



Tecnológico de Monterrey

Choo Choo Train

David Benitez - A01191731
Paulina Escalante - A01191962

March 6, 2017

1. Purpose of Project

The purpose of this project is to help children get an insight into the world of programming from a small age. Children are usually uninterested to learn how to program because they are not sure of what it actually is or what it's useful for. They think that because they don't know about computers, they will be bad at programming. The fact that programs look like huge files of unreadable text, sometimes with complicated formulas, doesn't help either. If children can relate programming to something that is familiar to them, then we might be able to help them understand what programming really is about, and possibly encourage them to pursue a Computer Science Degree in the future.

Children will use our visual language to "program" blocks, using a visual representation tool of code. Then they will be able to watch their code being executed, represented as a train going through its tracks. By mapping the elements of a language (such as loops, statements, etc.) to a specific type of train track section or element, we can help kids understand the logic and data flow of a program. This will introduce them to the world of programming while still maintaining an attractive and fun appearance that won't make them think that what they're about to do is rocket science (yet).

2. Main Objective and Area

The main objective of our language is to be understandable, visual, and easy to use so that children will be motivated to learn how to program. The main characteristic is that both the input and the output are visual, in a sense that the output encompasses the step by step execution of the program, as well as any output generated. With this visual representation, complicated programming terminology and structures are abstracted and represented with objects familiar to children, in this case a train track environment. Its objective is to be an educational, fun, and pleasing programming language for children.

3. Language Requirements

3.1 Tokens

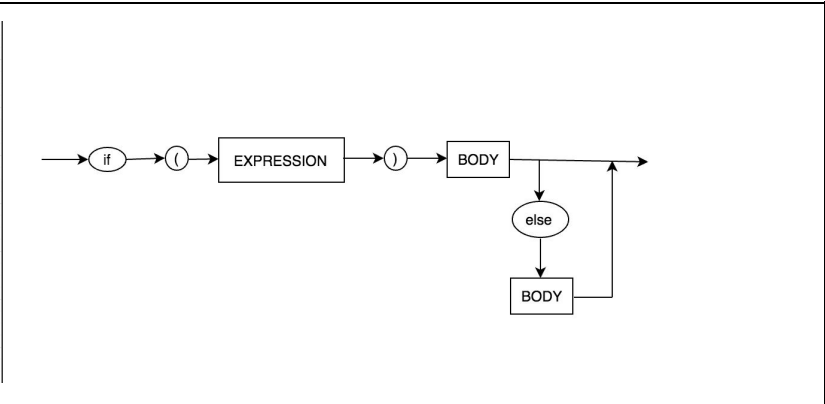
Token Name	Regular Expression
block	"block"
starting	"starting"
receives	"receives"
block_returns	"returns"
call	"call"
variable	"variable"
of_type	"of type"
return_statement	"return"
do	"do"
until	"until"
if	"if"
else	"else"
colon	"."
semicolon	"."
comma	"."
input	"input"
print	"print"
curlybraces_open	"{"
curlybraces_close	"}"
parenthesis_open	"("
parenthesis_close	")"
squarebracket_open	"["
squarebracket_close	"]"

op_assign	"="
op_less	
op_less_equal	
op_greater_equal	
op_or	
op_negation	
op_addition	
op_subtraction	
op_multiplication	
op_division	
whole	"whole"
decimal	"decimal"
words	"words"
boolean	"boolean"
cst_whole	[0-9]+
cst_decimal	[0-9]+\.[0-9]+([Ee][\+-]?[0-9]+)?
cst_words	\("[^"]"\)
cst_boolean	"true" "false"
id	[A-Za-z]([A-Za-z0-9])*

