Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Lenguajes Formales y de programación B-

Primer semestre 2023

Auxiliar: Pablo Daniel Rivas Marroquín

Proyecto 1

Manual Técnico

Andrea Jazmin Rodriguez Citalán

Carnet: 202110553

Datos del desarrollador:

- Nombre: Andrea Jazmin Rodriguez Citalán
- Carnet: 202110553
- Universidad: San Carlos de Guatemala

Detalles de desarrollo

- El programa fue desarrollado en el lenguaje Python.
- Se utilizó Visual Studio Code como IDE para el desarrollo de la aplicación

Logica del programa:

Glase Ventana:

En la función init se encuetra todo el código de tkinter como lo es el frame, los label, los botones y ahí se mandaba a llamar la función que se desea ejecutar cuando se necesita también se encuentra el área de texto donde se coloca la información del archivo de entrada.

```
class Ventana:
    def __init__ (self):
        self.a = analizador2()
        self.raiz = Tk()
        #Ventana principal
        self.raiz.title("Proyecto 1")
        self.raiz.geometry("550x500")
        self.raiz.resizable(False,False)
        #self.raiz.iconbitmap("Icono.ico")

        self.frame = Frame(self.raiz)
        self.frame.place(x=0, y=0, width=550, height=500)
        self.frame.config(bg="#e7aebc")
        #Textos

        self.archivo = Label(self.frame, text="Archivo", bg="#c63b97", font=("petfinder", 14))
        self.archivo.place(x=10, y= 0, width=150, height=30,)

        self.ayuda = Label(self.frame, text="Ayuda", bg="#c989d6", font=("petfinder", 14))
        self.ayuda.place(x=10, y= 270, width=150, height=30,)

        # Botones del laado de archivo
        self.b1 = Button(self.frame, text="Abrir",bg="#c866b9", command=self.abrirTXT, font=("petfinder", 12)).place(x=10, y= 30, width=150, height=30)
        self.b2 = Button(self.frame, text="Guardar", bg="#c866b9", command=self.guardar, font=("petfinder", 12)).place(x=10, y= 60, width=150, height=30)
        self.b3 = Button(self.frame, text="Guardar Como",bg="#c866b9", command= self.guardarComo,font=("petfinder", 12)).place(x=10, y= 90, width=150, height=30)
        self.b4 = Button(self.frame, text="Analizar", bg="#c866b9",command=self.analizar ,font=("petfinder", 12)).place(x=10, y= 120, width=150, height=30)
        self.b5 = Button(self.frame, text="Errores",bg="#c866b9",command= self.errores,font=("petfinder", 12)).place(x=10, y= 150, width=150, height=30)
        self.b6 = Button(self.frame, text="Salir", bg="#c866b9",command=self.raiz.destroy,font=("petfinder", 12)).place(x=10, y= 180, width=150, height=30)

        # Botones del laado de ayuda
        self.b7 = Button(self.frame, text="Manual de usuario",bg="#dfc9f5", command=self.manual_usuario,font=("petfinder", 12)).place(x=10, y= 300, width=150, height=30)
        self.b8 = Button(self.frame, text="Manual de Técnico",bg="#dfc9f5",command=self.manual_tecnico ,font=("petfinder", 12)).place(x=10, y= 330, width=150, height=30)
        self.b9 = Button(self.frame, text="Temas de ayuda",bg="#dfc9f5", command=self.Datosayuda ,font=("petfinder", 12)).place(x=10, y= 360, width=150, height=30)

        #Text area
        self.texto = Text (self.frame)
        self.texto.place (x=170, y=5,width=360, height=470 )
        self.scroll = Scrollbar(self.texto)
        self.scroll.pack(side=RIGHT, fill=Y)
        self.texto.config(yscrollcommand=self.scroll.set)
        self.scroll.config(command=self.texto.yview)

        self.raiz.mainloop()
```

