Universidad de San Carlos de Guatemala
Facultad de Ingeniería
Carrera: Ingeniería en ciencias y sistemas
Catedrático: Ing. Mario Bautista
Auxiliar: José Puac
Curso: Organización de lenguajes y compiladores 1
Sección **“N”**

MANUAL TECNICO

José Abraham Solórzano Herrera
201800937

Guatemala 4 de Julio del 2021

INTRODUCCION

El software es un intérprete de alto nivel, es un lenguaje exclusivo de la Universidad de San Carlos de Guatemala, el cual se llama JPR, consiste en un editor, cuya finalidad es proporcionar ciertas funcionalidades, características, herramientas que serán de utilidad al usuario. La funcionalidad del editor será el ingreso del código fuente que será analizado, donde podrá aceptar archivos con extensión ".jpr" y mostrará la línea actual, está diseñado en el lenguaje de programación Python, por medio de una librería PLY.

OBJETIVOS

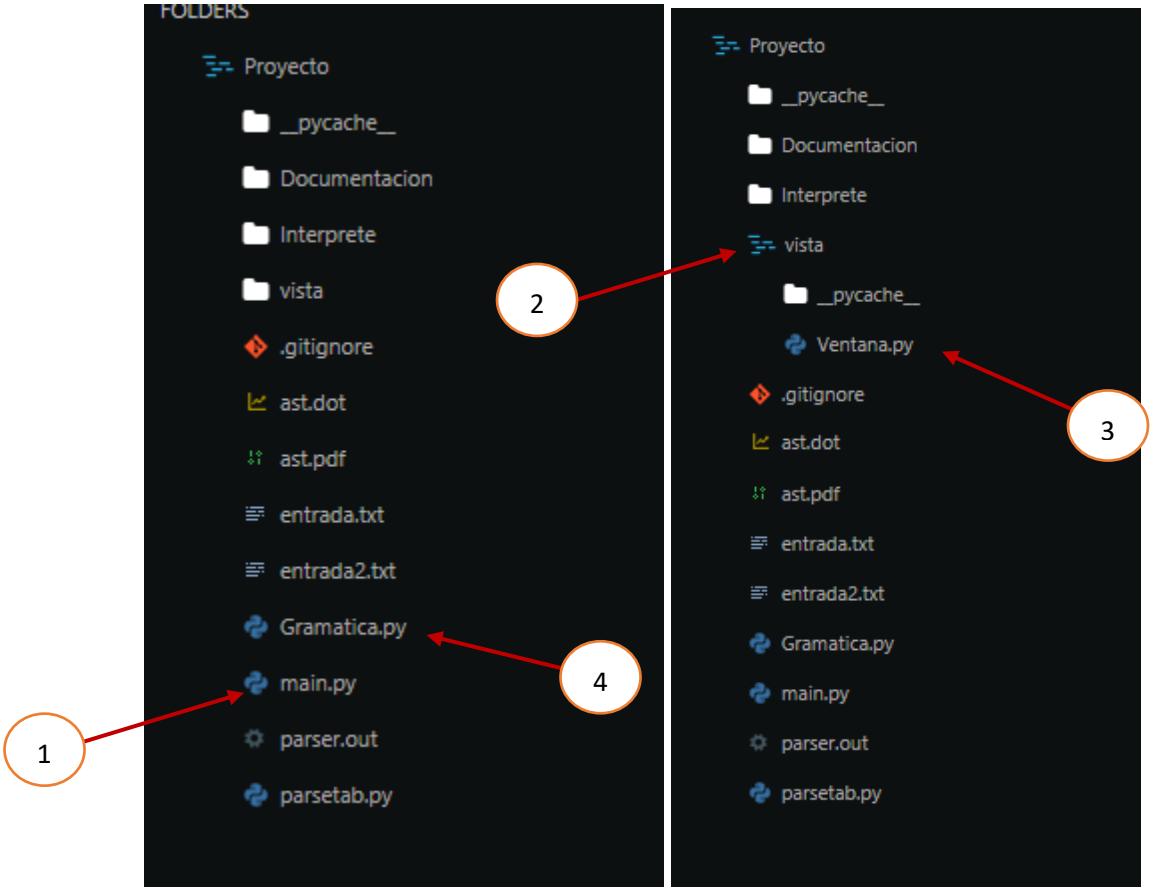
Objetivo General:

Aplicar los conocimientos sobre la fase de análisis léxico, sintáctico y semántico de un compilador para la realización de un intérprete completo, con las funcionalidades principales para que sea funcional.

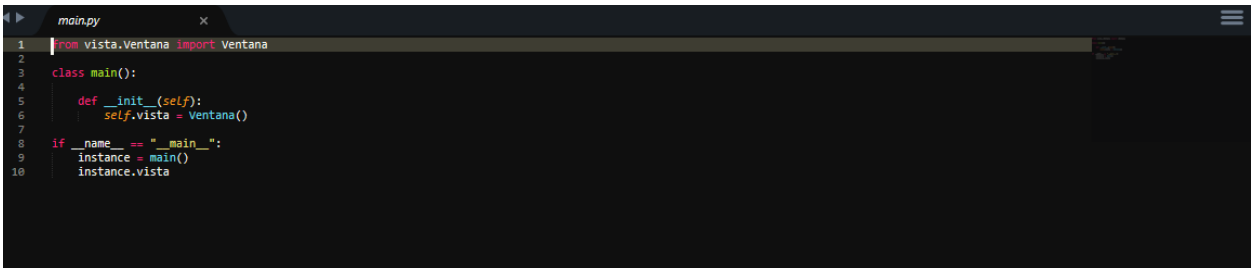
Objetivo Especifico:

1. Reforzar los conocimientos de análisis léxico, sintáctico y semántico para la creación de un lenguaje de programación.
2. Aplicar los conceptos de compiladores para implementar el proceso de interpretación de código de alto nivel.
3. Aplicar los conceptos de compiladores para analizar un lenguaje de programación y producir las salidas esperadas.
4. Aplicar la teoría de compiladores para la creación de soluciones de software.

