


# MANUAL TECNICO

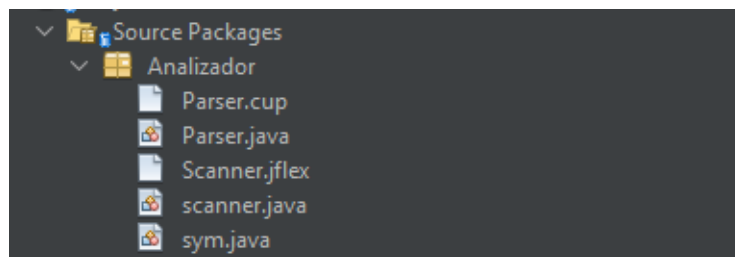
## HERRAMIENTAS → VERSION ( UTILIZADAS )

	JFlex	_____→	1.7.0
	JCup	_____→	11
	Jfreechart	_____→	1.5.3
	JDK	_____→	20

Este programa es principalmente un traductor de código, para este caso se traduce código Statpy a código python, otras de las funcionalidades de este programa es el de la recolección de datos y mediante de estructuras se hacen referencia para obtener valores y así poder generar gráficas ilustrando los datos encontrados o referidos. Por tanto en este documento se enfocará en la explicación de cómo funcionan estas funcionalidades descritas anteriormente y no en otras como la de generar interfaces o abrir archivos, etc.

## RECOLECCION DE DATOS

El algoritmo de uso de este programa es primero insertar archivos con estructura json para posteriormente utilizar estos datos en donde se haga referencia con el código statpy. Los primeros pasos son crear analizadores para reconocer la estructura json simple en la que vendrán los datos. Estos analizadores se encuentran en src/Analizadores



Primero se crea el archivo .jflex en este se definirá lo siguiente:

- Atributos con los que trabajará el scanner

```
%class scanner //Define como trabajara el scanner
%cup //trabajar con cup
%public //tipo publico
%line //conteo de lineas - linea por linea
%char //caracter por caracter
%unicode //tipo de codificacion para que acepte la ene u otro caracter
%ignorecase //case indispensable
```

- Expresiones regulares

```
// -----> Expresiones Regulares
ENTERO = [0-9]+
DECIMAL = [0-9]+("."[ 0-9])?
LETRA = [a-zA-ZÑñ]
ID = {LETRA}({LETRA}|{ENTERO}|"_")*
CADENA = [""](^\"\\n)+[\""]
CADENA1 = [""](^0-9("."[ 0-9])?"\"\\n)+[\""]
SPACE = [\ \r\t\f]
ENTER = \r|\\n|\\r\\n\\
CARACTER = [^\r\\n]
COMENTARIO_INLINE = "//" {CARACTER}* {ENTER}?
COMENTARIO_MULTII = /\/*[^*]*\*+([^\/*][^*]*\*+)*\/
//simbolos
LLA_IZQ = "{"
LLA_DER = "}"
MENOR_QUE = "<"
MAYOR_QUE = ">"
COMILLA_DOBLE = "\"\""
DOS_PUNTOS = ":"
COMA = ","
```

- Retornos de la informacion de los match encontrados (reglas lexicas)

```
//-----> Reglas lexicas -----
<YYINITIAL> {LLA_IZQ} { return new Symbol(sym.LLA_IZQ, yyline, yycolumn, yytext()); }
<YYINITIAL> {LLA_DER} { return new Symbol(sym.LLA_DER, yyline, yycolumn, yytext()); }
<YYINITIAL> {MENOR_QUE} { return new Symbol(sym.MENOR_QUE, yyline, yycolumn, yytext()); }
<YYINITIAL> {MAYOR_QUE} { return new Symbol(sym.MAYOR_QUE, yyline, yycolumn, yytext()); }
<YYINITIAL> {COMILLA_DOBLE} { return new Symbol(sym.COMILLA_DOBLE, yyline, yycolumn, yytext()); }
<YYINITIAL> {DOS_PUNTOS} { return new Symbol(sym.DOS_PUNTOS, yyline, yycolumn, yytext()); }
<YYINITIAL> {COMA} { return new Symbol(sym.COMA, yyline, yycolumn, yytext()); }

<YYINITIAL> {ENTERO} { return new Symbol(sym.ENTERO, yyline, yycolumn, yytext()); }
<YYINITIAL> {DECIMAL} { return new Symbol(sym.DECIMAL, yyline, yycolumn, yytext()); }
<YYINITIAL> {CADENA} { return new Symbol(sym.CADENA, yyline, yycolumn, yytext()); }
<YYINITIAL> {ID} { return new Symbol(sym.ID, yyline, yycolumn, yytext()); }

<YYINITIAL> {SPACE} { /*Espacion en blanco , ignorados*/}
<YYINITIAL> {ENTER} { /*Saltos de linea, ignorados*/}

// -----> Ignorados
<YYINITIAL> {COMENTARIO_MULTII} { /*Comentario multilinea, ignorados*/ System.out.println("Comentario mult
<YYINITIAL> {COMENTARIO_INLINE} { /*Comentario linea, ignorados*/ System.out.println("Comentario linea: "
```

- Excepciones

```
//-----> Errores lexicos

<YYINITIAL> . {
    String errLex = "Error lexico : '"+yytext()+"' en la linea: "+(yyline+1)+" y columna: "+(yycolumn+1);
    System.out.println(errLex);
}
```

Despues de terminar el archivo jflex se crea el archivo jcup en el siguiente orden

- Exportacion de paquete
- Definicion de terminales

```
//TERMINALES
terminal String LLA_IZQ,LLA_DER,MENOR_QUE,MAYOR_QUE,COMILLA_DOBLE,DOS_PUNTOS,COMA;
terminal String ENTERO,DECIMAL,LETRA,ID,CADENA,CADENA1,SPACE,ENTER,CARACTER;
```

- Definicio de no terminales

```
//NO TERMINALES
non terminal inicial, lista_instruccion;
non terminal miembro, par;
non terminal declaracion, valor;
```

- Definir la gramatica

```
//INICIA LA GRAMATICA
start with inicial;

//-----> Producciones <-----
inicial ::= lista_instruccion
;

lista_instruccion ::= LLA_IZQ miembro LLA_DER
;

miembro ::= par
|par COMA miembro
;

par ::= declaracion
;

declaracion ::= CADENA:x DOS_PUNTOS valor:val {:
    proyectol.Proyectol.carga(x, val.toString());:}
;

valor ::= DECIMAL:a {: RESULT = a; :}
|ENTERO:b {: RESULT = b; :}
|CADENA:c {: RESULT = c; :}
;
```

Esta gramatica unicamente sirve para reconocer archivos json simples, es decir no se puede tener una lista de claves dentro del valor de un clave

Para generar las clases que creen el arbol ascendente se utiliza el siguiente metodo:

```
public static void analizadores(String ruta, String jflexFile, String cupFile){
    try {
        String opcionesJflex[] = {ruta+jflexFile,"-d",ruta};
        jflex.Main.generate(opcionesJflex);
    }
}
```

```

        String opcionesCup[] = {"-destdir", ruta, "-parser", "Parser", ruta+cupFile};
        java_cup.Main.main(opcionesCup);
        System.out.println("Se crearon los analizadores ");

    } catch (Exception e) {
        System.out.println("No se ha podido generar los analizadores");
        System.out.println(e);
    }
}

```

Esta función recibe como parametro la ruta donde se encuentran los archivos jcup y jflex, el nombre del archivo jflex y el nombre del archivo jcup.

Para poder hacer uso del analizador que se creo ahora se hace uso de la siguiente función:

```

public static void analizar (String entrada){
    try {
        Analizador.scanner scanner = new Analizador.scanner(new StringReader(entrada));
        Analizador.Parser parser = new Analizador.Parser(scanner);
        parser.parse();
    } catch (Exception e) {
        System.out.println("Error fatal en compilación de entrada.");
        System.out.println(e);
    }
}

```

Esta función recibe la cadena que contiene el código fuente y es pasada por el archivo generado de .jflex para obtener la lista de tokens y esta lista de tokens es pasada al analizador generado del archivo. jcup para poder reconocer la tokens y poder manipularlos. Este código fuente es obtenido de la interfaz gráfica y esta función analizar es ejecutada al momento de precionar en la opción Analizador>Json, al momento de hacer esta acción se realiza la siguiente acción:

```

private void jMenuItem5ActionPerformed(java.awt.event.ActionEvent evt) {
    //Estructura para guardar clave : valor

    //Extraer codigo fuente
    String codigoFuente = jTextArea1.getText();

    //Recolectar nombre del json
    String nombreFile = selectedFile.getName();
    //Mandaar nombre del archivo
    proyectol.Proyectol.compilarjson(codigo: codigoFuente, files: nombreFile);
    //Creacion de analizador
    proyectol.Proyectol.analizadores(ruta: "src/Analizador/", jflexFile: "Scanner.jflex",
    //Ejecucion de analizadore
    proyectol.Proyectol.analizar(entrada: codigoFuente);

    System.out.println(x: "----- Termino el analisis -----");
    System.out.println(x: "tokens de los json: ");
    func.Funcion.mostrar();
}

```

Se extrae el nombre del json y el codigo fuente para poder almacenarlo en un HashMap definido en en package func y en la clase Funcion:

```

public static HashMap<String, List<HashMap<String, String>>> archivosJson = new HashMap<>();

```

Este hashMap recibe un string como clave, este almacenara especificamente el nombre de los archivos json y el valor de esto es una lista que contendra otro hashmap que almacenara un string como valor y un string como clave y con este se tiene una estructura de datos que almacenara todos los datos de los json por nombre de archivo y para recorrerla o buscar datos se utiliza esta funcion:

```

public static String buscaEstruc(String archivoBus, String claveBus){
    String valorBus = "";
    for (Map.Entry<String, List<HashMap<String, String>>> entry : archivosJson.entrySet()) {
        String archivo = entry.getKey();
        List<HashMap<String, String>> archivosData = entry.getValue();
        System.out.println("----- validacion nombres -----");
        System.out.println("nombreparam: "+archivoBus);
        System.out.println("nombreIter: "+archivo);
        if (archivoBus.equals(archivo)){
            System.out.println("Archivo: " + archivo);
            for (HashMap<String, String> archivoData : archivosData) {
                for (Map.Entry<String, String> innerEntry : archivoData.entrySet()) {
                    String clave = innerEntry.getKey();
                    String valor = innerEntry.getValue();

```

```

        //validacion de la clave
        if(claveBus.equals(clave)){
            valorBus = valor;
        }
    }
}
}
return valorBus;
}

```

## TRADUCCION

Para el proceso de traduccion se realiza el mis procedimiento para obtener los analizadores, pero en este caso la gramatica del archivo jcup es mucho mas compleja y por eso solo nos enfocaremos en ella.

Para obtener el codigo fuente se debe de presionar la opcion Analizador>statpy y se ejecutara el siguiente codigo

```

private void jMenuItem4ActionPerformed(java.awt.event.ActionEvent evt) {
    // Limpiar listas par barras:
    func.Funcion.EjeX.clear();
    func.Funcion.Titulo.clear();
    func.Funcion.TituloX.clear();
    func.Funcion.TituloY.clear();
    func.Funcion.Valores.clear();
    // Limpiar Listas para Pie
    func.Funcion.EjeX_pie.clear();
    func.Funcion.Titulo_pie.clear();
    func.Funcion.Valores_pie.clear();

    //Extraer codigo fuente
    System.out.println(x: "Inicia analisis statpy!!");
    String codigoFuente = jTextArea1.getText();

    //Ejecucion de analizadore
    proyectol.Proyectol.analizarP(entrada:codigoFuente);

    System.out.println(x: "-----TTermino analisis sp -----");
    //Muestra el codigo en la otra caja de texto
    jTextArea2.setText(x: func.Funcion.codigo);
    System.out.println(x: "-----Nuestra Dtos grafica -----");
    //func.Funcion.mostrarDataBarras();
}

