Instituto Tecnológico y de Estudios Superiores de Monterrey

Propuesta Inicial del Compilador

Diseño de Compiladores
Profesora Elda Quiroga / Hector Ceballos

Enrique Marroquín Artezán A01191578
Roel Castaño Moreno A01191163

18 de Febrero del 2017

Indice

1. Misión y Visión del Proyecto

La misión de este lenguaje es que tenga una curva de aprendizaje baja y que al mismo tiempo ofrezca características llamativas para audiencias artísticas. Lograremos esto al mantener una sintaxis clara y descriptiva, que pueda ser fácil de usar para los usuarios finales. Al mismo tiempo, estaremos desarrollando una documentación que describa el lenguaje de manera completa y concisa, y que muestre todas las capacidades y características que tenga el lenguaje.

La visión de integrar a individuos a la comunidad de la programación que presentan capacidades de aprendizaje visual. Así mismo, facilitar el proceso de aprendizaje de conceptos de la programación con el apoyo de representaciones gráficas.

2. Objetivo Principal

El objetivo principal de este lenguaje es incentivar el uso de la programación a personas sin experiencia previa por medio del uso de representaciones visuales de los resultados de sus programas. Al tener una salida visual, consideramos que el lenguaje cae dentro del área de lenguajes gráficos.

3. Requerimientos del Proyecto

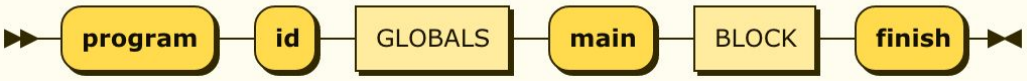
a. Elementos Básicos

int	dec	string	yesno	yes	no
point	triangle	circle	rectangle	canvas	center
width	height	with	add	paint	print
for	each	in	if	elsif	else
color					

b. Diagramas de Sintaxis

PROGRAM

PROGRAM:

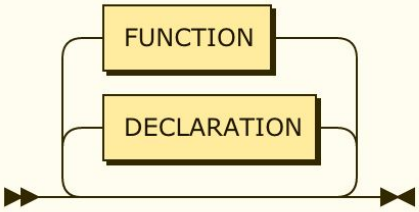



```
PROGRAM  ::= 'program' 'id' GLOBALS 'main' BLOCK 'finish'
```

no references

GLOBALS

GLOBALS:



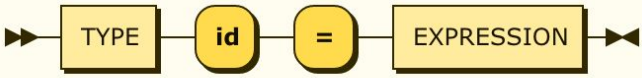

```
GLOBALS  ::= ( DECLARATION | FUNCTION )*
```

referenced by:

- PROGRAM

VAR

VAR:



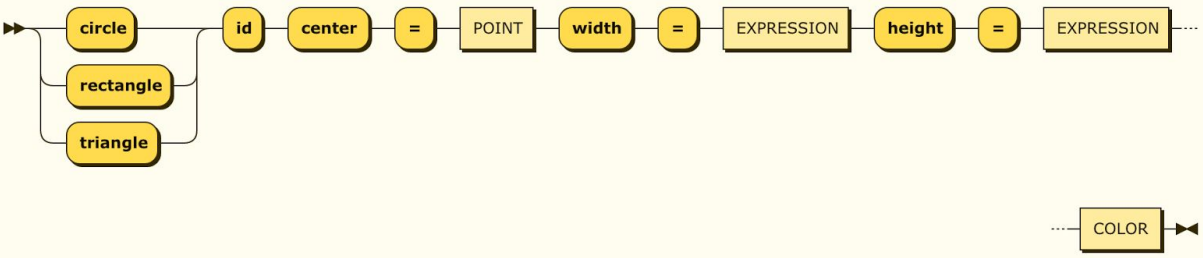

```
VAR      ::= TYPE 'id' '=' EXPRESSION
```

referenced by:

- DECLARATION

SHAPE

SHAPE:




```
SHAPE    ::= ( 'circle' | 'rectangle' | 'triangle' ) 'id' 'center' '=' POINT 'width' '=' EXPRESSION 'height' '=' EXPRESSION COLOR
```

referenced by:

- DECLARATION

BLOCK

BLOCK:

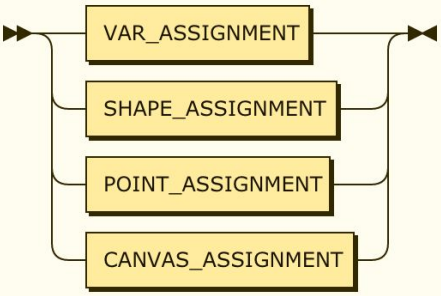

```
BLOCK    ::= '[' STATEMENT ']'
```

referenced by:

- CONDITIONAL
- FUNCTION
- LOOP
- PROGRAM

ASSIGNMENT

ASSIGNMENT:



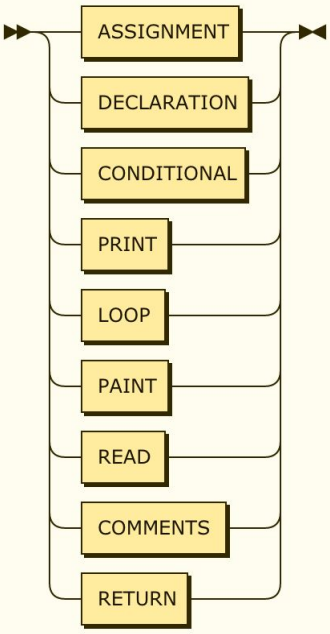

```
ASSIGNMENT
         ::= VAR_ASSIGNMENT
           | SHAPE_ASSIGNMENT
           | POINT_ASSIGNMENT
           | CANVAS_ASSIGNMENT
```

referenced by:

- STATEMENT

STATEMENT

STATEMENT:



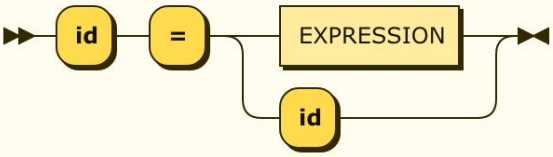

```
STATEMENT
          ::= ASSIGNMENT
            | DECLARATION
            | CONDITIONAL
            | PRINT
            | LOOP
            | PAINT
            | READ
            | COMMENTS
            | RETURN
```

referenced by:

- BLOCK

VAR_ASSIGNMENT

VAR_ASSIGNMENT:



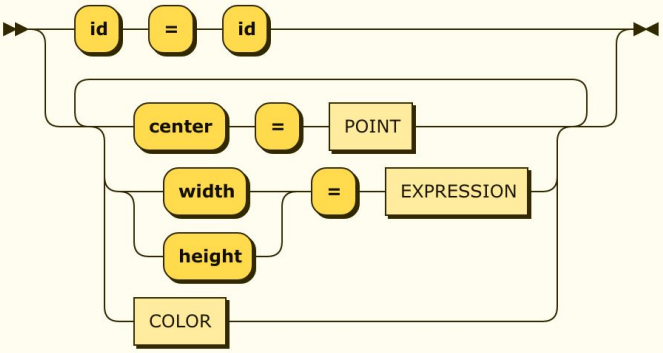

```
VAR_ASSIGNMENT
          ::= 'id' '=' ( EXPRESSION | 'id' )
```

referenced by:

- ASSIGNMENT

SHAPE_ASSIGNMENT

SHAPE_ASSIGNMENT:



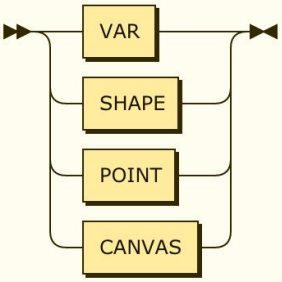

```
SHAPE_ASSIGNMENT
         ::= 'id' '=' 'id'
           | ( 'center' '=' POINT | ( 'width' | 'height' ) '=' EXPRESSION | COLOR )+
```

referenced by:

- ASSIGNMENT

DECLARATION

DECLARATION:




```
DECLARATION
          ::= VAR
            | SHAPE
            | POINT
            | CANVAS
```

referenced by:

- GLOBALS
- STATEMENT

POINT

POINT:

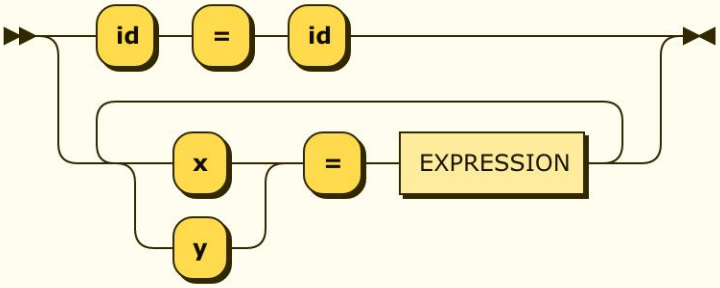

```
POINT    ::= 'point' 'id' 'x' '=' EXPRESSION 'y' '=' EXPRESSION
```

referenced by:

