



Tecnológico de Monterrey

Choo Choo Train

David Benitez - A01191731
Paulina Escalante - A01191962

February 27, 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

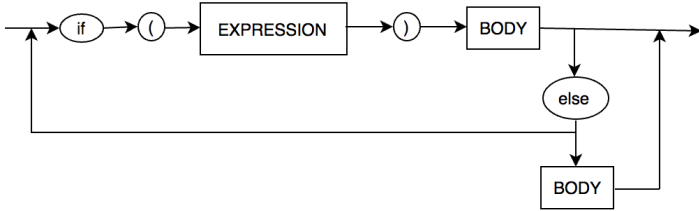
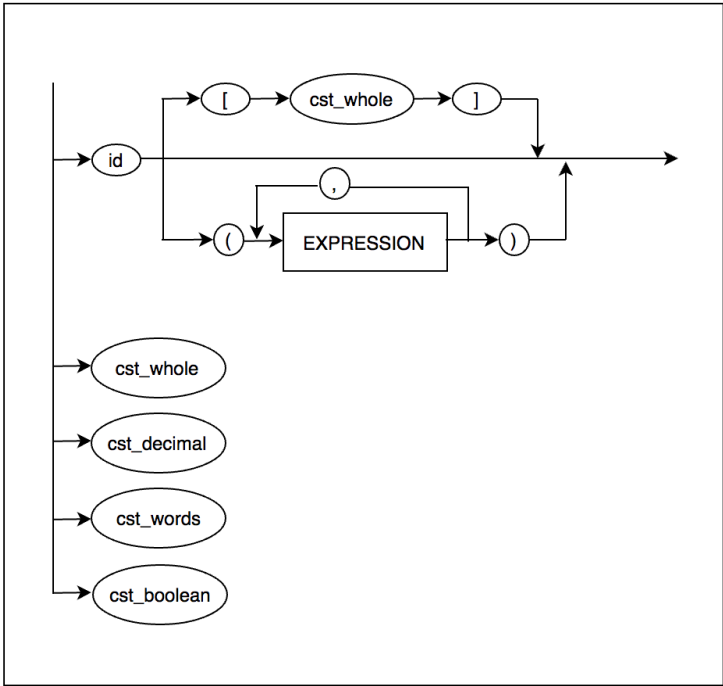
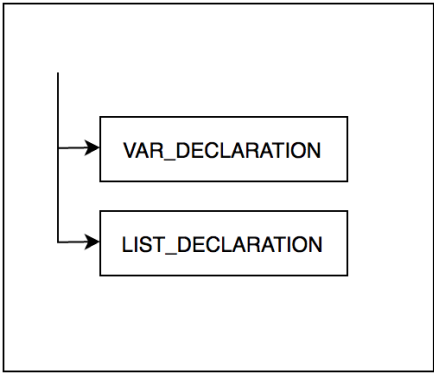
3.1 Tokens

Token Name	Regular Expression
block	"block"
receives	"receives"
call	"call"
variable	"variable"
ofType	"of type"
return	"return"
do	"do"
until	"until"
if	"if"
else	"else"
colon	"."
semicolon	","
comma	" "
input	"input"
print	"print"
curlyBracket_open	"{"
curlyBracket_close	"}"
parenthesis_open	"("
parenthesis_close	")"
squareBracket_open	"["
squareBracket_close	"]"
assign	"="
relOp	"<" "<=" ">" ">=" "==" "!="

