MANUAL TECNICO.

FUNCIONES.

Clase AutomataMenu.

Esta clase posee un analizador léxico que nos permite reconocer diferentes tokens y almacenarlos en una lista para su posterior manipulación para la realización de reportes de tokens y errores.

```
def analizador(self, entry):

    linea = 1
    columna = 0
    lexema = ""
    state = 1
    indice = 0
    while indice < len(entry):
        if state == 1:
            if re.search(r"[\']", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 3
            elif re.search(r"[0-9]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 5
            elif re.search(r"[a-zA-Z]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 6
            elif re.search(r"[\n]", entry[indice]):

                linea += 1
                columna = 0
                indice += 1
            elif re.search(r"[ \t]", entry[indice]):

                columna += 1
                indice += 1
            else:
                if re.search(r"[\=\[\]\;\:]", entry[indice]):
                    lexema = entry[indice]
                    self.tokens.append([linea, columna, self.signs[entry[indice]], lexema])
                    lexema = ""
                    indice += 1
                    columna += 1
```

```
    else:
        if re.search(r"[\=\[\]\;\:]", entry[indice]):
            lexema = entry[indice]
            self.tokens.append([linea, columna, self.signs[entry[indice]], lexema])
            lexema = ""
            indice += 1
            columna += 1
        else:
            self.error.append([linea, columna, entry[indice],"No se reconocio caracter"])
            indice += 1
            columna += 1

elif state == 3:
    if re.search(r"[^']", entry[indice]):
        lexema += entry[indice]
        columna += 1
        indice += 1
        state = 3
    else:
        state = 8
elif state == 5:
    if re.search(r"[0-9]", entry[indice]) or re.search(r"\.", entry[indice]):
        lexema += entry[indice]
        indice += 1
        columna += 1
        state = 5
    elif re.search(r"\.", entry[indice]):
        state = 11
    else:
        self.listTokens.append([linea, columna, "Numero", lexema])
        self.tokens.append([linea, columna, "Numero", lexema])
        state = 1
elif state == 6:
    if re.search(r"[a-zA-Z0-9_]", entry[indice]):
        lexema += entry[indice]
        indice += 1
        columna += 1
        state = 6
```

```
elif state == 6:
    if re.search(r"[a-zA-Z0-9_]", entry[indice]):
        lexema += entry[indice]
        indice += 1
        columna += 1
        state = 6
    else:
        self.listTokens.append([linea, columna, self.verificaLexema(lexema), lexema])
        self.tokens.append([linea, columna, self.verificaLexema(lexema), lexema])
        lexema = ""
        state = 1
elif state == 8:
    if re.search(r"[\']", entry[indice]):
        self.listTokens.append([linea, columna, "cadena", lexema + entry[indice]])
        self.tokens.append([linea, columna, "cadena", lexema + entry[indice]])
        indice += 1
        lexema = ""
        state = 1
    else:
        lexema += entry[indice]
        state = 1
elif state == 11:
    if re.search(r"[\.]", entry[indice]):
        lexema += entry[indice]
        state = 14
    else:
        self.listTokens.append([linea, columna, "entero", lexema])
        self.tokens.append([linea, columna, "entero", lexema])
        lexema = ""
        state = 1
elif state == 14:
    if re.search(r"[0-9]", entry[indice]):
        lexema += entry[indice]
        indice += 1
        columna += 1
        state = 14

    else:
        #self.listTokens.append([linea, columna, "decimal", lexema])

        state = 1
```

def verificaLexema(self, lexxe):

Esta función verifica si es una palabra reservada o un identificador.

```
def verificaLexema(self, lexxe):
    if lexxe.lower() == "restaurante":
        return "reservada"
    else:
        return "identificador"
```

def reporteToken(self):

Esta función nos permite realizar un reporte de la lista de tokens.

`

```
def reporteToken(self):
    contenido = ''
    htmFile = open("Reporte_Menu" + ".html", "w",encoding='utf8')
    htmFile.write("""<!DOCTYPE HTML PUBLIC"
        <html>

        <head>
            <title>Reporte de tokens Menu</title>
         <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        </head>
        <body>
        <div class="container">
      <h2>Reporte de tokens</h2>
      <p>Lista de tokens</p>
      <table class="table">
        <thead>
          <tr>
           <th>Fila</th>
            <th>Columna</th>
            <th>Token</th>
            <th>Lexemna</th>
          </tr>
        </thead>

        """)
    for i in range(len(self.listTokens)):
        contenido += (" <tbody>"
                      "<td>" + str(self.listTokens[i][0]) + "</td>"

                      "<td>" + str(self.listTokens[i][1]) + "</td>"
                      "<td>" + str(self.listTokens[i][2]) + "</td>"
                      "<td>" + str(self.listTokens[i][3]) + "</td>"
                      "</tbody>")
    htmFile.write(contenido)
```

tomataMenu › reporteToken()

def reporteError(self):

Esta función nos permite realizar un reporte de la lista de errores encontrados en el archivo de entrada.

```
def reporteError(self):
    contenido = ''
    htmFile = open("Error_menu" + ".html", "w", encoding='utf8')
    htmFile.write("""<!DOCTYPE HTML PUBLIC"

                    <html>

                    <head>
                        <title>Reporte de tokens Menu</title>
                     <meta charset="utf-8">
                  <meta name="viewport" content="width=device-width, initial-scale=1">
                  <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
                  <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
                  <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
                    </head>
                    <body>
                    <div class="container">
                  <h2>Reporte de tokens</h2>
                  <p>Lista de tokens</p>
                  <table class="table">
                    <thead>
                      <tr>
                       <th>Fila</th>
                        <th>Columna</th>
                        <th>caracter</th>
                        <th>descripcion</th>
                      </tr>
                    </thead>

                    """)
    for i in range(len(self.error)):
        contenido += ("<tbody>"
                      "<td>" + str(self.error[i][0]) + "</td>"
                      "<td>" + str(self.error[i][1]) + "</td>"
                      "<td>" + str(self.error[i][2]) + "</td>"
                      "<td>" + str(self.error[i][3]) + "</td>"
                      "</tbody>")
    htmFile.write(contenido)
    htmFile.write("""
                </table>
            </div>
```

def ordenarSecciones(self):

Esta función ordena las secciones del menú para su posterior manipulación.

```
def ordenarSecciones(self):
    listaSecc = []
    listaCuerpo = []
    seccion = ""

    for i in range(0,len(self.tokens)):
        if self.tokens[i][2] == 'reservada':
            listaSecc.append(self.tokens[i+2][3])
        elif self.tokens[i][2] == 'dosPuntos':
            seccion = self.tokens[i-1][3]
        elif self.tokens[i][2] == 'corcheteAbre':
            listaCuerpo.append(self.ordenarCuerpo(i+1,self.tokens))
        elif i+2 < len(self.tokens) and self.tokens[i][2] == 'corcheteCierra' and self.tokens[i+2][2] == 'dosPuntos':
            listaSecc.append([seccion,listaCuerpo])
            listaCuerpo = []
        elif i + 1 == len(self.tokens):
            listaSecc.append([seccion,listaCuerpo])
    return listaSecc
```

def ordenarCuerpo(self,indice,lista):

Ordena las secciones y se le introduce sus respectivos identificadores, precios y descripciones.

```
def ordenarCuerpo(self,indice,lista):
    listCuerpo = []
    for i in range(indice,len(lista)):
        if lista[i][2] != 'corcheteCierra':
            if lista[i][2] != 'puntoComa':
                listCuerpo.append(lista[i][3])
        else:
            return listCuerpo
```

def generarGraph(self,listaSecc):

Esta función genera un grafico mediante el lenguaje .dot

```
def generarGraph(self,listaSecc):
    aux = 0
    dot = "digraph G { \n"
    dot += "Inicio[label=\"Nombre\"]\n"
    dot += "Nombre[label=\"" + str(listaSecc[0]) + "\"]\n"
    dot += "Inicio -> Nombre \n"
    for i in range(1,len(listaSecc)):
        dot += "sec"+str(i)+"[label=\"" + str(listaSecc[i][0]) + "\"]\n"
        dot += "Nombre -> sec" + str(i)+"\n"
        listCuerpo = listaSecc[i][1]
        for x in range(0,len(listCuerpo)):
            dot += "son"+str(aux)+"[label=\"" + str(listCuerpo[x][0])+" : Q" + str(listCuerpo[x][2]) + "\n"+str(listCuerpo[x][3]) + "\"]\n"
            dot += "sec"+str(i) + "-> son" + str(aux) + "\n"
            aux += 1
    try:


        dot += "}"
        archivo = open("grafico.dot", 'w',encoding='utf8')
        archivo.write(dot)
        archivo.close()

        os.system("dot -Tpdf grafico.dot -o grafico.pdf")
        os.startfile("grafico.pdf")
    except Exception:

            return False
```

def vista(self,lista):

esta función genera una vista grafica del menú.

```
def vista(self,lista):
    contenido = ''
    htmFile = open("VistaMenu" + ".html", "w", encoding='utf8')
    htmFile.write("""<!DOCTYPE html>
                    <html lang="es">
                    <head>
                    <meta charset="UTF-8">
                    <meta name="viewport" content="width=device-width, initial-scale=1.0">
                    <title>Bootstrap 4. Tablas</title>
                    <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
                    </head>
                    <body>
                    <div class="container">
                    <div class="row">
                    <div class="col">
                    <table class="table table-striped table-bordered table-hover table-dark">
                    <thead>
                    <tr>
                        <th>"""+ str(lista[0]) +"""</th>

                    </tr>
                </thead>
                """)
    for i in range(1,len(lista)):
        listCuerpo = lista[i][1]
        contenido += "<tbody>" + "<tr>" + "<td>" + "<h2>" + str(lista[i][0])+ "</h2>"+ "</td >" + "</tr>" + "</tbody>"
        for x in range(0,len(listCuerpo)):
            contenido += "<td>" +"Nombre"+ str(listCuerpo[x][0])+" : PRECIO: Q:" + str(listCuerpo[x][2])+ " Descripcion " + str(listCuerpo[x][3]) + "</td>"+"</tr>" + "</tbody>"


    htmFile.write(contenido)
    htmFile.write("""
            </table>
        </div>
    </div>
<script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"></script>
 <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
 <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
ody>
tml>""")
```

Clase AutomataMenu.

Esta clase posee un analizador léxico que nos permite reconocer diferentes tokens y almacenarlos en una lista para su posterior manipulación para la realización de reportes de tokens y errores.

def analizadorF(self, entry):

```
def analizadorF(self, entry):

    linea = 1
    columna = 0
    lexema = ""
    state = 1
    indice = 0
    while indice < len(entry):
        if state == 1:
            if re.search(r"[\']", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 3
            elif re.search(r"[0-9]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 5
            elif re.search(r"[a-zA-Z]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 6
            elif re.search(r"[\n]", entry[indice]):
                linea += 1
                columna = 0
                indice += 1

            elif re.search(r"[\t]", entry[indice]):
                columna += 1
                indice += 1

            else:
                if re.search(r"[\,]", entry[indice]):
                    lexema = entry[indice]
                    #self.listTokens.append([linea, columna, self.signs[entry[indice]], lexema])
                    lexema = ""
                    indice += 1
                    columna += 1
                else:
                    self.error.append([linea, columna, entry[indice], " <----- No se esperaba caracter"])
```

```
            else:
                self.error.append([linea, columna, entry[indice], " <----- No se esperaba caracter"])
                indice += 1
                columna += 1

        elif state == 3:
            if re.search(r"[^']", entry[indice]):
                lexema += entry[indice]
                columna += 1
                indice += 1
                state = 3
            else:
                state = 8

        elif state == 8:
            if re.search(r"[\']", entry[indice]):
                self.listTokens.append([linea, columna, "cadena", lexema + entry[indice]])
                indice += 1

                state = 1
            else:
                lexema += entry[indice]
                state = 1

        elif state == 5:
            if re.search(r"[0-9]", entry[indice]) or re.search(r"\%", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 5

            else:
                self.listTokens.append([linea, columna, "NUMERO", lexema])
                lexema = ''
                state = 1

        elif state == 6:
            if re.search(r"[a-zA-Z0-9_]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
```

```
        elif state == 6:
            if re.search(r"[a-zA-Z0-9_]", entry[indice]):
                lexema += entry[indice]
                indice += 1
                columna += 1
                state = 6
            else:
                self.listTokens.append([linea, columna, 'identificador', lexema])
                lexema = ""
                state = 1
```

def reporteToken(self):

```
def reporteToken(self):
    contenido = ''
    htmFile = open("Reporte_Factura" + ".html", "w", encoding='utf8')
    htmFile.write("""<!DOCTYPE HTML PUBLIC"

        <html>

        <head>
            <title>Reporte de tokens Menu</title>
         <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        </head>
        <body>
        <div class="container">
      <h2>Reporte de tokens</h2>
      <p>Lista de tokens</p>
      <table class="table">
        <thead>
          <tr>
           <th>Fila</th>
            <th>Columna</th>
            <th>Token</th>
            <th>Lexemna</th>
          </tr>
        </thead>

        """)
    for i in range(len(self.listTokens)):
        contenido += (" <tbody>"
                      "<td>" + str(self.listTokens[i][0]) + "</td>"

                       "<td>" + str(self.listTokens[i][1]) + "</td>"
                       "<td>" + str(self.listTokens[i][2]) + "</td>"
                       "<td>" + str(self.listTokens[i][3]) + "</td>"
                       "</tbody>")
    htmFile.write(contenido)
    htmFile.write("""
     </table>
</div>
```

