



# DISEÑO DEL COMPILADOR

Andrea Reyes 20190265  
Katherine García 20190418

FASE 1

SCANNER

# DIRECTORIO **COMPILER** FASE 1 <sup>x</sup>

**01**

## **FILES**

inputFile.py  
outputFile.py

**02**

## **ABSTRACTS**

Symbols.py  
Tokens.py

**03**

## **SCANNER**

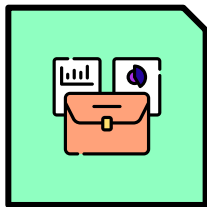
Accepting.py  
DFAs.py  
TypeSList.py  
Scanner.py

**04**

## **OUTPUTS**

.txt's con outputs

# COMPILER



## Compiler

Directorio con el árbol de todas nuestras clases para el funcionamiento correcto del compilador.



## Compiler.py

Es nuestro CLI, en donde se instancian todas las clases y se usan los métodos según las flags.

# CORRER CLI



☆☆☆☆ Python3 Compiler.py archivo.decaf - target scanner -o output

## **-o outputFileName**



Escribe el output a un archivo .txt con el nombre indicado.

## **-target scanner**



Corre la etapa especificada.

## **-debug scanner**



Imprime información de debugging.

## **-opt optStage**



Hace la optimización especificada.

# FILES



## inputFile.py

En esta clase leemos el archivo.decaf. Identificamos cada elemento del código ingresado, a qué línea pertenece cada elemento y filtramos comentarios y líneas en blanco.



## outputFile.py

Esta clase es la encargada de generar un archivo .txt con los tokens generados.

# ABSTRACTS



Es una clase con los atributos necesarios para cada objeto symbol (identificador, regex, nombre).

—**Symbols.py**

Clase con los atributos necesarios para cada objeto token (tipoSimbolo, id, value)

—**Tokens.py**



### Accepting.py

Verifica si cada palabra es aceptada por el DFA.

### DFAs.py

Es una clase con la representación de los DFA'S.

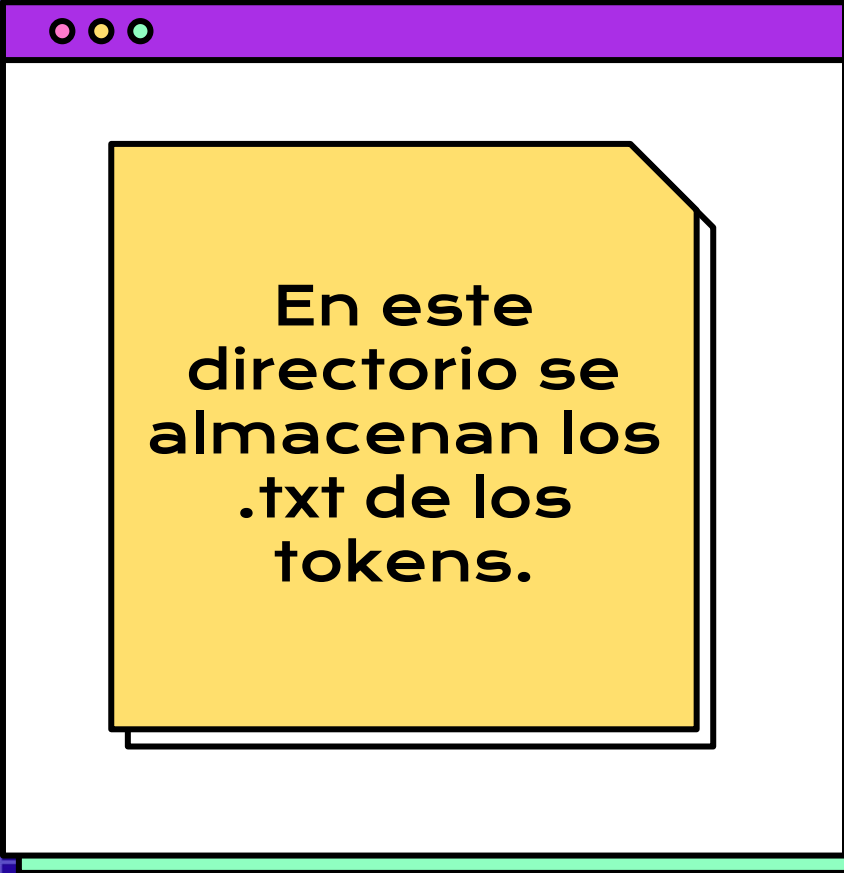
### TypeSList.py

Es una clase con la representación de cada símbolo aceptado.

### Scanner.py

Esta clase es la encargada de evaluar si se genera un token o un error.

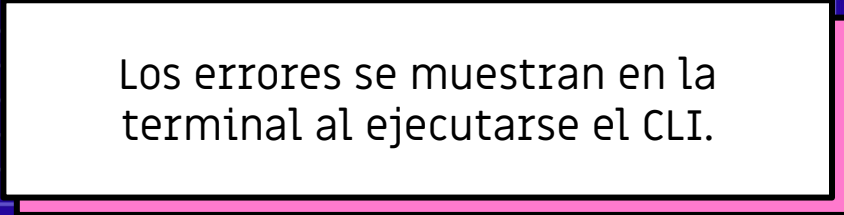
# SCANNER



En este  
directorio se  
almacenan los  
.txt de los  
tokens.



**outputs**



Los errores se muestran en la  
terminal al ejecutarse el CLI.