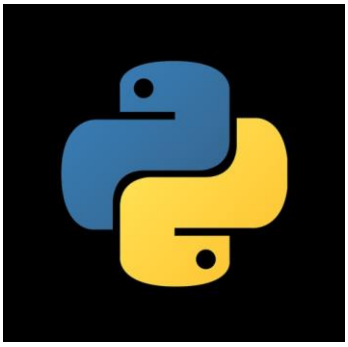
Laboratorio de Lenguajes Formales y de Programación, Sección B-.



PROYECTO 1 – FORMULARIOS DINAMICOS

MANUAL TECNICO

FECHA: 20/03/2022

Juan Josue Zuleta Beb
Carné: 202006353

Introducción

El presente documento describe los aspectos técnicos informáticos de la aplicación formularios dinámicos, deseñada a través de la interfaz gráfica de Tkinter. El documento familiariza al personal técnico especializado encargado de las actividades de mantenimiento, revisión, solución de problemas, instalación y configuración del sistema.

Objetivos

Instruir el uso adecuado del de la instalación y comprensión del código y de la implementación de métodos, para el acceso oportuno y adecuado en la inicialización de este, mostrando los pasos a seguir en el proceso de inicialización, así como la descripción de los archivos relevantes del sistema los cuales nos orienten en la configuración.

Requisitos Mínimos del Sistema

Sistema operativo 64 bits

- Microsoft Windows 10/8/7/Vista/2003/XP (incl.64-bit)
- macOS 10.5 o superior
- Linux GNOME o KDE desktop
- Procesador a 1.6 GHz o superior
- 1 GB (32 bits) o 2 GB (64 bits) de RAM (agregue 512 MB al host si se ejecuta en una máquina virtual)
- 3 GB de espacio disponible en el disco duro
- Disco duro de 5400 RPM
- Tarjeta de vídeo compatible con DirectX 9 con resolución de pantalla de 1024 x 768 o más.
- Navegador web (Recomendado: Google Chrome)

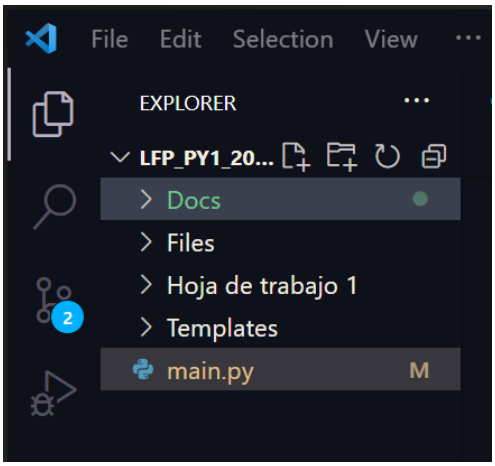
Software (Indispensable tenerlo instalado)

- IDE: Visual Studio Code
  Puede conseguirse en:
  https://code.visualstudio.com/

- Python 3.9 o posterior
  Puede conseguirse en:
  https://www.python.org/downloads/release/python-390/

Estructura raíz:

El proyecto tiene la siguiente estructura de directorios:



Directorio de la App: Raiz

La clase main:

Contiene el método principal de nuestra aplicación además de todas las clases y controlara todos los métodos y la implementación de la interfaz gráfica.

Las clases y métodos utilizados son los siguientes:

```
> class Token: ···

> class HTML_GUI: ···

> class Errors: ···

> def automata(starter: str): ···

> def scanner(tokens: List[Token]): ···

> def process_tokens(tokens): ···

> def process_errors(errs): ···

> def process_Gui(parameters): ···

> def process_input(text): ···

> class display_gui(): ···

if __name__ == '__main__':
    display_gui()
```

DESCRIPSION DE LOS METODOS Y CLASES:

Clase Token:

Tiene como función recibir un constructor con todos los datos de tipo token que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados.

```
class Token:
    def __init__(self, token: str, lexeme: str, row: int, col: int) -> None:
        self.token: str = token
        self.lexeme: str = lexeme
        self.row: int = row
        self.col: int = col
```