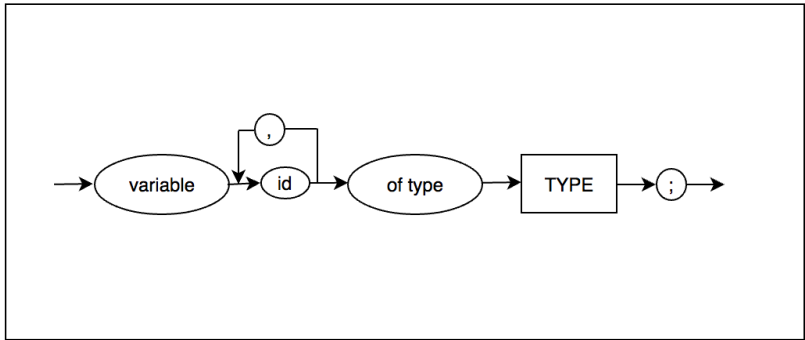
binaryOp	“and” ”or”
negationOp	“not”
plusOp	“+”
minusOp	“-”
multiplicationOp	“*”
divisionOp	“/”
whole	“whole”
decimal	“decimal”
words	“words”
boolean	“boolean”
cst_whole	[0-9]+
cst_decimal	[0-9]+\.[0-9]+([Ee][\+-]?[0-9]+)?
cst_words	\”[^\”]\”
cst_boolean	“true” “false”
id	[A-Za-z]([A-Za-z0-9])*

3.2 Syntax Diagrams

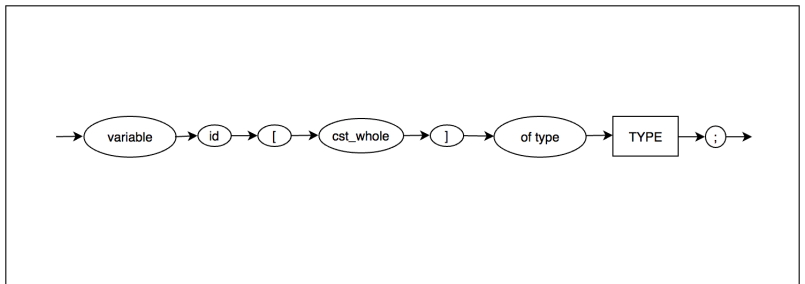
Syntactic Rule	Syntax Diagram
PROGRAM	
BLOCK	
TYPE	
BODY	

CONDITION	
CONSTANT	
DECLARATIONS	

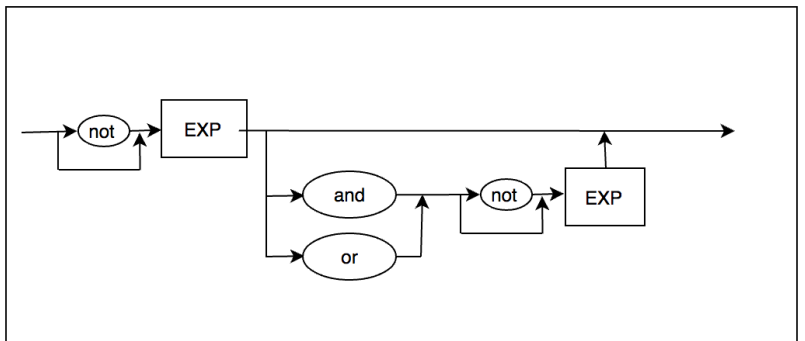
VAR_DECLARATION



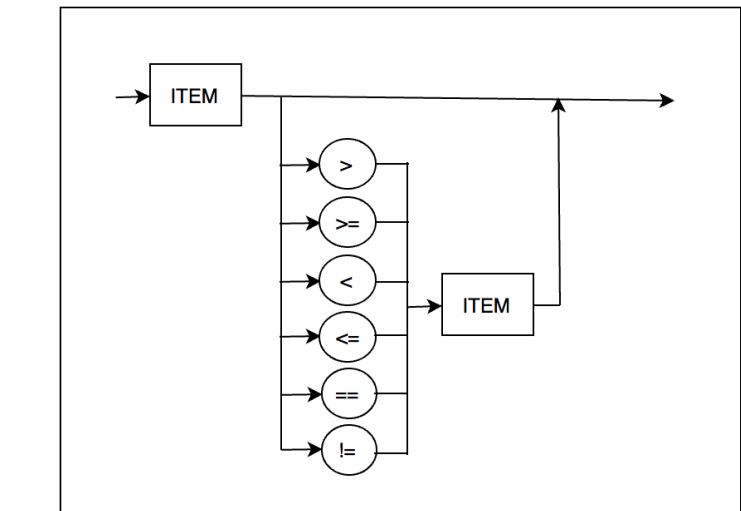
LIST_DECLARATION

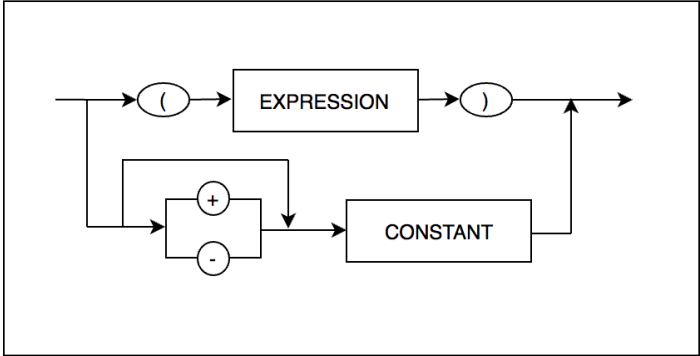
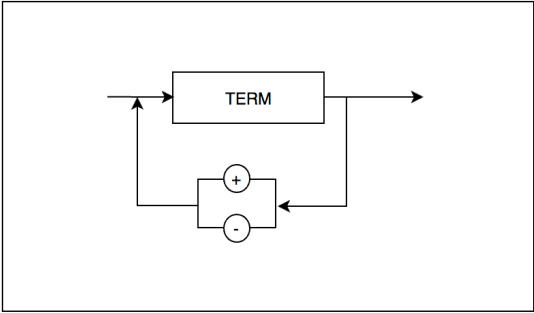
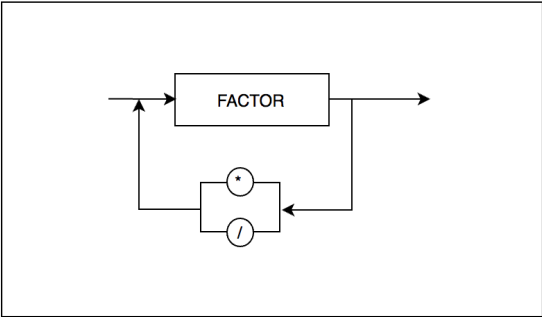
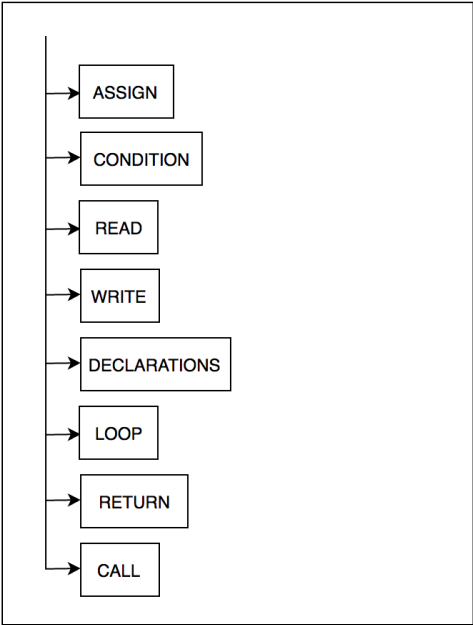


