x=150,y=50)
    label3 = tk.Label(mainWindow,text="Ana Carolina Arellano Valdez", font='Helvetica 10')
    label3.place(x=150,y=75)
    label4 = tk.Label(mainWindow,text="Arlyn Linette Medina Garcia", font='Helvetica 10')
    label4.place(x=150,y=100)
    label5 = tk.Label(mainWindow,text="GitHub", font='Helvetica 16')
    label5.place(x=200,y=150)
    label16 = tk.Label(mainWindow, text= "@carolinarellano Logic-Calculator", font= 'Helvetica 9')
    label16.place(x=150, y=180)

    mainWindow.wait_window()
    parentwindow.deiconify()

# Ventana principal

root = tk.Tk()
root.title('Logic Calculator')
root.geometry('335x325')
title = tk.Label(root, text='Calculadora Logica', font= 'Helvetica 15', width=20, height=3)
truthTableButton = tk.Button(root,
                             text='Crear tablas de verdad',
                             bg='dark salmon',
                             fg='black',
                             width=30,
                             height=3,
                             command=lambda: calcWindow(root, 0))
equivalence = tk.Button(root,
                        text='Calcular equivalencias logicas',
                        width=30,
                        height=3,
                        bg= 'dark salmon',
                        command=lambda: calcWindow(root, 1))

about = tk.Button(root,
                  text='Acerca de',
                  width=30,
                  height=2,
                  bg= 'SkyBlue4',
                  command=lambda: Acercade(root, 2))


title.place(x=55, y=30)
truthTableButton.place(x=55, y=100)
equivalence.place(x=55, y=170)
about.place(x=55, y=240)
root.mainloop()
```

- Código para identifical valores primitivos y empezar tabla de verdad

Values.py - /Users/arlynlinettemedinagarcia/OneDrive - ITESO/Values.py (3.9.6)

```python
# Funcion que regresa un diccionario con los valores de verdad de las primitivas

def valoresPrimitivos(proposition):

    #  Si existe un error, se recibe una string, en caso contrario, se recibe una lista que puede ser evaluada

    if type(proposition) != str:

        # Contar el numero de proposiciones primitivas

        primitives = ['p', 'q', 'r']
        nOfPrimitives = 0

        for primitive in primitives:
            if primitive in proposition:
                nOfPrimitives += 1

        # Asignar los valores de verdad de acuerdo a la cantidad de primitivas

        if nOfPrimitives == 1:

            o1 = {
                proposition[0]: ['V', 'F']
            }

            return o1

        elif nOfPrimitives == 2:

            o2 = {
                proposition[0]: ['V', 'V', 'F', 'F'],
                proposition[1]: ['V', 'F', 'V', 'F']
            }

            return o2

        else:

            o3 = {
                proposition[0]: ['V', 'V', 'V', 'V', 'F', 'F', 'F', 'F'],
                proposition[1]: ['V', 'V', 'F', 'F', 'V', 'V', 'F', 'F'],
                proposition[2]: ['V', 'F', 'V', 'F', 'V', 'F', 'V', 'F']
            }

            return o3

    else:
        return proposition
```

- Código de dividir proposiciones

DivideProposition.py - /Users/arlynlinettemedinagarcia/OneDrive - ITESO/DivideProposition.py (3.9.6)

```python
# Funcion para dividir una proposicion compuesta en proposiciones de una sola operacion
def divideProposition(proposition):
    syntaxStatus = checkSyntax(proposition)
    if syntaxStatus == 'Valid':
        dividedProposition = []
        getPrimitives(proposition, dividedProposition)
        negatePrimitives(proposition, dividedProposition)
        getOperations(proposition, dividedProposition)
        dividedProposition = sorted(dividedProposition, key=len)
        return dividedProposition
    else:
        return syntaxStatus


def checkSyntax(proposition):
    errorList = {
        '0': 'Valid',
        '00': 'Existen paréntesis\nsin abrir o cerrar',
        '01': 'No hay primitivas en\nla proposicion',
        '02': 'Existen proposiciones\nno separadas por un\noperador o dos operadores juntos',
        '03': 'La proposicion\nesta incompleta'
    }

    primitives = ['p', 'q', 'r', 't', 's']
    operators = ['^', 'v', '>', '<->']

    # Error 00: Parentesis sin abrir o cerrar

    if proposition.count('(') != proposition.count(')'):
        return errorList['00']