Clase HTML_GUI:

Tiene como función recibir un constructor con todos los datos de tipo dinámico que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados para la implementación del formulario dinámico.

```
class HTML_GUI:
    def __init__(self) -> None:
        self.tipo: str = ''
        self.valor: str = ''
        self.fondo: str = ''
        self.valores: list = []
        self.evento: str = ''
```

Clase Errors:

Tiene como función recibir un constructor con todos los datos de tipo error que sean reconocidos para que puedan ser almacenados en la memoria de la aplicación y posteriormente utilizados.

```
class Errors:
    def __init__(self, line: int, col: int, char: str) -> None:
        self.line: int = line
        self.col: int = col
        self.char: str = char
```

Función automata:

Tiene como función analizar el listado de caracteres escritos en el área de texto de la interfaz grafica de la aplicación. Esto lo realiza mediante la comparación de caracteres, reconociendo caracteres para los cuales fue preparado y para aquellos a los que detectara como errores, este es el principal encargado del funcionamiento de la aplicación, el diseño del autómata finito determinista se basó en el estudio del lenguaje presentado.

El proceso de diseño fue el siguiente:

Definir los Elementos:

S = símbolos = [~ < > [ ] , :]

R = Reservadas, eventos = [a - z, ñ]

A = Litarales, subLiterales= [A - Z, a – z, Ñ, ñ]

F = [: , - , espacio]

Nombre:	Patron:	Expression Regular:	Ejemplos:
Reservada	Inicia con una letra ala que le siguen cero o muchas veces cualquier combinación de letras.	**R+**	- formulario - tipo -valor
Símbolos	Símbolo representado directamente, puede aparecer una vez cualquier símbolo.	**S**	- ~ - > - <
Literal	Inicia con una letra mayúscula o minúscula a la que le sigue cero o muchas veces cualquier combinación de letras, y le sigue cero o una vez un sub-guion o dos puntos o un espacio y le sigue cero o muchas veces cualquier combinación de letras mayúsculas o minusculas.	**”A+F?A*”**	-etiqueta -grupo-radio -Nombre:
SubLiteral	Inicia con una letra mayúscula o minúscula a la que le sigue cero o muchas veces cualquier combinación de letras, y le sigue cero o una vez un sub-guion o dos puntos o un espacio y le sigue cero o muchas veces cualquier combinación de letras mayúsculas o minusculas.	**‘A+F?A*’**	-Masculino -Femenino -Guatemala

Expresión regular:

S | R+ | ”A+F?A*” | ‘A+F?A*’