MANUAL TECNICO



1: main.py: Esta clase hace el llamado de la instancia **Ventana.py "3"**.



2: Vista: Esta es la carpeta que contiene la clase **Ventana.py "3"**.

3: Ventana.py: En esta clase tendra todo lo del editor, se explicara a continuancion la funcionalidad.

4: Gramatica.py En esta clase se detalla la gramatica utilizada en el proyecto, puede revisar en el manual de Gramtica.

```python
'''
    DOCUMENTACION -> https://likegeeks.com/es/ejemplos-de-la-gui-de-python/
    DOCUMENTACION -> https://guia-tkinter.readthedocs.io/es/develop/chapters/6-widgets/6.1-Intro.html
    DOCUMENTACION -> https://python-intermedio.readthedocs.io/es/latest/args_and_kwargs.html
'''

from tkinter import ttk, Tk, Label, Menu, Button, Entry, StringVar, Scrollbar, Frame, Text, scrolledtext
import tkinter as tk
from tkinter import *
from tkinter import filedialog
from tkinter import messagebox
import tkinter.font as tkFont
import webbrowser as wb


class Ventana():

    def __init__(self):
        self.text_global = ""
        self.filename = ""

        self.root_window = Tk()
        self.root_window.configure(bg='#000000')

        self.root_window.geometry("1350x670+0+0")
        self.root_window.title("Ventana Principal")

```

Se inicia importando toda la Liberia tkinter. Luego se crea la raíz de tkinter, se le proporciona un tamaño y un color definido en hexadecimal.

```python
        # ------------------------------------ BARRA MENU ----------------------------------
        self.barra_menu = Menu(self.root_window)

        # ------------------------------------ CARGAR ARCHIVO ----------------------------------
        self.open_file = Menu(self.barra_menu, bg='#000000', foreground="#ffffff")
        self.open_file.add_command(label="Crear Archivo", command=lambda: self.crear_archivo())
        self.open_file.add_command(label="Abrir Archivo", command=lambda: self.abrir_archivo())
        self.open_file.add_command(label="Guardar", command=lambda: self.guardar())
        self.open_file.add_command(label="Guardar Como", command=lambda: self.guardar_como())


        self.barra_menu.add_cascade(label="Archivo", menu=self.open_file)
        # ------------------------------------ HERRAMIENTAS ----------------------------------
        self.herramienta = Menu(self.barra_menu, bg='#000000', foreground="#ffffff")
        self.herramienta.add_command(label="Interpretar", command=self.interpretar)
        self.herramienta.add_command(label="Debugger")


        self.barra_menu.add_cascade(label="Herramientas", menu=self.herramienta)
        # ------------------------------------ REPORTES ----------------------------------
        self.reporte = Menu(self.barra_menu, bg='#000000', foreground="#ffffff")
        self.reporte.add_command(label="Reporte de Errores")
        self.reporte.add_command(label="Generar Árbol AST (Árbol de Análisis Sintáctico)", command=lambda:self.reporte_ast())
        self.reporte.add_command(label="Reporte de Tabla de Simbolos")


        self.barra_menu.add_cascade(label="Reporte", menu=self.reporte)

        # ------------------------------------ AYUDA ----------------------------------
        self.ayuda = Menu(self.barra_menu, bg='#000000', foreground="#ffffff")
        self.ayuda.add_command(label="opcion 1")


        self.barra_menu.add_cascade(label="Reporte", menu=self.ayuda)

```

Se inicia creando la Barra de Menú, puede visualizarla en el manual de usuario Inciso [5-14]. Se crea la pestaña archive, herramientas, reportes y ayuda (No tiene funcionamiento).

```python
        self.barra_menu.add_cascade(label="Reporte", menu=self.ayuda)

        # ------------------------------------ CONSOLA Y PARTE DE CODIGO ----------------------------------

        self.editor = Label(self.root_window, text="Editor", bg='#000000', foreground="#ffffff")
        self.editor.pack()
        self.editor.place(x=20, y=1)

        self.label_position = Label(self.root_window, bg='#000000', foreground="#ffffff")
        self.label_position.pack()
        self.label_position.place(x=55, y=1)

        self.scroll = ScrollText(self.root_window, bg='#000000') # aqui va el codigo fuente
        self.scroll.pack()
        self.scroll.place(x=10, y=25)
        #self.scroll.text.focus()
        #self.root_window.after(200, self.scroll.redraw())


        self.fontStyle = tkFont.Font(family="Courier New",size=8)
        self.scroll_consola = scrolledtext.ScrolledText(self.root_window, height=23, width=80, font=self.fontStyle, bg='#000000', foreground="#ffffff") # conso
        #self.scroll_consola.configure(state='disabled')
        self.scroll_consola.pack()
        self.scroll_consola.place(x=750, y=25)

