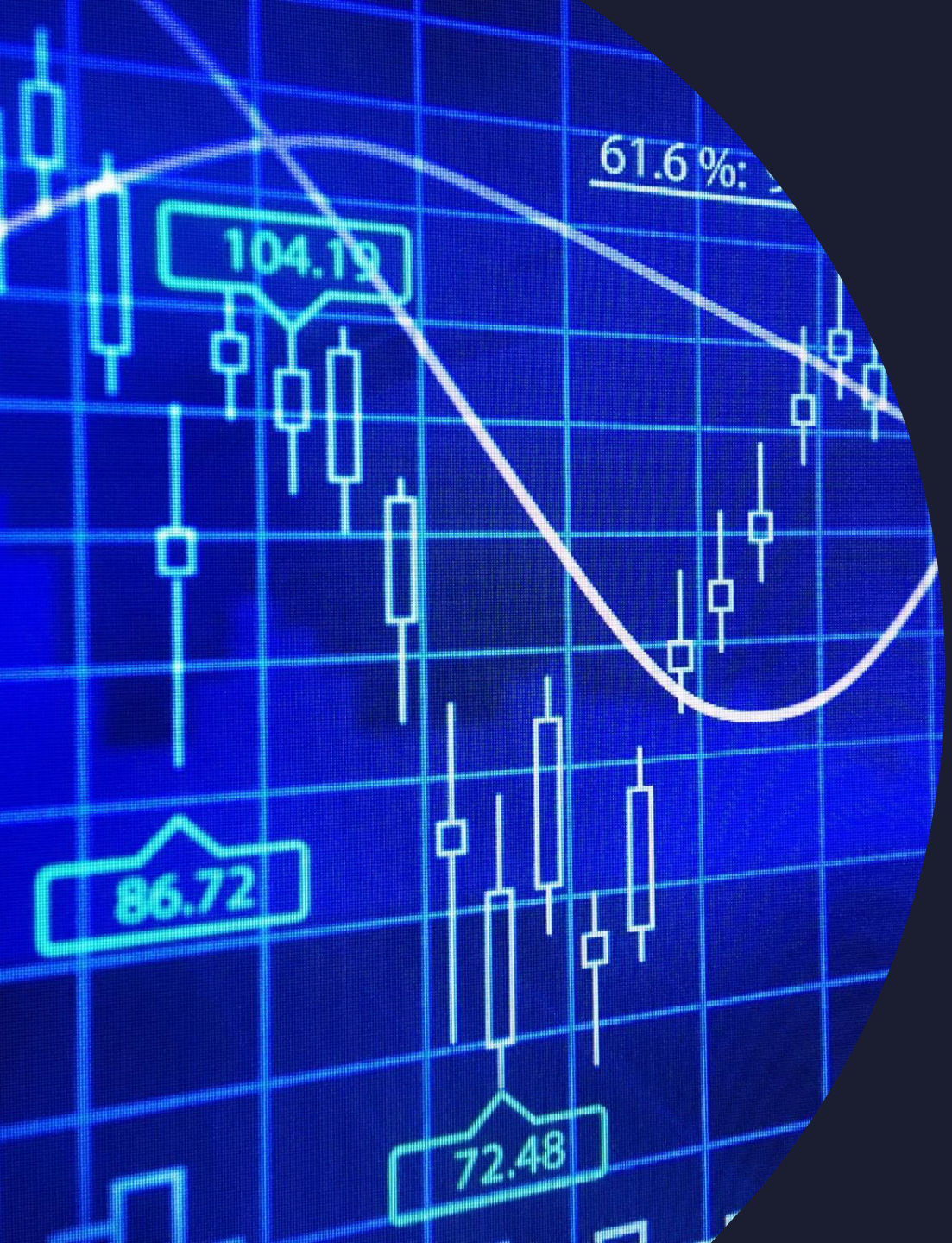


Proyecto Análisis Sintáctico

Juan Daniel Salazar Coronado - 17289193

Diseño de compiladores

Profesor : Ernesto Liñán García

Facultad de Sistemas – Universidad Autónoma de Coahuila

Léxico del proyecto :

Proyecto 12: Salazar Coronado Daniel

$\sim digito^+$	$*	@)	@>=
$\sim digito^+ edigito^+$	@/	@|	@<
$\sim digito^+.digito^+$	-	@&	@<=
$\sim digito^+.digito^+ edigito^+$	+	@[	“
$letra(letra\|digito\|\_\|-)^*$	@{	@]	;
<+	@}	<:-	.
<-	@(	@>	$
@	\	,	‘
:	!	\n	%=%

Palabras reservadas:

$suppose	$Out	$string	
$while	$Get	$cases	
$Repeat	$main	$case	
$for	$enter		
$enter	$integer		
$Finishing	$real		

Símbolos terminales :

- :
- @[
- @]
- @(
- @)
- .
- @{
- @}
- ;
- <+
- <-
- $case
- !
- %=%
- @>
- @<
- @<=
- @>=
- @|
- @&
- +
- -
- $*
- @/
- Todas las palabras reservadas
- Símbolo nulo
- Identificador
- numero

Símbolos NO terminales :

- P
- VARIABLES
- DECLARACION
- VARS
- TIPO
- LISTA
- LISTAPRIMA
- INSTRUCCIONES
- COMANDO
- CUERPOINSTRUCCION
- DETALLE-FOR
- CUERPOWHILE
- CUERPOIF
- INCREDECRE
- SIGNO
- ASIGNACION
- LISTACASES
- CONDICION
- OPERADOR
- EXPRESION
- EXPPRIMA
- TERMINO
- TERMPRIMO
- FACTOR

Símbolo inicial no terminal :

P

Conjunto de producciones :

Gramática:

1) P→$main: $enter VARIABLES INSTRUCCIONES $finishing
2) VARIABLES →@[DECLARACION @] VARIABLES
3) VARIABLES → ε
4) DECLARACION → VARS DECLARACION
5) DECLARACION → ε
6) VARS → @(TIPO LISTA @)
7) TIPO → $integer
8) TIPO → $real
9) TIPO → $string
10) LISTA → identificador LISTAPRIMA
11) LISTAPRIMA → LISTA
12) LISTAPRIMA → ε
13) INSTRUCCIONES → COMANDO. INSTRUCCIONES
14) INSTRUCCIONES → ε
15) COMANDO → $suppose CUERPOIF
16) COMANDO→ $for @ (DETALLE-FOR@) CUERPOINSTRUCCION
17) CUERPOINSTRUCCION→{{ INSTRUCCIONES }}
18) DETALLE-FOR→ ASIGNACION; identificador OPERADOR EXPRESION; INCREDECRE
19) COMANDO → $while CUERPOWHILE
20) CUERPOWHILE→ CONDICION CUERPOINSTRUCCION
21) CUERPOIF → CONDICION CUERPOINSTRUCCION
22) COMANDO → $Repeat CUERPOINSTRUCCION CONDICION
23) COMANDO → $cases identificador @[LISTACASES @]
24) COMANDO → $Out @[identificador @]
25) COMANDO → $Get @[identificador @]
26) COMANDO → ASIGNACION
27) INCREDECRE → identificador SIGNO
28) SIGNO →<+
29) SIGNO →<-
30) ASIGNACION → identificador <:- EXPRESION
31) LISTACASES → $case número CUERPOINSTRUCCION LISTACASES
32) LISTACASES → ε

33) CONDICION →@(EXPRESION OPERADOR EXPRESION@)
34) CONDICION→ ! @ (EXPRESION OPERADOR EXPRESION@)
35) OPERADOR → %=%
36) OPERADOR → @>
37) OPERADOR →@<
38) OPERADOR → @<=
39) OPERADOR → @>=
40) OPERADOR → @|
41) OPERADOR → @&
42) EXPRESION → TERMINO EXPPRIMA
43) EXPPRIMA → + TERMINO EXPPRIMA
44) EXPPRIMA → - TERMINO EXPPRIMA
45) EXPPRIMA → ε
46) TERMINO → FACTOR TERMPRIMO
47) TERMPRIMO → $* FACTOR TERMPRIMO
48) TERMPRIMO → @/ FACTOR TERMPRIMO
49) TERMPRIMO → ε
50) FACTOR → @(EXPRESION @)
52) FACTOR → identificador
52) FACTOR → número

First :

- First(P) = { $main (1) }
- First(VARIABLES) = { @[(2), nulo (3) }
- First(DECLARACION) = { @((4), nulo (5) }
- First(VARS) = { @((6) }
- First(TIPO) = { $integer (7), $real (8), $string (9) }
- First(LISTA) = { identificador (10) }
- First(LISTAPRIMA) = { identificador (11), nulo (12) }
- First(INSTRUCCIONES) = { $suppose (13), $for (13), $while (13), $Repeat (13), $cases (13), $Out (13), $Get (13), identificador (13), nulo (14) }
- First(COMANDO) = {$suppose (15), $for (16), $while (19), $Repeat (22), $cases (23), $Out (24), $Get (25), identificador (26) }
- First(CUERPOINSTRUCCION) = { @{ (17) }
- First(DETALLE-FOR) = { identificador (18) }
- First(CUERPOWHILE) = { @((20), ! (20) }

- First(CUERPOIF) = { @((21), ! (21) }
- First(INCREDECRE) = { identificador (27) }
- First(SIGNO) = { <+ (28), <- (29) }
- First(ASIGNACION) = { identificador (30) }
- First(LISTACASES) = { $case (31), nulo (32) }
- First(CONDICION) = { @((33), ! (34) }
- First(OPERADOR) = { %=% (35), @> (36), @< (37), @<= (38), @>= (39), @| (40), @& (41) }
- First(EXPRESION) = { @((42), identificador (42), numero (42) }
- First(EXPPRIMA) = { + (43), - (44), nulo (45) }
- First(TERMINO) = { @((46), identificador (46), numero (46) }
- First(TERMPRIMA) = { $* (47), @/ (48), nulo (49) }
- First(FACTOR) = {@((50), identificador (51), numero (52) }

Follow :

- Follow(P) = { EOF($) }
- Follow(VARIABLES) = {$suppose, $for, $while, $Repeat, $cases, $Out , $Get , identificador , $finishing }
- Follow(DECLARACION) = { @] }
- Follow(VARS) = { @(, @] }
- Follow(TIPO) = { identificador }
- Follow(LISTA) = { @) }
- Follow(LISTAPRIMA) = { @) }
- Follow(INSTRUCCIONES) = { $finishing , @} }
- Follow(COMANDO) ={ . }
- Follow(CUERPOINSTRUCCION) = { ., @(, !, $case }
- Follow(DETALLE-FOR) = { @) }
- Follow(CUERPOWHILE) = { . }

- Follow(CUERPOIF) = { . }
- Follow(INCREDECRE) = { @) }
- Follow(SIGNO) = { @) }
- Follow(ASIGNACION) = { ; . }
- Follow(LISTACASES) = { @] }
- Follow(CONDICION) = { @{, . }
- Follow(OPERADOR) = {@(, identificador, numero }
- Follow(EXPRESION) = { ; , . , %=% , @> , @< , @<= , @>= , @| , @&, @) }
- Follow(EXPPRIMA) = { ; , . , %=% , @> , @< , @<= , @>= , @| , @&, @) }
- Follow(TERMINO) = { + , - , ; , . , %=% , @> , @< , @<= , @>= , @| , @&, @) }
- Follow(TERMPRIMA) = {+ , - , ; , . , %=% , @> , @< , @<= , @>= , @| , @&, @) }
- Follow(FACTOR) = { $*, @/, + , - , ; , . , %=% , @> , @< , @<= , @>= , @| , @&, @) }

Tabla sintáctica :

Nota : Dar doble clic a la tabla
Para poder interactuar con el
Excel.

404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	3	3
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	13	13
404	404	404	404	404	404	404	404	404	404	404	404	404	15	19
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	20	404	404	404	404	404	404	404	404	404	404	404	404	404
404	21	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
31	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	34	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	35	36	37	38	39	40	41	404	404	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	45	45	45	45	45	45	45	43	44	404	404	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404
404	404	49	49	49	49	49	49	49	49	49	47	48	404	404
404	404	404	404	404	404	404	404	404	404	404	404	404	404	404

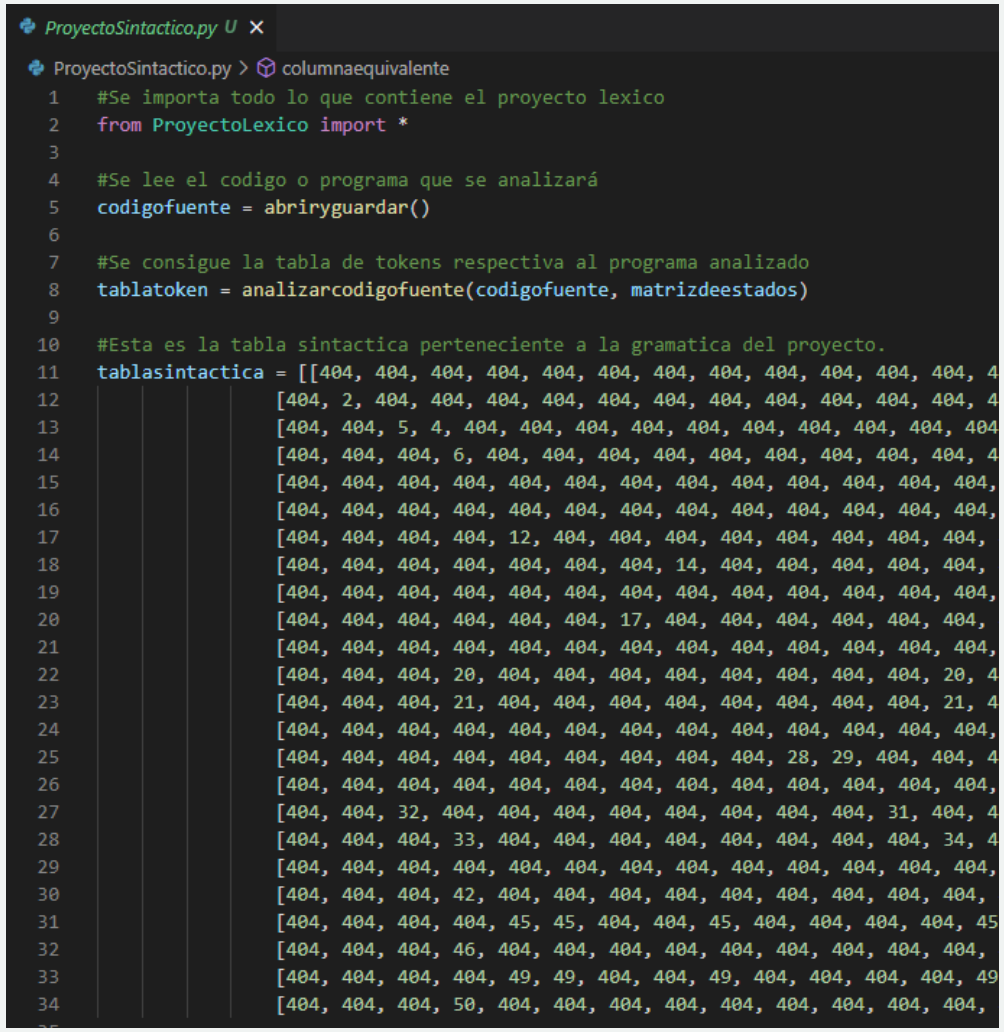
Token de cada símbolo no terminal :