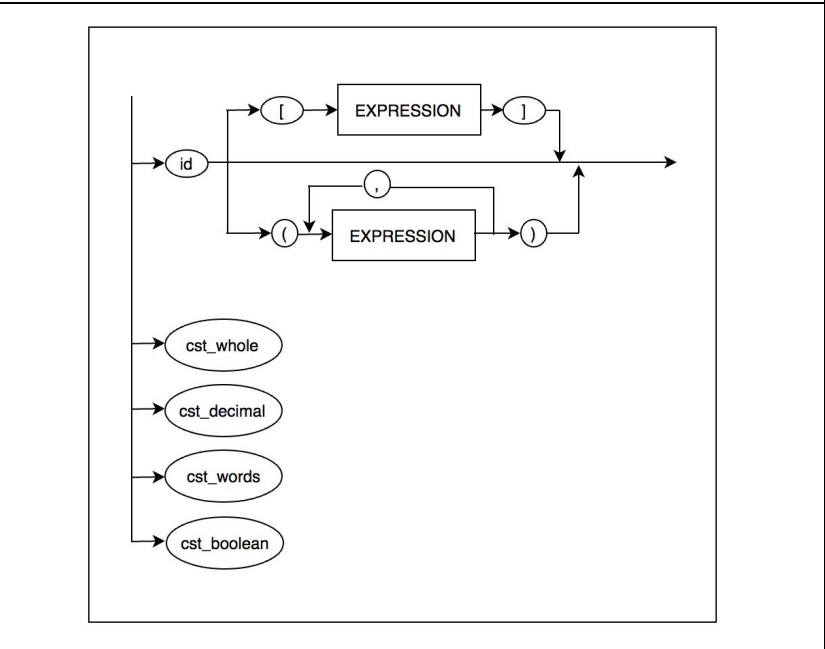
3.2 Syntax Diagrams

Syntactic Rule	Syntax Diagram
PROGRAM	
BLOCK	
BLOCK_BODY	
TYPE	
BODY	

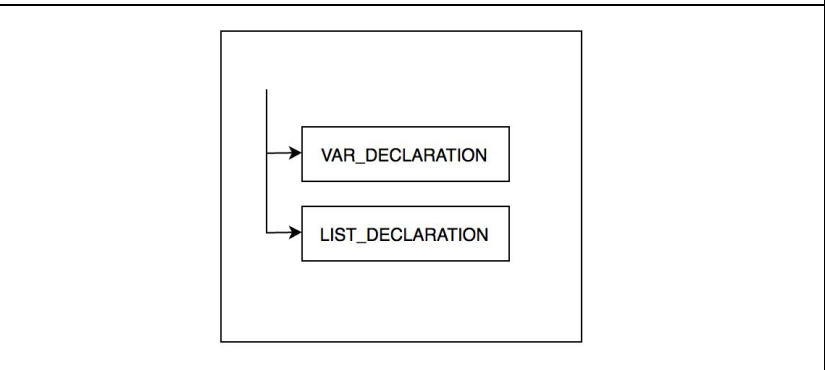
CONDITION



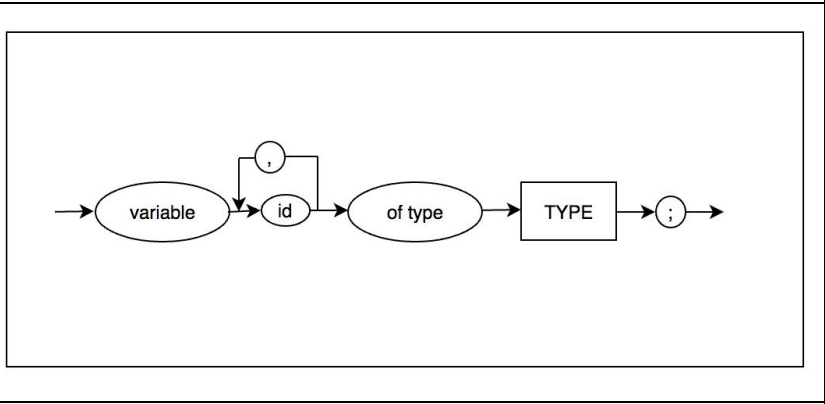
CONSTANT



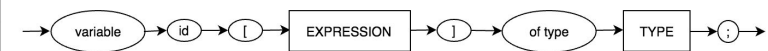
DECLARATIONS



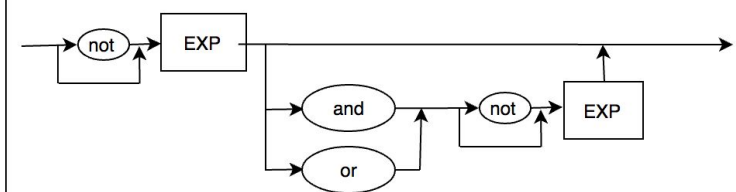
