“Manual Técnico”
"ML WEB"
“Analizador Lexico (html, css ,JavaScript)”

Autor: Widvin Josué Quiñónez Díaz
Universidad de San Carlos de Guatemala
Septiembre 2020

Introducción

Este manual describe el ambiente de desarrollo de una aplicación para el desarrollo de un juego con características similares a ML-WEB. De este modo, cualquier persona con bases de sistemas comunicacionales podrá configurar y dar mantenimiento al programa. Es importante mencionar que en este manual se indican las especificaciones mínimas de hardware y software para su correcta instalación.

Requerimiento Técnicos

Requerimientos mínimos de hardware:

- Procesador: PC con procesador de tipo Pentium II, 450 MHz
  Se recomienda: tipo Pentium III, 600 MHz.

- RAM: Windows XP Home: 96 MB; Windows XP Professional y Windows Server 2003: 192 MB.

  Se recomienda: 1 GB para el óptimo funcionamiento de la aplicación en cualquier sistema operativo.

- Disco Duro: 900 MB en la unidad del sistema y 4,1 GB en la unidad de instalación.

  Recomendación Python 3,4

Requerimientos mínimos de software:

- Sistema operativo: Windows 7, Windows Server 2003 o Windows NT 4.0 y todas sus versiones posteriores.

- Video: 800 x 600, 256 colores. Se recomienda: color de alta densidad, 16 bits.

Máquina de Compilación

- Python 3.4
- Windows 10
- Procesador Intel i7 de 6th generación.
- 8 Gb de Ram**.**

Ambiente de desarrollo de la aplicación

Python 3.4

Python es un lenguaje de programación interpretado cuya filosofía hace hincapié en la legibilidad de su código. Se trata de un lenguaje de programación multiparadigma, ya que soporta orientación a objetos, programación imperativa y, en menor medida, programación funcional. Es un lenguaje interpretado, dinámico y multiplataforma.

Es administrado por la Python Software Foundation. Posee una licencia de código abierto, denominada Python Software .

Dependencia :

Tkinter: Es un binding de la biblioteca gráfica Tcl/Tk para el lenguaje de programación Python. Se considera un estándar para la interfaz gráfica de usuario (GUI) para Python y es el que viene por defecto con la instalación para Microsoft Windows.

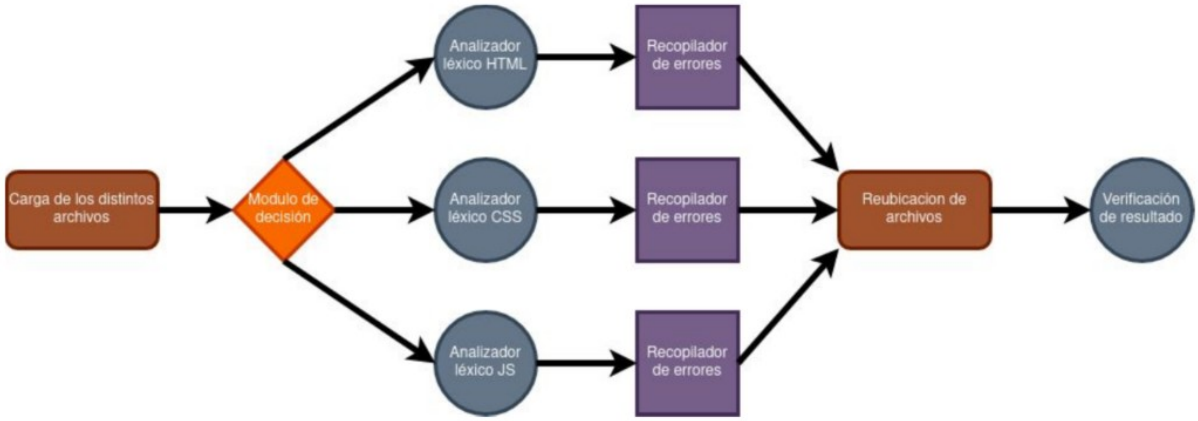
Graphviz: Es un conjunto de herramientas de software para el diseño de diagramas definido en el lenguaje descriptivo DOT

ML – WEB

ML – WEB es un analizador léxico orientado a los principales lenguajes para el diseño web : HTML , CSS , Javascript . Este analizador genera un analisis lexico de cada uno de los distintos archivos de entrada los cuales pueden ser de tipo Marcado como en el caso de HTML, de tipo diseño como CSS o de tipo interpretado como JavaScript.