    # Error 01: Proposicion sin primitivas

    errorFlag = 0  # Si llega a 5, no existen primitivas en la proposicion
    for primitive in primitives:
        if primitive not in proposition:
            errorFlag += 1
    if errorFlag == 5:
        return errorList['01']

    # Error 02: 2 operadores o 2 proposiciones primitivas juntas

    for i, char in enumerate(proposition):
        if char in primitives:
            if i != 0:
                if proposition[i - 1] in primitives:
                    return errorList['02']
            if i != len(proposition) - 1:
                if proposition[i + 1] == '~' and i != len(proposition) - 1:
                    return errorList['02']
        if char in operators and proposition[i - 1] in operators:
            return errorList['02']

        # Error 03: Operaciones con falta de operandos
```

```
    if proposition[-1] in operators or proposition[0] in operators:
        return errorList['03']

    return errorList['0']


def getPrimitives(proposition, columns):
    primitives = ['p', 'q', 'r', 't', 'e']

    for primitive in primitives:
        if primitive in proposition:
            columns.append(primitive)


def negatePrimitives(proposition, columns):
    for i, char in enumerate(proposition):
        if char == '~':
            negative = f'~{proposition[i + 1]}'
            columns.append(negative)


def getOperations(proposition, columns):
    operators = ['^', 'v', '→', '←→']

    # Quitar parentesis innecesarios

    if proposition[0] == '(' and proposition[-1] == ')':
        if proposition.find(')') == -1:
            proposition = proposition[1:-1]

    # En caso de doble parentesis para la misma expresion, se eliminan

    for i, char in enumerate(proposition):
        if char == '(' and proposition[i + 1] == '(':
            closingParenthesis = proposition.find(')', i)
            if proposition[closingParenthesis + 1] == ')':
                proposition = proposition[:i] + proposition[i + 1:]
                proposition = proposition[:closingParenthesis] + proposition[closingParenthesis + 1:]

    # Dividir por operador

    for i, char in enumerate(proposition):
        if char in operators:
            if char != '→' and char != '←→':
                if proposition[i - 1] != ')' and proposition[i + 1] != '(':
                    start = i - 1
                    end = i + 2
                    if proposition[i + 1] == '~':
                        end = i + 3
                    if proposition[i - 2] == '~':
                        start = i - 2
                    operation = proposition[start:end]
                    appendColumns(operation, columns)
            else:
                start = i - 1
                if proposition[i - 2] == '~':
                    start = i - 2
                if proposition[i - 1] == ')':
                    start = proposition[i::-1].find('(')
                    start = i - start

                if proposition[i + 1] == '(':
                    end = proposition.find(')', i) - 1
                else:
                    end = len(proposition)
                if proposition[i + 1] == '~':
                    end = i + 2
                if proposition[i + 1] == '(':
                    end = proposition.find(')', i)
                operation = proposition[start:end + 1]
                appendColumns(operation, columns)

        # Dividir las operaciones dentro de parentesis

        if char == '(':
            if proposition[i + 1] != '(' and proposition[i - 1] != '(':
                closingParenthesis = proposition.find(')', i)
                operation = proposition[i:closingParenthesis + 1]
                appendColumns(operation, columns)
            elif i < len(proposition) - 1 and proposition[i + 1] == '(':
                closingParenthesis = proposition.find(')', i)
                operation = proposition[i + 1:closingParenthesis + 1]
                appendColumns(operation, columns)
                closingParenthesis = proposition.find(')', closingParenthesis + 1)
                operation = proposition[i + 1:closingParenthesis + 1]
                appendColumns(operation, columns)

    # Añadir la proposicion completa

    appendColumns(proposition, columns)

# Añadir operacion dividida filtrando los registros duplicados


def appendColumns(operation, columns):
    duplicate = False
    for column in columns:
        if operation.strip('()') == column.strip('()'):
            duplicate = True
    if not duplicate:
        columns.append(operation)
```

- Código de clasificador de valores primitivos y operadores

Operators.py - /Users/arlynlinettemedinagarcia/OneDrive - ITESO/Operators.py (3.9.6)

```
# Funcion que realiza todas las operaciones necesarias en la proposicion

from switchcase import switch


def classifier(divided_proposition, primitive_table):

    # Manejo de errores

    if type(divided_proposition) == str:
        return divided_proposition

    operators = ['∧', '∨', '+', '←→']
    truth_table = primitive_table

    # Evaluar cada parte de la proposicion

    for proposition in divided_proposition:

        # Obtener valores de verdad de tautologia, contradiccion y negacion

        if proposition == 't':
            truth_table[proposition] = tautology(truth_table)
        if proposition == 'c':
            truth_table[proposition] = contradiction(truth_table)

        if '~' in proposition and len(proposition) == 2:
            truth_table[proposition] = not_function(truth_table[proposition[1]])

        # Operar cada parte de la propposicion

        for operator in operators:
            if operator in proposition:
                evaluator = operator
                start = 0
                end = len(proposition)

                # Interpretar correctamente con parentesis

                if proposition[0] == '(' and proposition[-1] == ')':
                    if proposition.find(')') == len(proposition) - 1:
                        proposition = proposition.strip('()')
                    else:
                        start = proposition.find(')')
                elif proposition[0] == '(':
                    start = proposition.find(')')

                # Ignorar operaderes dentro de parentesis para proposiciones compuestas

                if proposition[proposition.index(evaluator) - 1] == ')' or proposition.find(')') == -1\
                        or proposition[proposition.index(evaluator) + 1] == '(' or proposition.find('(') == 0:

                    # Obtener p y q de cada operacion

                    midOperator = proposition.find(evaluator, start, end)
                    p = truth_table[proposition[0:midOperator].strip('()')]
                    q = truth_table[proposition[midOperator + 1::].strip('()')]

                    for case in switch(evaluator):
                        if case('∧'):
                            truth_table[proposition] = and_function(p, q)
                        if case('∨'):
                            truth_table[proposition] = or_function(p, q)
                        if case('+'):
                            truth_table[proposition] = if_function(p, q)
                        if case('←→'):
                            truth_table[proposition] = bi_function(p, q)

    return truth_table

#Funcion para operador lógico and
def and_function(a, b):
    truth_table_and = []
    for index in range(len(a)):
        if a[index] == 'V' and b[index] == 'V':
            truth_table_and.append('V')
        else:
            truth_table_and.append('F')
    return truth_table_and

#Funcion para operador lógico or
def or_function(a, b):
    truth_table_or = []
    for index in range(len(a)):
        if a[index] == 'F' and b[index] == 'F':
            truth_table_or.append('F')
        else:
            truth_table_or.append('V')
    return truth_table_or

#Funcion para negacion
def not_function(a):
    truth_table_not = []
    for value in a:
        if value == 'V':
            truth_table_not.append('F')
        else:
            truth_table_not.append('V')
    return truth_table_not

#Funcion para operador lógico and
def if_function(a, b):
    truth_table_if = []
    for index in range(len(a)):
        if a[index] == 'V' and b[index] == 'F':
            truth_table_if.append('F')
        else:
            truth_table_if.append('V')
    return truth_table_if
```

```
#Funcion para operador lógico or
def or_function(a, b):
    truth_table_or = []
    for index in range(len(a)):
        if a[index] == 'F' and b[index] == 'F':
            truth_table_or.append('F')
        else:
            truth_table_or.append('V')
    return truth_table_or

#Funcion para negacion
def not_function(a):
    truth_table_not = []
    for value in a:
        if value == 'V':
            truth_table_not.append('F')
        else:
            truth_table_not.append('V')
    return truth_table_not

#Funcion para operador lógico and
def if_function(a, b):
    truth_table_if = []
    for index in range(len(a)):
        if a[index] == 'V' and b[index] == 'F':
            truth_table_if.append('F')
        else:
            truth_table_if.append('V')
    return truth_table_if

#Funcion para operador lógico de la implicacion
def bi_function(a, b):
    p = if_function(a, b)
    q = if_function(b, a)
    truth_table_bi = and_function(p, q)
    return truth_table_bi

#Funcion para la tautologia
def tautology(truth_table):
    truth_table_tautology = []
    for i in range(len(truth_table[list(truth_table.keys())[0]])):
        truth_table_tautology.append('V')
    return truth_table_tautology

#Funcion para la contradiccion
def contradiction(truth_table):
    truth_table_contradiction = []
    for i in range(len(truth_table[list(truth_table.keys())[0]])):
        truth_table_contradiction.append('F')
    return truth_table_contradiction
```

- Switchcase

```
"""
A pure-Python implementation of switch-case style control flow.

