



## Práctica 2: Gramáticas y generadores automáticos

591000 - Compiladores

---

Prof. Marçal Mora Cantallops, Universidad de Alcalá

14/10/2022

### Enunciado

Esta práctica consta de cuatro ejercicios en dos partes separadas.

#### 1. Primera parte: generación de árboles sintácticos para lenguajes específicos

Para esta primera parte se os proporcionarán características y un fichero de ejemplo y, partiendo de los mismos se deberá:

##### Nivel básico (1 puntos):

- Ser capaz de detectar e identificar automáticamente cada uno de los elementos del lenguaje (es decir, generar el lexer y el parser correspondientes al lenguaje).
- Construir el AST correspondiente.

##### Nivel medio (1 punto):

- Plantear alguna mejora que mejore o amplie la gramática.
- Mostrar el fichero del árbol AST en un formato legible de texto plano.

## 1.1. Árbol sintáctico de un CSV

Un fichero separado por comas es un formato muy habitual para conjuntos de datos en columnas. Algunos ejemplos podrían ser los siguientes:

Nombre; Frase; Tipo; Lanzamiento

Aatrox; the Darkin Blade; Juggernaut; 2013-06-13

Ahri; the Nine-Tailed Fox; Burst; 2011-12-14

Akali; the Rogue Assassin; Assassin; 2010-05-11

Type, Fast Moves, DPS, Power-PvP, Energy-PvP, Cast Time, Turns, DPT, EPT

Ste, Steel Wing, "13,8", 11-7, 6-5, "0,8s", 2, "3,5", "2,5"

Dra, Dragon Tail, "13,6", 15-9, 9-10, "1,1s", 3, "3,0", "3,3"

Generad un analizador léxico y sintáctico para poder construir el AST correspondiente a un CSV cualquiera, suponiendo que los separadores pueden ser la coma (,), el punto y coma (;) o la barra (|).

## 1.2. Un lenguaje para las formas

Vamos a crear un lenguaje simple para definir imágenes que estén compuestas de formas. El lenguaje *formitas* va a tener las siguientes características:

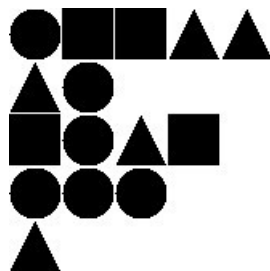
- El código fuente estará en una sola línea.
- El primero de los elementos será el tamaño del marco (o de la imagen de salida). Lo llamaremos *imgdim* y será, por ejemplo, *imgdim:256*, que significará que la imagen va a tener un tamaño de 256x256 píxeles.
- El tamaño de las formas individuales se especifica a nivel global y será con *shpdim* (e.g. *shpdim:64* para imágenes de 64x64).
- La descripción de las imágenes empieza con un delimitador (*>>>*).
- Las filas se separan con una pipe (*|*). No puede haber un símbolo *|* ni justo después del inicio (*>>>*) ni justo antes del final (*<<<*).
- Las formas individuales se separan por comas (*,*). Una fila no puede empezar o terminar con una coma.
- Cada fila está formada por una lista de formas (por ejemplo: (triangulo, cuadrado, circulo)).
- 

El cierre se hace con tres símbolos de menor (*<<<*).

```
imgdim:180,shpdim:32
```

```
>>>circulo,cuadrado,cuadrado,triangulo,triangulo|triangulo,circulo|  
cuadrado,circulo,triangulo,cuadrado|circulo,circulo,circulo|triangulo<<<
```

No es el objetivo en esta PL, pero la salida final de una línea como esta sería algo como esto, para que os hagáis una idea de lo que representa:



### 1.3. Lenguaje de programación mínimo

Imaginemos que tenemos un lenguaje muy simple de programación que es, además, más natural en su lenguaje. Lo llamaremos E++. Un ejemplo de programa podría ser el siguiente:

---

```
1  # Inicializacion de variable
2  asignar a = 20 ;P
3
4  # if-else sencillo
5  a > 2 ???
6  si ->
7  a = 5 - 2 ;P
8  terminar
9
10 # imprimir por pantalla 11
mostrar a ;P
```

---

Generad la gramática capaz de generar el AST para este lenguaje, que tiene:

- Asignación inicial para inicializar variables (asignar).
- Estructuras tipo if con (???) para la condición, (si ->, no->) para el condicional y (terminar) para cerrar el bloque.
- Las comparaciones pueden ser las habituales en los lenguajes de programación.
- Usa (;P) como fin de línea.
- Imprime por pantalla con (mostrar).
- Admite enteros, floats y cadenas de caracteres - strings.
- Los identificadores pueden tener la forma habitual en los lenguajes de programación.
- Solo puede sumar y restar como operadores aritméticos por ahora.

Los comentarios van con almohadilla y siempre en una línea para ellos solos (es decir, no hay comentarios en la misma línea que hay una instrucción).

## 2. Segunda parte: las recetas de la abuela

### Nivel básico (3 puntos):

- Definir un lenguaje en el que poder expresar recetas de cocina. Debe, por lo tanto, cubrir las instrucciones habituales, los verbos de cocina, los utensilios y las indicaciones de cantidades o productos. Debéis también limitar las elecciones posibles a un conjunto razonable, obviamente. En general, debe ser lo más informativo posible de la forma más sistemática que se os ocurra. Se valorará tanto la sencillez como el potencial (es decir, que de la forma más sistemática y condensada posible cubra gran cantidad de posibilidades).
- Ser capaz de detectar e identificar automáticamente cada uno de los elementos del lenguaje (es decir, generar el lexer y el parser correspondientes al lenguaje que habéis diseñado). Es, por lo tanto, necesario que generéis un juego de pruebas exhaustivo - de recetas de ejemplo en este caso.
- Construir el AST correspondiente. Fijaros que una vez diseñado lo anterior, el ejercicio se reduce a hacer lo mismo que en los apartados de la primera parte (la diferencia es que habéis diseñado vosotros el lenguaje y no viene dado).

### Nivel avanzado (1 puntos):

- Implementar un analizador para la lista de la compra que:
  - Pueda leer una receta.
  - Escriba la lista de ingredientes con sus cantidades por pantalla.
  - Escriba la lista de utensilios a utilizar.
  - (Y otras cosas que se os ocurran en función de vuestro diseño).
  - Nota/pista: aquí se trata de poner en práctica un simulacro de tabla de símbolos para acumular la información vista al visitar el árbol.