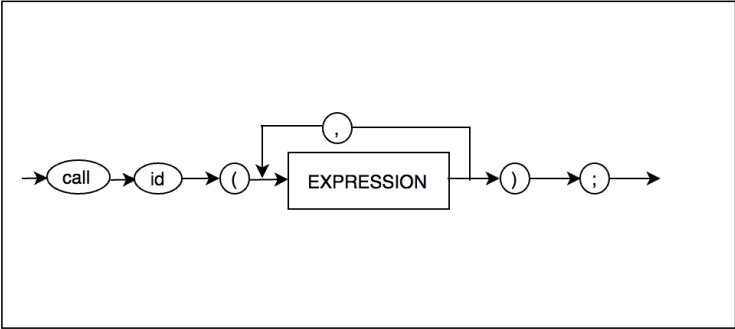
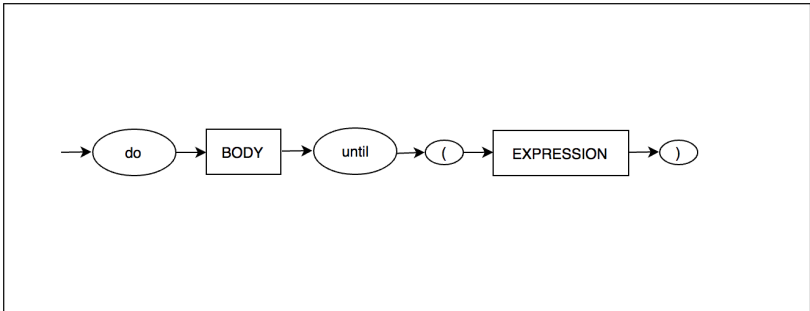
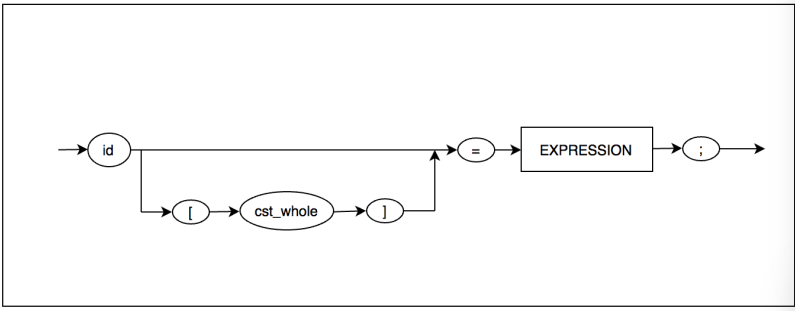
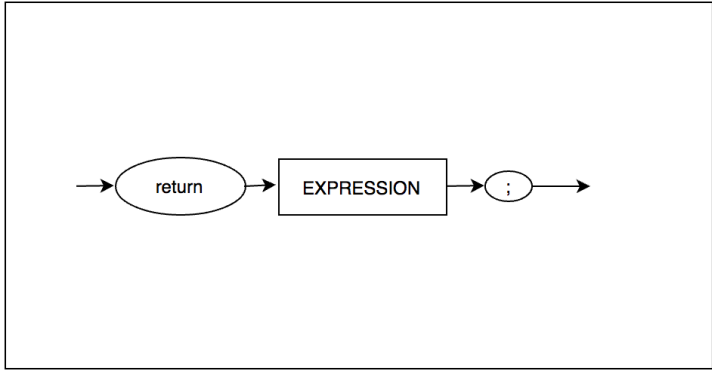
EXPRESSION

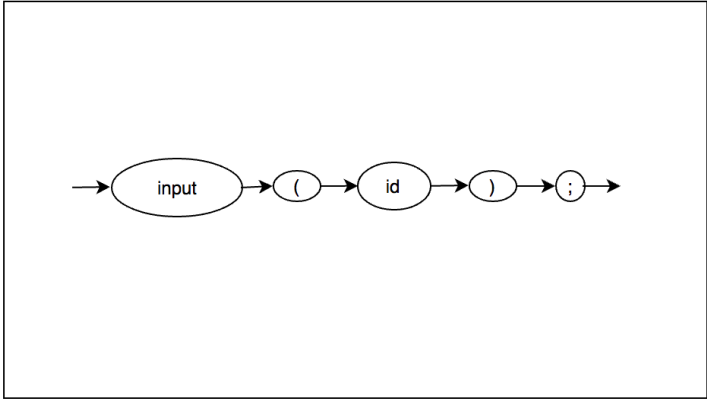
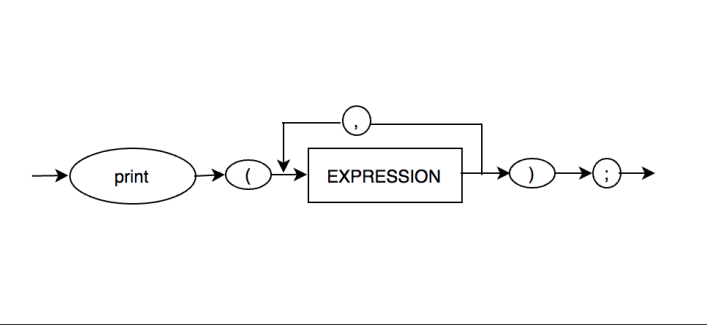


