Instituto Tecnológico de Costa Rica

Escuela de Computación

Ingeniería en computación

Compiladores e Intérpretes

Grupo 2

Profesor: Erika Marín

Análisis Sintáctico

Adrián López Quesada, 2014081634

Josué Arrieta Salas, 2014008153

Cartago

Miércoles 18 de junio

**Índice**

[Análisis de resultados 3](#_Toc453140567)

[Casos de pruebas 4](#_Toc453140568)

[Prueba 1: Funciones 4](#_Toc453140569)

[Prueba 1.1 – Primeras pruebas con Funciones 4](#_Toc453140570)

[Prueba 1.2 – Segundas pruebas con Funciones 4](#_Toc453140571)

[Prueba 2: Expresiones 5](#_Toc453140572)

[Prueba 2.1 – Primeras pruebas con Expresiones 5](#_Toc453140573)

[Prueba 2.1 – Segundas pruebas con Expresiones 6](#_Toc453140574)

[Prueba 3: Declaración de variables 7](#_Toc453140575)

[Prueba 4: Asignaciones 9](#_Toc453140576)

[Prueba 5: Print e Input 9](#_Toc453140577)

[Prueba 6: While 10](#_Toc453140578)

[Prueba numero 7: If 11](#_Toc453140579)

[Prueba numero 8: For (Adrián) 12](#_Toc453140580)

[Prueba numero 9: Try 13](#_Toc453140581)

[Prueba numero 10: Break - Continue (Adrián) 14](#_Toc453140582)

[Prueba numero 11: OOP 14](#_Toc453140583)

[Prueba 11.1 – Estructura Básica 14](#_Toc453140584)

[Prueba 11.2 – El ; 15](#_Toc453140585)

[Gramática 17](#_Toc453140586)

[¿Cómo compilar y correr el Parser? 23](#_Toc453140587)

# Análisis de resultados

|  |  |
| --- | --- |
| **Objetivo** | **Porcentaje de éxito (100%)** |
| Programa recibe código fuente escrito en Python y analiza el archivo. | 100% |
| Se listan errores léxicos encontrados (por línea y el error). | 100%: objetivo del proyecto pasado. |
| El programa ante un error léxico se recupera de este y no despliega errores en cascada. | 100%: objetivo del proyecto pasado. |
| Se despliega lista de errores sintácticos: despliega línea, columna, y el token incorrecto. | 100% |
| Se manejan mensajes específicos para reportar errores sintácticos. | 100% |
| El programa ante un error sintáctico se recupera de este y no despliega errores en cascada. | 100%: se recupera en gran medida; sin embargo se ha de mencionar que en algunos casos es simplemente imposible recuperarse del error y continuar parseando. |
| Se parsean correctamente declaraciones de funciones. | 100% |
| Se parsean correctamente declaraciones de variables de tipo: int, float, list, string, boolean y char. | 100% |
| Se procesa correctamente la estructura de un programa funcional. | 100% |
| Se parsea correctamente la estructura de un programa orientado a objetos. | 100%: se toma en cuenta además, que debe venir al menos un atributo y un método para este tipo de archivos. |
| Se parsean correctamente todo tipo de expresiones: aritméticas y booleanas. Una llamada a función también se considera una expresión. | 100%: también se considera que las asignaciones son expresiones. |
| Se parsean correctamente todo tipo de asignaciones. También son válidas la creaciones de clases. | 100% |
| Se define la estructura de la función print e input. | 100% |
| Se parsean correctamente las estructuras de control: if, while y for. | 100% |
| En un bloque se pueden usar las sentencias break y continue. | 100% |
| Se parsea correctamente la estructura de control de errores: Try-Except-Finally. | 100% |

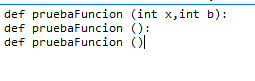
# Casos de pruebas

## Prueba 1: Funciones

### Prueba 1.1 – Primeras pruebas con Funciones

Pruebas simples de funciones, todavía no estaban claras todas las reglas que debían tener las funciones. No se tenía tanto manejo de número de líneas y errores.

Objetivo: Probar reglas básicas de las funciones del lenguaje MyPython.



Resultado:



Ya que no se habían definido “x” cantidad de funciones

Al definirlo para “x” cantidad de funciones:



Porque falta el ultimo “:”

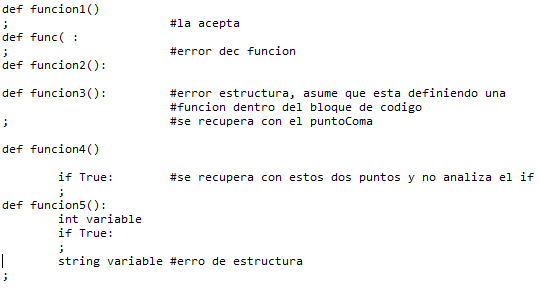
Al arreglar ese error en la prueba:



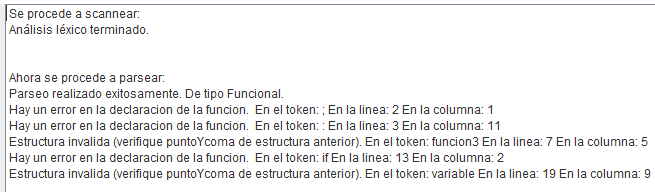
### Prueba 1.2 – Segundas pruebas con Funciones

Objetivo: realizar aún más tipos de prueba con funciones. En este caso se prueba fallos en la declaración de la función, falta de dos puntos o puntos y comas, y la estructura que debe mantener la función.

Se tiene el siguiente archivo de entrada:



Guiándose con los comentarios del archivo se esperan los siguientes resultados (Correctos):



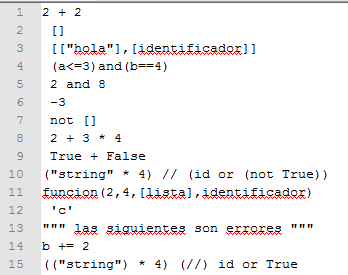
## Prueba 2: Expresiones

### Prueba 2.1 – Primeras pruebas con Expresiones

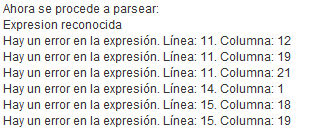
Esta prueba fue realizada en el transcurso de la realización del proyecto, y es considerada precisamente por esa razón. En el momento que se realizó las pruebas, una asignación no era considerada una expresión (ahora sí lo es) para que se tome en cuenta.

Objetivo: probar todo tipo de expresiones en el lenguaje MyPython.

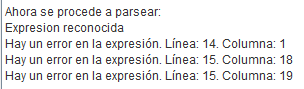
Se tiene el siguiente documento de texto de entrada:



Se espera errores en la línea 14 y en la línea 15, sin embargo las demás son expresiones válidas. Se tienen los siguientes resultados (incorrectos):

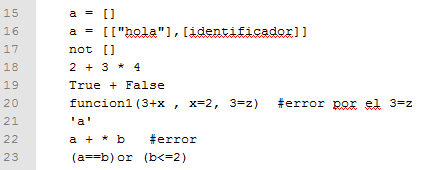


Se muestra que hay un error en la línea 11 debido a que las funciones no se contemplaron como expresiones. Se aplica un arreglo y al correr el mismo archivo se tienen los siguientes resultados (correctos):

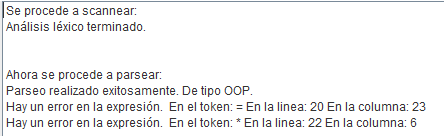


### Prueba 2.1 – Segundas pruebas con Expresiones

Objetivo: realizar aún más tipos de prueba con expresiones. En este caso las asignaciones si son tomadas en cuenta como expresiones. Se tiene el siguiente archivo de entrada:

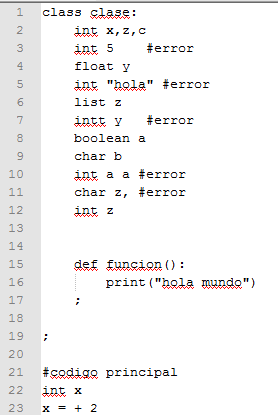


Resultados esperados: Se espera que haya errores en las líneas 22 y 20. Se obtuvieron los siguientes resultados (correctos):

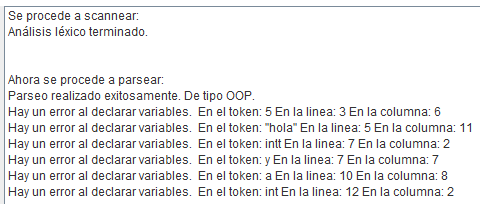


## Prueba 3: Declaración de variables

Objetivo: probar todo tipo de declaración de variables en el lenguaje MyPython. Se tiene el siguiente código de entrada:



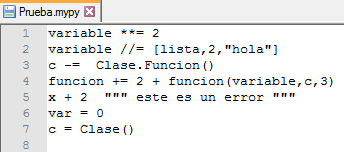
Resultados esperados: Se espera error en las líneas 3, 5, 7, 10 y 11. Se obtuvieron los siguientes resultados (correctos):



El error de la línea 11 se muestra en la línea 12 debido a la naturaleza del error y el error surge al encontrar “int” en la línea 12 y no un identificador. Este mensaje es considerado correcto.

## Prueba 4: Asignaciones

Objetivo: probar diferentes tipos de asignaciones posibles en el lenguaje MyPython. Se tiene el siguiente código de entrada:

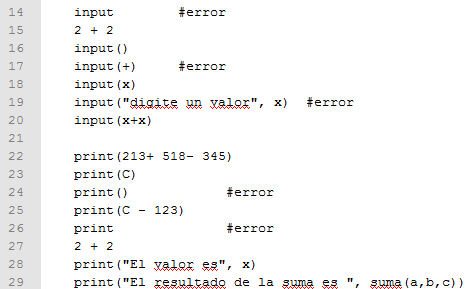


Resultados esperados: Se espera un error en la línea 5 ya que “x + 2” no es una asignación sino una expresión, sin embargo se espera que el compilador se recupere de este error. Se obtuvieron los siguientes resultados (correctos):

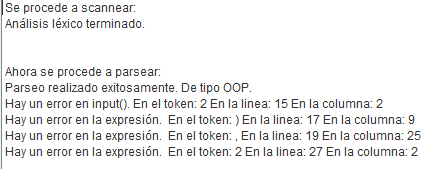


## Prueba 5: Print e Input

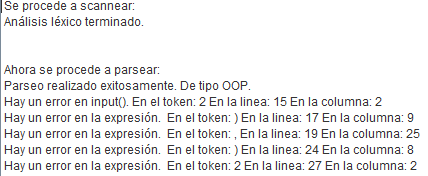
Objetivo: probar la estructura de las funciones print() e input() predefinidas. Se tiene el siguiente código de entrada:



Se espera que haya errores en las líneas 14, 17, 19,24 y 26. También se espera que se recupere del error y siga parseando. Se obtuvieron los siguientes resultados (incorrectos):



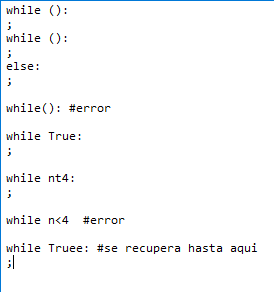
Como se ha mencionado, algunos errores los muestra en la línea después (en el caso del erro de la línea 14 y 26). Sin embargo el error de la línea 24 no se mostró. Print() es considerado correcto. Se aplica una corrección y se vuelve a correr el mismo archivo, y se obtienen los siguientes resultados (correctos):



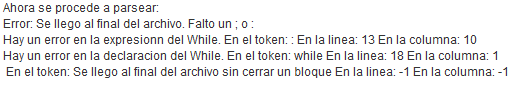
## Prueba 6: While

Objetivo: Probar todas las funcionalidades del While y su recuperacion de errores.

Se tiene el siguiente codigo de entrada:



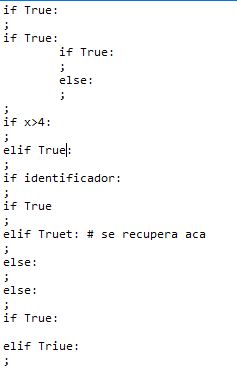
Se obtuvieron los siguientes resultados (Correctos):



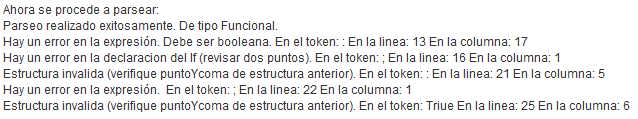
Los resultados son correctos, pero cabe destacar algunas observaciones. Se llego la captura de emergencia que se tiene por falta de un puntoYComa, esto es debido a que el while que le falta el puntoYComa tomo a todos los demás whiles como si fueran parte de su bloque, por lo tanto no se pudo recuperar con otro puntoYComa. Ademas se puede observar cómo no expreso error en el último while con el True, esto es correcto ya que el while anterior carece de los dos puntos, por lo que ignora todo hasta llegar a los siguiente dos puntos.

## Prueba numero 7: If

Objetivo: Probar la estructura del if, elif y else, ifs anidados, ifs con y sin else y elifs, recuperación de puntos y comas, dos puntos, mal ubicación de estructuras else elif y expresiones no booleanas.



Resultados obtenidos (Correctos):

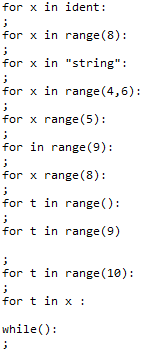


Se puede apreciar como las primeras estructuras las acepta sin ningún problema. Los demás errores son claros de porque suceden, dándose la recuperación donde los comentarios especifican. El error de la línea 22 se debe a que no acepta el else sin un if por lo tanto se recupera en los dos puntos y queda el punto y coma para analizarlo como expresión. Además no muestra el error de True ya que por falta de punto y coma del último if, anula la expresión elif ya que no la acepta sin tener un if antes; se recupera en los dos puntos y utiliza el punto y coma del elif para terminar el parseo.

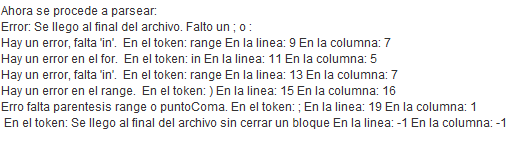
## Prueba numero 8: For (Adrián)

Objetivo: probar la estructura del for, todas sus posibilidades, control de errores y el uso de Range e In.

Se usó el siguiente trozo de código:



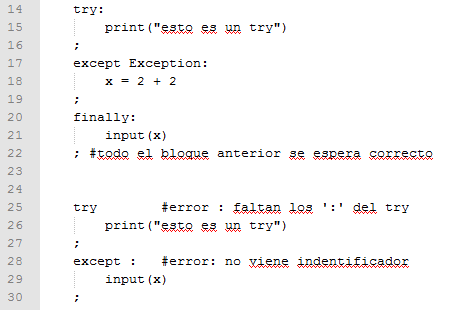
Se obtuvieron los siguientes resultados (Correctos):



Se puede apreciar como acepta todos los tipos de for y usos del range. Respecto a la parte de los errores, los identifica bien, pero es necesario aclararlos. En la línea 19 tira error, pero debido a la gramática es bastante complicado analizar donde si es por falta de paréntesis o dos puntos del for, por lo tanto, especifica al usuario que verifique ambos casos. Además, hay un error de EOF, debido al for de la línea 22, el cual no tiene punto y coma. El compilador ubica al while dentro del for y cierra el while pero falta cerrar el for.

## Prueba numero 9: Try

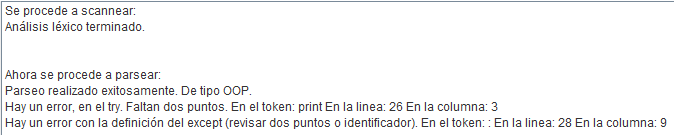
Objetivo: probar la estructura para el control de errores Try-Except en MyPython. Se tiene el siguiente archivo de entrada:



Resultados esperados:

* Error en la línea 25 ya que faltan los ‘:’ del try. Debe tirarlo en la línea 26 ya que ahí está el token que posee el error.
* Error en la línea 28 ya que falta un identificador en el except.

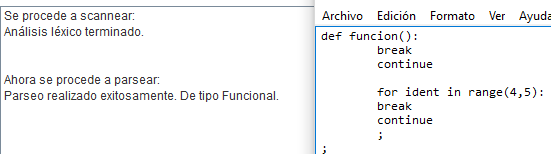
Resultados obtenidos (correctos):



## Prueba numero 10: Break - Continue (Adrián)

Objetivo: Probar el uso del break y continue.

Debido a ser una prueba sencilla se presentan los resultados y el archivo en uno solo:

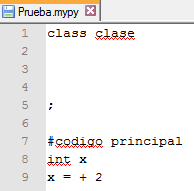


No tiro ningún error, esto esta correcto ya que el análisis del break y el continue se va a desarrollar dentro del analizador semántico. En la parte del sintáctico puede aparecer en cualquier parte de un bloque de código.

## Prueba numero 11: OOP

### Prueba 11.1 – Estructura Básica

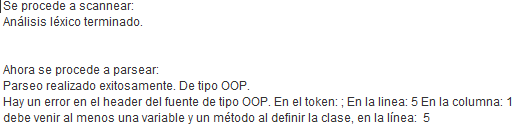
Objetivo: probar la estructura básica de la clase de un archivo MyPython. Se tiene el siguiente código de entrada:



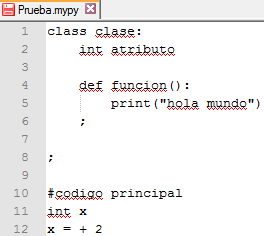
Resultados esperados: se espera que surjan los errores:

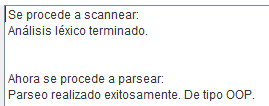
* Faltan los ‘:’ en el header de la clase.
* Se queje de que no vengan variables ni métodos (tienen que venir al menos 1):

Resultados obtenidos (correctos):



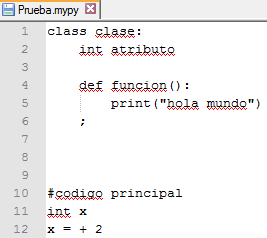
Si se añade el siguiente archivo (con los errores corregidos), no hay errores sintácticos:





### Prueba 11.2 – El ;

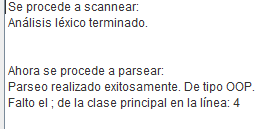
Objetivo: probar cuando no viene el ; al definir una clase. Se tiene el siguiente archivo:



Resultados esperados: error sintáctico al no cerrar la clase con el ;. Resultados obtenidos (incorrectos) :



Se aplica un parche que arregle el error y al correr el mismo archivo se obtiene (resultados correctos):



# Gramática

**start with Programa**

**Programa::=** Funcional | OOP | error

**---------------------------------------------------------Funcional--------------------------------------------------------**

**Funcional ::=** DefFunciones CodigoPrincipal

**DefFunciones ::=** DefFuncion DefFunciones |

**DefFuncion ::=** def identificador parenAbierto DefParametrosFact dosPuntos Variables CualquierCosas puntoComa

| def identificador parenAbierto DefParametrosFact dosPuntos Variables CualquierCosas errPuntoYComa

| def errDecFuncion dosPuntos Variables CualquierCosas puntoComa

| def identificador parenAbierto DefParametrosFact errDecFuncion dosPuntos

| def errDecFuncion puntoComa

| def parenAbierto errDecFuncion puntoComa

| def errPuntoYComa DefFuncion

| errDecFuncion DefFuncion

| errDecFuncion puntoComa

**DefParametrosFact ::=** Var identificador DefParametros | var errParametro1 DefParametros| identificador errParametro1 DefParametros | parenCerrado

**DefParametros::=** coma Var identificador DefParametros | parenCerrado | errParametro1

**---------------------------------------------------------OOP--------------------------------------------------------------**