- DECLARATION
- SHAPE
- SHAPE_ASSIGNMENT

POINT_ASSIGNMENT

POINT_ASSIGNMENT:




```
POINT_ASSIGNMENT
         ::= 'id' '=' 'id'
           | ( ( 'x' | 'y' ) '=' EXPRESSION )+
```

referenced by:

- ASSIGNMENT

CANVAS

CANVAS:


```
CANVAS   ::= 'canvas' 'id' 'width' '=' EXPRESSION 'height' '=' EXPRESSION COLOR
```

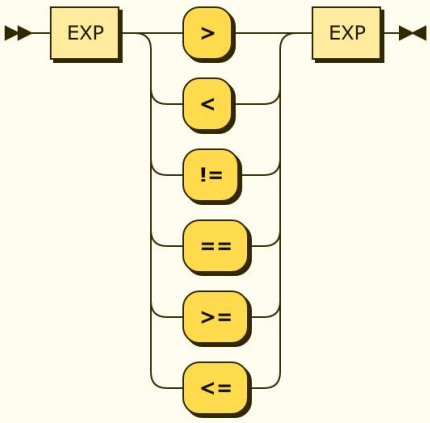
referenced by:

- DECLARATION

CANVAS_ASSIGNMENT

EXPRESSION

EXPRESSION:



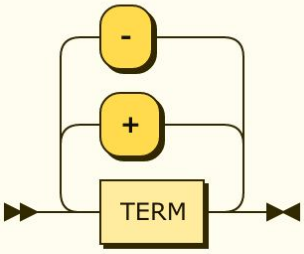

```
EXPRESSION
        ::= EXP ( '>' | '<' | '!=' | '==' | '>=' | '<=' ) EXP
```

referenced by:

- CANVAS
- CANVAS_ASSIGNMENT
- COLOR
- CONDITIONAL
- FACTOR
- POINT
- POINT_ASSIGNMENT
- PRINT
- RETURN
- SHAPE
- SHAPE_ASSIGNMENT
- VAR
- VAR_ASSIGNMENT

EXP

EXP:



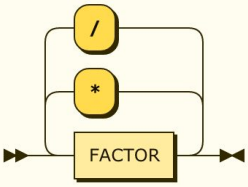

```
EXP       ::= TERM ( ( '+' | '-' ) TERM )*
```

referenced by:

- EXPRESSION

TERM

TERM:



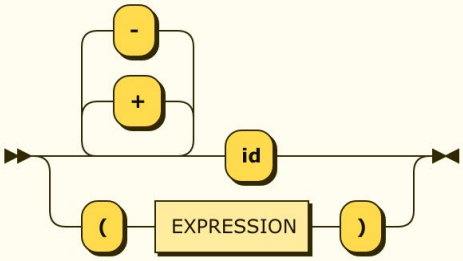

```
TERM     ::= FACTOR ( ( '*' | '/' ) FACTOR )*
```

referenced by:

- EXP

FACTOR

FACTOR:



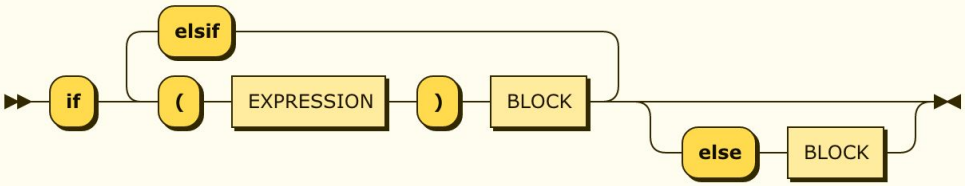

```
FACTOR   ::= '(' EXPRESSION ')'
           | ( '+' | '-' )* 'id'
```

referenced by:

- TERM

CONDITIONAL

CONDITIONAL:



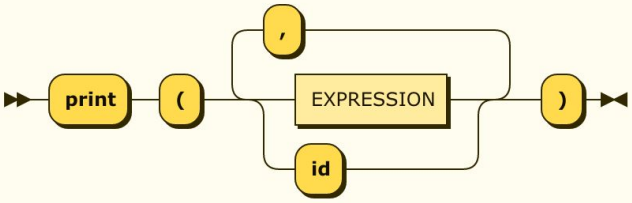

```
CONDITIONAL
         ::= 'if' '(' EXPRESSION ')' BLOCK ( 'elsif' '(' EXPRESSION ')' BLOCK )* ( 'else' BLOCK )?
```

referenced by:

- STATEMENT

PRINT

PRINT:



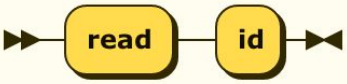

```
PRINT    ::= 'print' '(' ( EXPRESSION | 'id' ) ( ',' ( EXPRESSION | 'id' ) )* ')'
```

referenced by:

- STATEMENT

READ

READ:



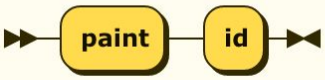

```
READ     ::= 'read' 'id'
```

referenced by:

- STATEMENT

PAINT

PAINT:



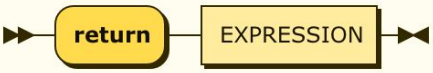

```
PAINT    ::= 'paint' 'id'
```

referenced by:

- STATEMENT

RETURN

RETURN:



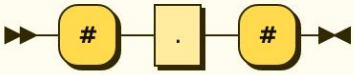

```
RETURN   ::= 'return' EXPRESSION
```

referenced by:

- STATEMENT

COMMENTS

COMMENTS:



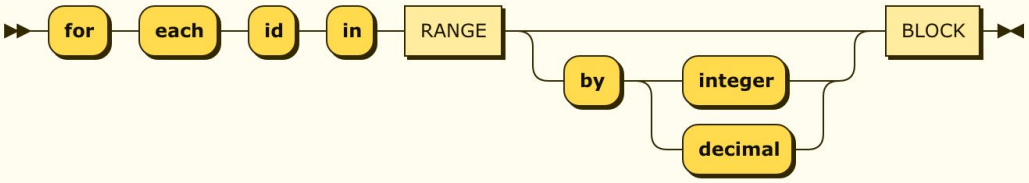

```
COMMENTS ::= '#' . '#'
```

referenced by:

- STATEMENT

LOOP

LOOP:



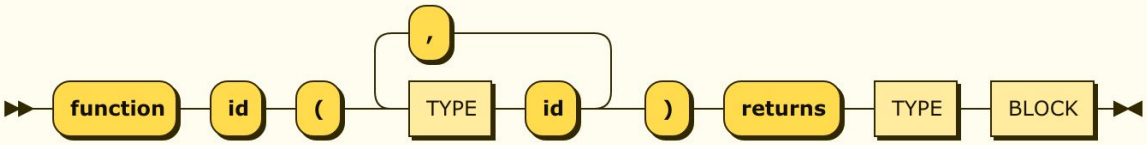

```
LOOP     ::= 'for' 'each' 'id' 'in' RANGE ( 'by' ( 'integer' | 'decimal' ) )? BLOCK
```

referenced by:

- STATEMENT

FUNCTION

FUNCTION:



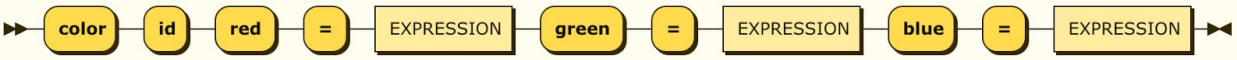

```
FUNCTION ::= 'function' 'id' '(' TYPE 'id' ( ',' TYPE 'id' )* ')' 'returns' TYPE BLOCK
```

referenced by:

- GLOBALS

COLOR

COLOR:




```
COLOR    ::= 'color' 'id' 'red' '=' EXPRESSION 'green' '=' EXPRESSION 'blue' '=' EXPRESSION
```

referenced by:

- CANVAS
- SHAPE
- SHAPE_ASSIGNMENT

Expresiones regulares
PROGRAM::= 'program' 'id' GLOBALS 'main' BLOCK 'finish' GLOBALS::= (DECLARATION | FUNCTION)* VAR::= TYPE 'id' '=' EXPRESSION SHAPE::= ('circle' | 'rectangle' | 'triangle') 'id' 'center' '=' POINT 'width' '=' EXPRESSION 'height' '=' EXPRESSION COLOR BLOCK::= '[' STATEMENT ']' STATEMENT::= (ASSIGNMENT | DECLARATION | CONDITIONAL | PRINT | LOOP | PAINT | READ | COMMENTS | RETURN) ASSIGNMENT::= (VAR_ASSIGNMENT | SHAPE_ASSIGNMENT | POINT_ASSIGNMENT | CANVAS_ASSIGNMENT) VAR_ASSIGNMENT::= 'id' '=' (EXPRESSION | 'id') SHAPE_ASSIGNMENT::= 'id' '=' 'id' | (('center' '=' POINT) | ('width' '=' EXPRESSION) | ('height' '=' EXPRESSION) | COLOR)+

```
DECLARATION::= VAR | SHAPE | POINT | CANVAS
POINT::= 'point' 'id' 'x' '=' EXPRESSION 'y' '=' EXPRESSION
POINT_ASSIGNMENT::= 'id' '=' 'id' | (('x' | 'y') '=' EXPRESSION)+
CANVAS::= 'canvas' 'id' 'width' '=' EXPRESSION 'height' '=' EXPRESSION COLOR
CANVAS_ASSIGNMENT::= 'id' ('add' | '=') 'id'  | (('width' '=' EXPRESSION) |
('height' '=' EXPRESSION) | ('color' '=' EXPRESSION))+
EXPRESSION::= EXP ( '>' | '<' | '!=' | '==' | '>=' | '<=') EXP
EXP::= TERM (('+' | '-') TERM)*
TERM::= FACTOR (('*' | '/') FACTOR)*
FACTOR::= ('(' EXPRESSION ')') | ('+' | '-')* 'id'
CONDITIONAL::= 'if' '(' EXPRESSION ')' BLOCK ('elsif' '(' EXPRESSION ')'
BLOCK)* ('else' BLOCK)?
PRINT::= 'print' '(' (EXPRESSION | 'id') (',' (EXPRESSION | 'id') )* ')'
READ::= 'read' 'id'
PAINT::= 'paint' 'id'
RETURN::= 'return' EXPRESSION
COMMENTS::= '#' . '#'
LOOP::= 'for' 'each' 'id' 'in' RANGE ('by' ('integer' | 'decimal'))? BLOCK
FUNCTION::='function' 'id' '(' TYPE 'id' (',' TYPE 'id')* ')' 'returns' TYPE
BLOCK
COLOR::= 'color' 'id' 'red' '=' EXPRESSION 'green' '=' EXPRESSION 'blue' '='
EXPRESSION
```

c. Características Semanticas

+				
	yesno	**int**	**string**	**dec**
yesno	ERROR	ERROR	ERROR	ERROR
int		int	ERROR	dec
string			string	ERROR
dec				dec

-, *, /				
	yesno	**int**	**string**	**dec**
yesno	ERROR	ERROR	ERROR	ERROR
int		int	ERROR	dec
string			ERROR	ERROR
dec				dec

d. Funciones Especiales

Paint	Función que recibe un objeto canvas y utiliza la libreria de Graphics.py para desplegar el grafico

e. Tipos de Datos

Primitivos			
int	string	yesno	dec
Figuras			
point	triangle	circle	rectangle
canvas	color		

4. Plataforma y Desarrollo

Git, Github	El repositorio del proyecto será creado y mantenido utilizando la herramienta de git y estará montado en la plataforma de Github.
Python	El lenguaje de programación con el que se implementara el compilador.
PLY	Implementación de Yacc y Lex (Originalmente C) en Python para el analisis lexico y semantico del compilador.
Graphics.py	Librería para la representación del output gráfico.

5. Firmas

_____ _____

Roel Castaño Enrique Marroquin

_____ _____

Elda Quiroga Hector Ceballos

6. Bibliografía

Zelle, John. "Simple Object Oriented Graphics Library." Http://mcsp.wartburg.edu. Web. <http://mcsp.wartburg.edu/zelle/python/graphics.py>.

Rademacher, Gunther. "Railroad Diagram Generator." Railroad Diagram Generator. 16 Feb. 2017. Web. <http://www.bottlecaps.de/rr/ui>.

Beazley, David. "PLY Python Lex-Yacc." PLY (Python Lex-Yacc). 31 Jan. 2017. Web. <http://www.dabeaz.com/ply/>.