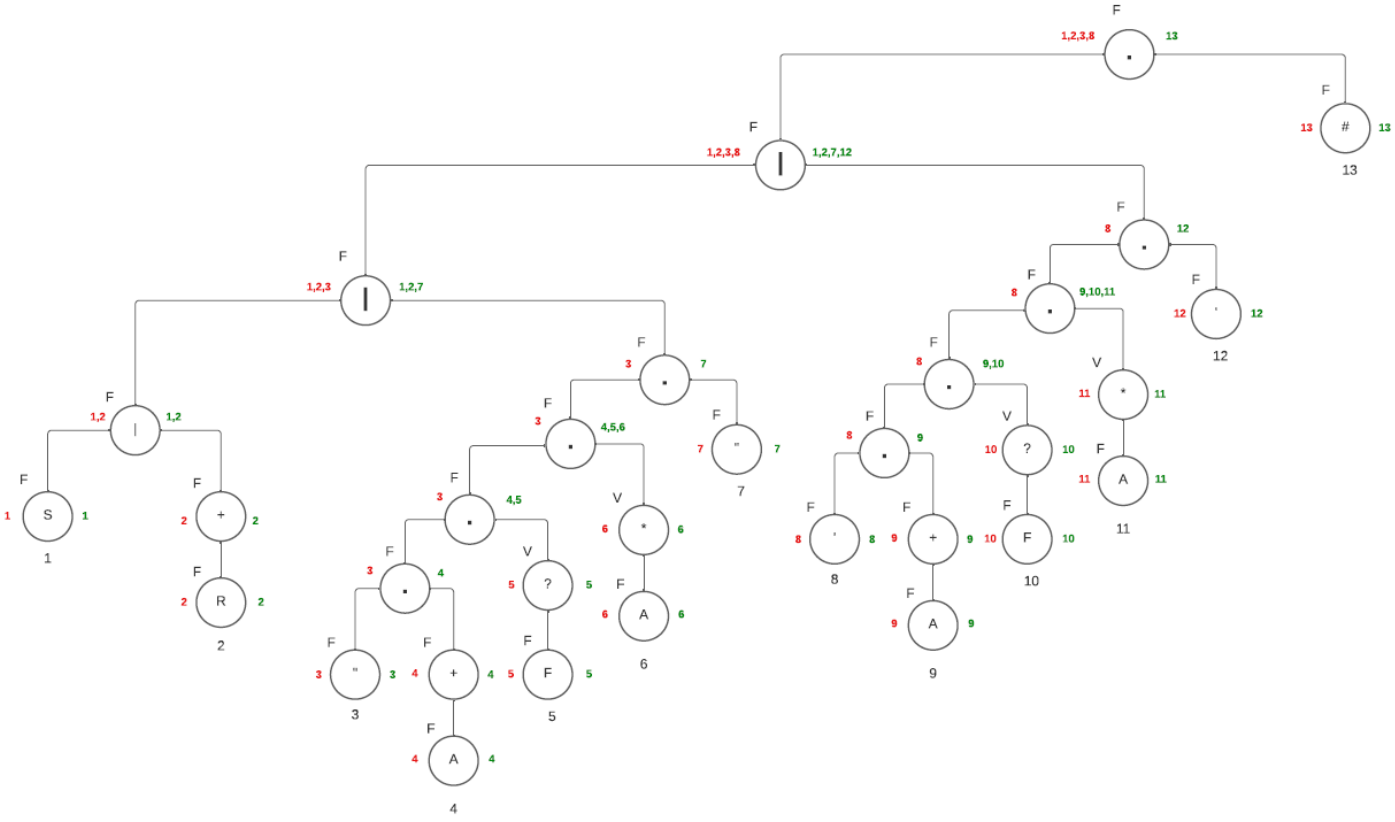
1. Concatenar símbolo de aceptación al final de la ER.

(S | R+ | ”A+F?A*” | ‘A+F?A*’)#

2. Construir el árbol binario de sintaxis

3. Identificar cada hoja con terminales.

4. Calcular por cada nodo del árbol: Anulable, First, Last.



5. Calcular Siguientes:

Valor	Hoja	Siguientes
S	1	13
R	2	2,13
“	3	4
A	4	4,5,6,7
F	5	6,7
A	6	6,7
“	7	13
‘	8	9
A	9	9,10,11,12
F	10	11,12
A	11	11,12
‘	12	13
#	13	--