**VariablesOOP ::=** Variable Variables;

**FuncionesOOP ::=** DefFunciones2 puntoComa CodigoPrincipal

| DefFunciones2:c {: usado para definir un error:} CodigoPrincipal

| puntoComa:c {: usado para definir un error:} CodigoPrincipal

**DefFunciones2 ::=** DefFuncion DefFunciones2 | DefFuncion

**FuncionesOOP2 ::=** DefFunciones puntoComa CodigoPrincipal

| DefFunciones:c {: usado para definir un error :} CodigoPrincipal

**OOP ::=** HeaderOOP

**BloqueOOP::=** VariablesOOP FuncionesOOP

| DefFuncion:c {: usado para definir un error :} FuncionesOOP2

| puntoComa:c {: usado para definir un error:} CodigoPrincipal

**HeaderOOP ::=** clas identificador dosPuntos BloqueOOP| clas errHeaderOOP BloqueOOP

**--------------------------------------------------------Bloques-----------------------------------------------------------**

**If ::=** iif ExpresionB dosPuntos BloquePuntoComa Elif

| iif ExpresionB error dosPuntos BloquePuntoComa Elif

| iif errExpresionB dosPuntos BloquePuntoComa Elif

**Elif ::=** elif ExpresionB dosPuntos BloquePuntoComa Elif

| elif error dosPuntos BloquePuntoComa Elif | Else

**Else::=** eelse dosPuntos BloquePuntoComa | eelse errElse dosPuntos BloquePuntoComa|

**ForFact ::=** ffor identificador in For dosPuntos BloquePuntoComa

| ffor errFor in For dosPuntos BloquePuntoComa

| ffor identificador in For errFor dosPuntos BloquePuntoComa

| ffor identificador errIn For dosPuntos BloquePuntoComa

**For ::=** STRING | identificador | Range | errFor

**Range ::=** range parenAbierto INT coma INT parenCerrado

| range parenAbierto INT parenCerrado | range errRange parenCerrado

| range parenAbierto INT error | range parenAbierto INT coma INT error| range errFor

**While ::=** wwhile ExpresionB dosPuntos BloquePuntoComa Else

| wwhile parenAbierto parenCerrado dosPuntos BloquePuntoComa Else

| wwhile error:e dosPuntos BloquePuntoComa Else

| wwhile ExpresionB error:e dosPuntos BloquePuntoComa Else

**Try::=** ttry dosPuntos CualquierCosas puntoComa Except

| ttry errTry BloquePuntoComa Except

| ttry dosPuntos CualquierCosas errExcept puntoComa

**Except::=** eexcept identificador dosPuntos BloquePuntoComa Finally

| eexcept error dosPuntos BloquePuntoComa Finally | errExcept2

**Finally::=** ffinally dosPuntos BloquePuntoComa

| ffinally errFinally dosPuntos BloquePuntoComa |

**BloquePuntoComa ::=** CualquierCosas puntoComa | errPuntoYComa puntoComa

**--------------------------------------------------------Expresiones-------------------------------------------------------**

**Funcion::=** identificador parenAbierto Parametros| identificador errParen parenCerrado

**Parametros ::=** Expresion Parametro | parenCerrado |errParen

**Parametro ::=** coma Expresion Parametro | parenCerrado | errParametro parenCerrado

**Literal ::=** INT|FLOAT|CHAR|STRING

**Lista::=** cuadradoAbierto ElementosLista cuadradoCerrado

**ElementosLista::=** Expresion MasElementosLista|

**MasElementosLista::=** coma Expresion MasElementosLista |

**Expresion ::=** Literal|Lista|identificador | parenAbierto Expresion parenCerrado | errExpresion

| parenAbierto errParen

| errParen parenCerrado;

**Expresion ::=** Expresion OpBinario Expresion

**Expresion ::=** OpUnario Expresion

**Expresion ::=** identificador punto Funcion

**Expresion ::=** ExpresionB

**Expresion ::=** FuncionPredefinida

**ExpresionB ::=** ffalse | ttrue| Funcion |opNot Expresion| Expresion OpBinarioB Expresion | parenAbierto ExpresionB parenCerrado