```

Las instrucciones del inicio son las listas que se utilizan para almacenar datos que hacen referencia a las graficas estadisticas las cuales se explicaran mas adelante. Primero se

definen los terminales necesarios y los no terminales para reconocimiento del lenguaje

```
terminal String PTCOMA, PAR_IZQ, PAR_DER, COR_IZQ, COR_DER, LLAV_IZQ, LLAV_DER, MAYOR_QUE, MENOR_QUE, COMA, PUNTO, D
terminal String IGUAL;
terminal String MAS, MENOS, POR, DIV;
terminal String ENTERO, DECIMAL, ID, CADENA;
terminal String UMENOS; // para la precedencia para los negativos y no en
terminal String RVOID, RMAIN, RDEFGLOBAL, RGRAFIBARRAS, RGRAFIPIE;
terminal String RTITULO, REJEX, RVALOR, RTITULOX, RTITULOY;
terminal String RINT, RDOUBLE, RCHAR, RSTRING, RBOOL;
terminal String RIF, RTRUE, RFALSE, RNEWVALOR, RELSE, RSWITCH, RBREAK, RCASE, RDEFAU, RFOR, RWHILE, RDOWHI
terminal String MAYOR_IGUAL, MENOR_IGUAL, COMPARADOR, DISTINTO, AND, OR, NOT;

//traduccion
non terminal inicio, lista_instruccion; // terminales para las transicio
non terminal instruccion, imprimir, asignacion, if, switch, bloquescases, case, cbreak, for, while, do_wh
non terminal declaracion, tipoDato, tipofuncion, main, defglobal, barras, gpie, condicion;
non terminal expresion;

//grafica globales
non terminal lista_instruccion_grafica;
non terminal instruccion_grafica, asignacion_grafica, tipo_dato_grafica;
non terminal expresion_grafica;

//grafica barras
non terminal lista_graf_barras;
non terminal barras_instruccion, barra_titu, arreglo, arreglo_valor, arregloD, arreglo_Dvalor;

//grafica pie
non terminal lista_graf_pie;
non terminal pie_instruccion, pie_titu, arreglo_pie, arreglo_pie_valor, arreglo_pie_D, arreglo_pie_Dvalor
```

Una vez definido esto se procede a realizar la gramatica para esto se define las expresiones mas interna es decir asignacion de variables, sentencias imprimir y las mas internar que son las expresiones logicas, aritmeticas y de comparacion.



```

expresion ::= NOT expresion: a                                {: RESULT = "not "+a.toString() ; :}
| MENOS expresion:a      {: RESULT = "-" +a.toString() ; :} %prec UMENOS
| ENTERO:a      {: RESULT = a; :}
| CADENA:a      {: RESULT = a; :}
| DECIMAL:a      {: RESULT = a; :}
| RTRUE:a      {: RESULT = a; :}
| RFALSE:a      {: RESULT = a; :}
| ID:a      {: RESULT = a; :}
| expresion:a MAS:b expresion:c  {: RESULT = a.toString()+" "+c.toString(); :}
| expresion:a MENOS:b expresion:c      {: RESULT = a.toString()+" - "+c.toString(); :}
| expresion:a POR:b expresion:c      {: RESULT = a.toString()+" * "+c.toString(); :}
| expresion:a DIV:b expresion:c      {: RESULT = a.toString()+" / "+c.toString(); :}
| expresion:a MAYOR_QUE:b expresion:c  {: RESULT = a.toString()+" > "+c.toString(); :}
| expresion:a MENOR_QUE:b expresion:c  {: RESULT = a.toString()+" < "+c.toString(); :}
| expresion:a MAYOR_IGUAL:b expresion:c  {: RESULT = a.toString()+" >= "+c.toString(); :}
| expresion:a MENOR_IGUAL:b expresion:c  {: RESULT = a.toString()+" <= "+c.toString(); :}
| expresion:a COMPARADOR:b expresion:c  {: RESULT = a.toString()+" == "+c.toString(); :}
| expresion:a DISTINTO:b expresion:c  {: RESULT = a.toString()+" != "+c.toString(); :}
| expresion:a IGUAL:b expresion:c      {: RESULT = a.toString()+" = "+c.toString(); :}
| expresion:b AND expresion:a      {: RESULT = b.toString() + "and" + a.toString() ; :}
| expresion:b OR expresion:a      {: RESULT = b.toString() + "or" + a.toString() ; :}
;

```

```

asignacion ::= tipoDato ID:a IGUAL expresion:b PTCOMA
{:
|
    RESULT = a + "=" + b+"\n";
:}
| ID:a IGUAL expresion:b PTCOMA
{:
|
    RESULT = a + "=" + b+"\n";
:}
| ID:a IGUAL expresion:b
{:
|
    RESULT = a + "=" + b;
:}
| tipoDato ID:a PTCOMA
{:
|
    RESULT = a+"\n";
:}
;

tipoDato ::= RINT
| RDOUBLE
| RCHAR
| RSTRING
| RBOOL
;

```

```

imprimir ::= RCONSOLA PUNTO RWRITE PAR_IZQ expresion:a PAR_DER PTCOMA
{:
    RESULT = "print(" + a + ")\n";
:}
;

```

Para poder reconocer varias instrucciones se utilizó esta estructura en donde se definen las posibles sentencias que pueden venir y las cuales pueden venir cero o muchas veces

```

//-----> PARA TRADUCCION <-----
lista_instruccion ::= lista_instruccion:a instruccion:b      (: RESULT = a.toString() + b.toString(); :)
|instruccion: a      (: RESULT = a.toString(); :)
;

instruccion ::= void:a      (: RESULT = a; :)
|imprimir:a      (: RESULT = a; :)
|asignacion:a      (: RESULT = a; :)
|if:a      (: RESULT = a; :)
|switch:a      (: RESULT = a; :)
|for:a      (: RESULT = a; :)
|while:a      (: RESULT = a; :)
|do_while:a      (: RESULT = a; :)
;

```

Con esto se asegura que pueda venir cualquier cantidad de estructuras anidadas. Y para todo esto la palabra RESULT se encarga de retornar los valores obtenidos desde el más interno del árbol hasta el más externo.

## TABULACION

Para traducir el código a python son muy importantes las tabulaciones y para eso se utilizó el siguiente código

```

if ::= RIF PAR_IZQ expresion:a PAR_DER LLAV_IZQ lista_instruccion:b LLAV_DER
{:
    String resultado = "";
    resultado = "if "+a+":\n";

    //arreglo
    String[] texto = b.toString().split("\n");
    for (String linea : texto){
        linea = "\t"+linea;
        resultado += linea+"\n";
    }
    RESULT = resultado;
:}

```

Se define este una variable string para almacenar todo el bloque de instrucciones analizadas en ese momento y se concatena con el código traducido a python posteriormente se crea un arreglo donde se separe la cadena con un \n y así tener el código separado por líneas, luego con un ciclo for se le agrega una tabulación y se concatena con la variable string, pero como todo eso es una gran cadena de resultados al momento de estar el if dentro de otra estructura, al momento de subir a esa estructura se podrá separar todo el resultado del if y el if incluidos en líneas y aquí también agregar una tabulación y con esto tener una representación exacta de código python

---

## GRAFICAS

Para los datos de las gráficas estadísticas también son extraídas de la gramática con la siguiente estructura

```
|DOLLAR LLAV_IZQ RNEWVALOR COMA CADENA:a COMA CADENA:b LLAV_DER
{
    String archivo = a.toString();
    String newArchi = archivo.substring(1);
    newArchi = newArchi.substring(0, newArchi.length() - 1);

    String clave = b.toString();

    RESULT = func.Funcion.buscaEstruc(newArchi, clave);
    :}
;
```

la palabra result retorna el valor que se haya encontrado en la estructura que almacena los datos de json y estos datos son almacenados en listas para poder poder graficar de una forma más sencillas, las listas utilizadas son las siguientes

```
//HashMap para GRAFICAAAAAA pie -----<<>>
public static Map<String, List<String>> Hash_pie = new HashMap<>();

//Lista para grafica de barras
public static List<String> Titulo_pie = new ArrayList<>();
public static List<String> EjeX_pie = new ArrayList<>();
public static List<String> Valores_pie = new ArrayList<>();
```

```
//HashMap para GRAFICAAAAA BARRAS -----<<>>
public static Map<String, List<String>> Hash_barras = new HashMap<>();

//Lista para grafica de barras
public static List<String> Titulo = new ArrayList<>();
public static List<String> EjeX = new ArrayList<>();
public static List<String> Valores = new ArrayList<>();
public static List<String> TituloX = new ArrayList<>();
public static List<String> TituloY = new ArrayList<>();
```

luego se utilizaron varios metodos para retornar valores especificos de estas listas los cuales se detallaron bastante bien con comentarios dentro del codigo.