Universidad San Carlos de Guatemala

MANUAL TECNICO

Nombre: Nataly Saraí Guzmán Duarte
Carné: 202001570

Manual

Técnico

Proyecto 1

Contenido

Introducción

El presente documento describe los aspectos técnicos informáticos del programa creado en lenguaje python, el programa este compuesto por distintos componentes gráficos que lo vuelven intuitivo para el usuario, el documento busca familiarizar todos los aspectos técnicos del programa, como lo es su funcionamiento correcto de la aplicación, como la aplicación de los distintos flujos que el sistema crea.

Lugar de realización: Guatemala

Fecha:17/03/2022

Responsable de elaboración: Nataly Saraí Guzmán Duarte

Información destacada

El programa “Generador de HTML” fue realizado utilizando como plataforma visual studio code , la realización del proyecto fue de aproximadamente 14 días, cuya realización fue separada en botones del menú, cada botón se realizo en aproximadamente 3 días su realización completa.

Objetivos y alcances del sistema

- Dar a conocer al programada la forma en que se realizó el programa desde su inicio cuando se instaló python, mostrando específicamente el código y explicación de porque se realizó cada bloque de código.
- Realizar un programa para que como estudiante practique con el lenguaje python.
- Construir aplicaciones que se ejecuten con interfaz gráfica creada a código.
- Crear un analizador para los archivos o contenido que se ingrese.
- Realizar un programa donde se pueda generar un html ingresando un archivo con un formulario.

Requerimientos

- RAM: 1 GB (mínimo).
- ROM: 250 MB (mínimo).
- Arquitectura x32 bits o x64 bits.
- Sistema operativo: Windows, Linux, MacOS.
- Lenguaje de programación: Python.

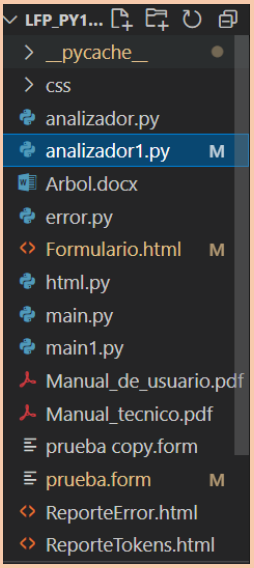
Datos técnicos para la realización del sistema

- Windows 10 Home Single Language.
- Procesador: Intel Core i5.
- RAM: 8.00 GB
- Plataforma IDE: Visual Studio Code
- Python 3.

Lógica del programa

El programa está constituido por una interfaz gráfica creada en el lenguaje de programación Python utilizando Qt Designer es simple e intuitivo con el usuario mostraremos a continuación las distintas clases que se usaron para su creación.

- Listado de clases creadas:



Cada clase cuenta con métodos que nos ayudan a que el sistema funciona de manera eficaz logrando así el uso correcto de la aplicación poniendo en práctica los distintitos condicionales y/o bucles, la utilización de hilos y recursividad.

- Descripción de realización de cada clase:
  - Main: Contiene el main para ejecutar el programa.
  - Analizador: Encargada de analizar los datos ingresados señalando los tokens y errores.
  - Tokens: Encargada de crear un objeto de tipo token.
  - Error: Encargada de crear un objeto de tipo error.
  - Html: Encargada de crear un objeto de tipo html.
- Codificación y explicación de cada clase:

  - Clase main:
    - ✓ Constructor: Llamado de la ventana que contiene la interfaz gráfica asignación de funciones a los botones.
    - ✓ item: botón que realiza la acción seleccionada en el select.
    - ✓ archivomemoria: Función que carga el archivo que se ingresa.
    - ✓ Lectura: Función que carga el contenido del archivo al área de texto.
    - ✓ Analizar: Función que llama a la clase del analizador.
    - ✓ Salir: Función que sale del programa

```python
def __init__(self):
    super().__init__()
    loadUi("ventana.ui", self)
    self.pushButton.clicked.connect(self.lectura)
    self.pushButton_2.clicked.connect(self.item)
    self.pushButton_3.clicked.connect(self.analizardoc)
    self.pushButton_4.clicked.connect(self.salir)

def salir(self):
    sys.exit(1)

def analizardoc(self):
    global anali
    anali= analizador1()
    archivo = self.plainTextEdit.toPlainText()
    anali.analizar(archivo)
    anali.imprimir()
    anali.CrearHtml()

def lectura(self):
```

```python
def lectura(self):
    global rutarecibida
    archi=filedialog.askopenfilename(filetypes=[("Archivos FORM", ".for
    rutarecibida=self.ArchivoMemoria(archi)
    self.plainTextEdit.setPlainText(rutarecibida)

def ArchivoMemoria(self, rutarchi):
    try:
        with open(rutarchi, encoding='utf-8') as file:
            datarchi = file.read()
            return datarchi
    except:
        messagebox.showwarning("Alert","No se pudo abrir el archivo")

def item(self):
    itms= self.comboBox.currentText()
    if str(itms)=="Reporte de tokens":
        anali.HTMLTOKENS()
    elif str(itms)=="Reporte de errores":
        anali.HTMLERRORES()
```