Función abrirTXT: En esta función se busca el archivo que se quiere cargar y se muestra un mensaje de que se ha cargado con éxito el archivo y se muestra en el área de texto de la interfaz.

```
def abrirTXT(self):
    self.texto.delete('1.0',END)
    self.text_file = filedialog.askopenfilename(initialdir = "/Descargas", title = "Holi busca tu archivo",filetypes = ( ("TXT", "*.txt*"),("All Files", "*.*") ))

    try:
        text = open(self.text_file,'r+', encoding="utf-8")
        contenido = text.read()
        self.texto.insert(1.0, contenido)
        self.leer = contenido
        text.close()
        messagebox.showinfo(title="¡Felicidades!", message="Archivo cargado con éxito")
    except UnicodeDecodeError:
     messagebox.showerror(title="Advertencia" ,message="Aún no hay archivo cargado")
```

Función para guardar: En esta función se pueden hacer cambios al archivo de entrada y al darle guardar se guardarán los cambios realizados al archivo y se mostrara un mensaje de que se ha guardado correctamente.

```
def guardar(self):
    texto1=self.texto.get(1.0,END)
    print(texto1)
    if self.text_file==None:
        self.guardarComo()
    else:
        self.text3=open(self.text_file,"w+",encoding="utf-8")
        self.text3.write(texto1)

        self.text3.close()

        messagebox.showinfo("Guardar", "Los datos se han guardado correctamente")
```

Función guardarComo: Esta función se usa para guardar un archivo nuevo

```
def guardarComo(self):

    archivo_guardar=filedialog.asksaveasfilename(initialdir = "/",title = "Guardar archivo",defaultextension=".txt", filetypes = (("txt files","*.txt"),("all files","*.*")))
    try:
        if archivo_guardar is not None:
            contenido=self.texto.get(1.0,END)
            archivo2=open(archivo_guardar,"w+",encoding="utf-8")
            archivo2.write(contenido)
            archivo2.close()
            self.text_file=archivo_guardar
            messagebox.showinfo("Guardar Como", "el archivo se guardo correctamente")
        else:
            print('no archivo ')
    except FileNotFoundError:
        messagebox.showerror("Error", "no se pudo guardar el archivo")
```

Función DatosAyuda: se muestra un mensaje emergente con la información del estudiante.

```
def Datosayuda(self):

    messagebox.showinfo(title="Temas de Ayuda", message="Lenguajes Formales de Programacion B- \nProyecto 1 \nNombre: Andrea Jazmin Rodriguez Citalán\nCarnet: 202110553")
```

Función analizar: Aquí se lee el archivo de entrada y se crea un archivo txt con los resultados de las operaciones realizadas.

```
def analizar(self):
    archi = open(self.text_file, "r")
    leer = ""
    for i in archi.readlines():
        leer += i

    self.a.instruccion(leer)
    self.a.operar_()
    with open('Resultados.txt', 'w',encoding="utf-8") as archivo:
        with redirect_stdout(archivo):
            self.a.instruccion(leer)
            self.a.operar_()
            #archivo.close()
    os.system('Resultados.txt')
```

Función de errores: Genera el archivo tipo json para enumerar los errores.

```
def errores(self):
    self.a.getErrores()
    #archivo = self.a.getErrores()
    with open('ERRORES_202110553.txt', 'w', encoding="utf-8") as archivo:
        with redirect_stdout(archivo):
            self.a.getErrores()
            #archivo.close()
    os.system('ERRORES_202110553.txt')
```

Estas dos funciones sirven para abrir los manuales al presionar el botón

```
def manual_tecnico(self):
    path = "tecnico.pdf"
    os.system(path)

def manual_usuario(self):
    path = "usuario.pdf"
    os.system(path)
```

Clase analizador:

Se colocan las variables globales que serán usadas en todo el programa

```
def __init__(self):
    self.col = 1
    self.fil = 1
    self.lista_lexemas = []
    self.lista_errores = []
```

Función armar_lexema: En esta función se recibe como parámetro una variable de tipo string, se declaran dos variables las cuales servirán para ir recorriendo el texto del archivo de entrada y donde encuentre la “ empieza a crear el lexema y se regresa la palabra ya armada y se retorna la variable que fue declarada en el parámetro con la longitud del punturo o hasta donde termino para que no se reinicie el ciclo.

```
def armar_lexema(self, cadena):
    lexema = ""
    count = ""
    for i in cadena:
        count += i
        if i == '\"':
            #lexema va recorriendo las palabras y arma la palabara para aceptarlo como un caracter valido
            #se regresa la cadena con la posición del count  hasta el final por eso los :
            return lexema, cadena[len(count):]
        else:
            lexema += i
    return None, None
```