133 : :
115 : @[
116 : @]
111 : @(
112 : @)
127 : .
109 : @{
110 : @}
126 : ;
123 : <+
121 : <-
104.16 : $case
134 : !
135 : %=%

119 : @>
117 : @<
118 : @<=
120 : @>=
113 : @|
114 : @&
138 : +
139 : -
106 : $*
108 : @/
104.2 : $suppose
104.3 : $while
104.4 : $Repeat
104.5 : $for

104.6 : $enter
104.7 : $Finishing
104.8 : $Out
104.9 : $Get
104.17 : $main
104.12 : $integer
104.13 : $real
104.14 : $string
104.15 : $cases
104.1 : identificador
100 : numero
105 : $ END OF FILE

Documentación



```
ProyectoSintactico.py U
ProyectoSintactico.py > columnaequivalente
1   #Se importa todo lo que contiene el proyecto lexico
2   from ProyectoLexico import *
3
4   #Se lee el codigo o programa que se analizará
5   codigofuente = abriryguardar()
6
7   #Se consigue la tabla de tokens respectiva al programa analizado
8   tablatoken = analizarcodigofuente(codigofuente, matrizdeestados)
9
10  #Esta es la tabla sintactica perteneciente a la gramatica del proyecto.
11  tablasintactica = [[404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 4
12                      [404, 2, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 4
13                      [404, 404, 5, 4, 404, 404, 404, 404, 404, 404, 404, 404, 404
14                      [404, 404, 404, 6, 404, 404, 404, 404, 404, 404, 404, 404, 4
15                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
16                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
17                      [404, 404, 404, 404, 12, 404, 404, 404, 404, 404, 404, 404,
18                      [404, 404, 404, 404, 404, 404, 404, 14, 404, 404, 404, 404,
19                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
20                      [404, 404, 404, 404, 404, 404, 17, 404, 404, 404, 404, 404,
21                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
22                      [404, 404, 404, 20, 404, 404, 404, 404, 404, 404, 404, 404, 20, 4
23                      [404, 404, 404, 21, 404, 404, 404, 404, 404, 404, 404, 404, 21, 4
24                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
25                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 28, 29, 404, 404, 4
26                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
27                      [404, 404, 32, 404, 404, 404, 404, 404, 404, 404, 404, 31, 404, 4
28                      [404, 404, 404, 33, 404, 404, 404, 404, 404, 404, 404, 404, 34, 4
29                      [404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404, 404,
30                      [404, 404, 404, 42, 404, 404, 404, 404, 404, 404, 404, 404, 404,
31                      [404, 404, 404, 404, 45, 45, 404, 404, 45, 404, 404, 404, 404, 45
32                      [404, 404, 404, 46, 404, 404, 404, 404, 404, 404, 404, 404, 404,
33                      [404, 404, 404, 404, 49, 49, 404, 404, 49, 404, 404, 404, 404, 49
34                      [404, 404, 404, 50, 404, 404, 404, 404, 404, 404, 404, 404, 404,
```

Para comenzar el código del análisis sintáctico se deben exportar y utilizar todo lo hecho en el análisis léxico, este es el procedimiento que se realiza de la línea 1 a la línea 8.

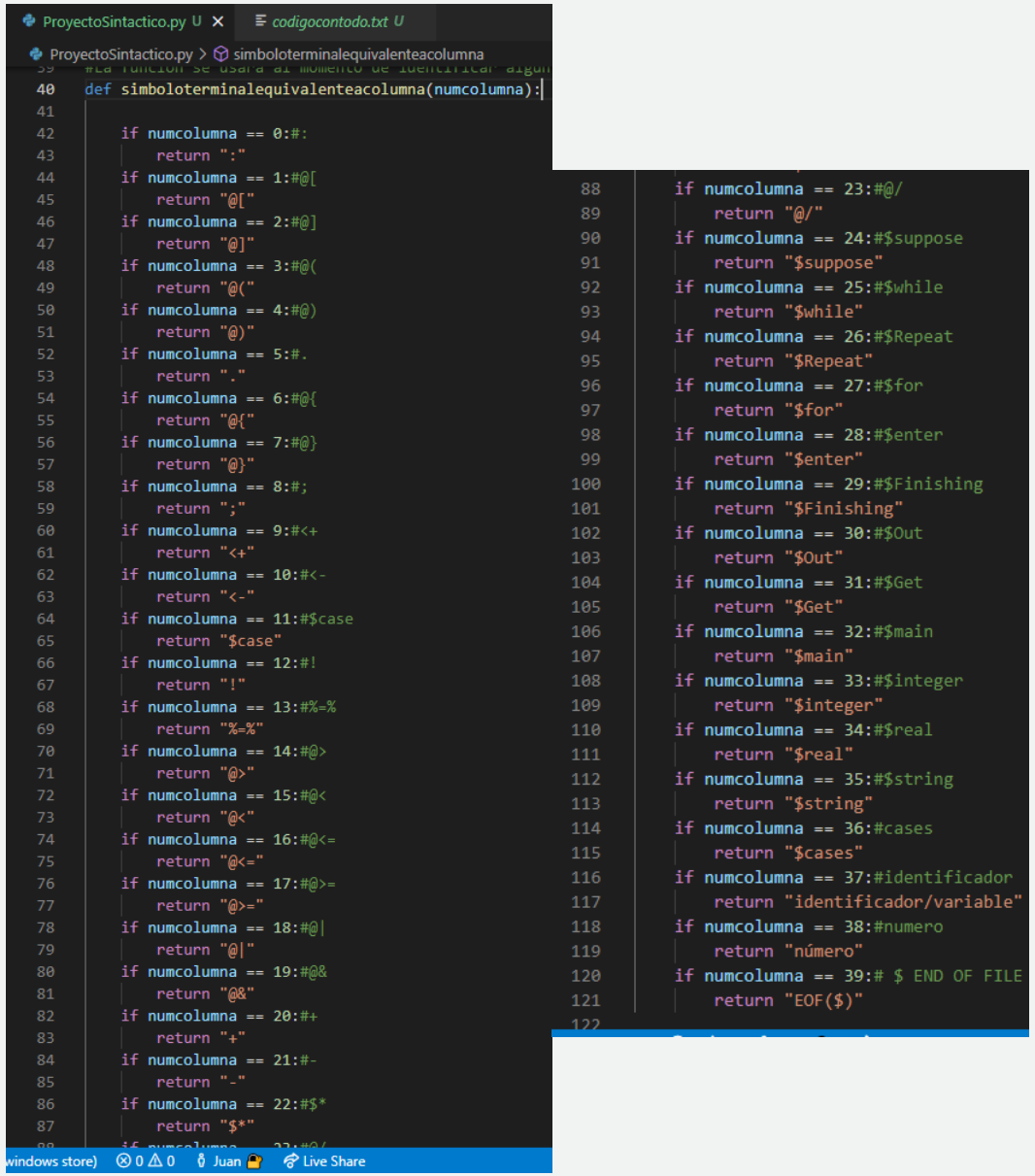
Línea 2: Se exporta todo lo que esta dentro de el archivo ProyectoLexico.py.