```

En Esta parte se crea la parte del codigo fuente y la salida, puede visualizala en el manal de usuario Inciso [1-2].

```python
# ------------------------  PARTE DE REPORTES Y TABLA DE SIMBOLOS --------------------------


self.root_tab_control = ttk.Notebook(self.root_window, height=210, width=1250) # Creando la raiz de las petañas


self.root_tab1 = tk.Frame(self.root_tab_control, bg='#000000') # Creando la pestaña
self.root_tab1.pack()
# self.root_tab1.config(bg='#000000')
self.root_tab_control.add(self.root_tab1, text="Tabla de Simbolos")

self.root_tab2 = tk.Frame(self.root_tab_control, bg='#000000') # Creeando otra pestaña
self.root_tab2.pack()
self.root_tab_control.add(self.root_tab2, text="Reporte de Errores")


self.lb1 = Label(self.root_tab1, text="Tabla de Simbolos", bg='#000000', foreground="#ffffff")
self.lb1.pack()
self.lb2 = Label(self.root_tab2, text="Tabla de Reportes", bg='#000000', foreground="#ffffff")
self.lb2.pack()

self.root_tab_control.pack(expand=1, fill="both")
self.root_tab_control.place(x=50, y=400)
```

En Este apartado se crea una raiz de pestaña que sera la encargada de almacenar la tabla de simbolos(No tiene funcionamiento) y tabla de errores. Puede visualizarla en el manual de usuario en el Inciso [21-23].

```python
self.root_tab_control.place(x=50, y=400)

#TABLA SIMBOLOS
self.table = ttk.Treeview(self.root_tab1)
self.table['columns'] = ('NO', 'ID', 'TIPO', 'DIMENSION', 'VALOR', 'AMBITO', 'REFERENCIAS')
self.table.column('#0', width=0, stretch=NO)
self.table.column('NO', anchor=CENTER, width=3)
self.table.column('ID', anchor=CENTER)
self.table.column('TIPO', anchor=CENTER)
self.table.column('DIMENSION', anchor=CENTER)
self.table.column('VALOR', anchor=CENTER)
self.table.column('AMBITO', anchor=CENTER)
self.table.column('REFERENCIAS', anchor=CENTER)

self.table.heading('#0', text='', anchor=CENTER)
self.table.heading('NO', text='#', anchor=CENTER)
self.table.heading('ID', text='Id', anchor=CENTER)
self.table.heading('TIPO', text='Tipo', anchor=CENTER)
self.table.heading('DIMENSION', text='Dimension', anchor=CENTER)
self.table.heading('VALOR', text='Valor', anchor=CENTER)
self.table.heading('AMBITO', text='Ambito', anchor=CENTER)
self.table.heading('REFERENCIAS', text='Referencias', anchor=CENTER)
"""
i = 0
while i < 10:
    self.table.insert(parent='', index=i, iid=i, text='', values=(f'{i+1}', f'id{i}',f'tipo{i}',f'dime{i}',f'val{i}',f'amb{i}',f'refe{i}'))
    i += 1
"""
# self.table.tag_configure("gray", background="gray")
self.table.pack()
```

En este apartado se le asignan las respectivas filas y columnas de la tabla de simbolos (No tiene funcionamiento). Puede visualizarla en el manual de usuario Inciso [22].

```python
"""
# self.table.tag_configure("gray", background="gray")
self.table.pack()

# TABLA DE ERRORES
self.table_error = ttk.Treeview(self.root_tab2)
self.table_error['columns'] = ('NO', 'TIPO', 'DESCRIPCION', 'LINEA', 'COLUMNA')
self.table_error.column('#0', width=0, stretch=NO)
self.table_error.column('NO', anchor=CENTER, width=3)
self.table_error.column('TIPO', anchor=CENTER)
self.table_error.column('DESCRIPCION', anchor=CENTER)
self.table_error.column('LINEA', anchor=CENTER)
self.table_error.column('COLUMNA', anchor=CENTER)

self.table_error.heading('#0', text='', anchor=CENTER)
self.table_error.heading('NO', text='#', anchor=CENTER)
self.table_error.heading('TIPO', text='Tipo', anchor=CENTER)
self.table_error.heading('DESCRIPCION', text='Dimension', anchor=CENTER)
self.table_error.heading('LINEA', text='Fila', anchor=CENTER)
self.table_error.heading('COLUMNA', text='Columna', anchor=CENTER)

self.table_error.pack()
```

En este apartado se le asignan las respectivas filas y columnas de la tabla de errores. Puede visualizarla en el manual de usuario Inciso [23].

```
        # ---------------------------------- POSICION DEL MOUSE Y COLORES---------------------------------

        self.scroll.text.bind("<Button-1>", self.get_posicion_mouse)
        self.scroll.text.bind("<Button-2>", self.get_posicion_mouse)
        self.scroll.text.bind("<Button-3>", self.get_posicion_mouse)


        self.scroll.text.tag_config('tk_reseverda',     foreground='#0317D8')     # Palabras reservadas   - Azul
        self.scroll.text.tag_config('tk_cadena',        foreground='#ff860d')     # Cadenas o caracteres  - Anaranjado
        self.scroll.text.tag_config('tk_numero',        foreground='#a33aff')     # Numeros               - Morado
        self.scroll.text.tag_config('tk_comentario',    foreground='#646464')     # Comentario            - Gris
        self.scroll.text.tag_config('tk_id',            foreground='#45f50d')     # id                    - Verde Claro
        self.scroll.text.tag_config('tk_otro',          foreground='#ffffff')     # Otros                 - Negro




        # ---------------------------------- FIN DE TK ----------------------------------------

        self.root_window.config(menu=self.barra_menu)
        self.root_window.mainloop()
```

En esta parte manda a setear todos los colores a al codigo fuente puede visualizarlo en el manual de usuario Inciso [15-20]

Ventana.py

```
    # ---------------------------------- POSICION ----------------------------------

    def get_posicion_mouse(self, *args, **kwargs):
        position = self.scroll.text.index(INSERT)
        self.label_position.destroy()
        self.label_position = Label(self.root_window, text=f"{position}", bg='#000000', foreground="#ffffff")
        self.label_position.pack()
        self.label_position.place(x=55, y=1)
```

En este apartdo se crea la parte donde muestra la posicion del cursor, puede visualizarlo en el manual de usuario Inciso [4].

```
    # ------------------------------------------------------- ARCHIVO -----------------------------------------------------

    def crear_archivo(self):
        print('\n---------------')
        result=self.scroll.text.get(1.0, tk.END+"-1c")
        print(result)
        print('---------------')

        self.filename = filedialog.asksaveasfilename(initialdir = "/",title = "Guardar como",filetypes = (("txt files","*.jpr"),("todos los archivos","*.*")))

        if self.filname!='':
            archi1=open(self.filename, "w")
            archi1.write(result)
            archi1.close()
```

En este apartado se crea una funcion que permita crear archivos por medio de la liberia filedialog, puede visuarlizarlo en el manual de usurio Inciso [6].

Ventana.py

```
    def abrir_archivo(self):
        try:
            ruta =  ""
            self.filename =  filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("JPR files","*.jpr"),("all files","*.*")))
            ruta = self.filename
            if ruta != "":
                try:
                    archivo = open(ruta, "r")
                    texto = archivo.read()

                    self.scroll.text.delete("1.0","end")
                    for array in self.set_paint(texto):
                        self.scroll.text.insert(INSERT, array[1], array[0])
                    archivo.close()
                except (FileNotFoundError):
                    print("Error")
            else:
                return None

        except IndexError as e:
            print(e)
```

En este apartado se crea una función donde se podrán abrir archivos con extensión .jpr, puede visualizarlo en el manual de usuario, Inciso [7].

```
231
232     def guardar(self):
233
234         if self.filename != "":
235             result=self.scroll.text.get(1.0, tk.END+"-1c")
236             archivo = open(self.filename, "w")
237             archivo.write(result)
238             archivo.close()
239         else:
240             messagebox.showinfo(message="No hay Documento abierto", title="Gaurdar")
241
242     def guardar_como(self):
243
244         self.filename = filedialog.asksaveasfilename(initialdir = "/", title = "Seleccione el Archivo", defaultextension=".jpr", filetypes = (("jpr files","*.jpr"),("all files","*.*")))
245         result = self.scroll.text.get(1.0, tk.END+"-1c")
246
247         if self.filename!='':
248             archi=open(self.filename, "w")
249             archi.write(result)
250             archi.close()
251
```

En este apartado puede visualizar el funcionamiento de la opción guardar y guardar como, puede visualizarlo en el manual de usuario Inciso [8-9].

Ventana.py

```
250             archi.close()
251
252     # ------------------------------------------------------- PINTAR ------------------------------------------------------
253     def set_paint(self, texto):
254         global_lista = []
255         temp = ''
256
257         i = 0
258         while i < len(texto):
259
260             if (re.search(r'[0-9]', texto[i])): # Aqui si el valor es solo un numero.
261                 temp += texto[i] # Esto hace que se concatene el temp con el texto.
262                 i = i + 1
263
264                 while (i != len(texto)):
265                     if (re.search(r'[0-9]', texto[i])):
266                         temp += texto[i]
267                         i = i + 1
268
269                     elif (texto[i] == '.'):
270                         temp += texto[i]
271                         i = i + 1
272
273                     if (not texto[i].isnumeric()) and (texto[i] != '.'):
274                         auxiliar_lista = ["tk_numero", temp]
275                         global_lista.append(auxiliar_lista)
276                         temp = ''
277                         i = i - 1
278                         break
279
280                 if (len(temp) != 0):
281                     auxiliar_lista = ['tk_numero', temp]
282                     global_lista.append(auxiliar_lista)
283                     temp = ''
284
285             elif (re.search(r'[a-zA-Z0-9_]', texto[i])): # Si el valor es una variable..
286                 temp += texto[i]
287                 i = i + 1
288
289                 while (i != len(texto)):
290                     if (i + 1 != len(texto)):
291                         if not (re.search(r'[a-zA-Z0-9_]', texto[i + 1])):
292                             temp += texto[i]
293                             auxiliar_lista = ['tk_id', temp]
294                             global_lista.append(auxiliar_lista)
295                             temp = ''
296                             break
297
```

En este apartado se crea un analizador para poder reconocer la entrada del codigo, puede visualizarlo en el Inciso [1] y [15-20].

```python
# ---------- HERRAMIENTAS ----------
def interpretar(self):
    #INTERFAZ
    import Gramatica as prueba

    result=self.scroll.text.get(1.0, tk.END+"-1c")
    self.scroll.text.delete("1.0","end")
    for array in self.set_paint(result):
        self.scroll.text.insert(INSERT, array[1], array[0])

    AST = prueba.interprete(result, self.scroll_consola)

    self.scroll_consola.delete("1.0","end")
    self.scroll_consola.insert(tk.INSERT, AST.get_consola())

    for d in self.table_error.get_children(): # Esto sirve para resetear la tabla de errores.
        self.table_error.delete(d)

    i = 0
    for pedo in AST.get_excepcion():
        self.table_error.insert(parent='', index=i, iid=i, text='', values=(f'{i+1}',f'{pedo.tipo}',f'{pedo.descripcion}',f'{pedo.fila}',f'{pedo.columna}
        print(pedo)
        i += 1
```

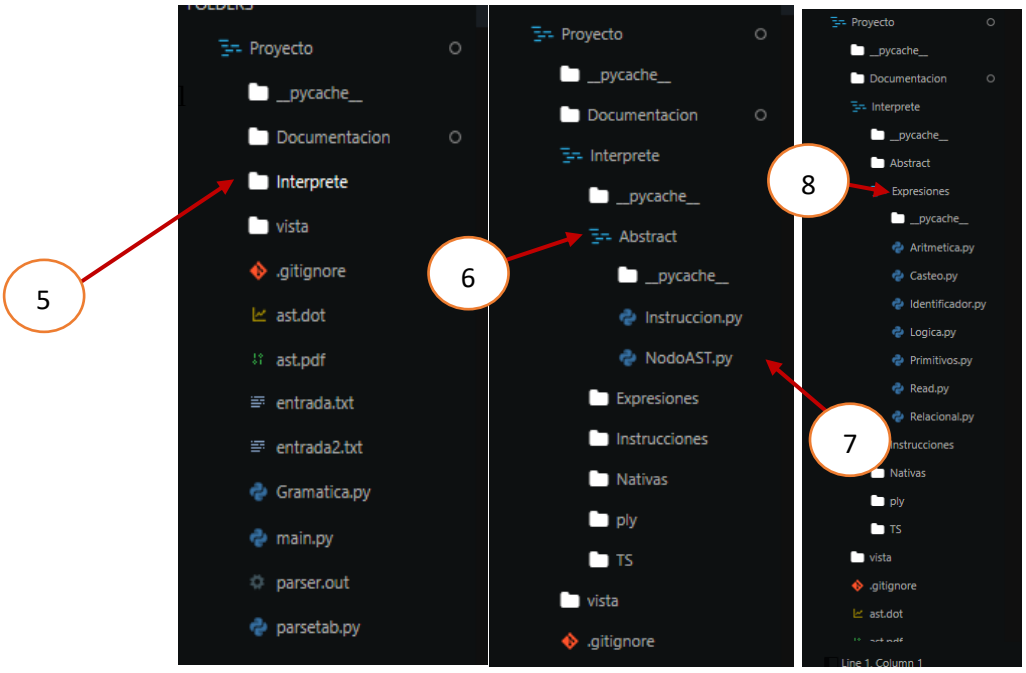
En enste apartado hace referencia a la Gramatica, la cual se crea una instancia de clase Gramatica la cual es encargada de interpretar todo del codigo fuente. Mas adelante se detallara a profundidad el punto **4, en el manual de gramatica.**

```python
            print(pedo)
            i += 1


    def debugger(self):
        pass

    # ---------- REPORTE ----------

    def reporte_error(self):
        pass
    def reporte_ast(self):
        try:
            #root = Tk()
            ruta =  ""
            filename =  filedialog.askopenfilename(initialdir = "/",title = "Select file",filetypes = (("TXT files","*.pdf"),("all files","*.*")))
            ruta = filename
            if ruta != "":
                wb.open_new(ruta)
                return ruta
            else:

                return None

        except IndexError as e:
            print(e)
```

En este apartado se encuentra el debugger (no tiene funcionalidad) y la visualización del reporte Árbol de análisis sintáctico (AST). Puede visualizar en el manual de usuario Inciso [11] y [13].

```python
class ScrollText(tk.Frame):
    def __init__(self, master, *args, **kwargs):
        tk.Frame.__init__(self, *args, **kwargs)

        self.fontStyle = tkFont.Font(family="Courier New",size=8)
        # bg -> color de fondo --- foreground -> color al texto --- selectbrackgroud -> color a lo que seleccione --- inserbackgroud -> color al puntero
        self.text = tk.Text(self, bg='#000000', foreground="#ffffff", selectbackground="#c2c2c2", insertbackground='#ffffff',  height=23, width=95, font=self.

        self.scrollbar = tk.Scrollbar(self, bg='#000000', orient=tk.VERTICAL, command=self.text.yview)
        self.text.configure(yscrollcommand=self.scrollbar.set)

        self.numero_lineas = TextoLinea(self, width=35, bg='#000000')
        self.numero_lineas.attach(self.text)
        self.scrollbar.pack(side=tk.RIGHT, fill=tk.Y)
        self.numero_lineas.pack(side=tk.LEFT, fill=tk.Y, padx=(5, 0))
        self.text.pack(side=tk.RIGHT, fill=tk.BOTH, expand=True)

        self.text.bind("<Key>", self.onPressDelay)
        self.text.bind("<Button-1>", self.numero_lineas.redraw)
        self.scrollbar.bind("<Button-1>", self.onScrollPress)
        self.text.bind("<MouseWheel>", self.onPressDelay)

    def onScrollPress(self, *args):
        self.scrollbar.bind("<B1-Motion>", self.numero_lineas.redraw)

    def onScrollRelease(self, *args):
        self.scrollbar.unbind("<B1-Motion>", self.numero_lineas.redraw)

    def onPressDelay(self, *args):
        self.after(2, self.numero_lineas.redraw)

    def get(self, *args, **kwargs):
        return self.text.get(*args, **kwargs)

    def insert(self, *args, **kwargs):
        return self.text.insert(*args, **kwargs)

    def delete(self, *args, **kwargs):
```

