

# Automatador

Manual de usuario

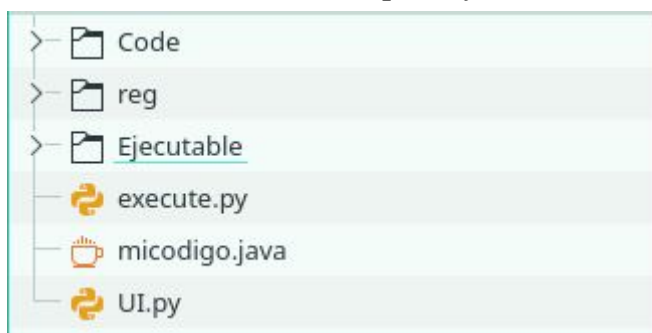
El siguiente documento es el manual del **Automatador** compilador lexicográfico escrito por ~~Juan Cardona~~ y Roy Maestre.

El lenguaje de programación que utilizamos para crearlo fue **python 3.8.5** utilizamos el **IDE Pycharm 2020.2.2** y utilizamos las **librerías tkinter** y **datetime** para realizar algunas operaciones específicas.

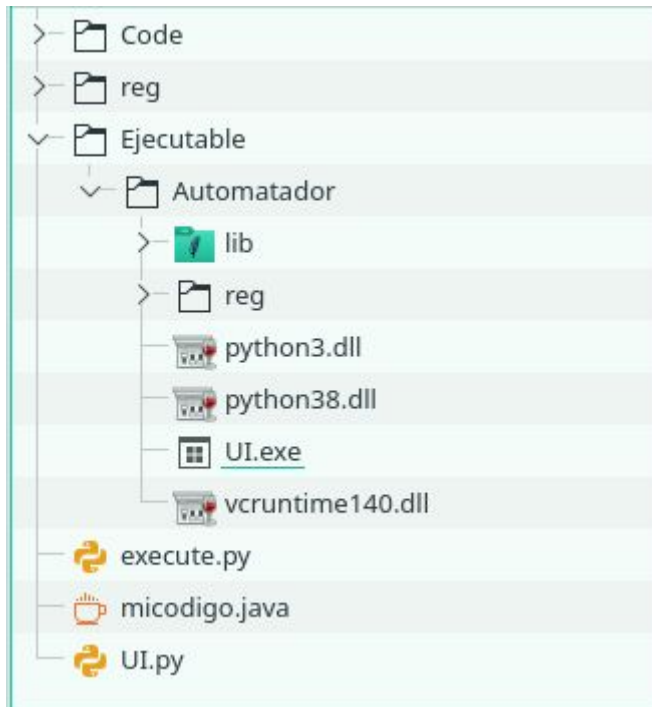
No es necesario tener ninguno de esos programas para correr el auto matador pero se anexaron pasos opcionales a la guía en los cuales se requiere tener instalado **python** de manera global para ejecutar desde consola algunos archivos.

Para obtener acceso al **Automatador** ingrese a la siguiente dirección de GitHub: <https://github.com/Royk8/TdLAutomataInfinito>, en ella encontrará el código fuente, el ejecutable y los documentos necesarios.

1. Descargue el archivo zip que proporciona la página.
2. Extraiga el contenido e ingrese a la carpeta **/TdLAutomataInfinito**
3. En ella encontrarán tres carpetas y tres archivos los cuales corresponden a:

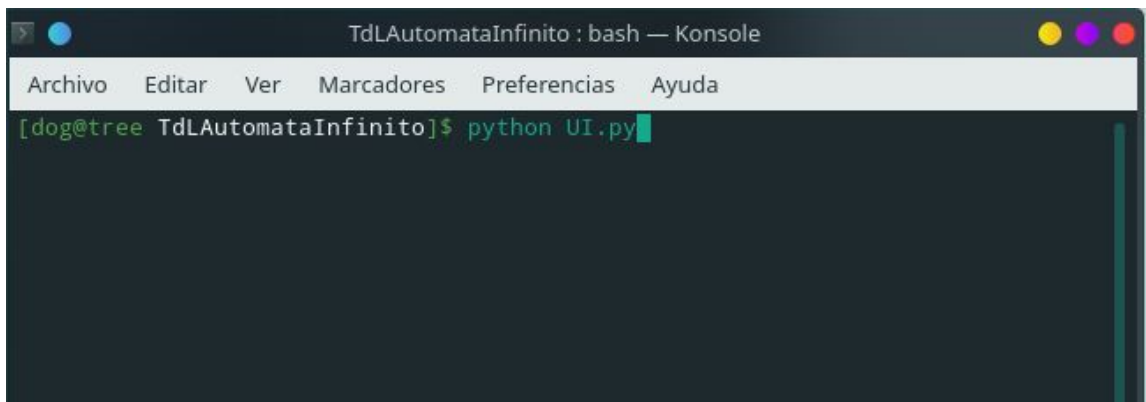


La carpeta **Code** está toda la lógica o BackEnd, el archivo **UI.py** contiene la vista o FrontEnd, el archivo **micodigo.java** sera el analizado lexicograficamente, el archivo **execute.py** se encarga de crear un archivo ejecutable \*.exe, la carpeta **reg** contiene los registros de ejecuciones y la carpeta **Ejecutable** contiene los archivos y el ejecutable del **Automatador**.

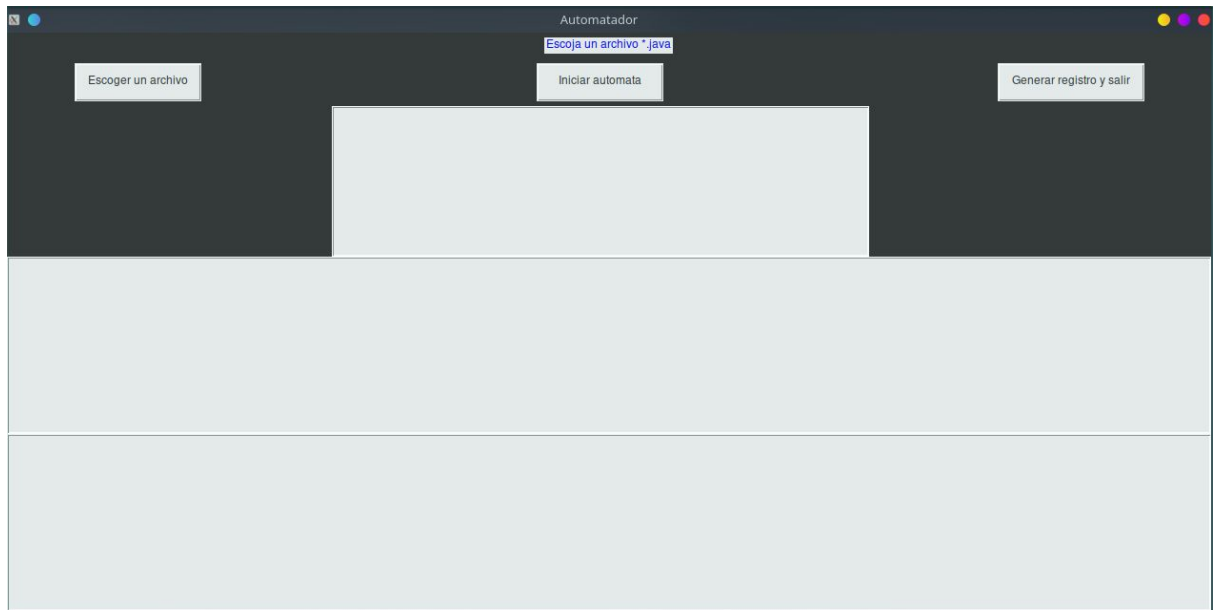


4. Para ejecutarlo debe dirigirse al archivo que se encuentra en `/TdeLAutomataInfinito/Ejecutable/Automatador` y hacer doble click en `UI.exe` o opcionalmente abrir una terminal y situarse en la carpeta `/TdeLAutomataInfinito` y ejecutar el comando(requiere python 3.8.5):

```
python UI.py
```

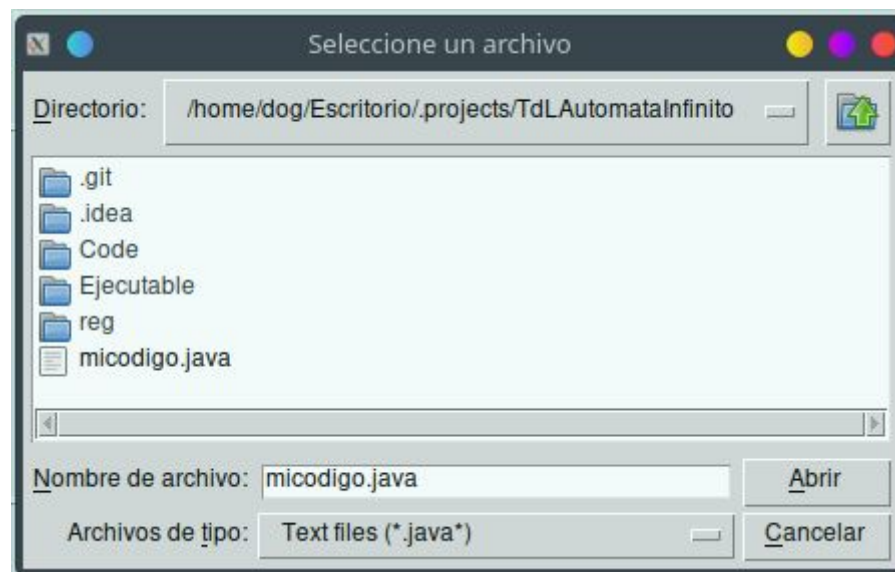


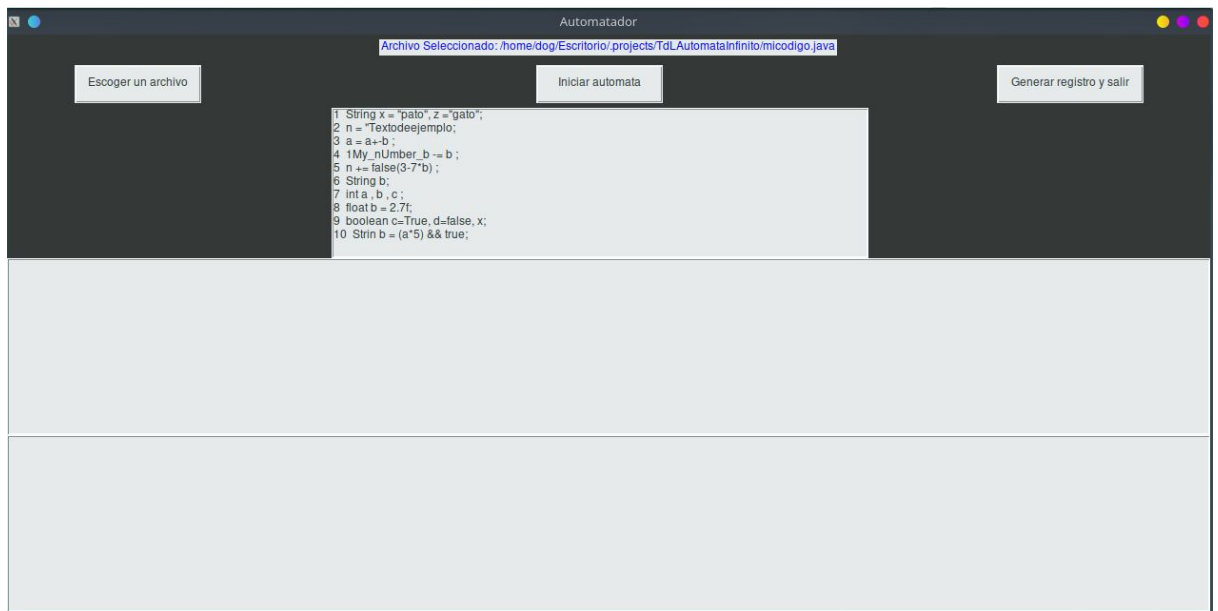
5. Una vez habiendo ejecutado el archivo `UI.exe` o el comando anterior, se desplegará la ventana del **Automatador**.



Este tiene un pequeño **campo de texto** que nos indica que debemos elegir un archivo \*.java, 3 tres botones **Escoger un archivo**, **Iniciar autómata** y **Generar registro y salir**. El **Automatizador** permite seleccionar un archivo a la vez y un número de archivos consecutivos ilimitado para analizar.

### 1. Escoger un archivo:





Aquí vemos que se cargó el archivo **micodigo.java**

## 2. Iniciar autómata:



Aquí se nos despliega la lista de nodos correspondientes a la *[clase | valor]* de cada sentencia en las líneas del archivo **micodigo.java** y los errores que encontró en el código.

Podemos escoger más archivos para seguir ejecutando el autómata.

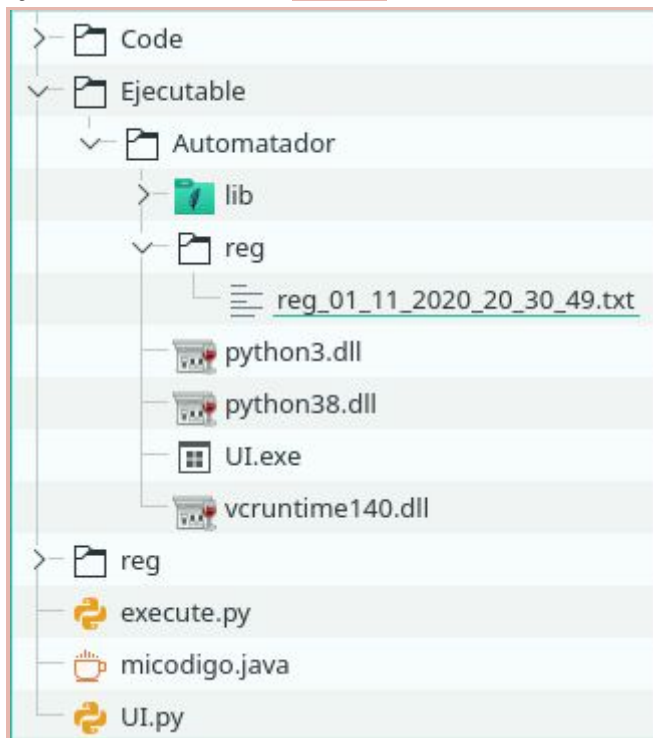
## 3. Generar registro y salir:



Aquí se nos abre un mensaje de texto que indica que un archivo *reg\_dd\_mm\_aa\_hh\_mm\_ss.txt* ha sido guardado en la carpeta `/TdeLAutomataInfinito/reg` si ejecutamos el **Automatador** desde consola



O en la carpeta `/TdeLAutomataInfinito/Ejecutable/Automatador/reg` si lo ejecutamos desde el **UI.exe**