Línea 5: Se llama a la función que abre y guarda el contenido del texto que se va a analizar.
Nota: para cambiar el archivo que se analiza
Se debe ir al archivo ProyectoLexico.py, en la función código fuente será donde escriba el nombre del archivo que quiere analizar.

Línea 8: Se llama a la función que hace el proceso de generar todos los token de los elementos que se encontraron y se guardan en una lista.

Después de estas líneas se declara la tabla sintáctica como una lista de listas en la línea 11.

Función “simboloterminalequivalenteacolumna”

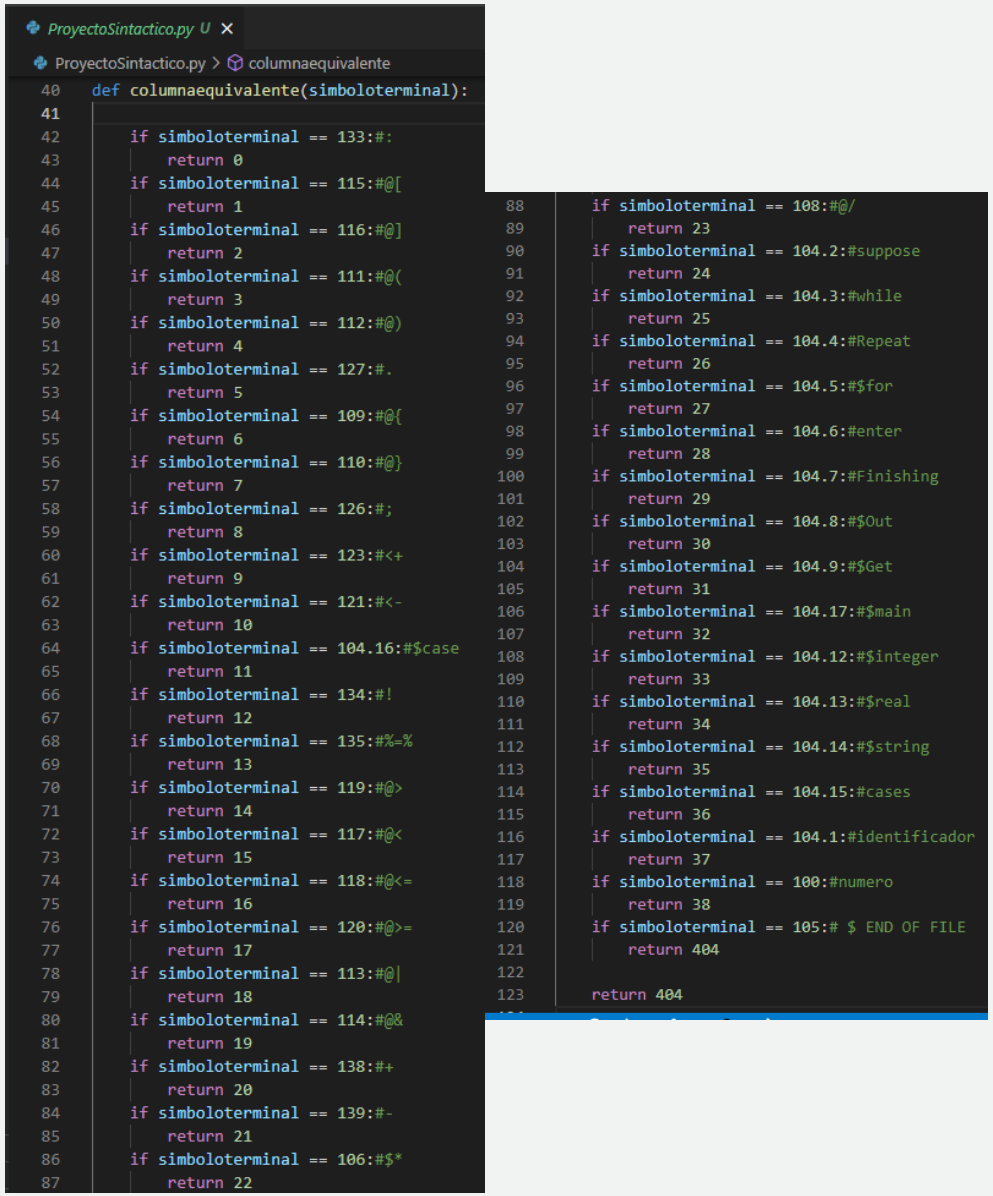


- Objetivo: Retornar el símbolo terminal que equivale al numero de columna de la tabla sintáctica que se introduzca.

- Parámetros: Numero de columna de la tabla sintáctica.

- Funcionamiento: La función recibe un numero de columna de la tabla sintáctica, ese numero se va comparando en muchos if para saber a que símbolo terminal equivale, cuando entra en un if se retorna el símbolo terminal que le pertenece a el numero de columna que se ingreso.