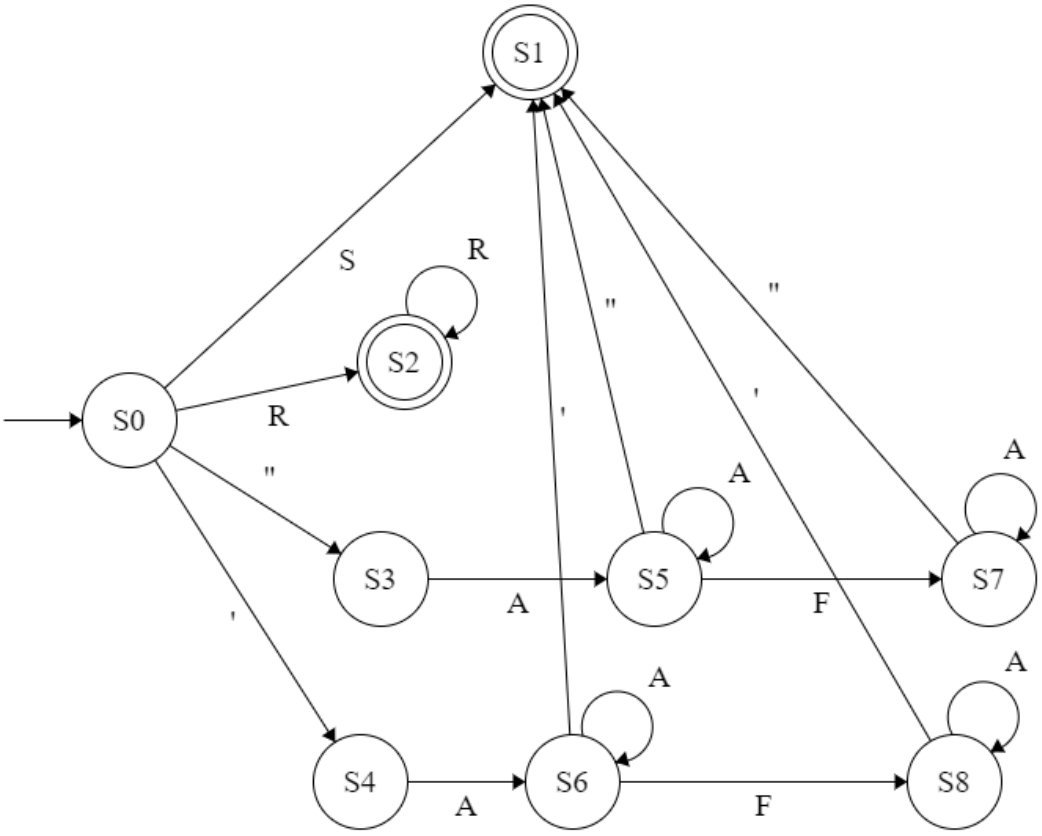
6. Tabla de transiciones:

	Estado	Valores	Siguientes
Inicio	**S0**	S, R,”,’ 1,2,3,8	S: {13} = S1 R: {2,13} = S2 “ : {4} = S3 ‘ : {9} = S4
#	**S1**	# 13	---
#	**S2**	R, # 2,13	R: {2,13} = S2
	S3	A 4	A: {4,5,6,7} = S5
	S4	A 9	A : {9,10,11,12} = S6
	S5	A,F,A,” 4,5,6,7	A : {4,5,6,7} = S5 F : {6,7} = S7 “ : {13} = S1
	S6	A,F,A,’ 9,10,11,12	A : {9,10,11,12} = S6 F : {11,12} = S8 ‘ : {13} = S1
	S7	A,” 6,7	A : {6,7} = S7 “ : {13} = S1
	S8	A,’ 11,12	A : {11,12} = S8 ‘ : {13} = S1

7. Tabla de estados

ESTADOS		SIGMA					
		S	**R**	**A**	**F**	**“**	**‘**
0	**S0**	S1	S2			S3	S4
#	**S1**						
#	**S2**		S2				
	S3			S5			
	S4			S6			
	S5			S5	S7	S1	
	S6			S6	S8		S1
	S7			S7		S1	
	S8			S8			S1

7.1 Autómata Finito Determinista



El código fuente del automata es el siguiente:

```
def automata(starter: str):
    #agregando al final
    starter += '\n'
    #lista de tokens
    tokens: List[Token] = []
    #lista de errores
    errores: List[Errors] = []
    #estado inicial
    state: int = 0
    tmp_state: int = 0
    #estado actual
    lexeme: str = ''
    #apuntador
    pointer: int = 0
    #Contador de filas y columnas
    row: int = 1
    col: int = 0

    while pointer < len(starter):
        char = starter[pointer]
        # state inicial
        if state == 0:
            #Lista de transiciones
            #Si el caracter es un simbolo [~>>[]<>,:]
            if(ord(char) == 60 or ord(char) == 62 or ord(char) == 58 or ord(char) == 44 or ord(char) == 91 or ord(char) == 93 or ord(char) == 126):
                state = 1
                tmp_state = 1
                pointer += 1
                col += 1
                lexeme += char
            #Si el caracter es una letra minuscula [a-z,ñ]
            elif((ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164):
                state = 2
                pointer += 1
                col += 1
                lexeme += char
```

```
            #Si el caracter es una comilla doble ["]
            elif char == '"':
                state = 3
                pointer += 1
                col += 1
                lexeme += char
            #Si el caracter es una comilla simple [']
            elif char == "'":
                state = 4
                pointer += 1
                col += 1
                lexeme += char

            # caracteres ignorados
            #si es un salto de linea [\n]
            elif (ord(char) == 10):
                row += 1
                col = 0
                pointer += 1
            #si es un tabulador horizontal [\t]
            elif (ord(char) == 9):
                col += 1
                pointer += 1
            #si es un espacio en blanco ['']
            elif (ord(char) == 32):
                row += 1
                col = 0
                pointer += 1

            else:
                errores.append(Errors(row, col, char))
                pointer += 1
                col += 1
```

```
        #estado 2 -> reservadas
        elif state == 2:
            if((ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164):
                pointer += 1
                col += 1
                lexeme += char
            else:
                if lexeme in ['formulario','tipo','valor','fondo','valores', 'evento']:
                    tokens.append(Token('reservada', lexeme, row, col))
                elif lexeme in ['info','entrada']:
                    tokens.append(Token('iframe', lexeme, row, col))
                else:
                    errores.append(Errors(row, col, lexeme))
                    pointer += 1
                    col += 1
                state = 0
                lexeme = ''

        #estado 3 -> literal
        elif state == 3:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                state = 5
                pointer += 1
                col += 1
                lexeme += char
            else:
                errores.append(Errors(row, col, char))
                pointer += 1
                col += 1
```

```
        elif state == 5:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                pointer += 1
                col += 1
                lexeme += char
            elif char == '"':
                state = 1
                tmp_state = 5
                pointer += 1
                col += 1
                lexeme += char
            elif ord(char) == 45 or ord(char) == 58 or ord(char) == 32:
                state = 7
                pointer += 1
                col += 1
                lexeme += char
            else:
                errores.append(Errors(row, col, char))
                pointer += 1
                col += 1

        elif state == 7:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                pointer += 1
                col += 1
                lexeme += char
            elif char == '"':
                state = 1
                tmp_state = 7
                pointer += 1
                col += 1
                lexeme += char
```

```
        #estado 4 -> subliteral
        elif state == 4:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                state = 6
                pointer += 1
                col += 1
                lexeme += char
            else:
                errores.append(Errors(row, col, char))
                pointer += 1
                col += 1

        elif state == 6:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                pointer += 1
                col += 1
                lexeme += char
            elif char == "'":
                state = 1
                tmp_state = 6
                pointer += 1
                col += 1
                lexeme += char
            elif ord(char) == 45 or ord(char) == 58 or ord(char) == 32:
                state = 8
                pointer += 1
                col += 1
                lexeme += char
            else:
                errores.append(Errors(row, col, char))
                pointer += 1
                col += 1
```

```
        elif state == 8:
            if((ord(char) >= 65 and ord(char) <= 90) or (ord(char) >= 97 and ord(char) <= 122) or ord(char) == 164 or ord(char) == 165):
                pointer += 1
                col += 1
                lexeme += char
            elif char == "'":
                state = 1
                tmp_state = 8
                pointer += 1
                col += 1
                lexeme += char

        #estado 1 -> Aceptacion General
        elif state == 1:
            if tmp_state == 5 or tmp_state == 7:
                state = 0
                tmp_state = 0
                tokens.append(Token('literal', lexeme, row, col))
                lexeme = ''
            elif tmp_state == 6 or tmp_state == 8:
                state = 0
                tmp_state = 0
                tokens.append(Token('subliteral', lexeme, row, col))
                lexeme = ''
            elif tmp_state == 1:
                state = 0
                tmp_state = 0
                tokens.append(Token('simbolo', lexeme, row, col))
                lexeme = ''

    return tuple(tokens), tuple(errores)
```