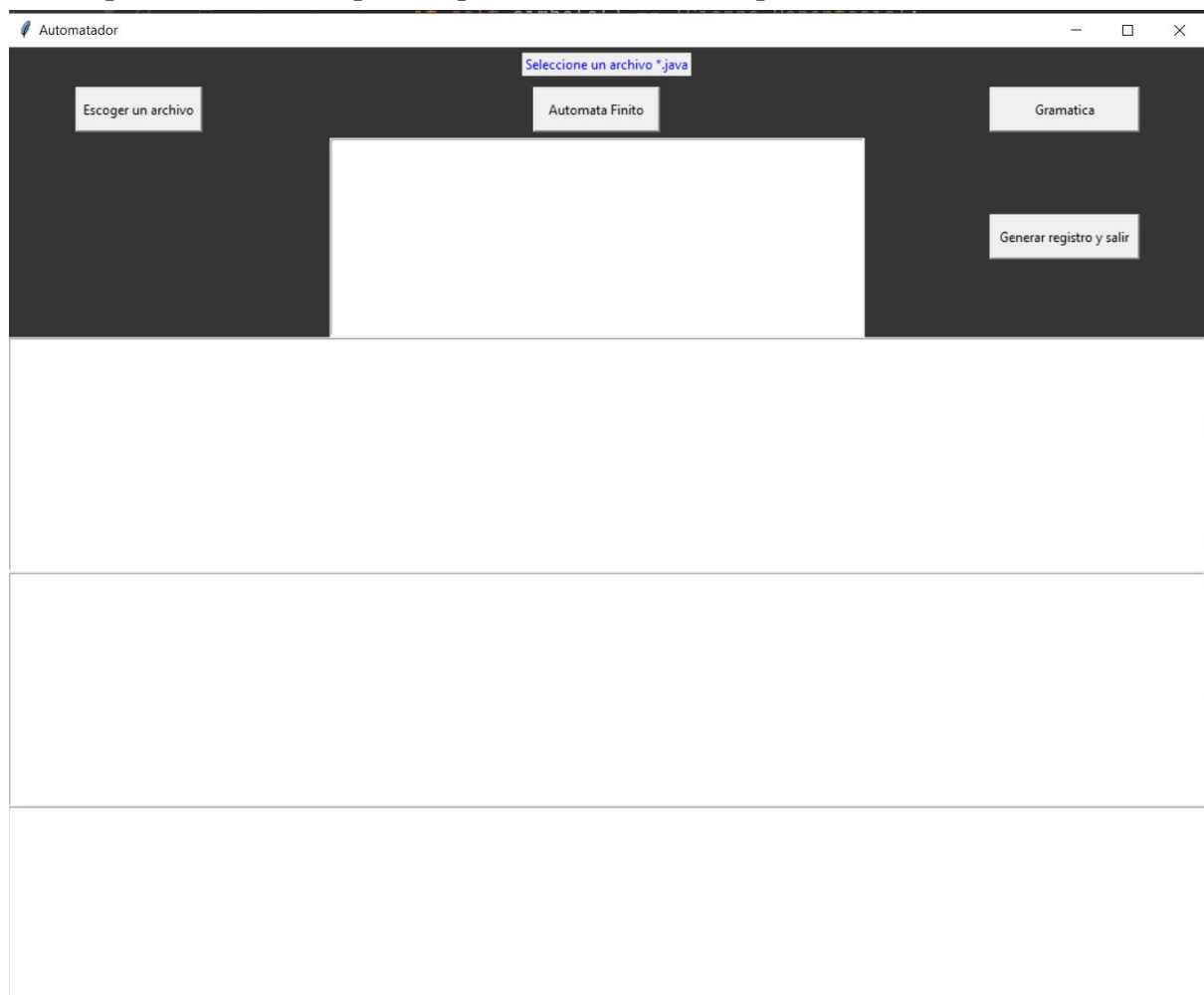
Al abrir el archivo de texto vemos lo siguiente:

```
reg_01_11_2020_21_17_43.txt X
reg > reg_01_11_2020_21_17_43.txt
1 1 [Tipo | String ] [Variable | x ] [Asignacion | = ] [Comillas | " ] [Variable | pato ] [Comillas | " ] [Separador | , ] [Variable | z ] [Asignacion
2 [Variable | n ] [Asignacion | = ] [Comillas | " ] [Variable | Textodeejemplo ] [Fin de linea | ; ]
3 [Variable | a ] [Asignacion | = ] [Variable | a ] [Operador Aritmetico | + ] [Operador Aritmetico | - ] [Variable | b ] [Fin de linea | ; ]
4 [Variable Invalida | 1My_nUmer_b ] [Modificador | -= ] [Variable | b ] [Fin de linea | ; ]
5 [Variable | n ] [Modificador | += ] [Parentesis | ( ] [Variable | false3 ] [int | 3 ] [Operador Aritmetico | - ] [int | 7 ] [Operador Aritmetico
6 [Tipo | String ] [Variable | b ] [Fin de linea | ; ]
7 [Tipo | int ] [Variable | a ] [Separador | , ] [Variable | b ] [Separador | , ] [Variable | c ] [Fin de linea | ; ]
8 [Tipo | float ] [Variable | b ] [Asignacion | = ] [float | 2.7f ] [Fin de linea | ; ]
9 [Tipo | boolean ] [Variable | c ] [Asignacion | = ] [Variable | True ] [Separador | , ] [Variable | d ] [Asignacion | = ] [Variable | false ] [Sep
10 10 [Variable | Strin ] [Variable | b ] [Asignacion | = ] [Parentesis | ( ] [Variable | a ] [Operador Aritmetico | * ] [int | 5 ] [Parentesis | ) ]
11
12
13 1 n = " Textodeejemplo ;
14                                     ^ ERROR: Quotation never closed
15 3 a = a + -
16                                     ^ ERROR: Expresion expected
17 5 1My_nUmer_b
18                                     ^ ERROR: Invalid variable
19 7 n += ( false3 3
20                                     ^ ERROR: Semicolon expected
21 9 Strin b
22 10                                     ^ ERROR: Not a statement
23 11
```

La lista de nodos por línea y los errores que fueron encontrados.

## PRACTICA 2

Para esta segunda práctica el funcionamiento es similar, solo se agregaron un botón y un campo de texto extra, pero la práctica 1 aun es completamente funcional.



Para ejecutar el reconocimiento recursivo de la segunda práctica es necesario haber escogido un archivo y dar clic en el botón “Gramática”. Al hacerlo, en el campo de texto inferior aparecerá el código con sus errores en caso de tenerlos.

---

EL CODIGO CONTIENE ERRORES

```
x = 25 ;  
17  
-ERROR: Se esperaba un Tipo, Variable o Estructura condicional, pero se recibió [int]-  
z = ( 37 * 10 ;  
-ERROR: Parentesis Desbalanceados-  
if ( hola ) { y = 33 ;  
} else { x = 15 ;  
}  
-ERROR: Se esperaba un punto y coma, pero se recibió [Cierra Llave]-  
String texto = " Habia una vez " ;  
String otroTexto = " Tambien  
-ERROR: Nunca se cierran las comillas
```

Además, de igual manera se puede generar un registro al dar clic en el botón “Generar Registro”