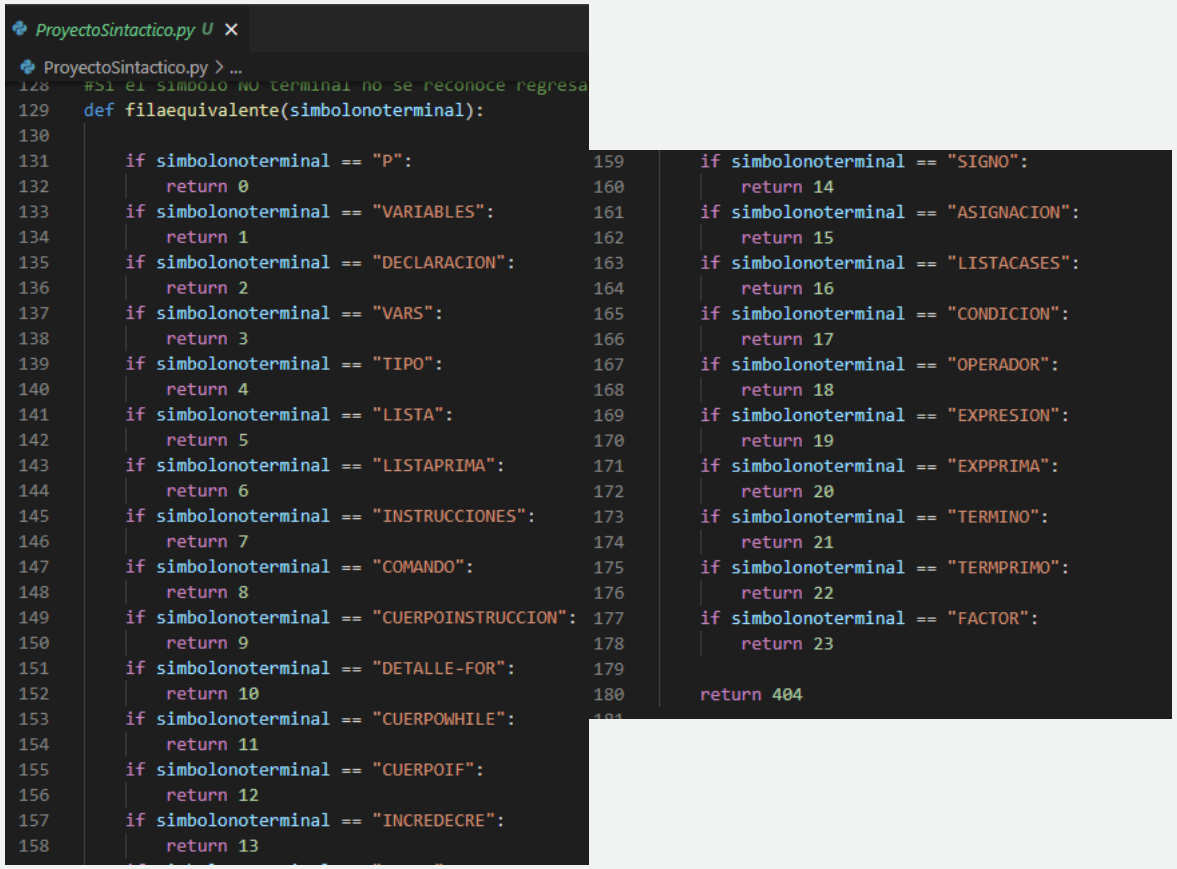
Función “columnaequivalente”



```
ProyectoSintactico.py U
ProyectoSintactico.py > columnaequivalente
40  def columnaequivalente(simboloterminal):
41
42      if simboloterminal == 133:#:
43          return 0
44      if simboloterminal == 115:#@[
45          return 1
46      if simboloterminal == 116:#@]
47          return 2
48      if simboloterminal == 111:#@(
49          return 3
50      if simboloterminal == 112:#@)
51          return 4
52      if simboloterminal == 127:#.
53          return 5
54      if simboloterminal == 109:#@{
55          return 6
56      if simboloterminal == 110:#@}
57          return 7
58      if simboloterminal == 126:#;
59          return 8
60      if simboloterminal == 123:#<+
61          return 9
62      if simboloterminal == 121:#<-
63          return 10
64      if simboloterminal == 104.16:#$case
65          return 11
66      if simboloterminal == 134:#!
67          return 12
68      if simboloterminal == 135:#%=%
69          return 13
70      if simboloterminal == 119:#@>
71          return 14
72      if simboloterminal == 117:#@<
73          return 15
74      if simboloterminal == 118:#@<=
75          return 16
76      if simboloterminal == 120:#@>=
77          return 17
78      if simboloterminal == 113:#@|
79          return 18
80      if simboloterminal == 114:#@&
81          return 19
82      if simboloterminal == 138:#+
83          return 20
84      if simboloterminal == 139:#-
85          return 21
86      if simboloterminal == 106:#$*
87          return 22
88      if simboloterminal == 108:#@/
89          return 23
90      if simboloterminal == 104.2:#suppose
91          return 24
92      if simboloterminal == 104.3:#while
93          return 25
94      if simboloterminal == 104.4:#Repeat
95          return 26
96      if simboloterminal == 104.5:#$for
97          return 27
98      if simboloterminal == 104.6:#enter
99          return 28
100     if simboloterminal == 104.7:#Finishing
101         return 29
102     if simboloterminal == 104.8:#$Out
103         return 30
104     if simboloterminal == 104.9:#$Get
105         return 31
106     if simboloterminal == 104.17:#$main
107         return 32
108     if simboloterminal == 104.12:#$integer
109         return 33
110     if simboloterminal == 104.13:#$real
111         return 34
112     if simboloterminal == 104.14:#$string
113         return 35
114     if simboloterminal == 104.15:#cases
115         return 36
116     if simboloterminal == 104.1:#identificador
117         return 37
118     if simboloterminal == 100:#numero
119         return 38
120     if simboloterminal == 105:# $ END OF FILE
121         return 404
122
123     return 404
```

- Objetivo: Retornar la columna que equivale al símbolo terminal que se introduzca.
- Parámetros: Token de un símbolo terminal.
- Funcionamiento: La función recibe el token de un símbolo terminal, ese token se va comparando en muchos if para saber a que token de símbolo terminal equivale, cuando entra en un if se retorna la columna que le pertenece en la tabla sintáctica. Si no se encuentra una coincidencia con ningún símbolo NO terminal establecido retorna un token 404, el cual quiere decir que hubo un error.

Función "filaquivalente"



```
ProyectoSintactico.py U
ProyectoSintactico.py > ...
#Si el simbolo NO terminal no se reconoce regresa
def filaequivalente(simbolonoterminal):

    if simbolonoterminal == "P":
        return 0
    if simbolonoterminal == "VARIABLES":
        return 1
    if simbolonoterminal == "DECLARACION":
        return 2
    if simbolonoterminal == "VARS":
        return 3
    if simbolonoterminal == "TIPO":
        return 4
    if simbolonoterminal == "LISTA":
        return 5
    if simbolonoterminal == "LISTAPRIMA":
        return 6
    if simbolonoterminal == "INSTRUCCIONES":
        return 7
    if simbolonoterminal == "COMANDO":
        return 8
    if simbolonoterminal == "CUERPOINSTRUCCION":
        return 9
    if simbolonoterminal == "DETALLE-FOR":
        return 10
    if simbolonoterminal == "CUERPOWHILE":
        return 11
    if simbolonoterminal == "CUERPOIF":
        return 12
    if simbolonoterminal == "INCREDECRE":
        return 13
    if simbolonoterminal == "SIGNO":
        return 14
    if simbolonoterminal == "ASIGNACION":
        return 15
    if simbolonoterminal == "LISTACASES":
        return 16
    if simbolonoterminal == "CONDICION":
        return 17
    if simbolonoterminal == "OPERADOR":
        return 18
    if simbolonoterminal == "EXPRESION":
        return 19
    if simbolonoterminal == "EXPPRIMA":
        return 20
    if simbolonoterminal == "TERMINO":
        return 21
    if simbolonoterminal == "TERMPRIMO":
        return 22
    if simbolonoterminal == "FACTOR":
        return 23

    return 404
```

- Objetivo: Retornar la fila que equivale al símbolo NO terminal que se introduzca.

- Parámetros: Token de un símbolo NO terminal.

- Funcionamiento: La función recibe un token de un símbolo NO terminal, ese token se va comparando en muchos if para saber a que token de símbolo NO terminal equivale, cuando entra en un if se retorna la fila que le pertenece en la tabla sintáctica. Si no se encuentra una coincidencia con ningún símbolo NO terminal establecido retorna un token 404, el cual quiere decir que hubo un error.