Example
-------
from switchcase import switch

for case in switch(x):
    if case(3):
        print("x was 3")
        break
    if case(4):
        print("x was 4")
        break
"""
from operator import eq

#Importar comando switchase para ser usado en la identificación de los diferentes operadores lógicos
def switch(value, comp=eq):
    return [lambda match: comp(match, value)]
```

Conclusiones:

Ana Carolina Arellano Valdez.

Al realizar este proyecto tuve más claridad acerca del uso de las tablas de verdad, de cómo funcionan los operadores, pues al desarrollar las funciones que darían los resultados de acuerdo con el operador que se utilizara pude analizar con la lógica para saber cómo asignar los valores usando las reglas básica de cada operador.

Por ejemplo, si el operador es conjunción (AND), sólo es verdadero cuando ambos valores (p y q) son verdaderos. De igual manera con el operador de disyunción (OR), condicional (IF), negación (NOT) y bicondicional (BI).

De igual manera, al estar buscando la lógica para separar operadores e ir por lista de prioridades fue importantes, pues los paréntesis complicaban un poco esta situación. Para poder validar que las entradas fueran correctas hicimos una implementación de errores de código para que el programa no tronara.

También puedo agregar que implementar herramientas como diccionarios, listas y switchcase permitieron que el código fuera mucho más conciso, además del uso de ciclos y condicionales en la implementación del código.

Luis Raúl Acosta Mendoza

Tras haber concluido el proyecto, me queda muy claro la importancia y utilidad que le puedo dar a la lógica proposicional en la programación. Pues puedo decir que en algunas ocasiones mientras realizaba alguna parte del proyecto, llegué a programar condiciones que a simple vista creí que resolverían mi problemática cuando, en realidad, era otro orden u otros operadores lo que tenía que utilizar.

Creo que el proyecto me ayudó a mejorar la forma en que escribo código porque con las complicaciones que surgieron (detectar proposiciones entre parentesis, encontrar el operador correcto, entre otras) tuvimos que buscar otras perspectivas para enfrentar el problema, lo cual consecuentemente nos llevó también a optimizar las funciones del código (por ejemplo, la función classifier del archivo Operators, encargado de realizar las operaciones correspondientes, sufrió un cambio muy drástico en los últimos días de realización)

Arlyn Linette Medina García

Este proyecto no solo me sirvió como practica de programación sino que también me ayudo a reforzar varios temas sobre las tablas de verdad. Al principio estuvimos un poco atorados en la división de proposiciones para el indice de la tabla de verdad, pero después de plantear todas las opciones que se podía haber, logramos sacar adelante esa parte. Esta actividad me ayudo a comprender cómo la lógica esta muy relacionada a la programación y que va a ser muy importante para futuros programas en los que sea indispensable tener eficiencia y eficacia.

Lo que más se me facilitó fue sacar los primeros valores de verdad y falso de los principales primitivos. Al principio planeábamos sacar esos valores con un sistema binario, pero después nos dimos cuenta que para el número de primitivos que se pedía, era más fácil tener listas predeterminadas y solamente identificar cual de las tres diferentes listas se necesitaba. Teníamos un poco de experiencia con la interfaz tkinter que usamos. Esta interfaz nos ayudo a tener un programa más limpio y ordenado.

Referencias:

Barceló Aspeitia, A. A. (2012, 26 marzo). Introducción a la Lógica Intensional - Lógica Temporal Proposicional. UNAM Filosoficas. Recuperado 1 de marzo de 2022, de http://www.filosoficas.unam.mx/~abarcelo/INTENSIONAL/2012/260312.pdf

Briceño, G. (2020, 23 julio). Python: Como se implementa una sentencia switch-case. Club de Tecnología. https://www.clubdetecnologia.net/blog/2017/python-como-se-implementa-una-sentencia-switch-case/

Plasencia Prado, C. E. (2017, 19 septiembre). Diccionarios en Python. DevCode Tutoriales. Recuperado 1 de marzo de 2022, de https://devcode.la/tutoriales/diccionarios-en-python/

Python, R. (2021, 27 agosto). Introducción a Tcl/Tk (tkinter). Recursos Python. Recuperado 1 de marzo de 2022, de https://recursospython.com/guias-y-manuales/introduccion-a-tkinter/

Python Software Foundation. (2020). tkinter — Interface de Python para Tcl/Tk — documentación de Python - 3.10.2. Python docs. Recuperado 1 de marzo de 2022, de https://docs.python.org/es/3/library/tkinter.html

Link a GitHub https://github.com/carolinarellano/Logic-Calculator