### Secuencia recomendada

Siempre elaborando la memoria en paralelo a lo avanzado:

1. Realizar y entender el nivel básico de la parte 1.
2. Aplicando lo anterior, realizar y entender el nivel básico de la parte 2.
3. Hechas ambas cosas, entender el funcionamiento básico de los Listeners de ANTLR para construir el árbol en formato de texto plano y sacarlo por pantalla o por fichero. Así se pueden realizar directamente ambos niveles medios.
4. Pensar en posibles mejoras o pequeñas extensiones de las propuestas 1.1, 1.2 y 1.3 e implementarlas.

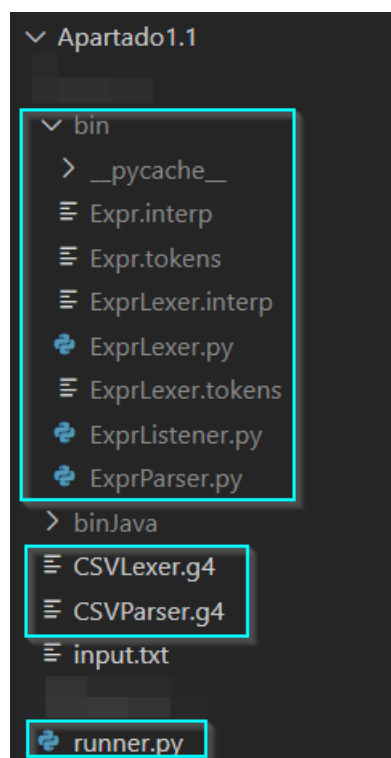
5. Entendido el funcionamiento de los Listeners es casi directo extraer la información para la parte avanzada de la segunda parte; solo quedará implementar el mecanismo.

## Contenido

Enunciado.....	1
1. Primera parte: generación de árboles sintácticos para lenguajes específicos.....	1
2. Segunda parte: las recetas de la abuela.....	5
Secuencia recomendada.....	5
Apartado 1.1:.....	7
Plantear alguna mejora que mejore o amplie la gramática.....	10
Apartado 1.2.....	13
Plantear alguna mejora que mejore o amplie la gramática.....	16
Apartado 1.3.....	19
Apartado 2.....	22

## Apartado 1.1

Se ha realizado la siguiente estructura de ficheros:



A continuación procederemos a explicar por orden la generación de los ficheros y cómo se ha llegado a esa estructura.

1. Nos generamos el CSVLexer.g4
  - a. Para realizarlo debemos tener en cuenta que el Lexer se trata de la parte que interpreta las cadenas de texto y las transforma a TOKENS, no interpreta sus relaciones.
  - b. El fichero es el siguiente:

```
Apartado1.1 > CSVLexer.g4 > ...
1  lexer grammar CSVLexer;
2
3  TXT_BLOCK_START : '"' -> skip, pushMode(TXT_BLOCK);
4
5  SEP      : [;,|?] -> skip;
6  NEWLINE : [\r\n]+;
7  CONTENT : ~[\r\n;|,"?]+;
8
9
10 mode TXT_BLOCK;
11 TXT_BLOCK_END : '"' -> skip, popMode;
12 TEXT_BLOCK: ~["]+;
13
```

- Como se puede observar se ha separado el Lexer y el Parser para poder simplificar el código.



- Podemos observar que está utilizando como separadores [;,|] como se describe en la práctica.

- Al encontrar un bloque de texto ["] entra en el modo TXT\_BLOCK para que nos pueda mostrar los elementos que se encuentran dentro del texto sin cambiar su contenido. Por ejemplo, si encuentra en el fichero "1,9s" , entrar en este modo permite que te retorne 1,9s sin quitar separadores ni ningún elemento que se encuentre dentro de las comillas dobles.
- Todo el contenido que se encuentre entre separadores lo va a modificar quitando los separadores y los NEWLINE.
- El AST permite distinguir la cabecera del resto de líneas del fichero. (Gracias al CSVParser.g4)

## 2. Nos generamos el CSVParser.g4

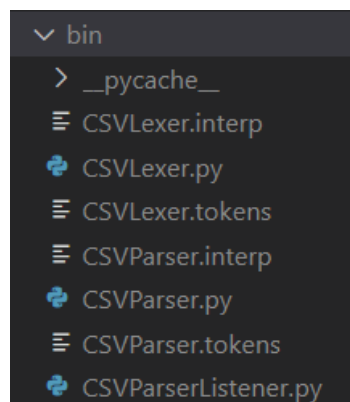
- Para generarlo necesitamos haber creado previamente el CSVLexer.g4
- El fichero CSVParser.g4 es el siguiente:

```
Apartado1.1 > CSVParser.g4 > ...
1  parser grammar CSVParser;
2
3  options{
4      tokenVocab=CSVLexer;
5  }
6
7  file    : (header)? (line)* EOF;
8
9  header  : cell* NEWLINE;
10
11 line    : cell* NEWLINE;
12 cell    : (TEXT_BLOCK | CONTENT);
13
```

- Como se puede observar el CSVParser se encarga de interconectar los TOKENS obtenidos por el CSVLexer y realiza una categorización entre file/line/cell [fichero/linea/celda].

## 3. Realizamos la compilación del Lexer y el Parser

```
emipley@DESKTOP-NFTDJAT:/mnt/e/Onedrive/Datos/Universidad/Cursos 3º/1º Cuatrimestre/Compiladores/Laboratorio/Práctica de Laboratorio 2/2.0 - Trabajo
/Apartado1.1$ antlr4 -Dlanguage=Python3 -o bin CSVLexer.g4 CSVParser.g4
```



#### 4. Ejecutamos lo que nos ha compilado el antlr con el siguiente comando:

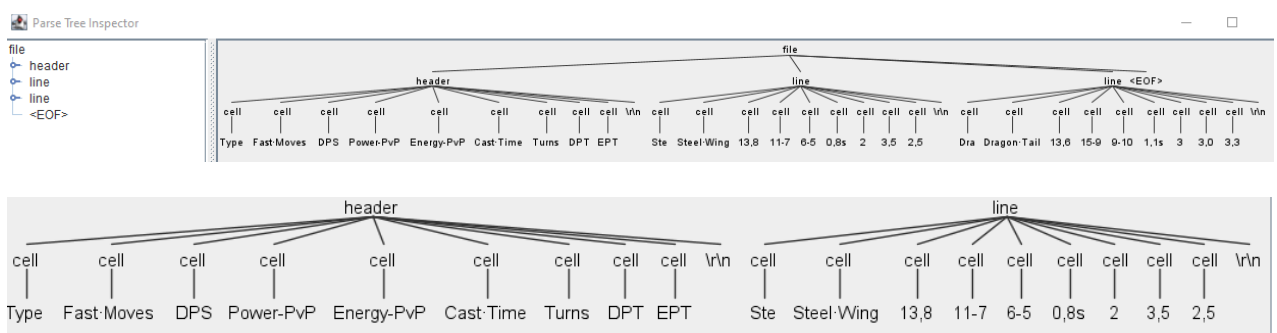
```
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo> python -u "e:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.1\runner.py" "E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.1\input.txt"
(file (header (cell Type) (cell Fast Moves) (cell DPS) (cell Power-PvP) (cell Energy-PvP) (cell Cast Time) (cell Turns) (cell DPT) (cell EPT) \r\n) (line (cell Ste) (cell Steel Wing) (cell 13,8) (cell 11-7) (cell 6-5) (cell 0,8s) (cell 2) (cell 3,5) (cell 2,5) \r\n) (line (cell Dra) (cell Dragon Tail) (cell 13,6) (cell 15-9) (cell 9-10) (cell 1,1s) (cell 3) (cell 3,0) (cell 3,3) \r\n) <EOF>)
```

- Podemos observar que nos retorna en texto plano el resultado de su ejecución quedando:

(file (header (cell Type) (cell Fast Moves) (cell DPS) (cell Power-PvP) (cell Energy-PvP) (cell Cast Time) (cell Turns) (cell DPT) (cell EPT) \r\n) (line (cell Ste) (cell Steel Wing) (cell 13,8) (cell 11-7) (cell 6-5) (cell 0,8s) (cell 2) (cell 3,5) (cell 2,5) \r\n) (line (cell Dra) (cell Dragon Tail) (cell 13,6) (cell 15-9) (cell 9-10) (cell 1,1s) (cell 3) (cell 3,0) (cell 3,3) \r\n) <EOF>)

#### 5. Ejecutamos lo anterior pero obteniendo una visualización gráfica

```
E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.1\binJava>java org.antlr.v4.gui.TestRig CSV file -tokens -gui "E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Ejemplos\datoscoma.csv"
[0,0:3='Type',<CONTENT>,1:0]
[1,5:14='Fast Moves',<CONTENT>,1:5]
[2,16:18='DPS',<CONTENT>,1:16]
[3,20:28='Power-PvP',<CONTENT>,1:20]
[4,30:39='Energy-PvP',<CONTENT>,1:30]
[5,41:49='Cast Time',<CONTENT>,1:41]
[6,51:55='Turns',<CONTENT>,1:51]
[7,57:59='DPT',<CONTENT>,1:57]
[8,61:63='EPT',<CONTENT>,1:61]
[9,64:64='\r',<NEWLINE>,1:64]
[10,65:65='\n',<NEWLINE>,1:65]
[11,66:68='Ste',<CONTENT>,2:0]
[12,70:79='Steel Wing',<CONTENT>,2:4]
[13,82:85='13,8',<TEXT_BLOCK>,2:16]
[14,88:91='11-7',<CONTENT>,2:22]
[15,93:95='6-5',<CONTENT>,2:27]
[16,98:101='0,8s',<TEXT_BLOCK>,2:32]
[17,104:104='2',<CONTENT>,2:38]
[18,107:109='3,5',<TEXT_BLOCK>,2:41]
[19,113:115='2,5',<TEXT_BLOCK>,2:47]
[20,117:117='\r',<NEWLINE>,2:51]
[21,118:118='\n',<NEWLINE>,2:52]
[22,119:121='Dra',<CONTENT>,3:0]
[23,123:133='Dragon Tail',<CONTENT>,3:4]
[24,136:139='13,6',<TEXT_BLOCK>,3:17]
[25,142:145='15-9',<CONTENT>,3:23]
[26,147:150='9-10',<CONTENT>,3:28]
[27,153:156='1,1s',<TEXT_BLOCK>,3:34]
[28,159:159='3',<CONTENT>,3:40]
[29,162:164='3,0',<TEXT_BLOCK>,3:43]
[30,168:170='3,3',<TEXT_BLOCK>,3:49]
[31,172:172='\r',<NEWLINE>,3:53]
[32,173:173='\n',<NEWLINE>,3:54]
[33,174:173='<EOF>',<EOF>,4:0]
```



## Plantear alguna mejora que mejore o amplie la gramática

Se ha añadido la posibilidad de interpretar las "?" como separadores.

1. Realizamos la modificación:

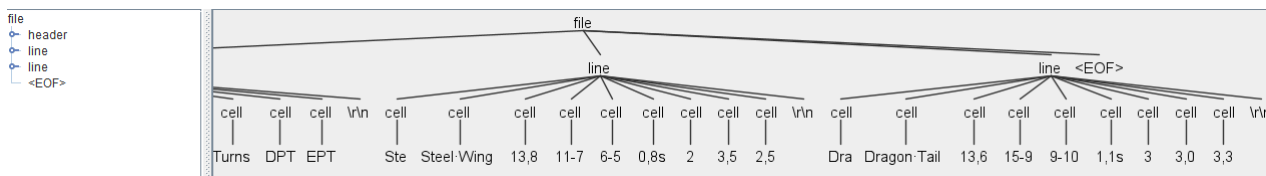
```
Apartado1.1 > ≡ CSVLexer.g4 > ...
1  lexer grammar CSVLexer;
2
3  TXT_BLOCK_START : '"' -> skip, pushMode(TXT_BLOCK);
4
5  SEP      : [;,|?] -> skip;
6  NEWLINE : [\r\n]+;
7  CONTENT : ~[\r\n;|, "?" ]+;
8
9
10 mode TXT_BLOCK;
11 TXT_BLOCK_END : '"' -> skip, popMode;
12 TEXT_BLOCK: ~[" ]+;
13
```

2. Modificamos los datos que le vamos a pasar para meter el nuevo separador "?":

```
Apartado1.1 > ≡ input.txt
1  Type, Fast Moves, DPS, Power-PvP, Energy-PvP, Cast Time, Turns, DPT, EPT
2  Ste?Steel Wing?"13,8"?11-7?6-5?"0,8s"?2?"3,5"?2,5"
3  Dra, Dragon Tail;"13,6",15-9,9-10|"1,1s",3,"3,0","3,3"
4
```

### 3. Realizamos la ejecución:

```
E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Labotatorio 2\2.0 - Trabajo\Apartado1.1\binJava>java org.antlr.v4.gui.TestRig CSV file -tokens -gui ..\input.txt
[0,0:3='Type',<CONTENT>,1:0]
[1,5:14='Fast Moves',<CONTENT>,1:5]
[2,16:18='DPS',<CONTENT>,1:16]
[3,20:28='Power-PvP',<CONTENT>,1:20]
[4,30:39='Energy-PvP',<CONTENT>,1:30]
[5,41:49='Cast Time',<CONTENT>,1:41]
[6,51:55='Turns',<CONTENT>,1:51]
[7,57:59='DPT',<CONTENT>,1:57]
[8,61:63='EPT',<CONTENT>,1:61]
[9,64:64='\r',<NEWLINE>,1:64]
[10,65:65='\n',<NEWLINE>,1:65]
[11,66:68='Ste',<CONTENT>,2:0]
[12,70:79='Steel Wing',<CONTENT>,2:4]
[13,82:85='13,8',<TEXT_BLOCK>,2:16]
[14,88:91='11-7',<CONTENT>,2:22]
[15,93:95='6-5',<CONTENT>,2:27]
[16,98:101='0,8s',<TEXT_BLOCK>,2:32]
[17,104:104='2',<CONTENT>,2:38]
[18,107:109='3,5',<TEXT_BLOCK>,2:41]
[19,113:115='2,5',<TEXT_BLOCK>,2:47]
[20,117:117='\r',<NEWLINE>,2:51]
[21,118:118='\n',<NEWLINE>,2:52]
[22,119:121='Dra',<CONTENT>,3:0]
[23,123:133='Dragon Tail',<CONTENT>,3:4]
[24,136:139='13,6',<TEXT_BLOCK>,3:17]
[25,142:145='15-9',<CONTENT>,3:23]
[26,147:150='9-10',<CONTENT>,3:28]
[27,153:156='1,1s',<TEXT_BLOCK>,3:34]
[28,159:159='3',<CONTENT>,3:40]
[29,162:164='3,0',<TEXT_BLOCK>,3:43]
[30,168:170='3,3',<TEXT_BLOCK>,3:49]
[31,172:172='\r',<NEWLINE>,3:53]
[32,173:173='\n',<NEWLINE>,3:54]
[33,174:173='<EOF>',<EOF>,4:0]
```



```
PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Labotatorio 2\2.0 - Trabajo> python -u "e:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Labotatorio 2\2.0 - Trabajo\Apartado1.1\runner.py" "E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Labotatorio 2\2.0 - Trabajo\Apartado1.1\input.txt"
(file (header (cell Type) (cell Fast Moves) (cell DPS) (cell Power-PvP) (cell Energy-PvP) (cell Cast Time) (cell Turns) (cell DPT) (cell EPT) \r\n) (line (cell Ste) (cell Steel Wing) (cell 13,8) (cell 11-7) (cell 6-5) (cell 0,8s) (cell 2) (cell 3,5) (cell 2,5) \r\n) (line (cell Dra) (cell Dragon Tail) (cell 13,6) (cell 15-9) (cell 9-10) (cell 1,1s) (cell 3) (cell 3,0) (cell 3,3) \r\n) <EOF>)
```

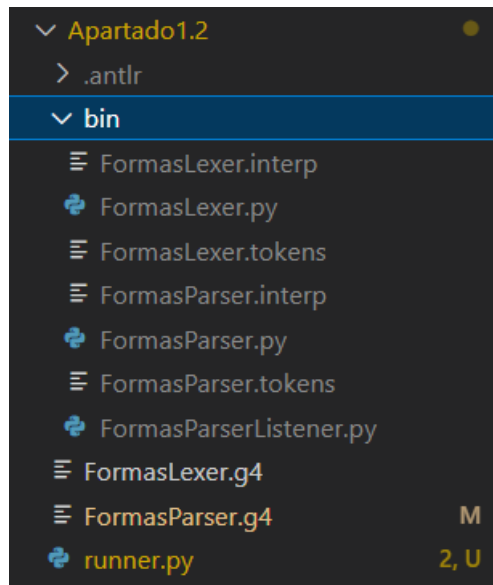
- Como podemos observar nos da exactamente el mismo resultado que en sin la mejora (habiendo añadido ?) por lo que podemos decir que la mejora se ha realizado correctamente.

El fichero (runner.py) que se ha utilizado para obtener en texto plano la salida es el siguiente:

```
Apartado1.1 > runner.py > main
1  import sys
2  from antlr4 import *
3
4
5
6  from antlr4.tree.Trees import Trees
7  from bin.CSVLexer import CSVLexer as Lexer
8  from bin.CSVParser import CSVParser as Parser
9  # from antlr4.tree.Trees import Trees
10
11 def main(argv):
12     # Inicializamos la entrada de datos
13     input_stream = FileStream(argv[1])
14     # Conectamos con el lexer
15     lexer = Lexer(input_stream)
16     # Inicializamos el canal de tokens
17     tokens = CommonTokenStream(lexer)
18     # Preparamos el parser
19     parser = Parser(tokens)
20     # Generamos el arbol a partir del axioma de la gramática
21     tree = parser.file_1()
22     # Mostrar el arbol por consola
23     print(Trees.toStringTree(tree, None, parser))
24
25
26 if __name__ == '__main__':
27     main(sys.argv)
28
```

## Apartado 1.2

Se ha realizado la siguiente estructura de ficheros:



A continuación procederemos a explicar por orden la generación de los ficheros y cómo se ha llegado a esa estructura.

1. Nos generamos el FormasLexer.g4
  - a. Para realizarlo debemos tener en cuenta que el Lexer se trata de la parte que interpreta las cadenas de texto y las transforma a TOKENS, no interpreta sus relaciones.
  - b. El fichero es el siguiente:

```

Apartado1.2 > ≡ FormasLexer.g4 > ...
1  lexer grammar FormasLexer;
2
3  // Header
4  IMGDIMSPEC      : 'imgdim: ';
5  SHPDIMSPEC      : 'shpdim: ';
6
7  INT              : DIGIT+;
8
9  fragment DIGIT   : [0-9];
10
11 // Shapes
12 SEP              : ',';
13 ROWSEP           : '|';
14
15 SHAPES_START     : '>>>';
16 SHAPES_END       : '<<<';
17
18 SHAPE            : [a-z]+;
19
20 NEWLINE          : [\n\r] -> skip;
21 WS               : ' ' -> skip;
22

```

- Como se puede observar se ha separado el Lexer y el Parser para poder simplificar el código.
  - Podemos observar que se encuentran las cabeceras del fichero que indicarán las dimensiones de la imagen y de las figuras (imgdim: , shpdim:). Se hará mediante INT (números naturales)
  - Se separan las figuras por fila usando el separador ROWSEP, y se separarán las propias figuras por SEP.
  - Las figuras podrán ser ["cuadrado","circulo","triangulo"]
  - Se indican los delimitadores de la especificación de las figuras con SHAPES\_START y SHAPES\_END.
2. Nos generamos el FormasParser.g4
- a. Para generarlo necesitamos haber creado previamente el FormasLexer.g4
  - b. El fichero FormasParser.g4 es el siguiente:



```

Apartado1.2 > ≡ FormasParser.g4 > ...
1  parser grammar FormasParser;
2
3  options{
4      tokenVocab=FormasLexer;
5  }
6
7  geom          : dimensions shapes EOF;
8  dimensions    : imgdim SEP shpdim;
9
10 imgdim        : IMGDIMSPEC size_=INT;
11 shpdim        : SHPDIMSPEC size_=INT;
12
13 shapes        : SHAPES_START row (ROWSEP row)* SHAPES_END;
14
15 row           : SHAPE (SEP SHAPE)*;
16

```

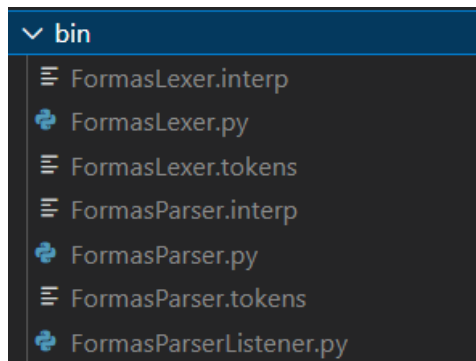
- Como se puede observar el FormasParser se encarga de interconectar los TOKENS obtenidos por el FormasLexer y realiza una categorización entre geom[programa]/dimensions[dimensiones/tamaños de la imagen y las figuras]/shapes[figuras]/row[filas].
- Este programa leerá solamente una línea. Como se especifica en el enunciado de la práctica.

### 3. Realizamos la compilación del Lexer y el Parser

```

emiplej@DESKTOP-NFTDJAT:/mnt/e/Onedrive/Datos/Universidad/Cursos 3º/1º Cuatrimestre/Compiladores/Laboratorio/Práctica de Laboratorio 2/2.0 - Trabajo
/Apartado1.2$ antlr4 -Dlanguage=Python3 -o bin FormasLexer.g4 FormasParser.g4
emiplej@DESKTOP-NFTDJAT:/mnt/e/Onedrive/Datos/Universidad/Cursos 3º/1º Cuatrimestre/Compiladores/Laboratorio/Práctica de Laboratorio 2/2.0 - Trabajo
/Apartado1.2$

```



#### 4. Ejecutamos lo que nos ha compilado el antlr con el siguiente comando:

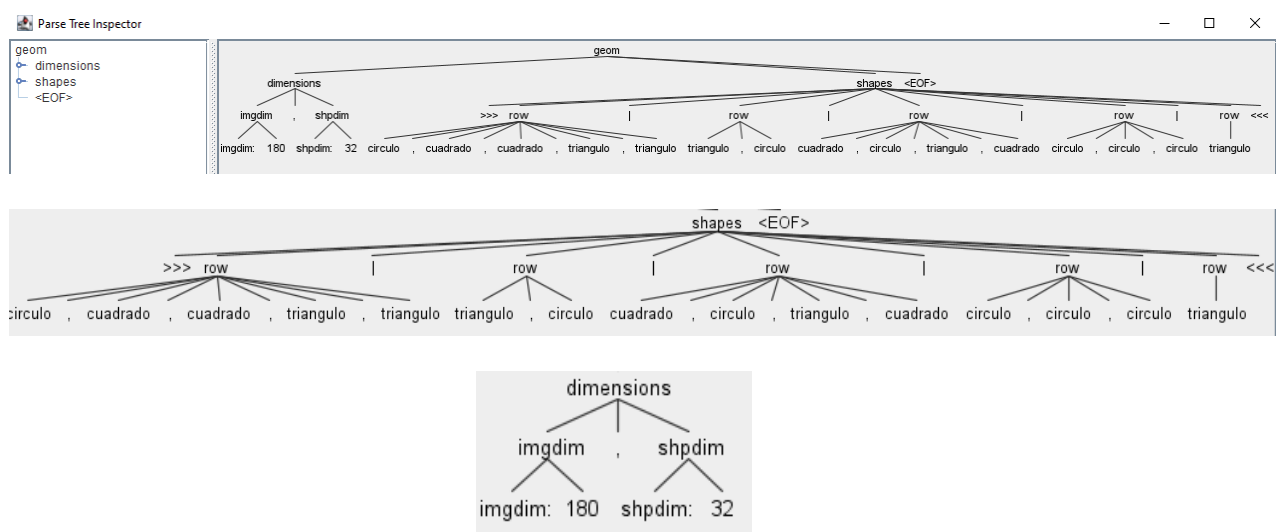
```
PS E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo> python -u "E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.2\runner.py" "E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Ejemplos\datosfigura.txt"
(ggeom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , cuadrado , triangulo , triangulo) | (row triangulo , circulo) | (row cuadrado , circulo , triangulo , cuadrado) | (row circulo , circulo , circulo) | (row triangulo) <<<) <EOF>)
```

- Podemos observar que nos retorna en texto plano el resultado de su ejecución quedando:

```
(ggeom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , cuadrado , triangulo , triangulo) | (row triangulo , circulo) | (row cuadrado , circulo , triangulo , cuadrado) | (row circulo , circulo , circulo) | (row triangulo) <<<) <EOF>)
```

#### 5. Ejecutamos lo anterior pero obteniendo una visualización gráfica

```
E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.2\bin\java>java org.gantlr.v4.gui.TestRig Formas ggeom -tokens -gui "E:\OneDrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Ejemplos\datosfigura.txt"
[0,0:6='imgdim:',<'imgdim:',1:0]
[1,7:9='180',<INT>,1:7]
[2,10:10='',<'>',1:10]
[3,11:17='shpdim:',<'shpdim:',1:11]
[4,18:19='32',<INT>,1:18]
[5,20:22='>>>',<'>>>',1:20]
[6,23:29='circulo',<SHAPE>,1:23]
[7,30:30='',<'>',1:30]
[8,31:38='cuadrado',<SHAPE>,1:31]
[9,39:39='',<'>',1:39]
[10,40:47='cuadrado',<SHAPE>,1:40]
[11,48:48='',<'>',1:48]
[12,49:57='triangulo',<SHAPE>,1:49]
[13,58:58='',<'>',1:58]
[14,59:67='triangulo',<SHAPE>,1:59]
[15,68:68='|',<'>',1:68]
[16,69:77='triangulo',<SHAPE>,1:69]
[17,78:78='',<'>',1:78]
[18,79:85='circulo',<SHAPE>,1:79]
[19,86:86='|',<'>',1:86]
[20,87:94='cuadrado',<SHAPE>,1:87]
[21,95:95='',<'>',1:95]
[22,96:102='circulo',<SHAPE>,1:96]
[23,103:103='',<'>',1:103]
[24,104:112='triangulo',<SHAPE>,1:104]
[25,113:113='',<'>',1:113]
[26,114:121='cuadrado',<SHAPE>,1:114]
[27,122:122='|',<'>',1:122]
[28,123:129='circulo',<SHAPE>,1:123]
[29,130:130='',<'>',1:130]
[30,131:137='circulo',<SHAPE>,1:131]
[31,138:138='',<'>',1:138]
[32,139:145='circulo',<SHAPE>,1:139]
[33,146:146='|',<'>',1:146]
[34,147:155='triangulo',<SHAPE>,1:147]
[35,156:158='<<<',<'<<<',1:156]
[36,161:160='<EOF>',<EOF>,2:0]
```



## Plantear alguna mejora que mejore o amplie la gramática

Se ha añadido la posibilidad de interpretar cualquier elemento de texto, por lo que puede interpretar cualquier figura. Por ejemplo: círculo, triángulo, cuadrado, rectángulo, pentágono, etc.

1. Realizamos la modificación:

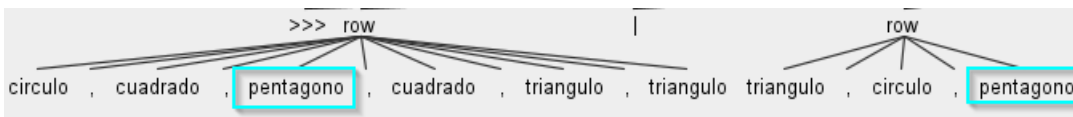
```
Apartado1.2 > ≡ FormasLexer.g4 > ...
1  lexer grammar FormasLexer;
2
3  // Header
4  IMGDIMSPEC      : 'imgdim:';
5  SHPDIMSPEC      : 'shpdim:';
6
7  INT              : DIGIT+;
8
9  fragment DIGIT   : [0-9];
10
11 // Shapes
12 SEP              : ',';
13 ROWSEP           : '|';
14
15 SHAPES_START     : '>>>';
16 SHAPES_END       : '<<<';
17
18 SHAPE            : [a-z]+;
19
20 NEWLINE          : [\n\r] -> skip;
21 WS               : ' ' -> skip;
22
```

2. Modificamos los datos que le vamos a pasar para meter la nueva figura:

```
1  imgdim:180,shpdim:32>>>circulo,cuadrado,pentagono,cuadrado,triangulo,triangulo|triangulo,circulo,pentagono|cuadrado,circulo,triangulo,cuadr
2  ado,pentagono|circulo,circulo,circulo|triangulo,pentagono<<<
```

### 3. Realizamos la ejecución:

```
E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.2\binJava>java org.antlr.v4.gui.TestRig Formas geom -tokens -gui ..\input.txt
[@0,0:6='imgdim:',<'imgdim:'>,1:0]
[@1,7:9='180',<INT>,1:7]
[@2,10:10='',<'>',1:10]
[@3,11:17='shpdim:',<'shpdim:'>,1:11]
[@4,18:19='32',<INT>,1:18]
[@5,20:22='>>>',<'>>>'>,1:20]
[@6,23:29='circulo',<SHAPE>,1:23]
[@7,30:30='',<'>',1:30]
[@8,31:38='cuadrado',<SHAPE>,1:31]
[@9,39:39='',<'>',1:39]
[@10,40:48='pentagono',<SHAPE>,1:40]
[@11,49:49='',<'>',1:49]
[@12,50:57='cuadrado',<SHAPE>,1:50]
[@13,58:58='',<'>',1:58]
[@14,59:67='triangulo',<SHAPE>,1:59]
[@15,68:68='',<'>',1:68]
[@16,69:77='triangulo',<SHAPE>,1:69]
[@17,78:78='|',<'|'>,1:78]
[@18,79:87='triangulo',<SHAPE>,1:79]
[@19,88:88='',<'>',1:88]
[@20,89:95='circulo',<SHAPE>,1:89]
[@21,96:96='',<'>',1:96]
[@22,97:105='pentagono',<SHAPE>,1:97]
[@23,106:106='|',<'|'>,1:106]
[@24,107:114='cuadrado',<SHAPE>,1:107]
[@25,115:115='',<'>',1:115]
[@26,116:122='circulo',<SHAPE>,1:116]
[@27,123:123='',<'>',1:123]
[@28,124:132='triangulo',<SHAPE>,1:124]
[@29,133:133='',<'>',1:133]
[@30,134:141='cuadrado',<SHAPE>,1:134]
[@31,142:142='',<'>',1:142]
[@32,143:151='pentagono',<SHAPE>,1:143]
[@33,152:152='|',<'|'>,1:152]
[@34,153:159='circulo',<SHAPE>,1:153]
[@35,160:160='',<'>',1:160]
[@36,161:167='circulo',<SHAPE>,1:161]
[@37,168:168='',<'>',1:168]
[@38,169:175='circulo',<SHAPE>,1:169]
[@39,176:176='|',<'|'>,1:176]
[@40,177:185='triangulo',<SHAPE>,1:177]
[@41,186:186='',<'>',1:186]
[@42,187:195='pentagono',<SHAPE>,1:187]
[@43,196:198='<<<',<'<<<'>,1:196]
[@44,201:200='<EOF>',<EOF>,2:0]
```





```
python -u "E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.2\runner.py" "E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.2\input.txt"
(geom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , pentagono , cuadrado , triangulo , triangulo) | (row triangulo , circulo , pentagono) | (row cuadrado , circulo , triangulo , cuadrado , pentagono) | (row circulo , circulo , circulo) | (row triangulo , pentagono) <<<) <EOF>)
```

(geom (dimensions (imgdim imgdim: 180) , (shpdim shpdim: 32)) (shapes >>> (row circulo , cuadrado , pentagono , cuadrado , triangulo , triangulo) | (row triangulo , circulo , pentagono) | (row cuadrado , circulo , triangulo , cuadrado , pentagono) | (row circulo , circulo , circulo) | (row triangulo , pentagono) <<<) <EOF>)

- Como podemos observar nos da exactamente el mismo resultado que sin la mejora (habiendo añadido pentagono) por lo que podemos decir que la mejora se ha realizado correctamente.

El fichero (runner.py) que se ha utilizado para obtener en texto plano la salida es el siguiente:

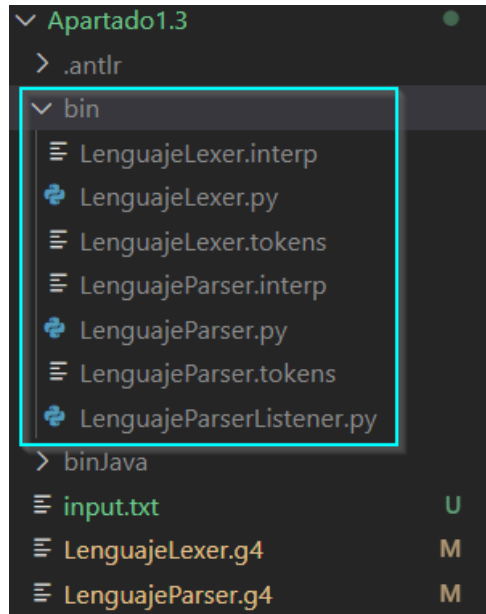
```
Apartado1.2 >  runner.py > ...
1  import sys
2  from antlr4 import *
3  from bin.FormasLexer import FormasLexer as Lexer
4  from bin.FormasParser import FormasParser as Parser
5  from antlr4.tree.Trees import Trees
6
7  def main(argv):
8      # Inicializamos la entrada de datos
9      input_stream = FileStream(argv[1])
10     # Conectamos con el lexer
11     lexer = Lexer(input_stream)
12     # Inicializamos el canal de tokens
13     tokens = CommonTokenStream(lexer)
14     # Preparamos el parser
15     parser = Parser(tokens)
16     # Generamos el arbol a partir del axioma de la gramática
17     tree = parser.geom()
18     # Mostrar el arbol por consola
19     print(Trees.toStringTree(tree, None, parser))
20
21
22  if __name__ == '__main__':
23      main([sys.argv])
```

## Apartado 1.3

No se ha podido realizar el Apartado 1.3 debido a errores.

A continuación, se mostrarán capturas del Lexer y el Parser, así como su ejecución:

- Estructura de ficheros:



- LenguajeLexer.g4

```

Apartado1.3 > LenguajeLexer.g4 > ...
1  lexer grammar LenguajeLexer;
2
3  // Separadores:
4  SEP : ';'P';
5  IG : '=';
6  OPL : [><];
7  OP : [+ -];
8
9  // Ignores
10 WH : ' ' -> skip;
11 NEWLINE : [\r\n]+;
12
13 COMENTARIOENTRAR : '#' -> skip, pushMode(COMENTARIO_MODE);
14 ASIGNAR : 'asignar' -> pushMode(ASIGNAR_MODE);
15 IF_SEP : '???' -> pushMode(IF_MODE);
16 MOSTRAR : 'mostrar' -> pushMode(MOSTRAR_MODE);
17
18 mode COMENTARIO_MODE;
19 COMMENT_STOP : [\n\r] -> popMode, skip;
20 COMMENT : ~[\n\r] -> skip;
21
22 mode ASIGNAR_MODE;
23 SEP_ASIGNAR : ';'P' -> popMode;
24 VARIABLE_ASIGNAR : ID;
25
26 mode IF_MODE;
27 IF_SALIR : 'terminar' -> popMode, skip;
28 SI_MODE : 'si ->';
29 NO_MODE : 'no ->';
30
31 mode MOSTRAR_MODE;
32 SEP_MOSTRAR : ';'P' -> popMode;
33 VARIABLE_MOSTRAR : ID;
34
35 // General identifiers
36 NUMERO : DIGITO+ ( '.' DIGITO+ )?;
37 ID : (LETRA | DIGITO)+;
38 // Low level rules
39 fragment LETRA : [a-zA-Z];
40 fragment DIGITO : [0-9];