Función “obtenerproducción”

ProyectoSintactico.py U

ProyectoSintactico.py > ...

```
#Los simbolos terminales se reemplazan por su token equivalente.
def obtenerproduccion(numproduccion, stacksintactico):

    if numproduccion == 1:
        stacksintactico.extend([104.7, "INSTRUCCIONES", "VARIABLES", 104.6, 133, 104.17])
        return stacksintactico
    if numproduccion == 2:
        stacksintactico.extend(["VARIABLES", 116, "DECLARACION", 115])
        return stacksintactico
    if numproduccion == 3:
        stacksintactico.extend([])#no se añade nada pues la producción es nula.
        return stacksintactico
    if numproduccion == 4:
        stacksintactico.extend(["DECLARACION", "VARS"])
        return stacksintactico
    if numproduccion == 5:
        stacksintactico.extend([])#no se añade nada pues la producción es nula.
        return stacksintactico
    if numproduccion == 6:
        stacksintactico.extend([112, "LISTA", "TIPO", 111])
        return stacksintactico
    if numproduccion == 7:
        stacksintactico.extend([104.12])
        return stacksintactico
    if numproduccion == 8:
        stacksintactico.extend([104.13])
        return stacksintactico
    if numproduccion == 9:
        stacksintactico.extend([104.14])
        return stacksintactico
    if numproduccion == 10:
        stacksintactico.extend(["LISTAPRIMA", 104.1])
        return stacksintactico
    if numproduccion == 11:
        stacksintactico.extend(["LISTA"])
        return stacksintactico
    if numproduccion == 12:
        stacksintactico.extend([])#no se añade nada pues la producción es nula.
        return stacksintactico
    if numproduccion == 13:
        stacksintactico.extend(["INSTRUCCIONES", 127, "COMANDO"])
        return stacksintactico
    if numproduccion == 14:
        stacksintactico.extend([])#no se añade nada pues la producción es nula.
        return stacksintactico
    if numproduccion == 15:
        stacksintactico.extend(["CUERPOIF", 104.2])
        return stacksintactico
    if numproduccion == 16:
        stacksintactico.extend(["CUERPOINSTRUCCION", 112, "DETALLE-FOR", 111, 104.5])
        return stacksintactico
    if numproduccion == 17:
        stacksintactico.extend([110, "INSTRUCCIONES", 109])
        return stacksintactico
    if numproduccion == 18:
        stacksintactico.extend(["INCREDECRE", 126, "EXPRESION", "OPERADOR", 104.1, 126, "ASIGNACION"])
        return stacksintactico
    if numproduccion == 19:
        stacksintactico.extend(["CUERPOWHILE", 104.3])
        return stacksintactico
    if numproduccion == 20:
        stacksintactico.extend(["CUERPOINSTRUCCION", "CONDICION"])
        return stacksintactico
    if numproduccion == 21:
        stacksintactico.extend(["CUERPOINSTRUCCION", "CONDICION"])
        return stacksintactico
    if numproduccion == 22:
        stacksintactico.extend(["CONDICION", "CUERPOINSTRUCCION", 104.4])
        return stacksintactico
    if numproduccion == 23:
        stacksintactico.extend([116, "LISTACASES", 115, 104.1, 104.15])
        return stacksintactico
    if numproduccion == 24:
        stacksintactico.extend([116, 104.1, 115, 104.8])
        return stacksintactico
    if numproduccion == 25:
        stacksintactico.extend([116, 104.1, 115, 104.9])
        return stacksintactico
    if numproduccion == 26:
        stacksintactico.extend(["ASIGNACION"])
        return stacksintactico
    if numproduccion == 27:
        stacksintactico.extend(["SIGNO", 104.1])
        return stacksintactico
    if numproduccion == 28:
        stacksintactico.extend([123])
        return stacksintactico
    if numproduccion == 29:
        stacksintactico.extend([121])
        return stacksintactico
    if numproduccion == 30:
        stacksintactico.extend(["EXPRESION", 122, 104.1])
        return stacksintactico
    if numproduccion == 31:
        stacksintactico.extend(["LISTACASES", "CUERPOINSTRUCCION", 100, 104.16])
        return stacksintactico
    if numproduccion == 32:
        stacksintactico.extend([])#no se añade nada pues la produccion es nula.
        return stacksintactico
    if numproduccion == 33:
        stacksintactico.extend([112, "EXPRESION", "OPERADOR", "EXPRESION", 111])
        return stacksintactico
    if numproduccion == 34:
        stacksintactico.extend([112, "EXPRESION", "OPERADOR", "EXPRESION", 111, 134])
        return stacksintactico
    if numproduccion == 35:
        stacksintactico.extend([135])
        return stacksintactico
    if numproduccion == 36:
        stacksintactico.extend([119])
        return stacksintactico
    if numproduccion == 37:
        stacksintactico.extend([117])
        return stacksintactico
    if numproduccion == 38:
        stacksintactico.extend([118])
        return stacksintactico
    if numproduccion == 39:
        stacksintactico.extend([120])
        return stacksintactico
    if numproduccion == 40:
        stacksintactico.extend([113])
        return stacksintactico
    if numproduccion == 41:
        stacksintactico.extend([114])
        return stacksintactico
    if numproduccion == 42:
        stacksintactico.extend(["EXPPRIMA", "TERMINO"])
        return stacksintactico
    if numproduccion == 43:
        stacksintactico.extend(["EXPPRIMA", "TERMINO", 138])
        return stacksintactico
    if numproduccion == 44:
        stacksintactico.extend(["EXPPRIMA", "TERMINO", 139])
        return stacksintactico
```

```
324     if numproduccion == 45:
325         stacksintactico.extend([])#no se añade nada pues la producción es nula
326         return stacksintactico
327     if numproduccion == 46:
328         stacksintactico.extend(["TERMPRIMO", "FACTOR"])
329         return stacksintactico
330     if numproduccion == 47:
331         stacksintactico.extend(["TERMPRIMO", "FACTOR", 106])
332         return stacksintactico
333     if numproduccion == 48:
334         stacksintactico.extend(["TERMPRIMO", "FACTOR", 108])
335         return stacksintactico
336     if numproduccion == 49:
337         stacksintactico.extend([])#no se añade nada pues la producción es nula
338         return stacksintactico
339     if numproduccion == 50:
340         stacksintactico.extend([112, "EXPRESION", 111])
341         return stacksintactico
342     if numproduccion == 51:
343         stacksintactico.extend([104.1])
344         return stacksintactico
345     if numproduccion == 52:
346         stacksintactico.extend([100])
347         return stacksintactico
348 
349     stacksintactico.extend([404])
350     return stacksintactico
```

- Objetivo: Añadir al stack sintáctico la siguiente producción.
- Parámetros: Número de la producción que va a añadirse, y lista que contenga el stack sintáctico.
- Funcionamiento: La función recibe el numero de la producción que va a añadirse, ese numero se compara en muchos if´s, si entra a alguno de ellos se añade esa producción al stack sintáctico actual y después retorna la lista que contiene el stack sintáctico que se añadió junto a los elementos de la producción que se añadieron, si el numero de la producción no entra en ningún if se añade un token 404 al stack sintáctico, el cual representa un error.

Nota: En las producciones los símbolos terminales están escritos con su token y los elementos de las producciones están escritos de derecha a izquierda.

Función “tokensqueseesperan”

```python
def tokensqueseesperan(fila, tablasintactica):
    columnas = []
    tokensesperados = []
    contador = 0

    for token in tablasintactica[fila]:
        if token != 404:
            columnas.append(contador)
        contador+=1

    for col in columnas:
        simboloterminal = simboloterminalequivalenteacolumna(col)
        tokensesperados.append(simboloterminal)

    return tokensesperados
```

- Objetivo: Retornar los símbolos terminales que son permitidos cuando se encuentra un error sintáctico.
- Parámetros: Tabla sintáctica y ultima fila a la que se ingreso antes de dar error.
- Declaraciones:

❑ Se declara una lista la cual servirá para guardar el numero de la columna en que el elemento que esta en la fila no es 404.