ML – WEB carga es un amigable editor de texto donde el usuario carga sus archivos y el propio editor analiza el tipo de archivo y genera el un reporte si se encuentran errores según sea la definición lexica de cada lenguaje . El analizador tiene la capacidad obtener todos los errores lexicos del archivo abierto .

Flujo de ejecución dentro de la aplicación:



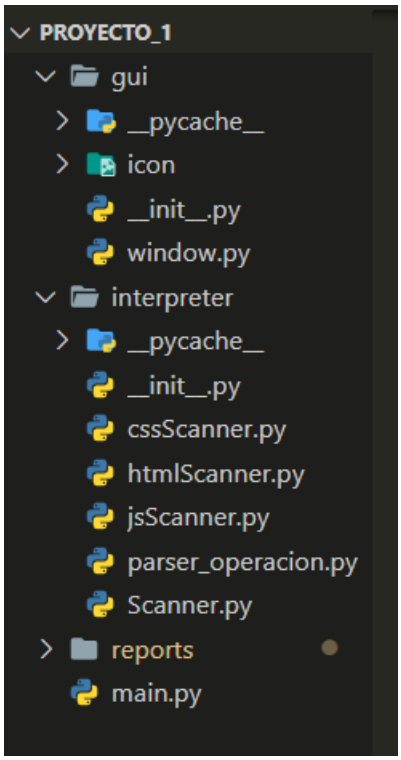
Archivos de Entrada :

El editor ML-WEB utiliza como archivos de entrada :

- **JavaScript :** Archivos con extensión *.js y estructura del mismo lenguaje.
- **Html:** Archivos con extensión *.html y estructura del mismo lenguaje
- **css :** Archivos con extensión *.css y estructura del mismo lenguaje
- **rmt :** Archivos con extensión *.rmt este tipo de archivo sera utilizado especialmente para el analisis de estructuras de expresiones aritmeticas con operadores del lenguaje Javascript ejemplo :

```
(a+65*964-6)
((aasd*9)+(8*9-6/4))
```

Estructura del Proyecto dentro de su Directorio:



Directorios :

/gui: Este directorio tiene todos los archivos relacionados con la interfaz grafica del editor .

- ***window.py*** : Es la clase de objeto que hereda los atributos del tk.Frame desde donde se inicializan los widgets de la interfaz y se toma la desición para saber que tipo de análisis realizar sobre el archivo de entrada.

```
class Application(tk.Frame):

    typeFileOpen = ""
    errorSintax = False
    scannerType = {
        "html":  lambda text: Html(text),
        "css":   lambda text: Css(text),
        "js" :   lambda text: JavaScript(text),
        "rmt":   lambda text: ParserOP(text)
    }


    def __init__(self, master=None):
        super().__init__(master)
        self.master = master
        self.pack()
        self.window_init()
        self.create_widgets()
```

La clase Application es quien hereda los atributos de tk.Frame y se puede ver como inicializa la ventana y crea los widgets definidos para todo el entorno del usuario .

Cabe destacar la utilizacion de un diccionario para la eleccion de que tipo de scanner se utilizara al realizara el analisis , donde se puede observar que cada llamado de llave según el tipo de la extension devuelve un objeto del tipo de scanner .

```
scannerType = {
    "html":  lambda text: Html(text),
    "css":   lambda text: Css(text),
    "js" :   lambda text: JavaScript(text),
    "rmt":   lambda text: ParserOP(text)
}
```

Una de los metodos de los cuales se puede hablar es :

```
def Analilizar(self):
    self.txtConsole.delete("1.0", tk.END)
    input_text = self.txtInputArea.get("1.0", tk.END)
    print("Se inicio el proceso de analisis")
    scan = self.scannerType[self.typeFileOpen](input_text) # Busca si existe el tipo de archivo dentro del diccionario y retorna una insta
    if self.typeFileOpen in self.scannerType:
        self.txtConsole.insert("1.0", "---------- EXISTEN ERROES LEXICOS----------") if scan.FindError()  else self.txtConsole.insert("1
        if self.typeFileOpen == "css":
            for a in scan.transicionCSS:
                self.txtConsole.insert("1.0",str(a)+"\n")
    else:
        print("Este tipo de archivo no se puede analizar")
```

El metodo Analizar() es el cual el boton con el mismo nombre llama siempre que se quiera iniciar el analisis , entonces lo que hace aquí es luego de abrir el archivo en el editor pasa como parametro al diccionario y obtiene un objeto del tipo del scanner para luego verificar a travez del metodo FindError si se existio un error reportarlo en consola para luego verificar el archivo de errores.

/interpreter : Es el directorio donde se encuentran todos los archivos del scanner respectivo para cada lenguaje .

Scanner.py : Esté archivo es el principal donde se encuentra el automata para resolver el analisis pero tomando como consideracion ciertos aspectos , en este caso se utilizo un solo scanner que recibe ciertos parametros que definen que tipo de lenguaje esta analizando y excluyendo algunos estados que no existen en alguno de los lenguajes por ejemplo los comentario unilinea en css que solo son aceptados en el lenguaje de JavaScript.

```
class Scanner(object):

    estado = 0
    error = False
    index = 0
    line = 0
    column = 0
    lexema = ""
    buscador= ' '
    cleanBuffer = ""
    comentarioDL = False
    contadorEstados = 0
    cadenaTransicion = ""



    def __init__(self, palabrasReservadas, simbolos , tipolenguaje, entrada):
        self.palabrasReservadas = palabrasReservadas
        self.simbolos= simbolos
        self.tipolenguaje = tipolenguaje
        self.entrada = entrada
        self.estado = 0
        self.fila = 1
        self.column = 1
        self.listErrors = list()
        self.transicionCSS = []
```

Se puede ver a simple vista que la clase Scanner hereda de Object para poder ser un objeto y se pueden ver sus principales atributos que se utilizan dentro de todo el analisis . Y en el metodo __init__ () la clase recibe como parametro:

- **palabrasReservadas:** una lista de palabras reservadas si se llegaran a validar para un posterior analisis sintactico.
- **simbolos:** una lista de simbolos que el lenguaje acepta como parte de su lexico .
- **tipoLenguaje:** la extension del tipo de lenguaje que se va a analizar para hacer las validaciones posteriores.
- **Cadena:** El texto del archivo a analizar para realizar el recorrido atravez del automata .

- **JsScanner.py** : Clase que hereda de scanner.py para implementar el automata, tambien tiene una lista de palabras reservadas y simbolos que el lenguaje acepta .

```python
from interpreter.Scanner import Scanner

class JavaScript(Scanner):
    palabrasReservadas = {'var', 'if', 'else','for', 'while' , 'do',
                          'continue' , 'break' , 'return' , 'function' ,
                          'constructor' , 'class' , 'math' , 'pow'

                         }
    simbolos = {'<', ">" , "/" , "=", "\"", "*","+","-", "'", ";","." , "(", ")" , "{", "}", ",", "!" ,"&" , "|"}
    tipolenguaje = "js"

    def __init__(self, entrada ):
        super().__init__(self.palabrasReservadas, self.simbolos, self.tipolenguaje, entrada)
```

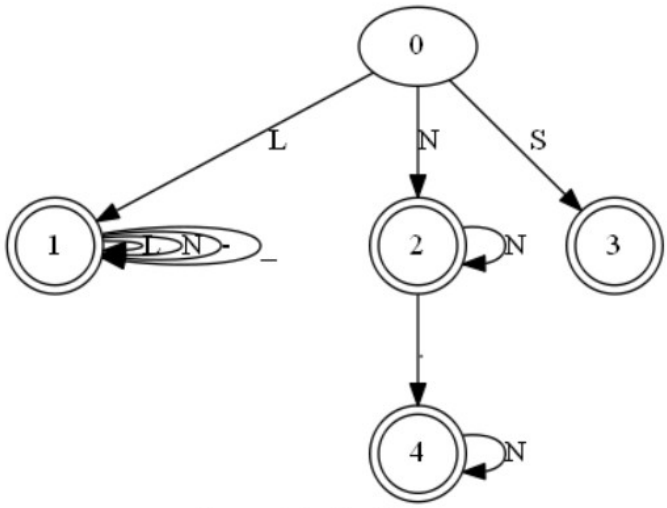
- **HtmlScanner.py :** Clase que hereda de scanner.py para implementar el automata, tambien tiene una lista de palabras reservadas y simbolos que el lenguaje acepta .

```python
from interpreter.Scanner import Scanner

class Html(Scanner):

    palabrasReservadas = {'html', 'head', 'title', 'body', 'h1', 'h2', 'h3' , 'h4', 'h5',
                          'h6'  , 'p'   , 'br'   , 'img' , 'src', "a" , 'href' , 'ol' ,
                          'ul'  , 'li'  , 'head' ,  'style' , 'table' , 'th' , 'tr'  ,
                          'td'  , 'caption', 'colgroup' , 'col' , 'thead' , 'tbody' , 'tfoot',
                          'div'

                         }
    simbolos = {'<', ">" , "/" , "=", "\"", "'"}
    tipolenguaje = "html"

    def __init__(self, entrada ):
        super().__init__(self.palabrasReservadas, self.simbolos, self.tipolenguaje, entrada)
```

- **CssScanner.py :** Clase que hereda de scanner.py para implementar el automata, tambien tiene una lista de palabras reservadas y simbolos que el lenguaje acepta .

```python
from interpreter.Scanner import Scanner

class Css(Scanner):

    palabrasReservadas = {'html', 'head', 'title', 'body', 'h1', 'h2', 'h3' , 'h4', 'h5',
                          'h6'  , 'p'   , 'br'   , 'img' , 'src', "a" , 'href' , 'ol' ,
                          'ul'  , 'li'  , 'head' ,  'style' , 'table' , 'th' , 'tr'  ,
                          'td'  , 'caption', 'colgroup' , 'col' , 'thead' , 'tbody' , 'tfoot',
                          'body', 'color'  , 'background-color', 'background-image' , 'border',
                          'opacity' , 'background' , 'text-align' , 'font-family' , 'font-style',
                          'font-weight' , 'font-size', 'font' ,      'padding-left' , 'padding-right',
                          'padding-bottom' , 'padding-top' , 'padding' , 'display' , 'line-height',
                          'width' , 'height', 'margin-top' , 'margin-right' , 'margin-bottom', 'margin-left',
                          'margin' , 'border-style' , 'display' , 'position' , 'bottom' , 'top' , 'right',
                          'left' , 'float' , 'clear' , 'max-width', 'min-width' , 'max-height' , 'min-height',
                          'px', 'em' , 'vh' , 'vw' , 'in' , 'cm' , 'mm' , 'pt' , 'pc'
                         }

    simbolos = {'{', "}" , "/" , ":", "\"", "*", "#", ",", ";", ".", "-", "%", "(",")", "'"}
    tipolenguaje = "css"

    def __init__(self, entrada ):
        super().__init__(self.palabrasReservadas, self.simbolos, self.tipolenguaje, entrada)
```

parser_operacion.py: Esta clase es especial porque se utiliza para realizar el analisis sintactico para el archivo de tipo rmt entonces esta clase construye su propio automata hereda de scanner pero sobrecarga algunos metodos ya que el automata que utiliza es mas pequeño que el de la clase Scanner



Recorrido de algunos estados del automata.

Esta clase de la misma forma implementa su propio metodo de parser para analizar las expresiones sintacticamente utilizando la siguiente gramatica :

```
add → mul
     |add2
mul → term
     |mul2
add2 →  ‘+’ mul add2
     | ‘-’  mul add2
     | epsilo
mul2 → ‘*’    term mul2
     | ‘/’  term mul2
     | epsilon
 term → id
     | num
     | ‘(’ add ‘)’
     | Math ‘.’ pow ‘(‘ term ‘,’ term ‘)’
```

/reports: Esté directorio almacena los reportes que se crean en ejecución .

Clase Principal :

main.py : *Donde se inicial el flujo del programa generando una instancia de window como un Frame:*

```python
import gui.window as Frame

def main():
    Frame.Window_gui()


if __name__ == "__main__":
    main()
```

INTERFAZ GRAFICA DE USUARIO FINAL :