```

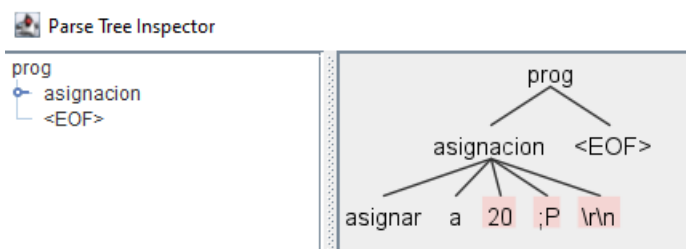
- LenguajeParser.g4

```

Apartado1.3 > LenguajeParser.g4 > linea
1  parser grammar LenguajeParser;
2
3  options{
4      tokenVocab=LenguajeLexer;
5  }
6  /*
7      * Parser rules
8      */
9  // prog      : linea* EOF;
10 prog      : asignacion EOF;
11
12 linea      : asignacion
13             | COMENTARIOENTRAR COMMENT
14             | comprobacion
15             | MOSTRAR ID SEP
16             ;
17
18 asignacion  : ASIGNAR VARIABLE_ASIGNAR IG expresion SEP_ASIGNAR;
19 comprobacion : ID OPL expresion IF_SEP;
20 expresion   : ID (OP ID)*;

```

- Ejecución y errores:



```

PS E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.3\binJava> gru
n Lenguaje prog -tokens -gui ..\input.txt

E:\Onedrive\Datos\Universidad\Cursos 3º\1º Cuatrimestre\Compiladores\Laboratorio\Práctica de Laboratorio 2\2.0 - Trabajo\Apartado1.3\binJava> java or
g.antlr.v4.gui.TestRig Lenguaje prog -tokens -gui ..\input.txt
line 1:7 token recognition error at: ' '
line 1:9 token recognition error at: ' '
line 1:10 token recognition error at: '='
line 1:11 token recognition error at: ' '
line 1:14 token recognition error at: ' '
[0,0:6='asignar',<'asignar'>,1:0]
[1,8:8='a',<VARIABLE_ASIGNAR>,1:8]
[2,12:13='20',<VARIABLE_ASIGNAR>,1:12]
[3,15:16=';',<SEP_ASIGNAR>,1:15]
[4,17:18='\r\n',<NEWLINE>,1:17]
[5,19:18='<EOF>',<EOF>,2:0]
line 1:12 mismatched input '20' expecting '='

```



## Apartado 2

Funcionamiento del lenguaje

Antes de definir la gramática se tiene que especificar una estructura y funcionamiento general del lenguaje que se está pidiendo.

Se ha llegado a tres divisiones jerárquicas que son:

- Secciones: se comportan como un separador que permite localizar las partes generales de una receta. Como por ejemplo la preparación de ingredientes, la cocción de los mismos y el emplatado.
- Indicaciones de tiempo (timestamps): permiten definir los momentos en los que se tienen que realizar las acciones. Todas las expresiones que se encuentren dentro de un bloque de estos se realizan “al mismo tiempo” hasta que se indique una posición de tiempo nueva.
- Expresiones: unidad mínima que indica una acción única a realizar. Estas también se pueden dividir en varias partes:
  - o Almacenamiento: indica dónde se almacenarán los resultados de la acción indicada
  - o Verbo: indica que acción se aplicará a los ingredientes
  - o Ingredientes: pueden ser de varios tipos (desde ingredientes puros (entre comillas) a otros almacenamientos que se hallan declarado anteriormente). Estos pueden ser tantos como se quiera y están separados por una coma.
  - o Utensilio: muchos verbos tienen utensilios implícitos (por ejemplo cortar tiene cuchillo y tabla). Además de estos, se pueden especificar otros mediante el uso de la palabra “con”.
  - o Temperatura: solo aplicable a algunos verbos, permite indicar valores de temperatura para los almacenamientos. Esto se puede hacer en expresiones con verbos como “calentar”.
- Aunque hay comentarios dentro de la gramática no hemos conseguido hacerlo funcionar correctamente en todos los casos así que no se recomienda su uso.

*Ejemplo de sintaxis*

El siguiente es un ejemplo de la sintaxis del lenguaje:

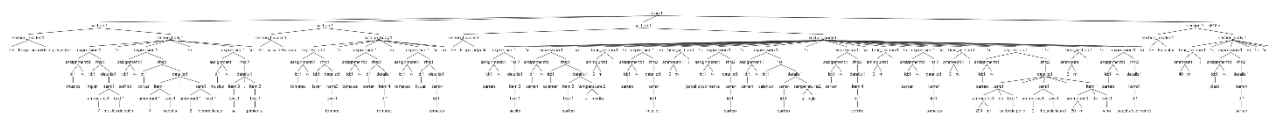
```
>> Preparacion de ingredientes
muslos <- limpiar 7 "muslos de pollo"
sofrito <- cortar 1 "cebolla", 2 "diente de ajo"
muslos <- "sal", "pimienta"
>> Salsa de tomate
tomates <- lavar "tomate"
tomates <- cortar tomates
tomates <- licuar tomates

>> Preparar pollo
sarten <- "aceite"
```

```

sarten <- calentar "sarten" a medio
5m:
sarten <- muslos
5m:
papel_absorbente <- sarten
sarten <- calentar sarten a bajo
sarten <- sofrito
3m:
sarten <- tomates
5m:
sarten <- 200ml de "caldo de pollo", 2 "hoja de laurel", 30ml de "vino"
3m:
sarten <- papel_absorbente
>> Emplatar
40m:
plato <- sarten

```



#### Parte avanzada

Una vez se tiene un parse-tree se genera una estructura de objetos de Python que representan la receta introducida. Esta se genera mediante visitors y permite el cálculo de métricas así como la ejecución (si fuese necesario) de una forma sencilla.

Para las recetas se ha seguido una división similar a la de la gramática. Teniendo un array de secciones, que contiene un array de timestamps y por cada uno de estos un array de expresiones. Todas estas partes se encuentran dentro de la carpeta lang.

Un hecho importante es que las keywords y verbos no están definidas dentro de la gramática, si no dentro del archivo keywords.py (en lang). Por lo que añadir nuevos verbos y utensilios implícitos es muy sencillo.

La salida para la receta anterior es la siguiente. Los cálculos se pueden encontrar dentro de interpreter.reader.

#### Lista de ingredientes:

muslos de pollo:	7
cebolla:	1
diente de ajo:	2
sal:	1
pimienta:	1
tomate:	1

aceite: 1  
sarten: 1  
caldo de pollo: 0.2l  
hoja de laurel: 2  
vino: 0.03l

Lista de utensilios:

cuchillo

tabla

licuadora

Tiempo total: 3660.0s