❑ Se declara una lista donde se guardarán los tokens permitidos en donde dio error sintáctico.

❑ Se declara un contador como auxiliar para llevar la cuenta de cuantas iteraciones ha dado el primer ciclo for en la función.

Funcionamiento:

En un ciclo for se itera sobre todos los token que pertenecen a una fila, cada vez que uno de esos token es diferente a 404 el numero de la columna en la que esta se añade a otra lista que lleva por nombre “columnas”.

A continuación en otro ciclo for se recorre la lista que se lleno en el for anterior y se obtiene el símbolo terminal al que le pertenece las columnas que se encontraron y se añaden a la lista que lleva por nombre “tokensesperados”.

Al final se retorna la lista “tokensesperados”.

Función “analisissintactico”

```
332 def analisissintactico(tablasintactica, tablatoken):
333     stacksintactico = [105, "P"]
334 
335     while len(tablatoken) > 0:
336 
337         if stacksintactico[-1] == tablatoken[0]:
338             print("------------------------------------------------------------")
339             print("Los elementos coinciden, se reconoce un simbolo terminal.")
340             print(f"Token de simbolo terminal reconocido: {tablatoken[0]}")
341             print(f"Estado de stack sintactico: {stacksintactico}")
342             print(f"Estado de tabla de tokens: {tablatoken}")
343             stacksintactico.pop()
344             tablatoken.pop(0)
345             print(f"Estado de stack sintactico después de pop: {stacksintactico}")
346             print(f"Estado de tabla de tokens después de pop: {tablatoken}")
347             print("------------------------------------------------------------")
348         else:
349 
350             columna = columnaequivalente(tablatoken[0])
351             fila = filaequivalente(stacksintactico[-1])
352             print(columna)
353             print(fila)
354 
355             if columna == 404 or fila == 404:
356                 print(f"SE ENCONTRO UN ERROR, SE ESPERABA: {stacksintactico[-1]}")
357                 break
358 
359             numproduccion = tablasintactica[fila][columna]
360 
361             print("------------------------------------------------------------")
362             print("No se reconocio ningún simbolo terminal")
363             print(f"Estado de stack sintactico antes de añadir producción: {stacksintactico}")
364             print(f"Producción que va a incluirse: {numproduccion}")
365             stacksintactico.pop()
366             stacksintactico = obtenerproduccion(numproduccion, stacksintactico)
367             print(f"Estado de stack sintactico después de añadir producción: {stacksintactico}")
368             print(f"Tabla de tokens: {tablatoken}")
369             print("------------------------------------------------------------")
370 
371     if len(tablatoken) == 0:
372         print("EL PROGRAMA SE ANALIZO SIN PROBLEMAS, NO HAY ERROR SINTACTICO. ")
373 
```

- Objetivo: Desarrollar el proceso de un análisis sintáctico.
- Parámetros: Tabla sintáctica y stack de tokens de símbolos terminales generado por el análisis léxico.
- Declaraciones:

  Se declara una lista la cual servirá para guardar el stack sintáctico, y se inicializa con el símbolo EOF y el símbolo inicial no terminal de la gramática.

- Funcionamiento:

Dentro de un ciclo while que continuara hasta que la tabla de tokens este vacía se indica un if, este if compara el primer elemento en la tabla token y el ultimo elemento en el stack sintáctico, si estos elementos son iguales los dos se popean/eliminan de su respectiva lista, si estos elementos no son iguales se llama a las funciones “columnaequivalente” con el primer elemento en la tabla token y “filaequivalente” con el ultimo elemento en el stack sintáctico para obtener la fila y la columna en la cual esta el numero de la producción que se va a añadir, si la fila o columna obtuvieron un token que representa error se ingresará al if en la línea 355 para informar del error y parar el ciclo, después se obtiene el numero de la producción que se va a añadir, a continuación se popea/elimina el ultimo elemento en el stack sintáctico y se llama a la función “obtenerproducción” para añadir los elementos de la producción respectiva. Al salir del ciclo while se tiene un if, al cual solo se entra si la tabla de tokens fue vaciada por completo, si esto es así se informa que el código fuente se analiza sin problemas y no se encontraron errores sintácticos.

```
423 analisissintactico(tablasintactica, tablatoken)
```

La ultima línea del código llama a la función “análisissintactico”

Código añadido a la función "analisissintactico"

El if que estaba en la línea 355 es editado para tener mucha más funcionalidad, y ahora imprimir el símbolo terminal que se esperaba y no solo informar que hubo un error.

```
503             #Se entra a este if cuando no se encuentra el simbolo terminal que debe seguir
504             #el if primero checa la columna donde se fallo, después nos da el simbolo
505             #terminal al que pertenece esa columna y se imprime que se espera ese simbolo terminal.
506             if fila == 404 or columna == 404:
507                 columnadesimboloterminal = columnaequivalente(stacksintactico[-1])
508                 tokenesperado = simboloterminalequivalenteacolumna(columnadesimboloterminal)
509                 print(f"SE ENCONTRO UN ERROR, SE ESPERA: ' {tokenesperado} '")
510                 print(f"Nota: si alguno de los tokens de simbolos terminales es de 200 a 203 el error es de lexico.")
511                 break
```

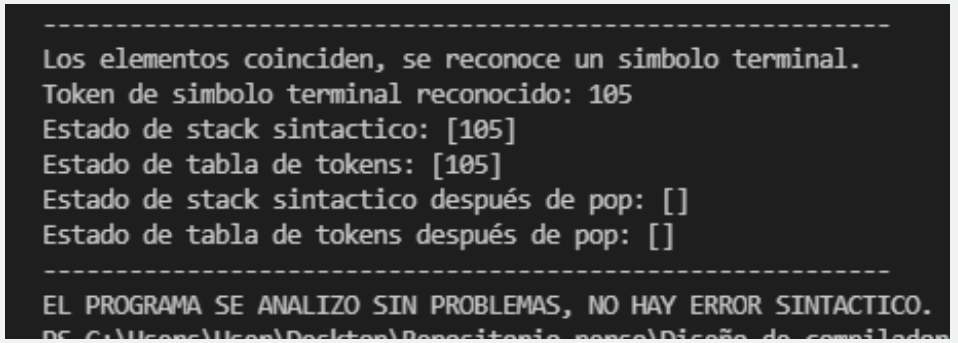
También se añade otro if al final del recorrido del ciclo while, este if nos informara específicamente si el error que se encontró es sintáctico, y nos dirá los símbolos terminales que son admitidos en donde hubo error.

```
533             print("------------------------------------------------------------")
534
535             #Si hay un error sintactico se entra a este if, y se imprimen los
536             #simbolos terminales que se pueden usar a continuación.
537             #el if hace uso de la función tokensqueseesperan.
538             if stacksintactico[-1] == 404:
539                 tokensesperados = tokensqueseesperan(fila, tablasintactica)
540                 print(f"SE ENCONTRO UN ERROR SINTACTICO, SE ESPERABA ALGUNO DE ESTOS TOKENS TERMINALES: {tokensesperados}")
541                 print(f"Nota: si alguno de los tokens de simbolos terminales es de 200 a 203 el error es de lexico.")
542                 break
```

El funcionamiento de la función continua exactamente igual, solo se añadieron esos ifs para ser más explicativos a la hora de encontrar algún error.