def reporterror(self):

```
def reporterror(self):
    contenido = ''
    htmFile = open("Reporte_error_factura" + ".html", "w", encoding='utf8')
    htmFile.write("""<!DOCTYPE HTML PUBLIC"

        <html>

        <head>
            <title>Reporte de errores</title>
         <meta charset="utf-8">
      <meta name="viewport" content="width=device-width, initial-scale=1">
      <link rel="stylesheet" href="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/css/bootstrap.min.css">
      <script src="https://ajax.googleapis.com/ajax/libs/jquery/3.5.1/jquery.min.js"></script>
      <script src="https://maxcdn.bootstrapcdn.com/bootstrap/3.4.1/js/bootstrap.min.js"></script>
        </head>
        <body>
        <div class="container">
      <h2>Reporte de errores</h2>
      <p>Lista de errores</p>
      <table class="table">
        <thead>
          <tr>
           <th>Fila</th>
            <th>Columna</th>
            <th>Token</th>
            <th>Lexemna</th>
          </tr>
        </thead>

        """)
    for i in range(len(self.error)):
        contenido += (" <tbody>"
                      "<td>" + str(self.error[i][0]) + "</td>"

                       "<td>" + str(self.error[i][1]) + "</td>"
                       "<td>" + str(self.error[i][2]) + "</td>"
                       "<td>" + str(self.error[i][3]) + "</td>"
                       "</tbody>")
    htmFile.write(contenido)
    htmFile.write("""
     </table>
```

def vistaFactura(self):

```
def vistaFactura(self):
    contenido = ''
    htmFile = open("VistaFactura" + ".html", "w", encoding='utf8')
    htmFile.write("""<!DOCTYPE html>
                            <html lang="es">
                            <head>
                            <meta charset="UTF-8">
                            <meta name="viewport" content="width=device-width, initial-scale=1.0">
                            <title>Bootstrap 4. Tablas</title>
                            <link rel="stylesheet" href="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/css/bootstrap.min.css">
                            </head>
                            <body>
                            <div class="container">
                            <div class="row">
                            <div class="col">
                            <table class="table table-striped table-bordered table-hover table-dark">
                            <thead>
                            <tr>
                                    <th>Factura</th>

                            </tr>
                        </thead>
                        """)

    contenido += "<tbody>" + "<tr>" + "<td>" + "Nombre: " +str(self.listTokens[0][3]) + "</td >" + "</tr>" + "</tbody>"
    contenido += "<tbody>" + "<tr>" + "<td>" + "NIT: "+str(self.listTokens[1][3]) + "</td >" + "</tr>" + "</tbody>"
    contenido += "<tbody>" + "<tr>" + "<td>" +"Direccion"+ str(self.listTokens[2][3]) + "</td >" + "</tr>" + "</tbody>"
    htmFile.write(contenido)
    htmFile.write("""
                    </table>
                </div>
            </div>
        <script src="https://code.jquery.com/jquery-3.4.1.slim.min.js"></script>
        <script src="https://cdn.jsdelivr.net/npm/popper.js@1.16.0/dist/umd/popper.min.js"></script>
        <script src="https://stackpath.bootstrapcdn.com/bootstrap/4.4.1/js/bootstrap.min.js"></script>
    </body>
    </html>""")
    htmFile.close
```

Menu.

Menu de opciones para moverse en las diferentes funcionabilidades.

```python
from AutomataMenu import *
from funtion import *
from AutomataFactura import *
entrada = automataMenu()
entrada2 = AutomataFactura()
def menu():

    print("Proyecto 1")
    print("-----------------")
    print("1.- Cargar Menu")
    print("2.- Cargar Orden")
    print("3.- Generar Menu")
    print("4.- Generar Factura")
    print("5.- Generar arbol")
    print("6.- Salir")
    opcion = input('>>Ingrese una opcion: ')
    return opcion
```

```python
def menuP():

    while True:
        opcion = menu()
        if opcion == '1':

            cadena = openFile()
            entrada.analizador(cadena)
            entrada.sintaxisError()
            entrada.reporteToken()
            entrada.reporteError()


        elif opcion == '2':
            factura = openFile()
            entrada2.analizadorF(factura)
            entrada2.reporteToken()
            entrada2.reporterror()

        if opcion == '3':

            listaOrdenada = entrada.ordenarSecciones()
            entrada.vista(listaOrdenada)
        if opcion == '4':
            entrada2.vistaFactura()
        elif opcion == '5':
            listaOrdenada = entrada.ordenarSecciones()
            entrada.generarGraph(listaOrdenada)

        if opcion == '6':
            break
```