En la función getErrores se inicia el contador en 1 se imprime la llave de apertura después se van eliminando e imprimiendo los caracteres que no pertenecen a las palabras reservadas para después generar el archivo de errores

```
def getErrores(self):
    #self.lista_errores: list = self.getErrores()
    contador = 1
    print("{")
    for error in self.lista_errores:
        error = self.lista_errores.pop(0)
        print(error.operar(contador),",")
        contador+= 1
    print("}")
```

Función num: El objetivo de esta función es reconocer si un número es decimal esto se logró recorriendo el numero como un string y si se encontraba un . en el texto significaba que el número es un decimal. Si no se regresaba como un numero entero.

```
def num(self, cadena:str):
    num = ""#token
    count = ""
    decimal = False
    for i in cadena:
        count += i
        if i ==".":
            decimal = True
        if i == '\"' or i == ' ' or i == '\n' or i == '\t':
            if decimal:
                return float(num), cadena[len(count)-1:]
            else:
                return int(num), cadena[len(count)-1:]
        else:
            num += i

    return None, None
```

la función operar: Es utilizada para procesar la lista de lexemas (tokens) y construir una instancia de la clase "Aritmetica" o "Trigonometricas", dependiendo de los tokens que se encuentren en la lista. La función utiliza un bucle while para recorrer la lista de tokens, y en cada iteración, comprueba si el token actual es una operación, un primer valor o un segundo valor. Si se encuentran los tres, se crea una instancia de la clase correspondiente con los valores y la operación, y se devuelve esa instancia. Si no se encuentran los tres, la función devuelve None.

```
def operar(self):
    operacion = ""
    n1 = ""
    n2 = ""
    #lexema1 = list
    while self.lista_lexemas:
        lexema1 = self.lista_lexemas.pop(0)
        if lexema1.operar(None) == "Operacion":
            operacion = self.lista_lexemas.pop(0)
        elif lexema1.operar(None)== "Valor1":
            n1 = self.lista_lexemas.pop(0)
            if n1.operar(None) == "[":
                n1 = self.operar()
        elif lexema1.operar(None)== "Valor2":
            n2 = self.lista_lexemas.pop(0)
            if n2.operar(None) == "[":
                n2 = self.operar()

        if operacion and n1 and n2:
            return Aritmetica(n1,n2,operacion, f'Inicio: {operacion.getFila()}:{operacion.getColumna()}', f'Fin: {n2.getFila()}:{n2.getColumna}')
        elif operacion and n1 and operacion.operar(None) == ('Seno' or 'Coseno' or 'Tangente'):
            return Trigonometricas(n1, operacion, f'Inicio: {operacion.getFila()}: {operacion.getColumna()}', f'Fin: {n1.getFila()}:{n1.getColumna()}')
    return None
```

Función operar_: Es un bucle que llama repetidamente al método operar() y almacena los resultados de cada operación en una lista y luego imprime los resultados de cada operación. También cuenta el número de operaciones realizadas y lo muestra junto con los resultados.

```
def operar_(self):
    instruc= []
    contador = 1
    while True:
        operacio = self.operar()
        if operacio:
            instruc.append(operacio)
        else:
            break
    for i in instruc:
        print(f"Resultado operación {contador}: ",i.operar(None))
        #print(i.operar.tipo(None))
        print("**********************************************")
        contador+=1
```

La función instruccion: se encarga de analizar una cadena de texto y convertirla en una lista de lexemas (tokens) y una lista de errores. La función utiliza un bucle while para recorrer la cadena de texto caracter por caracter, y en cada iteración, comprueba si el caracter actual es una comilla, un número, un corchete, una tabulación, un salto de línea o un carácter de espacio en blanco, entre otros. Dependiendo del tipo de caracter que se encuentre, la función crea un nuevo lexema y lo agrega a la lista de lexemas, o bien agrega un nuevo error a la lista de errores. Al final de la función, se devuelve la lista de lexemas y la lista de errores.

```
def instruccion(self, cadena:str):
    lexema = ""
    count = 0
    while cadena:
        i = cadena[count]
        count += 1
        if i == '\"':
            lexema, cadena = self.armar_lexema(cadena[count:])
            #Si cadena no retorna nulo
            if lexema and cadena:
                self.col +=1
                l = Lexema(lexema, self.fil, self.col)

                #Aqui almacena todas las palabras lexemas
                self.lista_lexemas.append(l)
                #se le suma el tamaño del lexema a la columna y se le suma 1 como inicia el 0
                self.col += len(lexema) + 1
                #y se reinicia el count
                count = 0
        #elif i.isdigit():
        elif ord(i) >= 48 and ord(i) <= 57:
            #int(i)
            token, cadena = self.num(cadena)
            if token and cadena:
                self.col +=1
                #arma el lexema
                n = Lexema(token, self.fil, self.col)
                self.lista_lexemas.append(n)

                self.col += len(str(token))+1
                count = 0

        elif i == "[" or i == "]":
            c = Lexema(i, self.fil, self.col)
            self.lista_lexemas.append(c)
            cadena = cadena[1:]
            count = 0
            self.col += 1
        #Si encinetra una taulación
        elif i == "\t":
            #Se le suman 4 espacios a la columna
            self.col += 4
            # y se eliminan esos 4 espacios
            cadena = cadena[4:]
            count = 0
        elif i =="\n":
            cadena = cadena[1:]
            count = 0
            self.fil += 1
            #Se reinicia la columna a 1
            self.col = 1
        elif i == ' ' or i == '\r' or i == "{" or i == '}' or i == ','or i == '.' or i == ':':
            self.fil += 1
            cadena = cadena[1:]
            count = 0

        else:
            self.lista_errores.append(Errores(i, self.fil, self.col))
            cadena = cadena[1:]
            count = 0
            self.col += 1

    return self.lista_lexemas, self.lista_errores
```

Clase abstracción: Esta clase se usa como plantilla la cual hereda la clase ABC y el @abstractmethod son todas las funciones que van a heredar donde se vaya a heredar esta clase de Expresion.

```python
from abc import ABC, abstractmethod

class Expresion(ABC):
    def __init__(self, fila, columna):
        self.fila = fila
        self.columna = columna

    @abstractmethod
    def operar(self, arbol):
        pass

    @abstractmethod
    def getFila(self):
        return self.fila

    @abstractmethod
    def getColumna(self):
        return self.columna
```

Clase Lexema: Aquí se heredan los atributos y funciones de la clase Expresion solo que aquí se agrega la variable lexema la cual va a ser retornada cuando se mande a llamar la función operar

```python
from abstraccion import Expresion

class Lexema(Expresion):
    def __init__(self, lexema, fila, columna):
        self.lexema = lexema
        super().__init__(fila, columna)

    def operar(self, arbol):
        return self.lexema

    def getFila(self):
        return super().getFila()

    def getColumna(self):
        return super().getColumna()
```

Clase Trigonometricas: Se herada lo de la plantilla de Expresion y con la función operar se realiza la validación igualando a que si una palaba corresponde al lexema encontrado se realizan las operaciones de seno, cose, tangente con la ayuda la de la librería math. En esta clase solo se agrega las variables tipo de operación y un valor porque las trigonométricas solo necesitan de un valor para ser operadas.

```python
from abstraccion import Expresion
from math import *

class Trigonometricas(Expresion):
    def __init__(self, L, tipo, fila, columna):
        self.L = L
        self.tipo = tipo
        super().__init__(fila, columna)

    def operar(self, arbol):
        left = ''
        if self.L != None:
            left = self.L.operar(arbol)

        if self.tipo.operar(arbol) == "Seno":
            print(f"La operación es Seno de {left}" )
            return sin(left)
        if self.tipo.operar(arbol) == "Coseno":
            print(f"La operación es Coseno de {left}" )
            return cos(left)
        if self.tipo.operar(arbol) == "Tangente":
            print(f"La operación es Tangente de {left}" )
            return tan(left)
        else:
            return 0

    def getFila(self):
        return super().getFila
    def getColumna(self):
        return super().getColumna
```

Clase errores: En esta clase se heredan los atributos y las funciones de errores y en la función de operar se debe de recibir como parámetro la variable operar esta servirá para recibir todos los caracteres que no pertenecen a las palabras reservadas y se realiza un return con todos los atributos necesarios para el archivo de errores.

```python
from abstraccion import Expresion
class Errores(Expresion):
    def __init__(self,lexema, fila, columna):
        self.lexema = lexema
        super().__init__(fila, columna)

    def operar(self, error):
        no_= f'\t\t"No.": {error}\n'
        desc = '\t\t"Descripción-Token": {\n'
        lex = f'\t\t\t"Lexema": {self.lexema}\n'
        tipo = '\t\t\t"Tipo": Error\n'
        fila = f'\t\t\t"Fila": {self.fila}\n'
        col = f'\t\t\t"Columna": {self.columna}\n'
        fin = '\t\t},\n'
        return '\t{\n' + no_+ desc + lex + tipo + fila + col + fin + '\t}'

    def getColumna(self):
        return super().getColumna()
    def getFila(self):
        return super().getFila()
```

Clase Aritmeticas: En esta clase se heredan los atributos de Expresion y se le agregan 3 atributos más los cuales son los 2 dígitos para ser operados y el tipo de operación. En la función operar se realiza la validación que si el tipo de operación pertenece al lexema que se ha igualado regrese y realice la operación asignada.

```python
from abstraccion import Expresion

class Aritmetica(Expresion):

    def __init__(self,L ,R , tipo, fila, columna):
        self.L = L
        self.R = R
        self.tipo = tipo
        super().__init__(fila, columna)


    def operar(self, arbol):
        Lvalue = ''
        Rvalue = ''

        if self.L != None:
            Lvalue = self.L.operar(arbol)
            print(self.L.operar(arbol))
        if self.R != None:
            Rvalue = self.R.operar(arbol)

        if self.tipo.operar(arbol) == 'Suma':
            print(f"La operación es Suma {Lvalue} + {Rvalue}" )
            #print(self.tipo.operar)
            return Lvalue + Rvalue
        elif self.tipo.operar(arbol) == 'Resta':
            print(f"La operación es Resta {Lvalue} - {Rvalue}" )
            #print(self.tipo.operar)
            return Lvalue - Rvalue
        elif self.tipo.operar(arbol) == 'Multiplicacion':
            print(f"La operación es Multiplicación {Lvalue} * {Rvalue}" )
            return Lvalue * Rvalue
        elif self.tipo.operar(arbol) == 'Division':
            print(f"La operación es División {Lvalue} / {Rvalue}" )
            return Lvalue / Rvalue
        elif self.tipo.operar(arbol) == 'Modulo':
            print(f"La operación es Modulo {Lvalue} % {Rvalue}" )
            return Lvalue % Rvalue
        elif self.tipo.operar(arbol) == 'Potencia':
            print(f"La operación es potencia {Lvalue}^{Rvalue}" )
            return Lvalue ** Rvalue
        elif self.tipo.operar(arbol) == 'Raiz':
            print(f"La operación es Raiz {Lvalue}^1/{Rvalue}" )
            return Lvalue ** (1/Rvalue)
        elif self.tipo.operar(arbol) == 'Inverso':
            print(f"La operación es Inverso 1/{Lvalue}")
            return 1/Lvalue

        else:

            return 0


    def getFila(self):
        return super().getFila

    def getColumna(self):
        return super().getColumna
```

Paradigmas

Librería Math:

La librería "math" en Python proporciona un conjunto de funciones matemáticas para realizar operaciones matemáticas avanzadas. Estas funciones están escritas en C y son mucho más eficientes que si las escribiéramos en Python desde cero.

Librería Tkinter:

Tkinter es una librería del lenguaje de programación Python y funciona para la creación y el desarrollo de aplicaciones de escritorio. Esta librería facilita el posicionamiento y desarrollo de una interfaz gráfica de escritorio con Python. Tkinter es el paquete estándar de Python para interactuar con Tkt.