Pruebas

- Para probar el proyecto y obtener un resultado satisfactorio se debe analizar un código que respete la gramática del proyecto y no contenga errores léxicos ni sintácticos, cuando esto sucede la impresión en pantalla terminará así, con la lista de token y el stack sintáctico vacíos:



Ejemplos de código fuente que no da error:

Go Run Terminal Help codigocontodo.txt - ProyectoAnalisisSintactico-DanielSalazar-17289193 - Visual Studio

ProyectoSintactico.py U | codigocontodo.txt U

codigocontodo.txt

```
$main:

$enter
    @[ @( $integer $contador @) @]
    @[ @( $integer $seleccionoperacionprueba @) @]
    @[ @( $integer $result @) @]
    @[ @( $real $contador2 @) @]
    @[ @( $real $parasuppose @) @]
    @[ @( $real $num1 @) @]
    @[ @( $real $num2 @) @]
    @[ @( $string $nombreprueba @) @]

$Get @[ "Ingrese el valor del contador 1 " $contador @].
$Get @[ "Ingrese el valor del contador 2 " $contador2 @].
$Get @[ "Ingrese el nombre de quien quiere probar " $nombreprueba @].

$for @(  $i <:- ~1 ; $i %=% $contador ; $i <+ @)
    @{
        $Out @[ "Iteraciones for de prueba de " $nombreprueba @].
    @}.

$while @( $contador2 @<= ~2 @)
    @{
        $Out @[ 'Iteraciones while de prueba de ' $nombreprueba @].
    @}.

$Repeat @{
            $Out @[ "Iteraciones Repeat de prueba de " $nombreprueba @].
        @}
        @( $contador @>= ~5.5 @).

$Get @[ "Ingresa 1 para suma 2 para resta 3 para multiplicacion y 4 para division " $seleccionoperacionprueba @].
$Get @[ "Ingresa el primer numero " $num1 @].
$Get @[ "Ingresa el segundo numero " $num2 @].

$cases $seleccionoperacionprueba @[
    $case ~1
    @{
        $result <:- $num1 + $num2 .
        @}
    $case ~2
    @{
        $result <:- $num1 - $num2 .
        @}
    $case ~3
    @{
        $result <:- $num1 $* $num2 .
        @}
    $case ~4
```

windows store) ⊗ 0 ⚠ 0 Juan Live Share

```
    @{
        $result <:- $num1 @/ $num2 .
        @}
    @].

$parasuppose <:- ~3e5 .

$suppose @( $num1 @> $parasuppose @)
    @{
        $Out @[ "el numero 1 es mayor que " $parasuppose @].
    @}.

$parasuppose <:- ~4.16e2 .

$suppose ! @( $num2 @< $parasuppose @)
    @{
        $Out @[ "el numero 2 es menor que " $parasuppose @].
    @}.

$Finishing
```

indows store) ⊗ 0 ⚠ 0 Juan Live Share

programacongramatica.txt

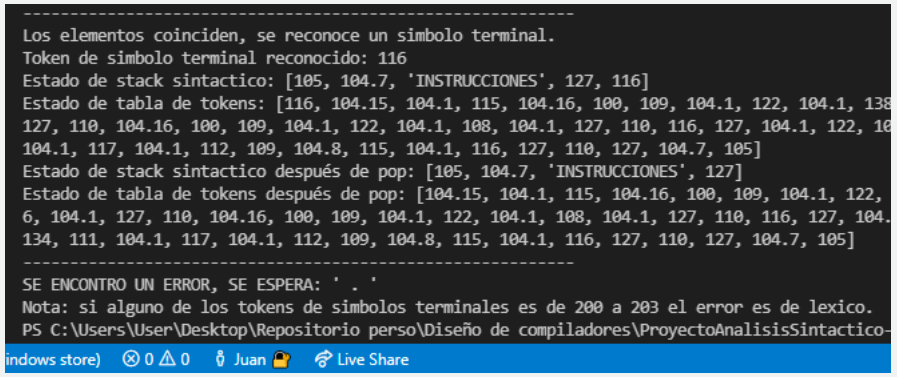
```
$main:

$enter
    @[ @( $integer $nummaterias @) @]
    @[ @( $integer $calificacion @) @]
    @[ @( $integer $califacum @) @]
    @[ @( $integer $promedio @) @]
    @[ @( $string $nombrealumno @) @]

$Get @[ "Ingrese el numero de materias " $nummaterias @].
$Get @[ "Ingrese el nombre del alumno " $nombrealumno @].
$califacum <:- ~0 .

$for @(  $i <:- ~1 ; $i %=% $nummaterias ; $i <+ @)
    @{
        $Get @[ "Ingrese la calificacion del alumno en esta materia " $calificacion @].
        $califacum <:- $califacum + $calificacion .
    @}.

$promedio <:- $califacum @/ $nummaterias .

$Out @[ "El promedio del alumno es " $promedio @].

$Finishing
```

holamundogramatica.txt

```
$main:

$enter
    @[ @( $string $nombre @) @]

$Get @[ "Ingresa tu nombre " $nombre @].

$Out @[ "Hola " $nombre @].

$Finishing
```

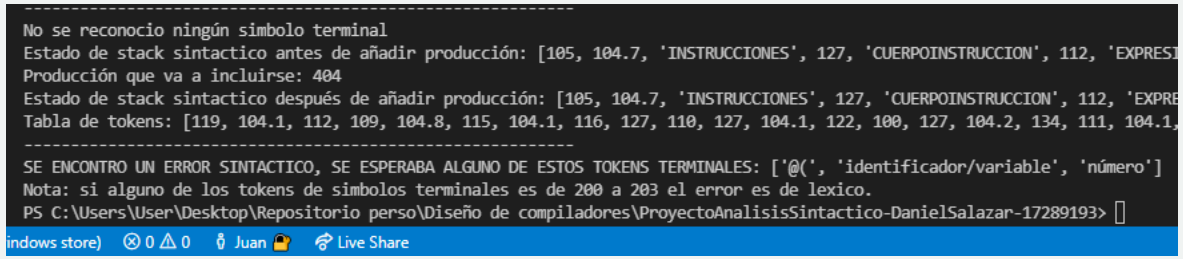
Errores:

- Cuando se encuentran un error en el código fuente en el cual hay un símbolo terminal especifico el cual debe continuar la impresión en pantalla se verá así:



En este ejemplo se le quito un punto a uno de los archivos de código fuente funcionales y el programa nos indica que nos falta un punto.

- Cuando se encuentra un error en el código fuente que puede ser sintáctico o faltan símbolos terminales y hay varios que pueden ser usados la impresión en pantalla se verá así:



Opinión de la clase:

- La clase fue muy concisa y exacta en lo que debíamos aprender a pesar del poco tiempo que teníamos disponible, eso fue muy bueno, por otro lado somos un grupo que no pudo aprender mucho en la materia anterior a esta y aun así el profesor se encargo de explicarnos los autómatas para poder diseñar un léxico funcional, en conclusión la experiencia con la clase fue buena, no hubo algún tema que no se ocupara ni actividades de relleno, haciendo la materia disfrutable y eficiente.