Esta función ScrollText se encarga de crear las líneas de código fuente, puede visualizarlo en el manual de usuario Inciso [14]



5. En la carpeta Interprete: Se desarrolla el patron de diseño Interprete.

6. Carpeta Abstract: Se inicia creando una clase abstracta.

7. Instrucción.py y NosoAST.py

Estas clases son las encargadas de representar el patron de diseño interprete, son las encargadas de heredar su funcion. A continuacion se muestra las clases.

Instruccion.py

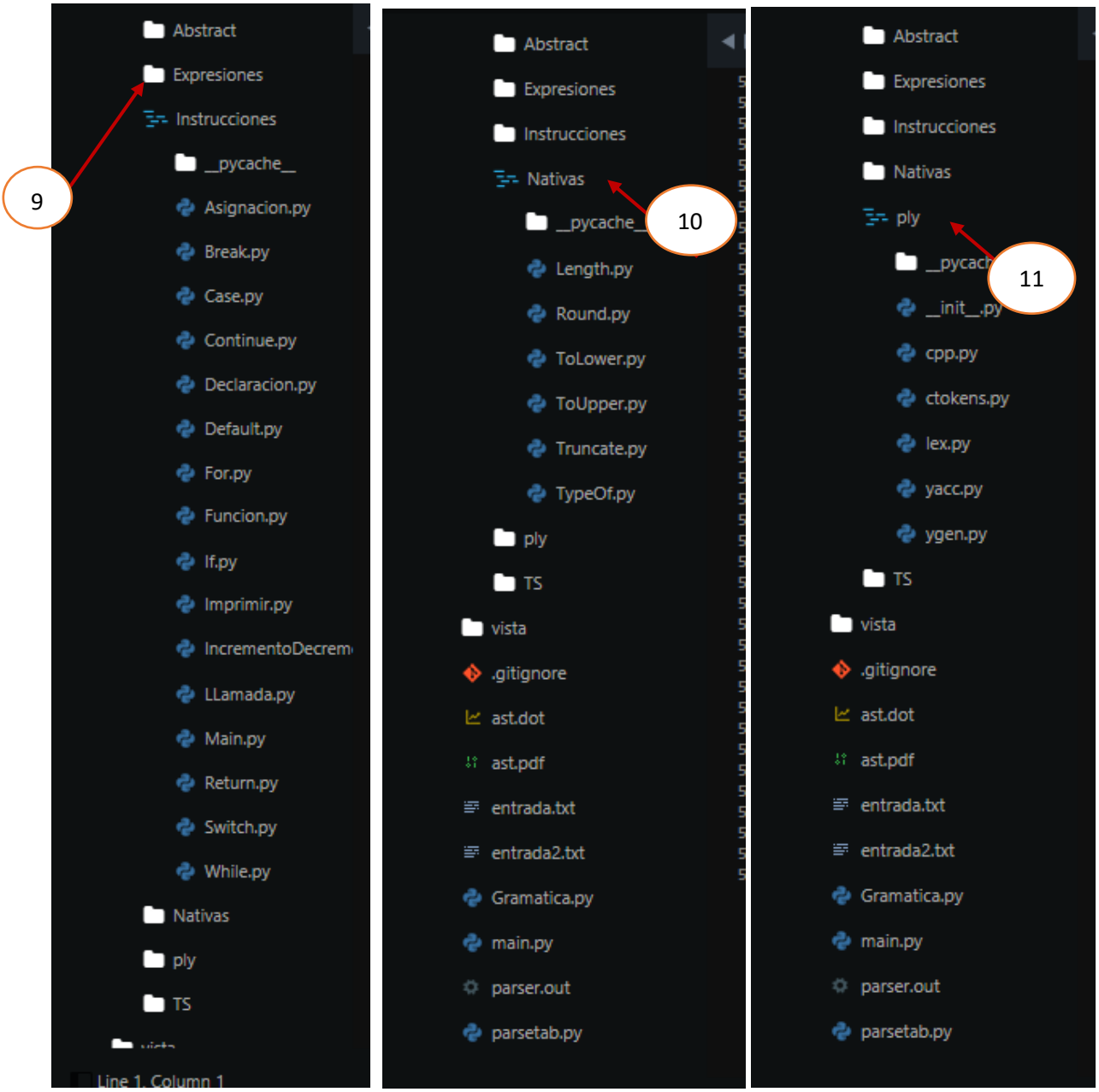
```python
'''
    ---------------------------------
    Implementación de clase  Abstracta.
    ---------------------------------
'''
from abc import ABC, abstractmethod

class Instruccion(ABC):

    def __init__(self, fila, columna):      # fila y columna, ingresa el punto exacto de la entrada
        self.fila = fila
        self.columna = columna
        super().__init__()

    @abstractmethod
    def interpretar(self, tree, table):
        pass

    @abstractmethod
    def getNodo(self):
        pass

```

Esta es la clase que se hereda a las siguiente.