**Expresion ::=** identificador opAsignaciones Expresion

**Expresiones ::=** Expresiones Expresion |

**FuncionPredefinida::=** Input | Print | IntFuncion

**IntFuncion ::=** intReservado parenAbierto FuncionPredefinida parenCerrado

| intReservado parenAbierto Funcion parenCerrado

|intReservado parenAbierto STRING parenCerrado | errInput

**Input ::=** iinput parenAbierto Expresion parenCerrado | iinput parenAbierto parenCerrado

| iinput errInput | errInput

**Print ::=** pprint parenAbierto ElementosLista2 parenCerrado | errPrint

| pprint parenAbierto errPrint parenCerrado

**ElementosLista2::=** Expresion MasElementosLista2

**MasElementosLista2::=** coma Expresion MasElementosLista2|

**OpAritmeticos ::=** opSuma | opResta | opMultiplicacion | opDivisionE | opDivision | opModulo | opPotencia

**OpUnario ::=** opSuma | opResta

**OpBinario ::=** OpAritmeticos

**OpBinarioB ::=** opComparadores | OpLogicos

**OpLogicos ::=** opAnd | opOr | opNot

**--------------------------------------------------------General-------------------------------------------------------**

**CualquierCosa::=** Expresion |If | ForFact | While | Try | bbreak|ccontinue

| def errMalUbicacion dosPuntos | eexcept errMalUbicacion dosPuntos

| var errMalUbicacion | clas errMalUbicacion puntoComa

| ffinally errMalUbicacion dosPuntos

| eelse errMalUbicacion dosPuntos

| elif errMalUbicacion dosPuntos;

**CodigoPrincipal::=** Variables CualquierCosas

**CualquierCosas::=** CualquierCosas CualquierCosa |

**Var ::=** var | intReservado

**Variable::=** Var DefVariablesFact | Var errDecVariables | errDecVariables identificador

**Variables ::=** Variables Variable|

**DefVariablesFact ::=** identificador DefVariables | errDecVariables DefVariables

**DefVariables::=** coma identificador DefVariables | errDecVariables identificador DefVariables | errDecVariables DefVariables |

**--------------------------------------------------------Errores-------------------------------------------------------**

**errHeaderOOP::=** error

**errDecVariables::=** error

**errPuntoYComa::=** error

**errExpresion::=** error

**errExpresionB::=** error

**errInput::=** error

**errPrint::=** error

**errDecFuncion::=** error

**errParametro1::=** error

**errParametro::=** error

**errRange::=** error

**errIn::=** error

**errFor::=** error

**errElse::=** error

**errDosPuntos ::=** error

**errTry ::=** error

**errExcept ::=** error

**errExcept2 ::=** error

**errFinally ::=** error

**errParen ::=** error

**errMalUbicacion ::=** error

# ¿Cómo compilar y correr el Parser?

La parte esencial para compilar el proyecto es agregar el archivo “jflex-1.6.1.jar” y “java-cup-11b.jar” como JAR a las librerías que tiene acceso. De esta manera se pueden utilizar todas las funcionalidades del Jflex y JCup. En caso de duda, por favor seguir las instrucciones del video siguiente (referente a JFlex): https://www.youtube.com/watch?v=w-KfjJdRas8 cerca del minuto 10, tomando en cuenta que se importa como JAR, no como librería. El proceso para JCUP es el mismo que en el video. Otra manera es cargar el proyecto a Netbeans el cual ya está ya tiene los JAR’s agregado. Además, se necesita que el archivo flex.lexer y parser.cup esté presente en el src y aunque no es necesario para compilar se recomienda utilizar la ubicación actual del archivo de pruebas.

Debido a problemas para él envió se deben hacer modificaciones a 3 archivos, a todos se les debe quitar la “a” al final de la extensión .jar. Los archivos son: ejecutable/EjecutableScanner.jara , ejecutable/lib/jflex-1.6.1.jara , y CodigoFuente/Lexer/jflex-1.6.1.jara (Modificar paths antes de enviar, me avisa cuando esté listo todo)

Para correr el scanner se puede utilizar el .jar “Scanner ejecutable” para hacer las pruebas con el archivo “Pruebas.MYPY” que se encuentra dentro de la carpeta ejecutable.