VAR_DECLARATION



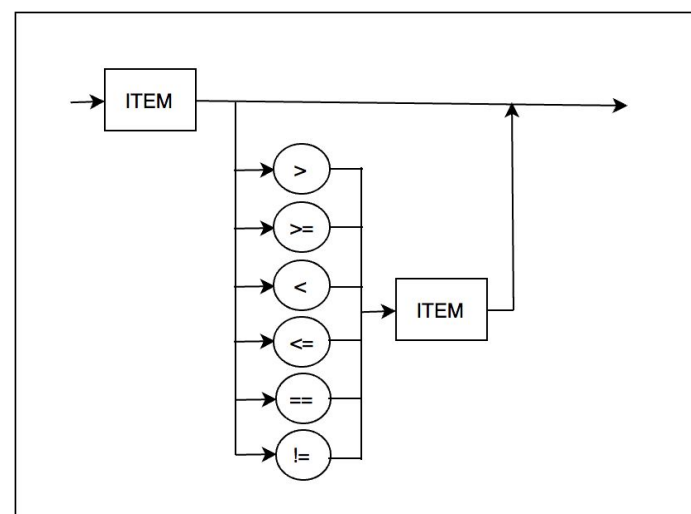
LIST_DECLARATION



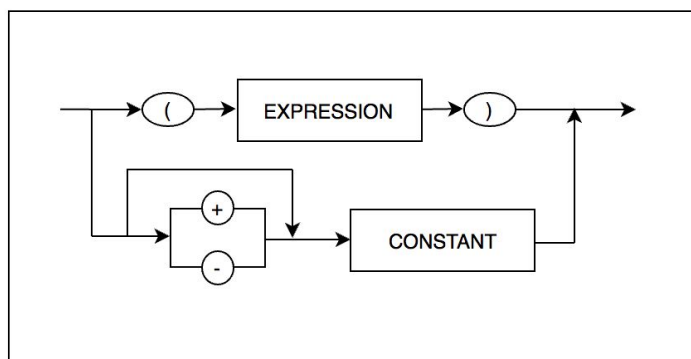
EXPRESSION

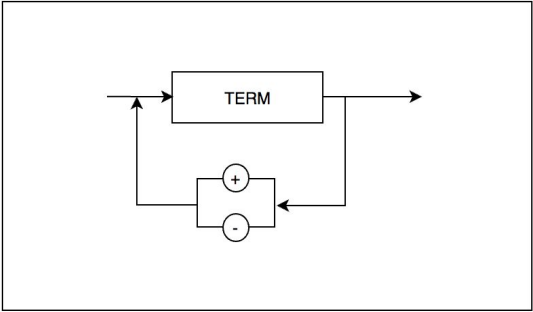
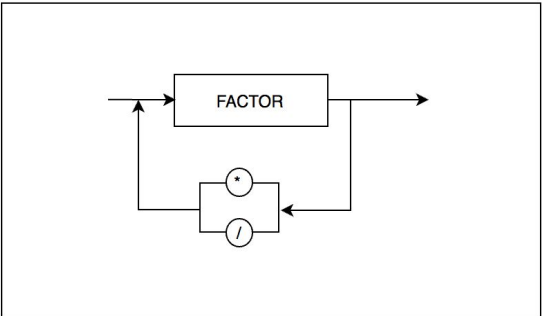
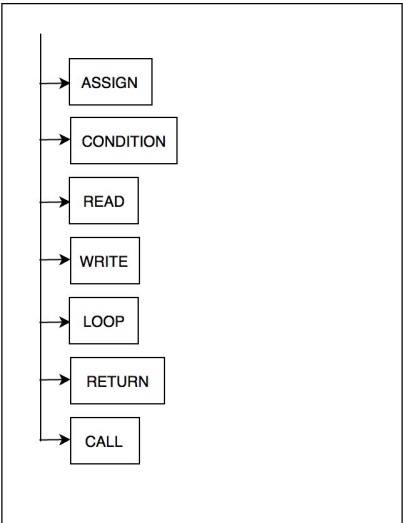
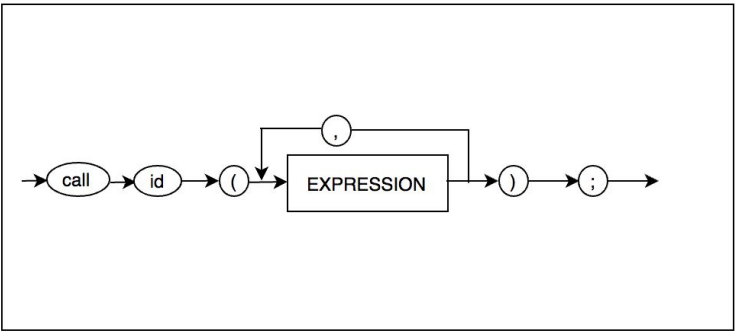