NodoAST.py

```python
class NodoAST():
    def __init__(self, valor):
        self.hijos = []
        self.valor = valor

    def setHijos(self, hijos):
        self.hijos = hijos

    def agregarHijo(self, valorHijo):
        self.hijos.append(NodoAST(valorHijo))

    def agregarHijos(self, hijos):
        for hijo in hijos:
            self.hijos.append(hijo)

    def agregarHijoNodo(self, hijo):
        self.hijos.append(hijo)

    def agregarPrimerHijo(self, valorHijo):
        self.hijos.insert(0, NodoAST(valorHijo))

    def agregarPrimerHijoNodo(self, hijo):
        self.hijos.insert(0, hijo)

    def getValor(self):
        return str(self.valor)

    def setValor(self, valor):
        self.valor = valor

    def getHijos(self):
        return self.hijos
```

Esta es la construccion del Arbol de analisis sintactico AST.



9: Carpeta Expresiones: Se encarga de analizar todas las instrucciones posibles que la gramatica contiene, a continuacion se va a mostrar una imagen de la representacion de Expresiones.

Asignacion.py

```python
from Interprete.Expresiones.Identificador import Identificador
from Interprete.TS.Exception import Exception
from Interprete.Abstract.Instruccion import Instruccion
from Interprete.TS.Simbolo import Simbolo
from Interprete.Abstract.NodoAST import NodoAST


class Asignacion(Instruccion):

    def __init__(self, identificador, expresion, fila, columna):
        self.identificador = identificador
        self.expresion = expresion
        self.fila = fila
        self.columna = columna

    def interpretar(self, tree, table):
        value = self.expresion.interpretar(tree, table) # Valor a asignar a la variable
        if isinstance(value, Exception): return value

        simbolo = Simbolo(str(self.identificador).lower(), self.expresion.tipo, self.fila, self.columna, value)
        result = table.actualizarTabla(simbolo)

        if isinstance(result, Exception): return result
        return None

    def getNodo(self):
        nodo = NodoAST("ASIGNACION")

        nodo.agregarHijo(str(self.expresion.tipo))
        nodo.agregarHijo(str(self.identificador))
        nodo.agregarHijoNodo(self.expresion.getNodo())
        return nodo
```

Esta imagen representa la clase Asignacion, donde se crea su constructor con un identificador, que viene siendo el id, su expresion, que puede ser un entero, una cadena, decimal, carácter, etc, luego se guarda la variable en una tabla de simbolos, por ultimo se hace una comprobacion si es una excepion se guardaria en la tabla de errores.

10: Nativas, son las fucniones ya defenidas, son las que ya vienen por defecto, a continuacion se muestra un ejemplo de una funcion Nativa.

Length.py

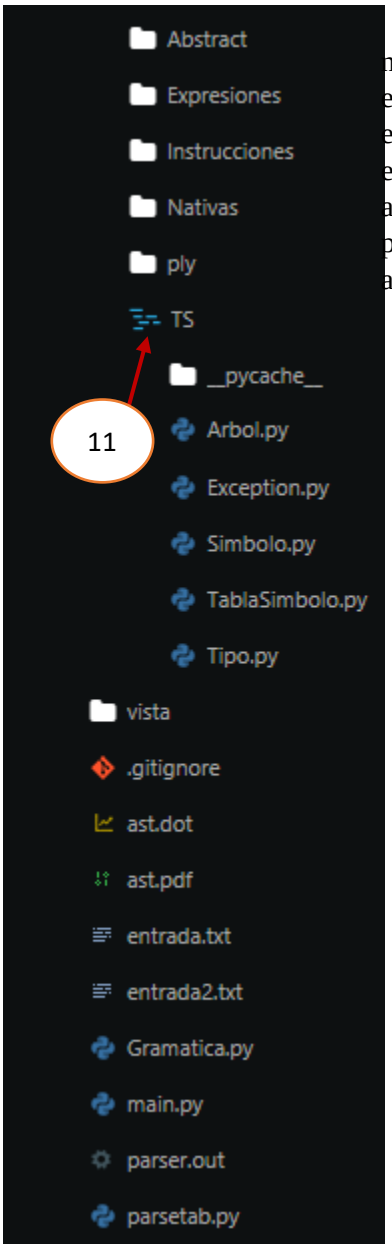
```python
from Interprete.TS.Exception import Exception
from Interprete.TS.Tipo import Tipo
from Interprete.Instrucciones.Funcion import Funcion

class Length(Funcion):
    def __init__(self, nombre, parametros, instrucciones, fila, columna):
        self.nombre = nombre.lower()
        self.parametros = parametros
        self.instrucciones = instrucciones
        self.fila = fila
        self.columna = columna
        self.tipo = Tipo.NULO

    def interpretar(self, tree, table):
        simbolo = table.getTabla("length##Param1")
        if simbolo == None : return Exception("Semantico", "No se encontró el parámetro de Length", self.fila, self.columna)

        if simbolo.get_tipo() != Tipo.CADENA:
            return Exception("Semantico", "Tipo de parametro de Length no es cadena.", self.fila, self.columna)

        self.tipo = simbolo.get_tipo()
        return len(simbolo.get_valor())
```

Esta clase Length, se encarga de crear un constructor que recibe como parametros el nombre, de la funcion, sus parametros que vienen siendo el Tipo de dato, identificador que es su id, y las instrucciones encargadas. Por medio de la tabla de simbolos podemos obtener el valor y asi retornar la longitud de dicho valor.