Función scanner:

Tiene como función analizar el listado de tokens ya clasificados por el autómata y comparando los tokens puede definir cuales tokens tiene un lexema igual al lexema esperado para la generación del formulario dinámico, por lo tanto, la función del scanner es encontrar las palabras que nos ayudaran a construir el formulario dinámico, esto se llevara a cabo mediante comparaciones y mediante ordenamiento de palabras.

Función process_tokens:

Tiene como función analizar el texto que se encuentre en el área de texto de la interfaz grafica y está por medio del automata retornara un listado de tokens por medio del cual se desplegara un formulario con el listado de tokens actuales.

```
def process_tokens(tokens):
    env = Environment(loader=FileSystemLoader('Templates/'),
                      autoescape=select_autoescape(['html']))
    template = env.get_template('report_tokens.html')

    html_file = open('oficial_report_tokens.html', 'w+', encoding='utf-8')
    html_file.write(template.render(tokens=tokens))
    html_file.close()
    startfile('oficial_report_tokens.html')
```

Función process_errors:

Tiene como función analizar el texto que se encuentre en el área de texto de la interfaz gráfica y está por medio del autómata retornara un listado de errores por medio del cual se desplegara un formulario con el listado de errores actuales.

```
def process_errors(errs):
    env = Environment(loader=FileSystemLoader('Templates/'),
                      autoescape=select_autoescape(['html']))
    template = env.get_template('report_errors.html')

    html_file = open('oficial_report_errors.html', 'w+', encoding='utf-8')
    html_file.write(template.render(errs=errs))
    html_file.close()
    startfile('oficial_report_errors.html')
```

Función process_Gui:

Tiene como función analizar los parámetros enviados por el scanner para ser enviados a la plantilla del formulario dinámico y de esta forma generar el formulario con las propiedades de entrada previamente evaluadas por el autómata.

```
def process_Gui(parameters):
    env = Environment(loader=FileSystemLoader('Templates/'),
                      autoescape=select_autoescape(['html']))
    template = env.get_template('gui.html')
    html_file = open('oficial_gui.html', 'w+', encoding='utf-8')
    html_file.write(template.render(parameters=parameters))
    html_file.close()
    startfile('oficial_gui.html')
```

Clase display_gui:

Tiene como función desplegar la interfaz gráfica al usuario mediante la librería Tkinter.

```
class display_gui():
    def __init__(self) -> None:
        self.root = tk.Tk()
        self.frame = Frame()
        self.text = []

        self.root.title('Generador de formularios dinamicos')
        self.root.geometry('700x400')
        self.frame.config(width=700, height=400)
        self.frame.place(x=0, y=0)

        self.btn_load_files = Button(self.frame, text="Cargar Archivo",command=self.load_file)
        self.btn_load_files.place(x=50, y=15)

        self.lbl_combo = Label(self.frame, text="Reportes")
        self.lbl_combo.place(x=540, y=30)

        self.combo_report = Combobox(self.frame, values=['Reporte de Tokens', 'Reporte de Errores', 'Manual de Usuario', 'Manual Tecnico'])
        self.combo_report.place(x=500, y=50)

        self.text_area = Text(self.frame,  height = 15, width = 80)
        self.text_area.place(x=30, y=90)

        self.btn_process = Button(self.frame,text="Analizar", command=self.analyzer)
        self.btn_process.place(x=50, y=350)

        self.btn_clear = Button(self.frame,text="Limpiar", command=self.clearText_area)
        self.btn_clear.place(x=590, y=350)

        self.btn_clear = Button(self.frame,text="Go", command=self.print_report)
        self.btn_clear.place(x=645, y=47)

        self.root.resizable(0,0)
        self.root.mainloop()
```