



INSTITUTO POLITÉCNICO NACIONAL.

ESCUELA SUPERIOR DE CÓMPUTO.

TEORÍA COMPUTACIONAL

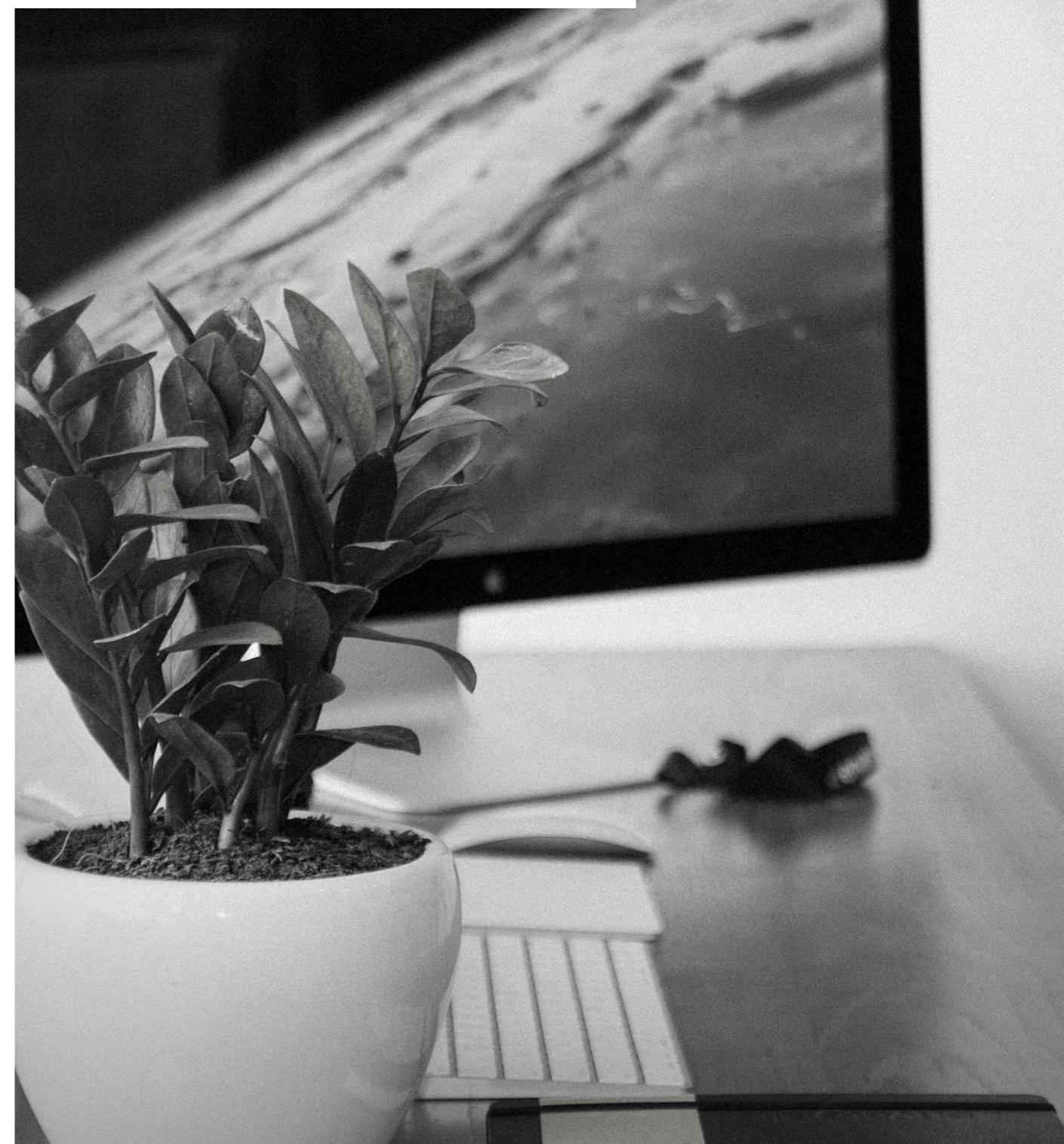
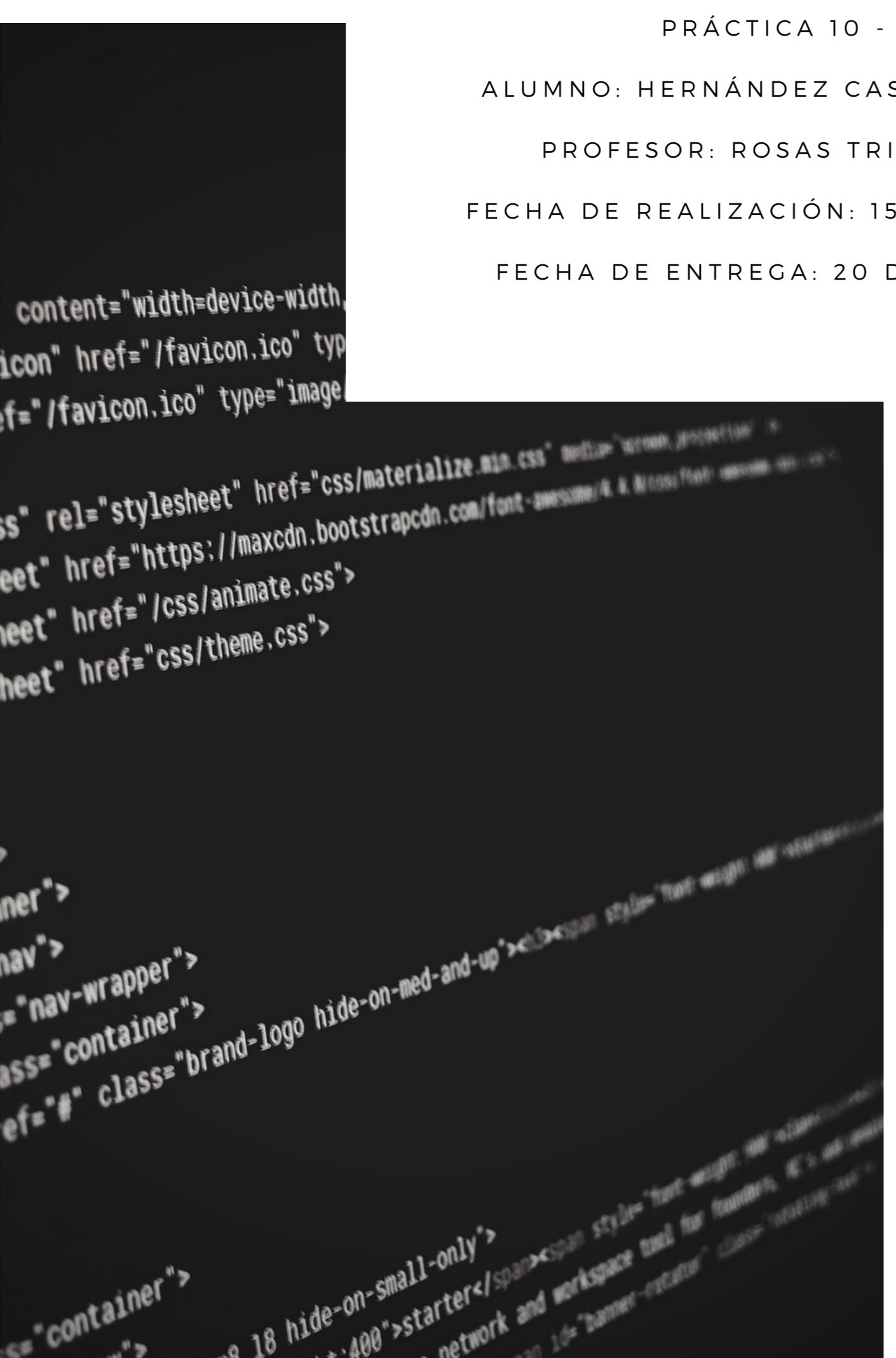
PRÁCTICA 10 - GRAMÁTICAS.

ALUMNO: HERNÁNDEZ CASTELLANOS CÉSAR URIEL.

PROFESOR: ROSAS TRIGUEROS JORGE LUIS.

FECHA DE REALIZACIÓN: 15 DE NOVIEMBRE DEL 2017

FECHA DE ENTREGA: 20 DE NOVIEMBRE DEL 2017



Marco teorico

Una gramática formal es una estructura matemática con un conjunto de reglas de formación que definen las cadenas de caracteres admisibles en un determinado lenguaje formal o lengua natural. Las gramáticas formales aparecen en varios contextos diferentes: la lógica matemática, las ciencias de la computación y la lingüística teórica, frecuentemente con métodos e intereses divergentes.

En un lenguaje formal, a las cadenas formadas según las reglas de la gramática formal se las llama fórmulas bien formadas, y el conjunto de todas las fórmulas bien formadas constituye un lenguaje formal. Una gramática formal no describe el significado de las fórmulas bien formadas, sino solamente su forma. La teoría de los lenguajes formales estudia las gramáticas formales y los lenguajes formales, y es una rama de la matemática aplicada. Sus aplicaciones se encuentran en la ciencia computacional teórica, la lingüística, la semántica formal, la lógica matemática y otras áreas.

Una gramática formal es un conjunto de reglas para reescribir cadenas de caracteres, junto con un símbolo inicial desde el cual debe comenzar la reescritura. Por lo tanto, una gramática formal generalmente se piensa como una generadora de lenguajes. Sin embargo, a veces también puede ser usada como la base para un "reconocedor": una función que determina si una cadena cualquiera pertenece a un lenguaje o es gramaticalmente incorrecta.

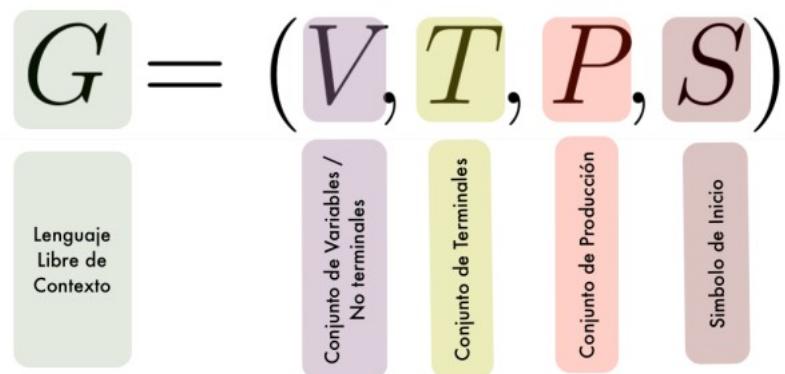


Imagen 1.0 - Gramática

Una gramática es:

$$G = \{N, T, S, P\}$$

donde:

N = Conjunto de no terminales

T = Conjunto de terminales

S = Símbolo inicial de la gramática

P = Conjunto de reglas de producción

Y de aquí podemos partir a diferentes tipos de gramáticas:

1.1. Gramáticas lineales

Sólo tienen un no terminal en el lado derecho de una producción.

1.1.1. Gramática lineal por la derecha

Todas las producciones son de la forma:

$$A \rightarrow xB \text{ ó } A \rightarrow x$$

1.1.2. Gramática lineal por la izquierda:

Todas las producciones son de la forma:

$$A \rightarrow Bx \text{ ó } A \rightarrow x$$

1.1.3. Gramática regular:

Es cualquier gramática lineal por la izquierda o por la derecha.

Material y equipo.

El material utilizado en la práctica es el siguiente:

Herramientas de software:

* Oracle VM VirtualBox

* Ubuntu 17.04

* Python 3.6.2

* eric6 Web Browser (QtWebKit)

Herramientas de hardware:

- * Computadora personal

Desarrollo de la práctica.

La práctica número 11 de teoría computacional, consistió en hacer uso de la herramienta PLY, el cual nos facilitó bastante el trabajo al implementar las gramáticas planteadas, a continuación se presentan los resultados obtenidos:

La primera gramática, el profesor nos dió el código a modo ilustrativo.

$$\begin{aligned} S &\rightarrow bA \\ A &\rightarrow aaaA|b|\varepsilon \end{aligned}$$

```
tokens = ('a','b')
t_a = r'a'
t_b = r'b'
def t_error(t):
    print('caracter ilegal', t.value[0])
    t.lexer.skip(1)

import ply.lex as lex
lex.lex()

def p_S(p):
    "S : b A"
    pass

def p_A(p):
    "A : a a a A
       | b
       | empty"
    pass

def p_empty(p):
    'empty : '
    pass

S = ''
def p_error(p):
    global s
    if p:
        print('error de sintaxis en', p.value)
    else:
        print('error de sintaxis en EOF')
        print(s, 'no esta en el lenguaje')
import ply.yacc as yacc
yacc.yacc()
while(1):
    try:
        s = input('> ')
    except EOFError:
        break;
    if not s:
        continue
    t = yacc.parse(s)
```

Código empleado.

Salida del programa:

```
> ab
> aabb
> abb
> abbbbb

ERROR EN LA SINTAXIS

NO ESTA EN EL LENGUAJE

>
```

Ejercicio número dos.

$$L = \{a^n b^n \mid n, m \geq 0\}$$

$$S \rightarrow aSb|\varepsilon$$

```
tokens = ('a','b')
t_a = r'a'
t_b = r'b'
def t_error(t):
    print('caracter ilegal', t.value[0])
    t.lexer.skip(1)

import ply.lex as lex
lex.lex()

def p_S(p):
    "S : a S b
     | empty"
    pass

def p_empty(p):
    'empty : '
    pass

S = ' '

def p_error(p):
    global s
    if p:
        print('error de sintaxis en', p.value)
    else:
        print('error de sintaxis en EOF')
        print(s, 'no esta en el lenguaje')
import ply.yacc as yacc
yacc.yacc()
while(1):
    try:
        s = input('> ')
    except EOFError:
        break;
    if not s:
        continue
    t = yacc.parse(s)
```

Código empleado.

Salida del programa:

```
> ab
> aabb
> aaabbb
> aaab

ERROR

>
```

Ejercicio número tres.

$$L = \{a^n b^m \mid n \leq m \leq 2n\}$$

$$S \rightarrow aSb \mid aSbb \mid \varepsilon$$

Salida del programa:

```
> ab
> aabb
> abb
> abbbb
ERROR
abbbbb no está en el lenguaje.
```

```
tokens = ('a','b')
t_a = r'a'
t_b = r'b'
def t_error(t):
    print('caracter ilegal', t.value[0])
    t.lexer.skip(1)

import ply.lex as lex
lex.lex()

def p_S(p):
    """S : a S b b
        | a S b
        | empty """
    pass

def p_empty(p):
    'empty : '
    pass

S = ' '

def p_error(p):
    global s
    if p:
        print('error de sintaxis en', p.value)
    else:
        print('error de sintaxis en EOF')
    print(s, 'no esta en el lenguaje')
import ply.yacc as yacc
yacc.yacc()
while(1):
    try:
        s = input('> ')
    except EOFError:
        break;
    if not s:
        continue
    t = yacc.parse(s)
```

Código empleado.

Ejercicio número cuatro.

$$L = \{w \in \{a,b\}^* \mid a \text{ impar}, b \text{ impar}\}$$

$$\begin{aligned}S &\rightarrow aA|bB \\A &\rightarrow aS|bC \\B &\rightarrow aC|bS \\C &\rightarrow bA|aB|\varepsilon\end{aligned}$$

Salida del programa:

```
> ab

> aaabbb

> aaaaabbbbb

> abb

ERROR

abb no se encuentra en el lenguaje.
```

```
tokens = ('a','b')
t_a = r'a'
t_b = r'b'
def error(t):
    print('caracter ilegal', t.value[0])
    t.lexer.skip(1)

import ply.lex as lex
lex.lex()

def p_S(p):
    """S : a A
       | b B"""
    pass

def p_A(p):
    """A : b C
       | a S"""
    pass

def p_B(p):
    """B : a C
       | b S"""
    pass

def p_C(p):
    """C : b A
       | a B
       | empty"""
    pass

def p_empty(p):
    'empty : '
    pass

S = ''

def p_error(p):
    global S
    if p:
        print('error de sintaxis en', p.value)
    else:
        print('error de sintaxis en EOF')
```

```
- def p_C(p):
    "C : b A
     | a B
     | empty"
    pass

- def p_empty(p):
    'empty :
    pass

S = ' '

- def p_error(p):
    global S
    if p:
        print('error de sintaxis en', p.value)
    else:
        print('error de sintaxis en EOF')
        print(S, 'no esta en el lenguaje')
import ply.yacc as yacc
yacc.yacc()
while(1):
    try:
        S = input('> ')
    except EOFError:
        break;
    if not S:
        continue
    t = yacc.parse(S)
```

Código empleado.

Conclusiones y recomendaciones.

La herramienta de PLY, resultó ser de gran utilidad en la práctica, ya que nos facilitó de manera significativa la realización de la misma, en cuanto a las complicaciones de la práctica, únicamente se tuvo en la última gramática, ya que las pruebas de escritorio no correspondían con los resultados deseados de ahí en fuera todo salió bien.

Referencias

Es.wikipedia.org. (2017). *Gramática formal*. [online] Available at: https://es.wikipedia.org/wiki/Gram%C3%A1tica_formal [Accessed 20 Nov. 2017].

Scribd. (2017). *Teoría de la Computación: Gramática libre de contexto*. [online] Available at: <https://es.scribd.com/doc/97246521/Teoria-de-la-Computacion-Gramatica-libre-de-contexto> [Accessed 20 Nov. 2017].

Teoría de Lenguajes, Gramáticas y Autómatas Para Informáticos. (2000). [Place of publication not identified]: Digitalia, Inc.