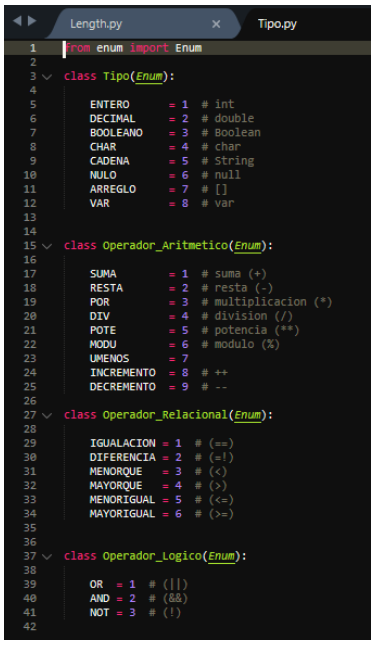
11: PLY.py: Es la liberia que se implento para el proyecto. Una breve descripcion y direccion de donde se puede obtener dicha documentacion https://www.dabeaz.com/ply/.

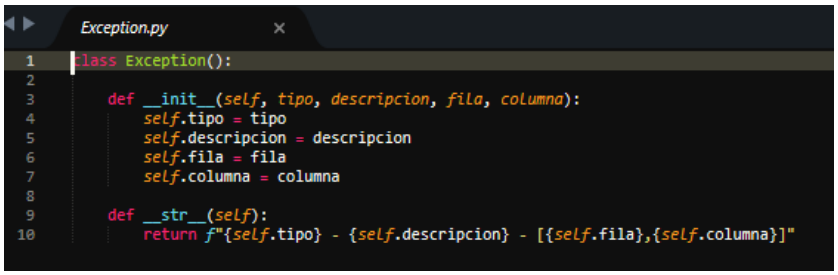
En pocas palabras, PLY no es más que una implementación sencilla de lex / yacc. Aquí hay una lista de sus características esenciales:

- Está implementado completamente en Python.
- Utiliza análisis LR que es razonablemente eficiente y muy adecuado para gramáticas más grandes.
- PLY proporciona la mayoría de las funciones estándar de lex / yacc, incluido el soporte para producciones vacías, reglas de precedencia, recuperación de errores y soporte para gramáticas ambiguas.
- PLY es fácil de usar y proporciona una comprobación de errores muy extensa.
- PLY no intenta hacer nada más o menos que proporcionar la funcionalidad básica de lex / yacc. En otras palabras, no es un gran marco de análisis o un componente de algún sistema más grande.



11: En la carpeta TS, se crean una de las clases mas importates que nos sirve para los ambitos, que viene sinedo Tabla simbolos, que es la encargada de crear los ambitos y guardarlos en sus siguientes, la clases excepcion es la encargada de guardar los errores, la clase simbolo es la encargada de gurdar todos los id, la clase tipo es la encargada de almacenar una clase enum donde se podran virificar todos los tipos que puedan venir, ya sean un entero, decimal, mas que todo operadores arimeticos, logicos, relacional.





TablaSimbolo.py

```python

from Interprete.TS.Exception import Exception
from Interprete.TS.Tipo import *

class TablaSimbolo:
    def __init__(self, anterior = None):
        self.tabla = {} # Diccionario Vacio
        self.anterior = anterior
        self.funciones = []

    def setTabla(self, simbolo):      # Agregar una variable
        if simbolo.id.lower() in self.tabla :
            return Exception("Semantico", "Variable " + simbolo.id + " ya existe", simbolo.fila, simbolo.columna)
        else:
            self.tabla[simbolo.id.lower()] = simbolo
            return None

    def getTabla(self, id):           # obtener una variable
        tablaActual = self
        while tablaActual.tabla != None:
            if id.lower() in tablaActual.tabla :
                return tablaActual.tabla[id.lower()]
            else:
                tablaActual = tablaActual.anterior
                if tablaActual is None:
                    return None
        return None

    def actualizarTabla(self, simbolo):
        tablaActual = self

        while tablaActual != None:
            if simbolo.id in tablaActual.tabla :
                if tablaActual.tabla[simbolo.id.lower()].get_tipo() == simbolo.get_tipo() or tablaActual.tabla[simbolo.id.lower()].get_tipo() is Tipo.NULO or simbolo.tipo is Tipo.NULO:
                    tablaActual.tabla[simbolo.id.lower()].set_valor(simbolo.get_valor())
                    tablaActual.tabla[simbolo.id.lower()].set_tipo(simbolo.get_tipo())
                    return None
                return Exception("Semantico", "Tipo de dato Diferente en Asignacion", simbolo.get_fila(), simbolo.get_columna())
            else:
                tablaActual = tablaActual.anterior
                if tablaActual is None:
                    return None
        return Exception("Semantico", "Variable No encontrada en Asignacion", simbolo.get_fila(), simbolo.get_columna())

```