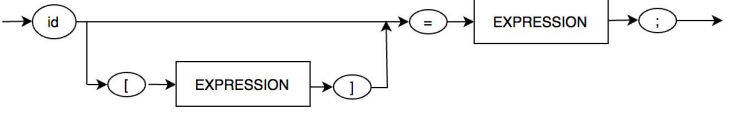

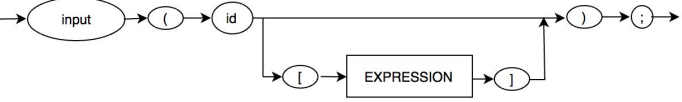
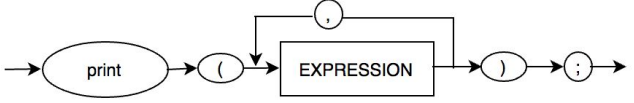
EXP



FACTOR



ITEM	
TERM	
STATEMENT	
CALL	

<p>LOOP</p>	 <pre> graph LR Start(()) --> do((do)) do --> BODY[BODY] BODY --> until((until)) until --> LP("(") LP --> EXPRESSION[EXPRESSION] EXPRESSION --> RP(")") RP --> End(()) </pre>
<p>ASSIGN</p>	 <pre> graph LR Start(()) --> id((id)) id --> EQ("=") EQ --> EXPRESSION1[EXPRESSION] EXPRESSION1 --> SEMI(";") SEMI --> End(()) id --> L "[" L --> EXPRESSION2[EXPRESSION] EXPRESSION2 --> R "]" R --> EQ </pre>
<p>RETURN</p>	 <pre> graph LR Start(()) --> return((return)) return --> EXPRESSION[EXPRESSION] EXPRESSION --> SEMI(";") SEMI --> End(()) </pre>
<p>READ</p>	 <pre> graph LR Start(()) --> input((input)) input --> LP("(") LP --> id((id)) id --> RP(")") RP --> SEMI(";") SEMI --> End(()) id --> L "[" L --> EXPRESSION[EXPRESSION] EXPRESSION --> R "]" R --> RP </pre>
<p>WRITE</p>	 <pre> graph LR Start(()) --> print((print)) print --> LP("(") LP --> COMMA(",") COMMA --> EXPRESSION[EXPRESSION] EXPRESSION --> RP(")") RP --> SEMI(";") SEMI --> End(()) </pre>

3.3 Context Free Grammar

Syntactic Rule	Context Free Grammar (LR Parsing)
PROGRAM	PROGRAM'
PROGRAM'	BLOCK PROGRAM' BLOCK
BLOCK	BLOCK' block id RECEIVES_AUX RETURNS_AUX BLOCK_BODY
BLOCK'	starting ϵ
RECEIVES_AUX	receives colon id of_type TYPE RECEIVES_AUX' ϵ
RECEIVES_AUX'	comma id of_type TYPE RECEIVES_AUX' ϵ
RETURNS_AUX	block_returns TYPE ϵ
BLOCK_BODY	curlybraces_open BLOCK_BODY' curlybraces_close
BLOCK_BODY'	DECLARATIONS BLOCK_BODY'' BLOCK_BODY''
BLOCK_BODY''	STATEMENT BLOCK_BODY'' ϵ
TYPE	whole decimal words boolean
BODY	curlybraces_open BODY' curlybraces_close
BODY'	STATEMENT BODY' ϵ
CONDITION	if parenthesis_open EXPRESSION parenthesis_close BODY CONDITION'
CONDITION'	else BODY ϵ
CONSTANT	id CONSTANT' cst_whole cst_decimal cst_words cst_boolean
CONSTANT'	squarebracket_open EXPRESSION squarebracket_close

	parenthesis_open EXPRESSION CONSTANT'' parenthesis_close ε
CONSTANT''	comma EXPRESSION CONSTANT'' ε
DECLARATIONS	VAR_DECLARATION LIST_DECLARATION
VAR_DECLARATION	variable id VAR_DECLARATION' of_type TYPE semicolon
VAR_DECLARATION ,	comma id VAR_DECLARATION' ε
LIST_DECLARATION	variable id squarebracket_open EXPRESSION squarebracket_close of_type TYPE semicolon
EXPRESSION	EXPRESSION' EXP EXPRESSION''
EXPRESSION'	op_negation ε
EXPRESSION''	op_and EXPRESSION' EXP op_or EXPRESSION' EXP
EXP	ITEM EXP'
EXP'	op_less ITEM op_less_equal ITEM op_greater ITEM op_greater_equal ITEM op_equal ITEM op_not_equal ITEM ε
FACTOR	parenthesis_open EXPRESSION parenthesis_close FACTOR'
FACTOR'	op_addition CONSTANT op_subtraction CONSTANT CONSTANT
ITEM	TERM ITEM'
ITEM'	op_addition ITEM op_subtraction ITEM ε
TERM	FACTOR TERM'
TERM'	op_multiplication TERM op_division TERM ε

