# Proyecto 2 Manual Técnico

**Índice**

## Introducción

A continuación se describen las principales funciones para el uso de gramáticas y autómatas de pila como declaración de nodo, generación de árbol, reportes entre otros.

## 1. Técnica de Programación

### 1.1 Programación Orientada a Objetos

La programación orientada a objetos se basa en el concepto de crear un modelo del problema de destino en sus programas. La programación orientad a objetos disminuye los errores y promociona la reutilización del código, Python es un lenguaje orientado a objetos, por este motivo se decidió trabajar de este modo.

## 2. Convenciones de Nomenclatura

### 2.1 Declaración de Variables

Para la declaración de variables, se usó un nombre el cual hiciera referencia a lo que se contendrá en la variable, por ejemplo, al guardar las gramáticas que se iban ingresando, la variable correspondiente que contendrá los cursos fue nombrada “Gramaticas”.

```python
# create class Afd


class Gr:
    # static variable to autoincrement id
    id = 0
    # constructor

    def __init__(self, name, no_terminals, terminals, initial_no_terminal, accepting_no_terminals, productions):
        # autoincrement id
        Gr.id += 1
        self.name = name
        self.no_terminals = no_terminals
        self.terminals = terminals
        self.initial_no_terminal = initial_no_terminal
        self.productions = productions
        self.accepting_no_terminals = accepting_no_terminals
        self.dot = ''
        self.last_node = 0
```

El nombre de cada variable corresponde a lo que contendrá.

## 2.2 Métodos

Para la declaración de Métodos, el nombre de cada método corresponde a la acción que estos deben realizar, cabe mencionar que los métodos solo realizan las instrucciones que indiquemos, en algunos se retornan valores utiles.

Ejemplo, generar árbol de derivación

```
def generateTree(self):
    # quitar espacios
    name = self.name.replace(' ', '')
    name = name + 'tree'
    productions = self.productions.copy()

    self.dot = ''
    self.last_node = 0
    self.dot += f'graph {name} {"{"} \n'
    self.dot += f'node [color=lightblue2, fontcolor=black, shape=box]; \n'
    self.dot += f'layout=dot; rankdir=TB; shape=circle \n'
    self.dot += f'nodo0 [label=init shape=plain]\r'
    self.recursiveExpression(productions, 0, self.initial_no_terminal)
    self.dot += '}'

    document = open(f'dots/{name}.dot', 'w')
    document.write(self.dot)
    document.close()

    os.system(
        f'dot -Tpng dots/{name}.dot -o pngs/{name}.png')

    return f'pngs/{name}.png'

def recursiveExpression(self, productions, node, expression):
    make_node = node
    response = self.searchProduction(expression, productions)
    production = response['production']
    productions.pop(response['index'])
    self.writeDot(expression, self.last_node, make_node)
    for terminal in production.terminals:
        if self.typeExpression(terminal) == 'terminal':
            make_node += 1
            self.writeDot(terminal, node, make_node)
        if self.typeExpression(terminal) == 'no_terminal':
            make_node += 1
            make_node = self.recursiveExpression(productions, make_node, terminal)
    return make_node


def writeDot(self, expression, node, node_make):
    type = self.typeExpression(expression)
    if type == 'no_terminal':
        self.last_node = node_make
    shape = type == 'terminal' and 'plain' or 'circle'
    self.dot += f'nodo{node_make} [label={expression} shape={shape}]\r'
    if node_make > 0:
        self.dot += f'nodo{node} -- nodo{node_make}\r'
```

## 2.3 Funciones de apoyo

```python
    def typeExpression(self, expression):
        for terminal in self.terminals:
            if terminal == expression:
                return 'terminal'

        for no_terminal in self.no_terminals:
            if no_terminal == expression:
                return 'no_terminal'

    def searchProduction(self, no_terminal, productions):
        for production in productions:
            if production.no_terminal == no_terminal:
                return {'production' : production, 'index': productions.index(production)}

```

```python
    def searchTransition(self, state, letter):
        for transition in self.transitions:
            if transition[0] == state and transition[1] == letter:
                return transition
        return False

    def validateStringRoute(self):
        response = "Ruta: \n"
        self.last_route.reverse()
        for transition in self.last_route:
            for transition2 in transition:
                response += f'{transition2},'
            response += f' \n'

        return response
```

## 3. Métodos Principales

### 3.1 Leer archivos

Este método analiza el archivo que hallamos seleccionado, lee línea por línea y separa cada parte de la gramática guardando sus valores en una variable correspondiente a los valores que se obtengan para luego crear el objeto de tipo Gramática esto es muy similar para autómatas de pila.

main.py > ...

```
415     def frameAutomatasFile(self):
416         # esconder alerta
417         if self.label_alert is not None:
418             self.label_alert.grid_forget()
419 
420         self.content_frame_sub.grid_remove()
421         self.content_frame_sub = customtkinter.CTkFrame(self.content_frame)
422         self.content_frame_sub.grid(row=0, column=1,  rowspan=7, padx=(
423             20, 20), pady=(20, 20), sticky="nsew")
424         self.content_frame_sub.grid_rowconfigure(10, weight=1)
425         self.content_frame_sub.grid_columnconfigure(1, weight=1)
426 
427         init = "./"
428 
429         file = filedialog.askopenfilename(
430             initialdir=init, title="Select file", filetypes=(("Text files", "*.ap"), ("all files", "*.*")))
431 
432         if file == "":
433             return
434 
435         # validar que sea un archivo txt
436         if file.split(".")[-1] != "ap":
437             self.alert("Error: El archivo debe ser un archivo ap")
438             return
439 
440         f = open(file, "r")
441         lines = f.readlines()
442         f.close()
443 
444         state = 0
445         name = ""
446         alphabet = []
447         simbols = []
448         states = []
449         initial_state = ""
450         acceptance_state = ""
451         transitions = []
452         flag = True
```

```
        for line in lines:
            if state == 0:
                name = self.replaceSpacing(line)
                state = 1
            elif state == 1:
                alphabet = self.replaceSpacing(line)
                alphabet = alphabet.split(",")
                state = 2
            elif state == 2:
                simbols = self.replaceSpacing(line)
                simbols = simbols.split(",")
                state = 3
            elif state == 3:
                states = self.replaceSpacing(line)
                states = states.split(",")
                state = 5
            elif state == 5:
                initial_state = self.replaceSpacing(line)
                state = 6
            elif state == 6:
                acceptance_state = self.replaceSpacing(line)
                state = 7
            elif state == 7:
                if self.replaceSpacing(line) == "%" or self.replaceSpacing(line) == "":
                    # crear afd
                    for ap in self.aps:
                        if ap.name == name:
                            self.alert(
                                "Error: Ya existe un ap con el mismo nombre")
                            flag = False

                    if flag:
                        self.aps.append(
                            Ap(name, alphabet, simbols, states, initial_state, acceptance_state, transitions))
                    transitions = []
                    state = 0
                    flag = True

                else:
                    line = self.replaceSpacing(line)
                    # separar por comas o puntos y comas
                    line = line.split(",")
                    sep = line[2].split(";")
                    line = [line[0], line[1], sep[0], sep[1], line[3]]
                    transitions.append(line)

        self.success("Grs creados con exito")
```

## 4. Requerimiento o Funcionalidad Especifica

Para la elaboración de este proyecto se usaron los siguientes programas y herramientas:

4.1 Visual Studio Code

Se uso este editor de código debido a que ya se tenía conocimiento sobre él, por este motivo fue más sencillo llevar a cabo el proyecto en él.

4.2 Tkinter

Se uso la librería Tkinter para la interfaz gráfica del proyecto, se decidió usar debido a que esta librería se pide como requisito para la elaboración del proyecto y próximos trabajos a realizar en el curso.

## 5. Glosario

I. IndexError: es un error usual al momento de agregar un valor de índice fuera del rango de nuestras listas.
II. Identación: este término significa mover un bloque de texto hacia la derecha insertando espacios o tabuladores, para así separarlo del margen izquierdo y distinguirlo mejor del texto adyacente
III. ValueError: es un error que sucede cuando trato de convertir un valor a otro tipo que no es posible, por ejemplo, una cadena de caracteres a números.
IV. Importaciones de orden Superior: un error que sucede cuando se trata de importar algo que no está al mismo nivel que la raíz.