Modulo ABC:

En Python, el módulo "abc" proporciona la funcionalidad para definir clases abstractas. Las clases abstractas son clases que no se pueden instanciar directamente, sino que se utilizan como plantillas para otras clases que heredan de ellas. La sintaxis "from abc import ABC, abstractmethod" importa dos elementos clave del módulo "abc" en su programa:

La clase ABC: Esta es una clase de Python que se utiliza para definir clases abstractas. Para crear una clase abstracta, simplemente haga que su clase herede de ABC

El decorador abstractmethod: Este es un decorador que se utiliza para marcar un método como abstracto en una clase abstracta. Los métodos abstractos son métodos que no tienen una implementación concreta en la clase abstracta, pero que deben ser implementados en las clases que heredan de ella. Para marcar un método como abstracto, simplemente agregue el decorador @abstractmethod encima de su definición.

Gramáticas

Gramática de analizador

Σ = { { , “Operación”, : ,"Suma", "Resta", "Multiplicación", "División", ”Potencia”, ”Raiz”, ”Inverso”, ”Seno”, ”Coseno”, ”Tangente”, ”Mod” , “Valor1”, DIGITO, [,], “Valor2”}

N = {X0, X1, X2, X3, X4, X5, X6, X7, X8, X9, X10, X11, X12, X13, X14,X15}

Inicio = X0

Producciones

X0 -> { X1

X1 -> “Operación” X2

X2 -> : X3

X3 -> "Suma"|"Resta"|"Multiplicación"|"División"|”Potencia”|”Raiz”|”Inverso”|”Seno”

|”Coseno”|”Tangente”|”Mod” X4

X4 -> "Valor1" X5

X5 -> : X6

X6 -> DIGITO X8 | [X7

X7 -> X1 |] X8

X8 -> "Valor2" X19

X9 -> : X10

X10 -> DIGITO X12| [X11

X11 -> X1 |] X12

X12 -> } X13 | }, X0

X13 Epsilon

Gramática de DIGITO

V = {0,1,2,3,4,5,6,7,8,9}

P = {.}

Σ = {V, .}

N = {D0, D1, D2}

Inicio = D0

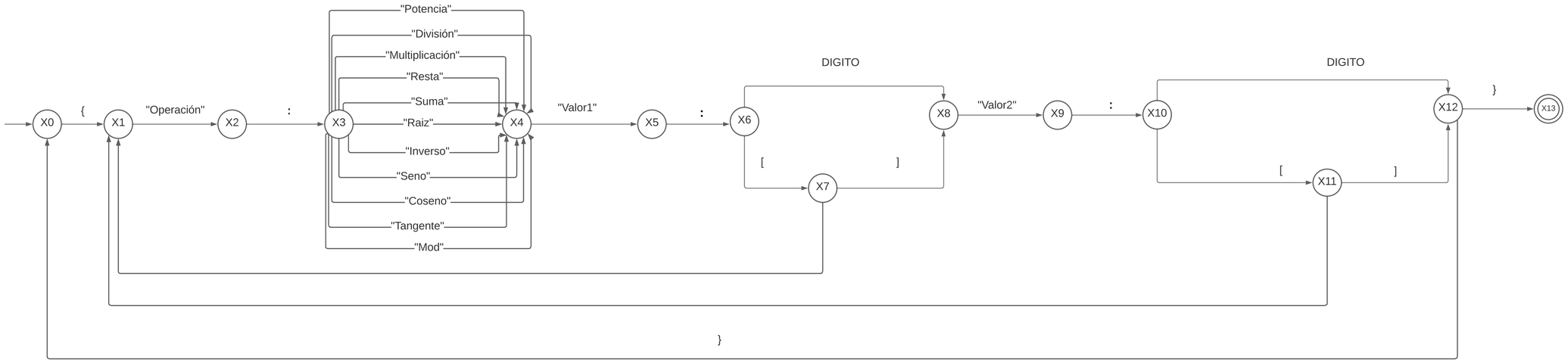
Producciones

D0 -> V D0 | . D1| Epsilon

D1 -> V D2

D2 -> V D2 | Epsilon

AUTOMATA ANALIZADOR



AUTOMATA DIGITO