- Clase analizador:
  - ✓ _init_: Creación de listas para los tokens, errores, y html.
  - ✓ Analizar: Función que analiza cada elemento que se encuentra en el archivo, obtiene los tokens, los errores, y llena la lista de html para luego su creación.
  - ✓ HTMLTOKENS: Crea un html con una tabla donde se encuentra cada token que se encontró en el momento de analizar.
  - ✓ HTMLERRORES: Crea un html con una tabla donde se encuentra cada error que se encontró en el momento de analizar.

- ✓ CrearHtml: Crea un html con el formulario que se ingreso en el archivo.

```
from tokens import tokens
from error import error
from html import html
from valorees import valorees
import re
import webbrowser

class analizador1:
    def __init__(self):
        self.listaTokens = []
        self.listaErrores = []
        self.listaHtml = []

    def analizar(self, codigo):
        self.listaTokens = []
        self.listaErrores = []
        self.listaHtml = []
        global valoress
        valoress=[]
        valoress.append(valorees("Masculino"))
```

```
def HTMLTOKENS(self):
    texto1 = """<!doctype html
            <html lang="en">
            <head>
            <title>Reporte de
            <meta charset="utf
            <meta name="viewpo
            <link href='https:
            <link rel="stylesh
            <link rel="stylesh
            <H1><font color="C
            </head>
            <body style="backg
                <section class
                <div class="co
                    <div class
                        <div c
                            <h2 class="heading-section">Tabla de tokens<
```

```
def CrearHtml(self):
    texto1 = """<!doctype html>
            <html lang="en">
            <head>
            <title>Formulario</title>
            <meta charset="utf-8">
            <meta name="html" content="width=device-width, initial-scale
            </head>
            <body>"""
    for f in range(0,len(self.listaHtml)):
        if(self.listaHtml[f-1].tipo=="etiqueta"):
            texto1 = texto1 + "<label>"+self.listaHtml[f-1].valor+"</lab
        if(self.listaHtml[f-1].tipo=="texto"):
            texto1 = texto1 + " <input type=\"text\" id=\""+self.listaHt
        if(self.listaHtml[f-1].tipo=="grupo-radio"):
            for u in range(0,len(valoress)):
                texto1 = texto1 + "<input type=\"radio\"  value=\""+self
        if(self.listaHtml[f-1].tipo=="grupo-radio"):
            for o in range(0,len(valoress)):
                texto1 = texto1 + "<select name=\"select\"><option>"+val
```

- Clase error:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ strError: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

```
class error:
    def __init__(self,tipo, descripcion, fila, columna):
        self.tipo = tipo
        self.descripcion = descripcion
        self.fila = fila
        self.columna = columna-len(descripcion)

    def strError(self):
        print("\n================================")
        print("Tipo:" +self.tipo)
        print("Descripción:  "+self.descripcion)
        print("Fila:  " +str(self.fila))
        print("Columna: "+str(self.columna))
```

- Clase html:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.

- ✓ _repr_: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

```
class html:
    def __init__(self, tipo, valor ,fondo, listavalores, evento,nombre):
        self.tipo=tipo
        self.valor=valor
        self.fondo=fondo
        self.listavalores=listavalores
        self.evento=evento
        self.nombre = nombre

    def __repr__(self):
        print(self.tipo+" "+self.valor+" "+self.fondo+" "+self.evento+" "+se
```

- Clase tokens:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ strToken: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

```
class tokens:
    def __init__(self, tipo,lexema, fila, columna):
        self.tipo = tipo
        self.lexema = lexema
        self.fila = fila
        self.columna = columna-len(lexema)

    def strToken(self):
        print("\n================================")
        print("Tipo:" +self.tipo)
        print("Lexema:  "+self.lexema)
        print("Fila:  " +str(self.fila))
        print("Columna: "+str(self.columna))
```

- Clase tokens:
  - ✓ _init_: método encargado de inicializar de variables y atributos de la clase.
  - ✓ _repr_: método encargado de generar una cadena para la impresión de información de la instancia de la clase.

```
class valorees:
    def __init__(self, valor):
        self.valor = valor
    def __repr__(self):
        return self.valor
```

Tabla de tokens y patrones

La tabla de tokens y patrones presentada a continuación, presenta los tokens que nuestro autómata y/o analizador léxico reconoce como parte de nuestro alfabeto:

ID	Token	Patrón	Lexema
0	Formulario	[a-zA-Z]	Formulario
1	Tipo	[a-zA-Z]	Tipo
2	Valor	[a-zA-Z]	Valor
3	Fondo	[a-zA-Z]	Fondo
4	Nombre	[a-zA-Z]	Nombre
5	Valores	[a-zA-Z]	Valores
6	Evento	[a-zA-Z]	Evento
7	Virgulilla	~	~
8	Signo_mayor	>	>
9	Signo_menor	<	<
10	Corchete_Abrir	[	[
11	Corchete_Menor	]	]
12	Coma	,	,
13	Cadena	"[A-Za-z0-9_]+"	“nombre”
14	Centinela	$	$

Método el árbol y expresiones regulares

En esta sección utilizamos las expresiones regulares (patrones) de la tabla de tokens para la elaboración de los distintos arboles sintácticos.

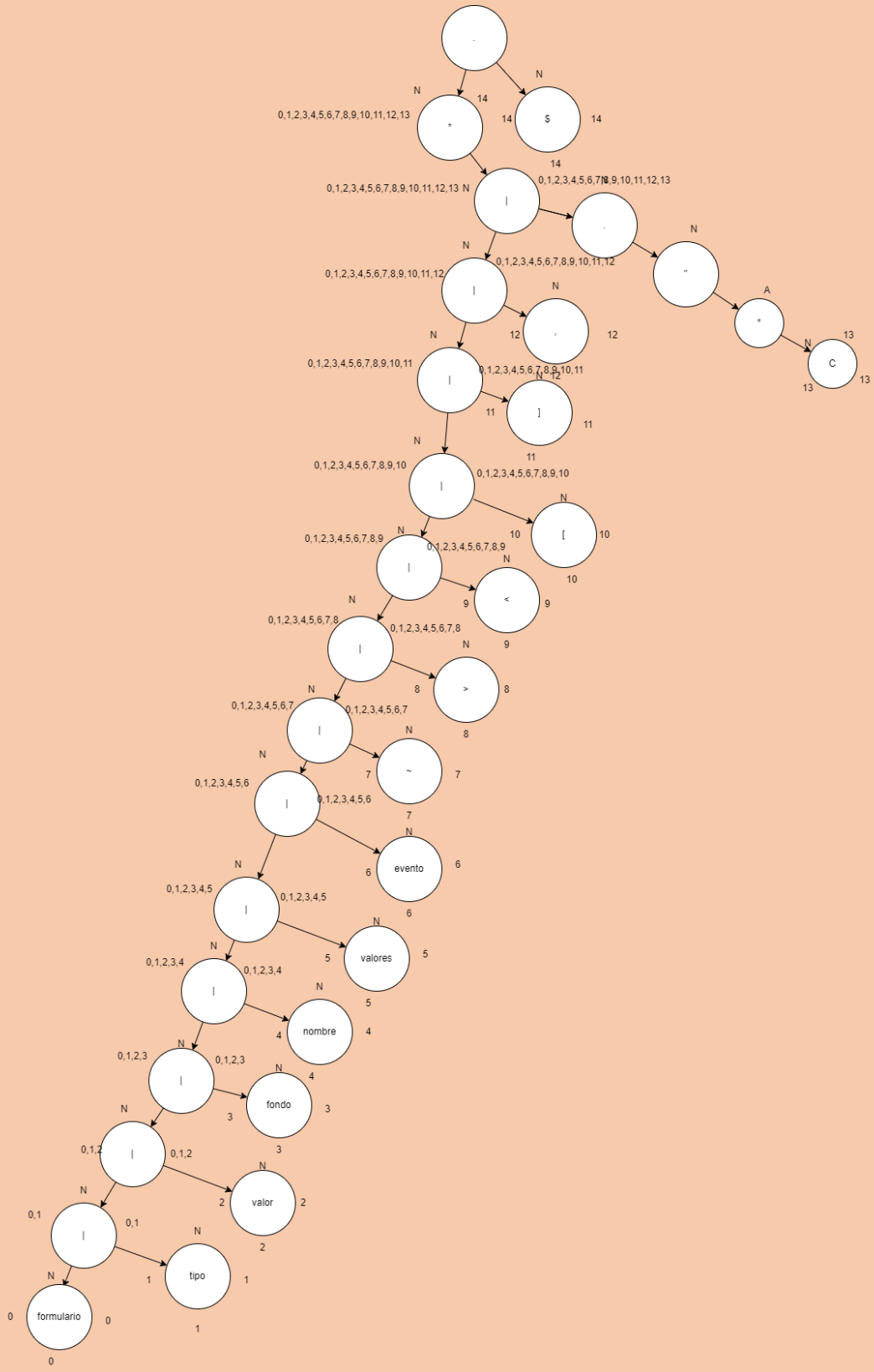


Diagrama del autómata

En esta sección mostramos el diagrama del autómata aplicado en el analizador léxico construido para el proyecto: