Language Proposal: Pp

Dr. Héctor Ceballos
Ing. Elda Quiroga
TC3048 Compilers Design

Made by
Eduardo Enrique Trujillo Ramos
A01187313

Hugo Oswaldo Garcia Perez
A00815354

2019-09-23

Index

Purpose of the project **2**

Language main objective **2**

Language Requirements **3**

Examples of the language 3

Function declaration 3

Variable declarations and assignments (Basic types) 3

Matrices and datasets 3

Conditionals 4

Loops 4

IO 4

Complete Example: Fibonacci number 5

Basic Elements (Tokens) 6

Syntax Diagrams 8

Main Semantic Characteristics 15

Special Functions 16

Data Types 16

Development Environment **17**

1. Purpose of the project

The purpose of this project is to develop a language focused mainly on engineers and scientists who do statistical analysis and use linear algebra in their day to day life. The main data structure the language will support is the matrix or dataset, which will allow the storage of two-dimensional data. This language will allow the users to speed up their workflow and have a nicer syntax than other languages with this focus such as R but also not be so advanced and difficult to learn for non-programmers such as Python.

2. Language main objective

The main objective of Pp is to be readable by both, developers and non-developers, to be a common ground of collaboration between them and be intuitive and easy to program. The language will be a high-level imperative scripting programming language for statistical analysis, having as the basic data structure the matrix, supporting all of its operations as well as basic statistic functions.

3. Language Requirements

3.1. Examples of the language

3.1.1. Function declaration

```
func int myFunctionName(int a, float b) {
     return a;
}
```

3.1.2. Variable declarations and assignments (Basic types)

```
let int a, b, c;
let float f, g;
let string s;
let bool t;

a = 1;
a = 2+2;
a = 2^10;
a = myFunctionName(a, f);

f = 1.0E18;
f = 1.;
f = 3.14159;
f = 2.0E-10;

t = true and false;
t = not true;
t = not false;

s = “Hello World”;
s = “a”;
```

3.1.3. Matrices and datasets

```
let matrix<int>[3][3] matA, matB;
let dataset<int, float, string, float> myData;

matA = [[1, 2, 3], [4, 5, 6], [7, 8, 9]];
matA = matA*matB;
matA[0][0] = 10;
```

```
myData.add(1, 3.14, “Eduardo”, 4.5);
myData[0][0] = 10;
myData[0][1] = 3.5;
myData[0][2] = “Hugo”;
myData[0][3] = 20.3;

readcsv(myData);
```

3.1.4. Conditionals

```
if (a == 10) {
     ...
} elseif (b > 5) {
     ...
} else {
     ...
}
```

3.1.5. Loops

```
while (a > 0) {
     a = a - 1;
     ...
}
```

3.1.6. IO

```
read(a);
write(10);
write(“Hello world”);
plot(matX, matY);
```

3.1.7. Complete Example: Fibonacci number

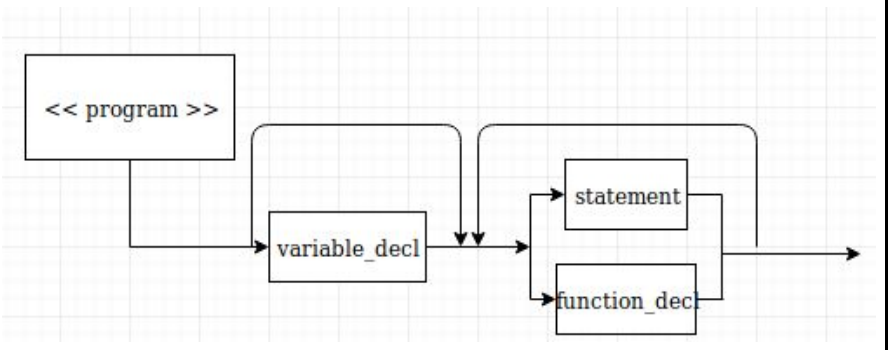
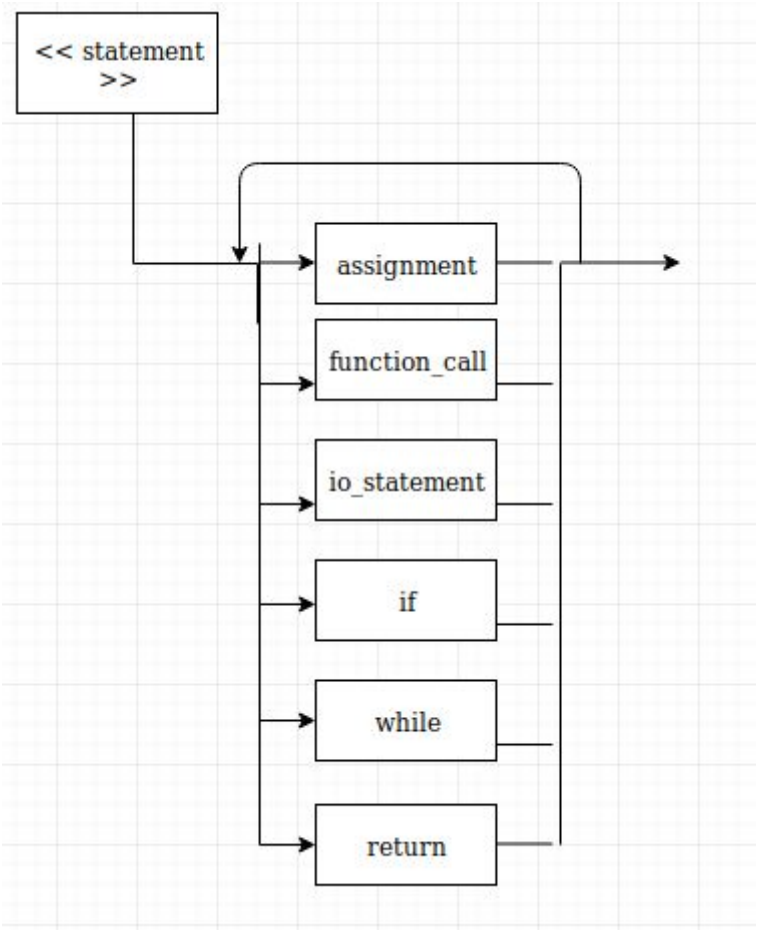
```
let int fib;

func int fibonacci(int n) {
     let int f1, f2, i;
     f1 = 1;
     f2 = 1;
     i = 0;
     while (i < n) {
          let int aux;
          aux = f2;
          f2 = f1 + f2;
          f1 = f2;
          i = i + 1;
     }
     return f1;
}

write("Which Fibonacci number do you want?");
read(fib);

write(fibonacci(fib));
```

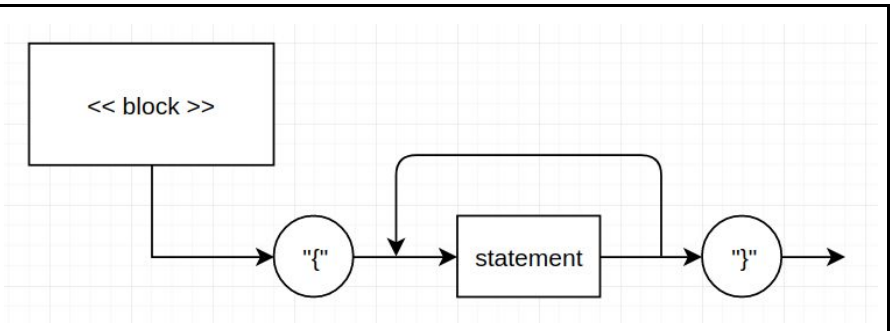
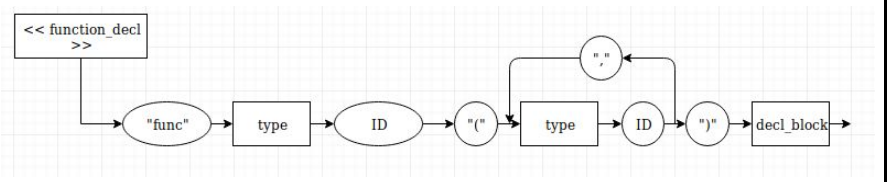
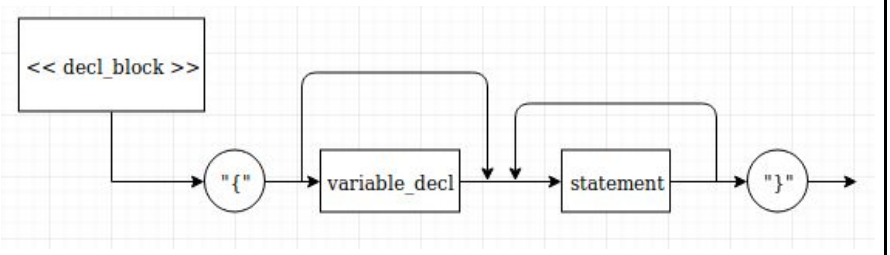
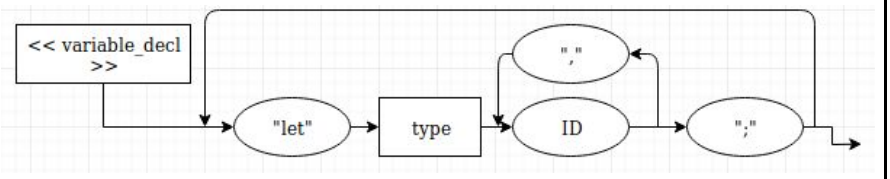
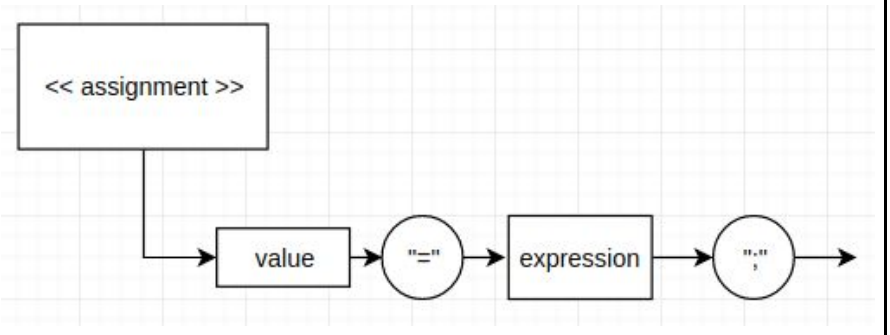
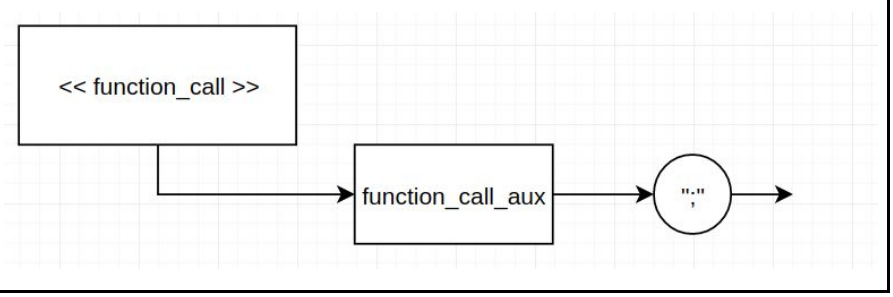
3.2. Basic Elements (Tokens)

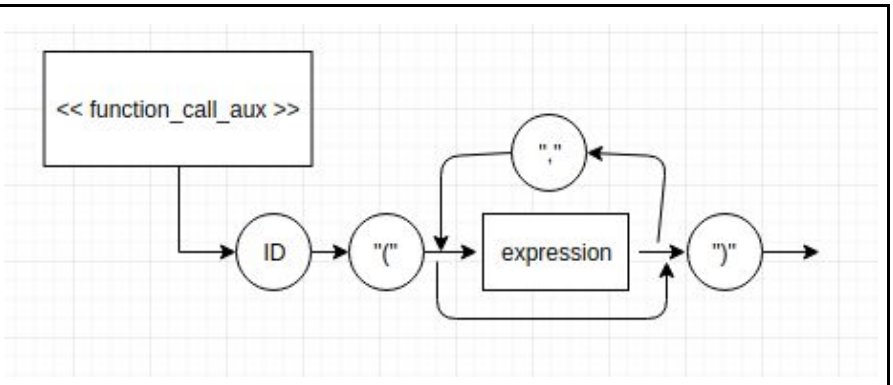
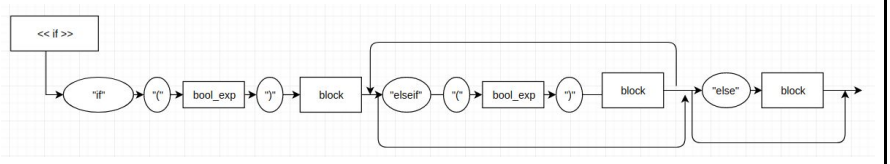
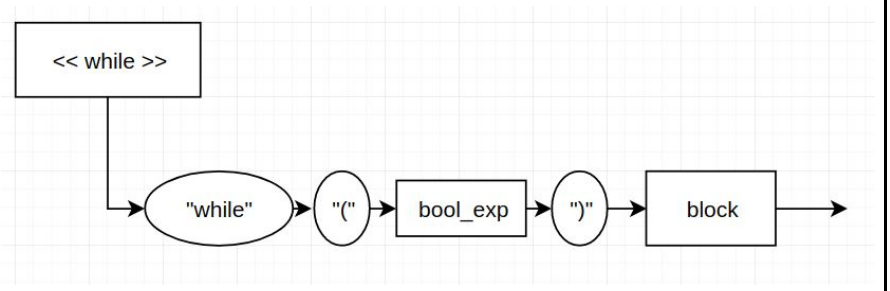
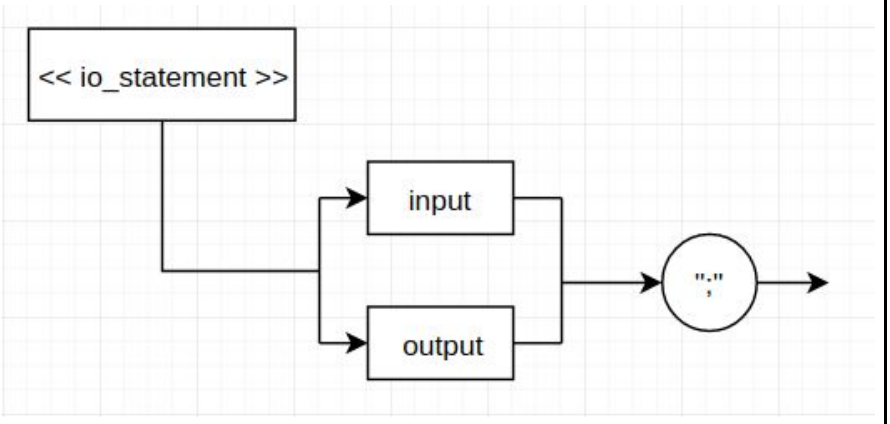
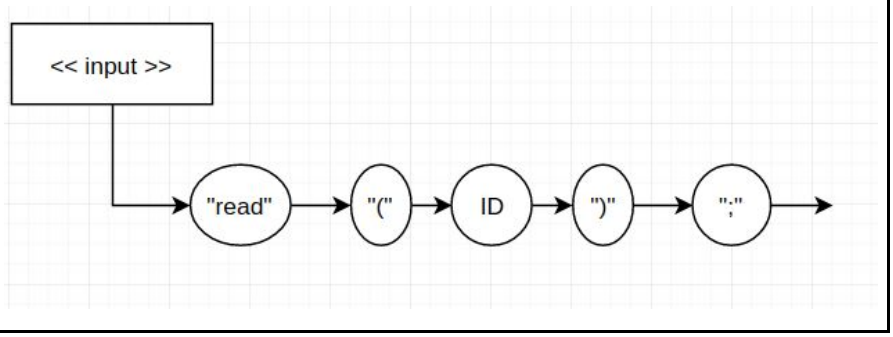
Allowed Tokens	Regular Expression
ADDITION_OPERATOR	“+”
SUBTRACTION_OPERATOR	“-”
MULTIPLICATION_OPERATOR	“*”
DIVISION_OPERATOR	“/”
EXPONENTIATION_OPERATOR	“^”
MODULUS_OPERATOR	“%”
ASSIGNMENT_OPERATOR	“=”
BOOLEAN_AND_OPERATOR	“and”
BOOLEAN_OR_OPERATOR	“or”
BOOLEAN_NOT_OPERATOR	“not”
EQUALITY_OPERATOR	“==”
INEQUALITY_OPERATOR	“!=”
LESS_THAN_OPERATOR	“<”
LESS_THAN_EQUAL_OPERATOR	“<=”
GREATER_THAN_OPERATOR	“>”
GREATER_THAN_EQUAL_OPERATOR	“>=”
SEMICOLON_DELIMITER	“;”
COMMA_DELIMITER	“,”
INPUT_STATEMENT	“read”
CSV_INPUT_STATEMENT	“readcsv”
OUTPUT_STATEMENT	“write”
PLOT_STATEMENT	“plot”
VARIABLE_STATEMENT	“let”
FUNCTION_STATEMENT	“func”

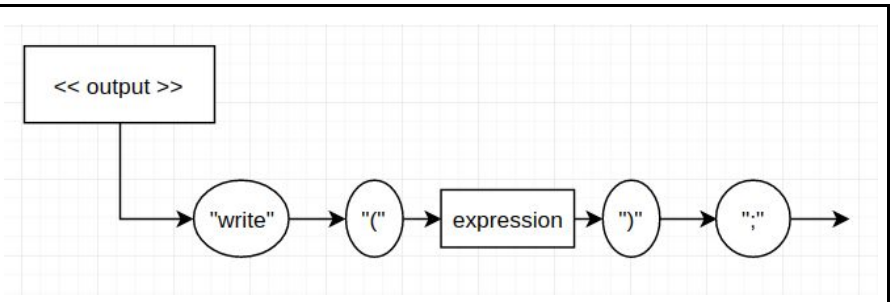
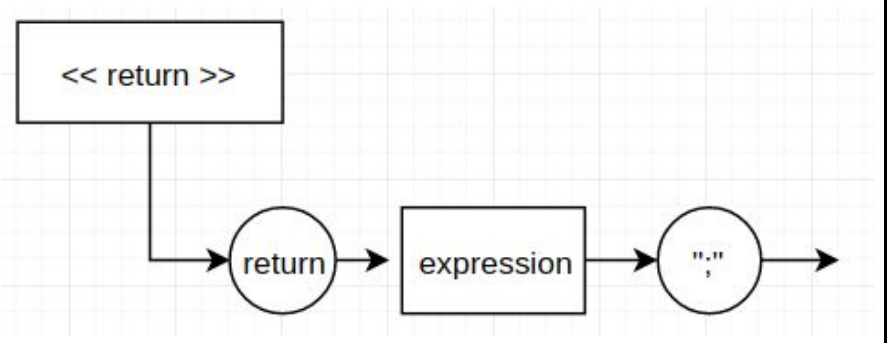
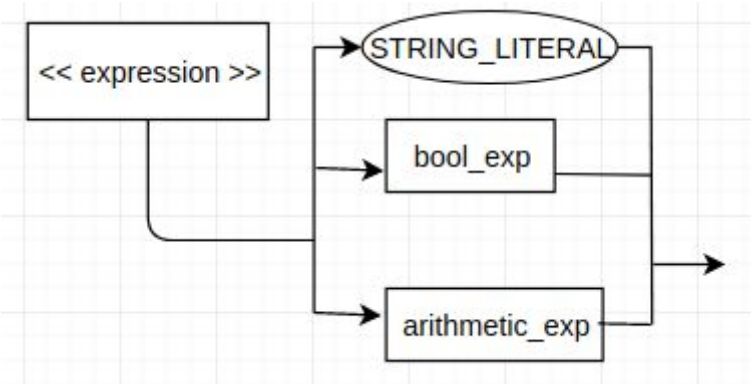
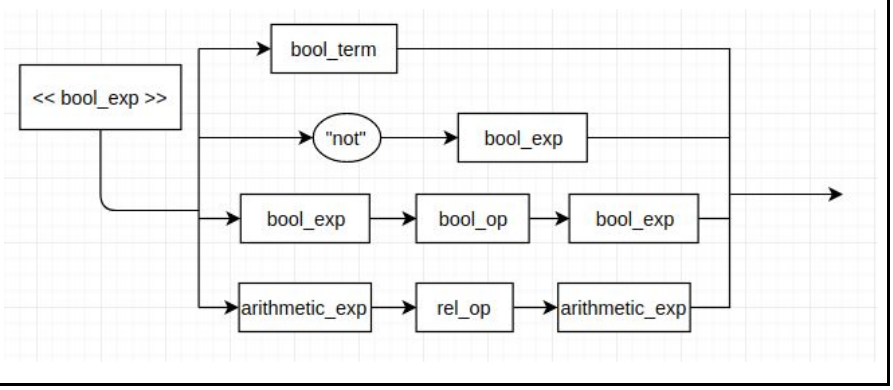
RETURN_STATEMENT	“return”
IF_STATEMENT	“if”
ELSE_IF_STATEMENT	“elseif”
ELSE_STATEMENT	“else”
WHILE_LOOP_STATEMENT	“while”
LEFT_SQUARE_BRACKET	“[“
RIGHT_SQUARE_BRACKET	“]”
LEFT_CURLY_BRACKET	“{“
RIGHT_CURLY_BRACKET	“}”
LEFT_PARENTHESIS	“(“
RIGHT_PARENTHESIS	“)”
INT_TYPE	“int”
BOOLEAN_TYPE	“bool”
FLOAT_TYPE	“float”
STRING_TYPE	“string”
DATESET_TYPE	“dataset”
MATRIX_TYPE	“matrix”
ID	“_”?[a-zA-Z][a-zA-Z0-9_]*
INT_NUMBER	[+|-]?[0-9]+
FLOAT_NUMBER	[+|-]?[0-9]+”.”([0-9]+)(E[-|+]?[0-9]+)
BOOLEAN_LITERAL	(“true” | ”false”)
STRING_LITERAL	““‘.*?’”’

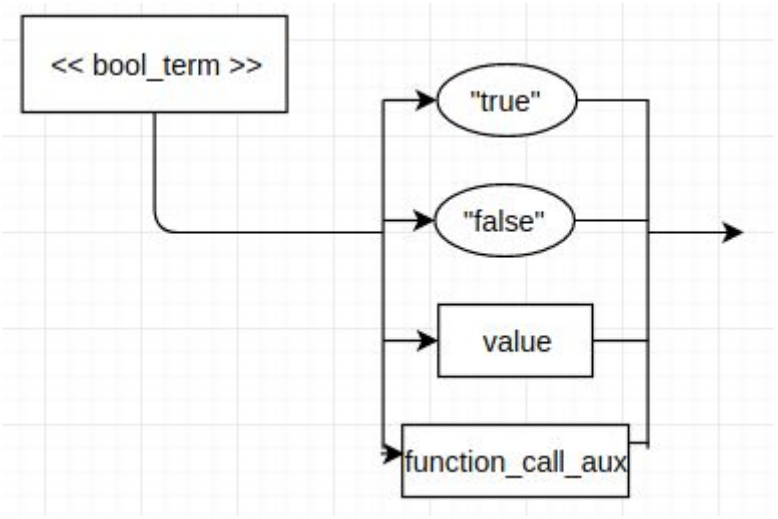
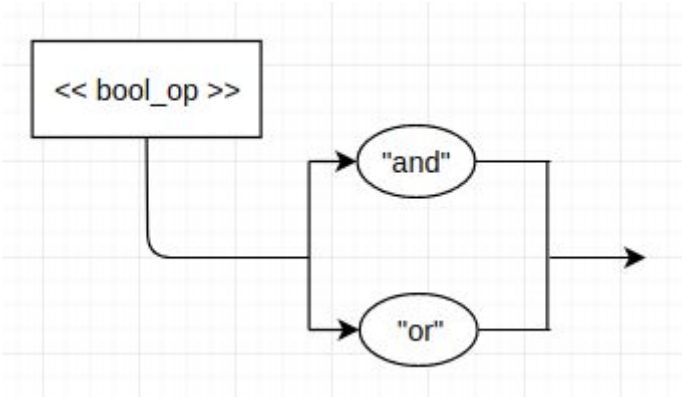
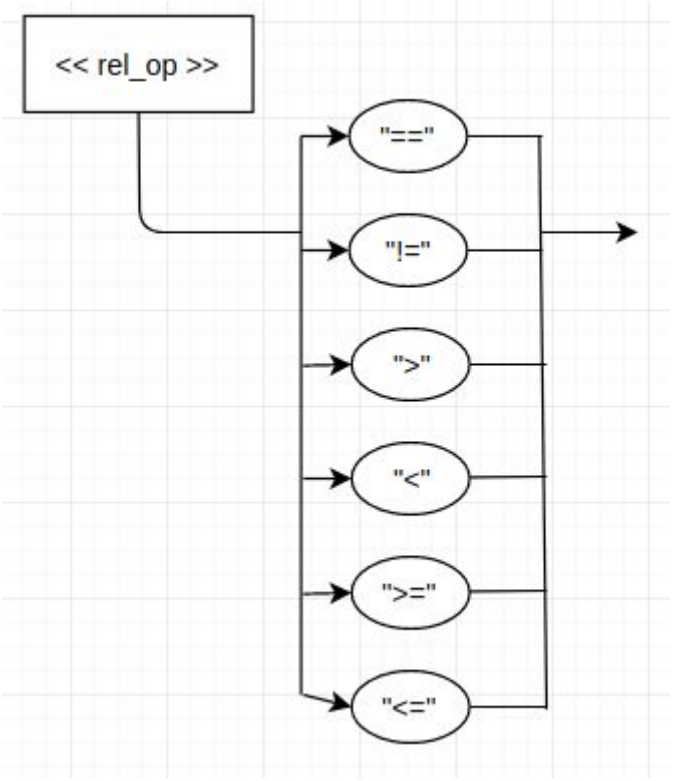
3.3. Syntax Diagrams

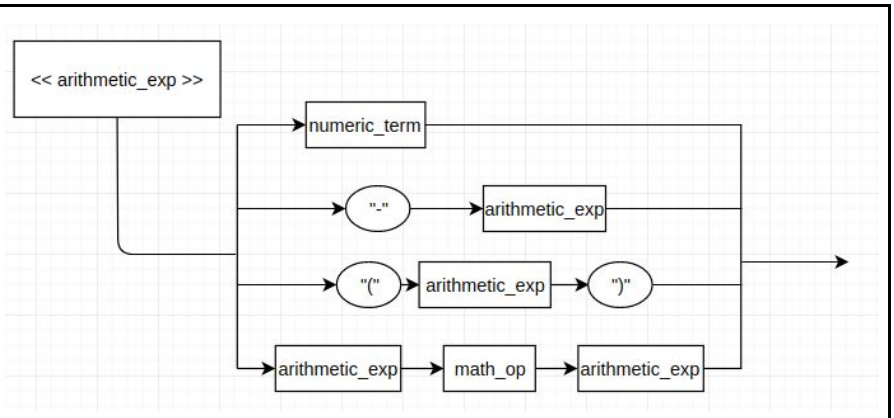
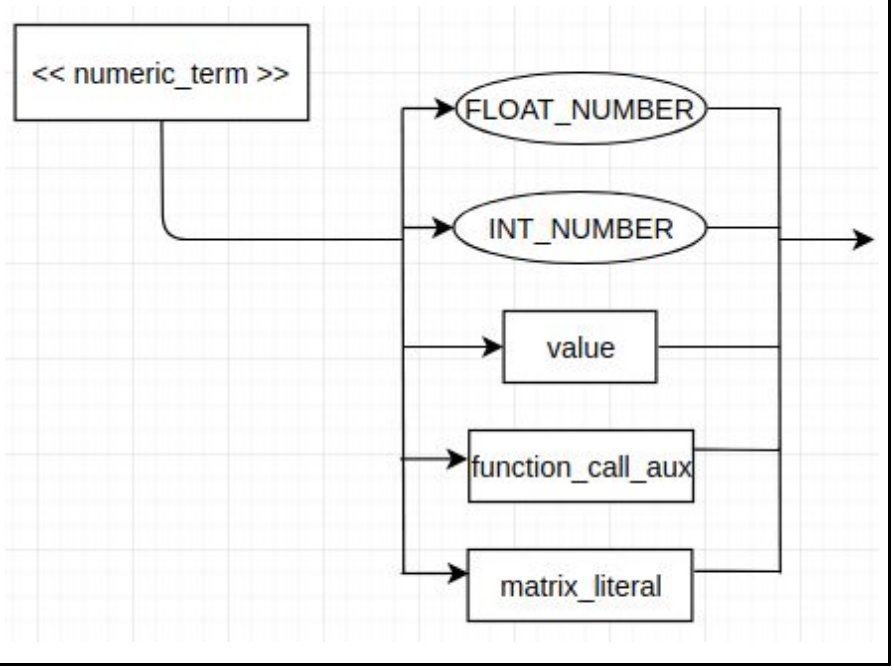
Rule	Diagram
program	
statement	

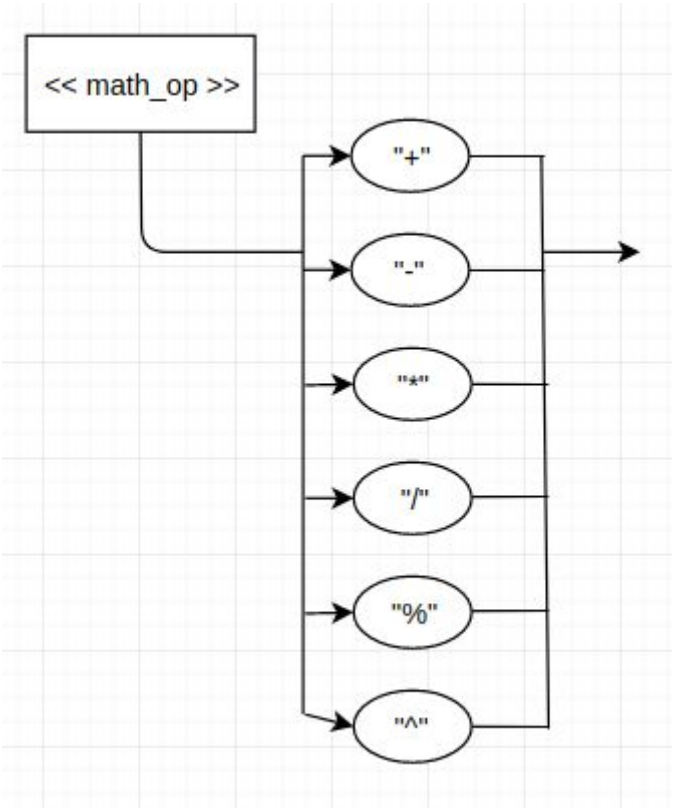
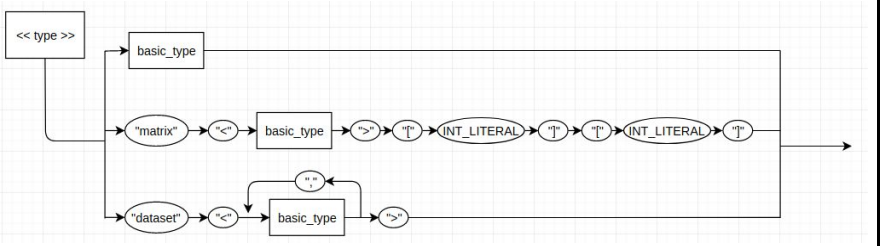
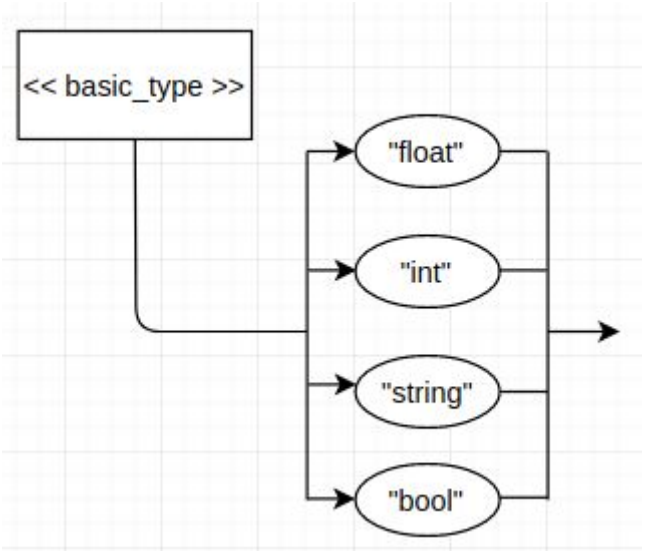
block	
function_decl	
decl_block	
variable_decl	
assignment	
function_call	

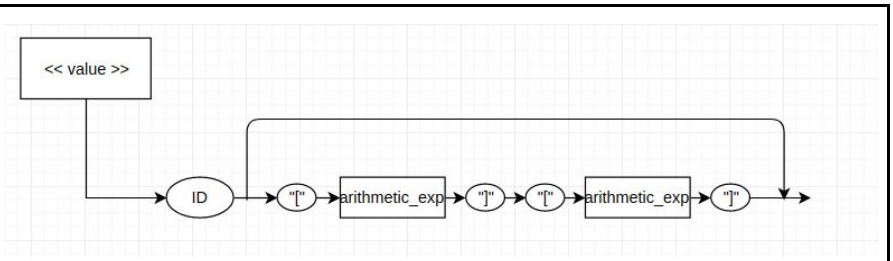
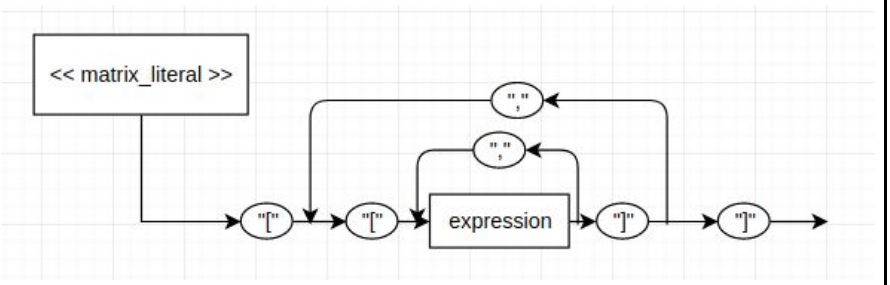
function_call_aux	
if	
while	
io_statement	
input	

output	
return	
expression	
bool_exp	

bool_term	
bool_op	
rel_op	

arithmetic_exp	
numeric_term	

math_op	
type	
basic_type	

value	
matrix_literal	

3.4. Main Semantic Characteristics

The following arithmetic operation will be supported (order of operands does not matter):

Left operand	operator	Right operand	Result
int	+, -, *, %, ^	int	int
int	/	int	float
int	+, -, *, /, ^	float	float
float	+, -, *, /, ^	float	float
matrix	+, -, *	matrix	matrix (The dimensions must be compatible and of type int or float)
int	+, -, *	matrix	matrix
float	+, -, *	matrix	matrix
matrix	^	int	matrix (The order does matter here!)

3.5. Special Functions

The language will include the common matrix operations overloaded so it will be as easy to add to matrices as it is to add to integers. It will also have the capacity to plot points and join them given two column matrices (n*1 dimension) representing the x values and y values.

The language will also include a dataset type which allows the user to have a matrix-like data structure where each column may be of a different type. This data structure will be able to grow freely by adding several values or read them from a CSV.

We will include a small set of statistical functions as part of the standard library such as:

- mean(): Arithmetic mean of the data
- median(): Middle value of the sorted data
- mode(): Most common value of the data
- stdev(): Standard deviation of data
- variance(): Variance of data

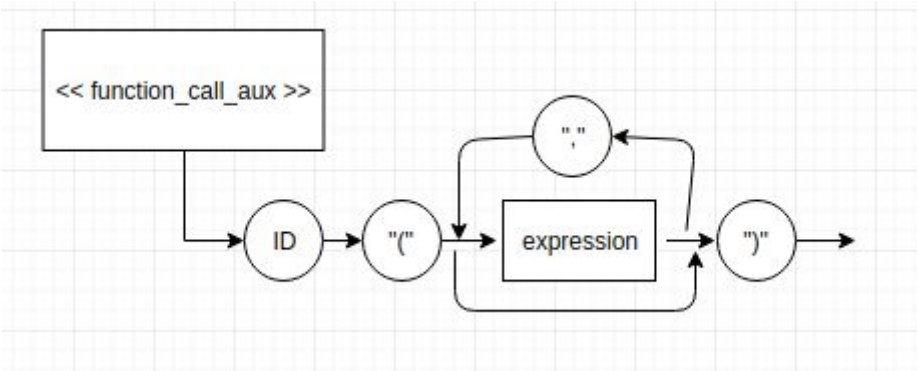
These functions will operate on numerical arrays or columns (for the dataset structure). Also some distributions can be calculated:

- dbeta(): Beta distribution
- dbinom(): Binomial distribution (Bernoulli included)
- dexp(): Exponential distribution
- dgamma(): Gamma distribution
- dgeom(): Geometric distribution
- dnorm(): Normal distribution
- dpois(): Poisson distribution
- dunif(): Uniform distribution

For every distribution function there are two more versions where the first letter changes, for example cbeta() is the cumulative beta function and rbeta() is the random variable beta function.

3.5.1. Syntax diagrams for special functions

The basic syntax for all of the special functions follow the function_call_aux form:



Where the first ID is the name of the special function and has a determined number of parameters depending on the definition of the function. All the functions return a numeric value.

Functions definition:
The following functions operate column or row matrices (dimensions Nx1 or 1xN) or dataset columns of type int or float. The mode function can operate on strings and bools as well. For simplicity we will just define the syntax for the matrices type and the variable T is any of the types supported.

- Mean

Function:

```
func float mean(matrix<T> m)
```

- Median

Function:

```
func T median(matrix<T> m)
```

- Mode

Function:

```
func T mode(matrix<T> m)
```

- Standard Deviation

Function:

```
func float stdev(matrix<T> m)
```

- Variance

Function:

```
func float variance(matrix<T> m)
```

- Beta distribution

Γ(a+b)/(Γ(a)Γ(b))x^(a-1)(1-x)^(b-1)

Functions:

```
func float dbeta(float x, float a, float b)
func float cbeta(float x, float a, float b)
func float rbeta(float a, float b)
```

- Binomial distribution (Bernoulli included)

choose(n, x) p^x (1-p)^(n-x)

Functions:

```
func float dbinom(int x, int n, float p)
func float cbinom(int x, int n, float p)
func int rbinom(int n, float p)
```

- Exponential distribution

λ {e}^{- λ x}

Functions:

```
func float dexp(float x, float lambda)
func float cexp(float x, float lambda)
func float rexp(float lambda)
```

- Gamma distribution

1/(s^a Gamma(a)) x^(a-1) e^-(x/s)

Functions:

```
func float dgamma(float x, float a, float s)
func float cgamma(float x, float a, float s)
func float rgamma(float a, float s)
```

- Geometric distribution

p (1-p)^x

Functions:

```
func float dgeom(int x, float p)
func float cgeom(int x, float p)
func int rgeom(float p)
```

- Normal distribution

1/(√(2 π) σ) e^-((x - μ)^2/(2 σ^2))

Functions:

```
func float dnorm(float x, float mean, float sd)
func float cnorm(float x, float mean, float sd)
func float rnorm(float mean, float sd)
```

- Poisson distribution

λ^x exp(-λ)/x!

Functions:

```
func float dpois(int x, float lambda)
func float dpois(int x, float lambda)
func int rpois(float lambda)
```

- Uniform distribution

1/(max-min)

Functions:

```
func float dunif(float x, float min, float max)
func float cunif(float x, float min, float max)
func float runif(float min, float max)
```

3.6. Data Types

The basic data types supported by this programming language are:

- int: Signed integer with no memory limitation. Will work as BigIntegers in Java or like integers in Python.
- float: Signed double-precision floating-point number.
- string: Chain of 8 bit ASCII characters.

The other more complex data types supported will be:

- matrix: 2-dimensional arrays of contiguous memory. The matrix is a collection of the same basic data type. This data type will also be used for the traditional array, being either a column matrix (n*1 dimensions) or row matrix (1*n dimensions).
- dataset: A dataset is a set of column matrices where each matrix can have a different data type. All matrices should have the same number of rows and only one column.

4. Development Environment

The programming language here specified will be developed using ANTLR as the syntax and lexical analyzer with Python. The environment in which it will be developed and tested will be Linux Ubuntu 16.04+