STATEMENT	ASSIGN CONDITION READ WRITE LOOP RETURN CALL
CALL	call id parenthesis_open EXPRESSION CALL' parenthesis_close semicolon
CALL'	comma EXPRESSION CALL' ϵ
LOOP	do BODY until parenthesis_open EXPRESSION parenthesis_close
ASSIGN	id ASSIGN' op_assign EXPRESSION semicolon
ASSIGN'	squarebracket_open EXPRESSION squarebracket_close
RETURN	return_statement EXPRESSION semicolon
READ	input parenthesis_open id READ' parenthesis_close semicolon
READ'	squarebracket_open EXPRESSION squarebracket_close ϵ
WRITE	print parenthesis_open EXPRESSION WRITE' parenthesis_close semicolon
WRITE'	comma EXPRESSION WRITE' ϵ

3.3 Semantic Characteristics

- The keyword “starting” denotes the initial function.
- Variables have to be declared before being used in any way.
- Id names can’t be repeated.
- A square bracket should only be used after an id that represents an array.
- Variables have a local scope.
- Only the following operations between types are allowed (note that the order of the types are irrelevant and every other combination is considered an error):

Valid Operations			
Operand 1	Operator	Operand 2	Result
whole	+ - * /	whole	result of type whole number: operation is mathematically evaluated
whole	+ - * /	decimal	result of type decimal number: operation is mathematically evaluated
decimal	+ - * /	decimal	result of type decimal number: operation is mathematically evaluated
words	+	words	result of type words: Operand 2 is added to Operand 1 (string concatenation)
whole	< <= > >= == !=	whole	result of type boolean: evaluates the relational operation
whole	< <= > >= == !=	decimal	result of type boolean: evaluates the relational operation
decimal	< <= > >= == !=	decimal	result of type boolean: evaluates the relational operation
words	== !=	words	result of type boolean: evaluates the relational operation
boolean	== !=	words	result of type boolean: evaluates the relational operation
boolean	and	boolean	result of type boolean: evaluates the logical operation
boolean	or	boolean	result of type boolean: evaluates the logical operation
-	not	boolean	result of type boolean: evaluates the unary operation

3.4 Special Functions

There are some functions that work similar to those used in C++ or Python. The logic is the same but keywords are changed to facilitate the reading of code for children. For example:

Conventional Name	Choo Choo Train Name
Function/Method	block
var	variable
while	do until
&&	and
	or
!	not

There are also three modifications for the language. Parameters in a block definition are defined with the keyword “receives” so that it is known that parameters are variables that the block receives. There is a keyword for “of type” that specifies any variable’s type, instead of the conventional way of defining variables as first specifying type and then the variable id. While loops are called “do until” loops, since this improves the clarity and readability of the code for little children.

3.5 Primitive Data Types

There are 4 primitive data types in our language. We wanted to make it simple for children to understand, so while we kept the types:

- integer
- float
- string
- boolean

We decided to change their names to: whole, decimal and words, respectively; excluding the boolean type. These are more commonly used terms in the english language, so it will help children understand the values being represented by each type. The boolean type remains untouched, and is named the same, for there is no term that would semantically represent the type boolean better than its own name. To make it easier for children, values will be restricted when declaring boolean variables only to True and False.

Lists are also part of the language, and can hold values for any of the available data types mentioned above. They are declared by following the same format as a

variable declaration, but square brackets enclosing the size of the list follow the id name.

4. Language and Computers

The project will be developed using Python, Google's Blockly Library, and using two Macbook Pros.

5. Bibliography

- Programiz. "Python Variables and Data types". *Programiz*. Web.
<<https://www.programiz.com/python-programming/variables-datatypes>>
- Esthier T., Guyot J. "The BNF Web Club Language". Web. 1998.
<<http://cuiwww.unige.ch/db-research/Enseignement/analyseinfo/BNFweb.html>>
- MIT. "Scratch - Imagine, Program, Share." *Scratch - Imagine, Program, Share*.
<<https://scratch.mit.edu/>>
- Google for Education. "Blockly". *Google*. Web.
<<https://developers.google.com/blockly/>>