EXP



<p>FACTOR</p>	
<p>ITEM</p>	
<p>TERM</p>	
<p>STATEMENT</p>	

CALL	 <pre>graph LR; Start(()) --> call([call]); call --> id([id]); id --> LP("("); LP --> EXPRESSION[EXPRESSION]; EXPRESSION --> RP(")"); RP --> SEMI(";"); SEMI --> End(())</pre> <p>The flowchart for a CALL statement starts with an entry arrow pointing to an oval labeled 'call'. This is followed by an oval labeled 'id', then an oval labeled '('. An arrow from '(' points to a rectangular box labeled 'EXPRESSION'. From 'EXPRESSION', an arrow points to an oval labeled ')'. Finally, an arrow points to an oval labeled ';', which then leads to the exit arrow.</p>
LOOP	 <pre>graph LR; Start(()) --> do([do]); do --> BODY[BODY]; BODY --> until([until]); until --> LP("("); LP --> EXPRESSION[EXPRESSION]; EXPRESSION --> RP(")"); RP --> End(())</pre> <p>The flowchart for a LOOP statement starts with an entry arrow pointing to an oval labeled 'do'. This is followed by a rectangular box labeled 'BODY', then an oval labeled 'until'. An arrow from 'until' points to an oval labeled '(', which points to a rectangular box labeled 'EXPRESSION'. From 'EXPRESSION', an arrow points to an oval labeled ')', which then leads to the exit arrow.</p>
ASSIGN	 <pre>graph LR; Start(()) --> id([id]); id --> LP("("); LP --> cst_whole([cst_whole]); cst_whole --> RP(")"); RP --> EQ("="); EQ --> EXPRESSION[EXPRESSION]; EXPRESSION --> SEMI(";"); SEMI --> End(())</pre> <p>The flowchart for an ASSIGN statement starts with an entry arrow pointing to an oval labeled 'id'. An arrow from 'id' points to an oval labeled '(', which points to an oval labeled 'cst_whole'. From 'cst_whole', an arrow points to an oval labeled ')'. This is followed by an oval labeled '=', then a rectangular box labeled 'EXPRESSION'. From 'EXPRESSION', an arrow points to an oval labeled ';', which then leads to the exit arrow.</p>
RETURN	 <pre>graph LR; Start(()) --> return([return]); return --> EXPRESSION[EXPRESSION]; EXPRESSION --> SEMI(";"); SEMI --> End(())</pre> <p>The flowchart for a RETURN statement starts with an entry arrow pointing to an oval labeled 'return'. This is followed by a rectangular box labeled 'EXPRESSION'. From 'EXPRESSION', an arrow points to an oval labeled ';', which then leads to the exit arrow.</p>

READ	 <p>The diagram shows a linear sequence of nodes in a control flow graph. It starts with an entry arrow pointing to an oval node labeled 'input'. This is followed by a circular node containing '(', then an oval node labeled 'id', then a circular node containing ')', and finally a circular node containing ';'. An exit arrow points away from the semicolon node.</p>
WRITE	 <p>The diagram shows a control flow graph for a write statement. It begins with an entry arrow pointing to an oval node labeled 'print'. This is followed by a circular node containing '('. An arrow from this node leads to a rectangular node labeled 'EXPRESSION'. From the 'EXPRESSION' node, an arrow points to a circular node containing ')'. Above this arrow is a small circular node containing a comma ','. A feedback loop arrow originates from the arrow between the '(' and 'EXPRESSION' nodes, goes up and around the comma node, and then points down to the arrow between 'EXPRESSION' and ')'. Finally, an arrow from the closing parenthesis node leads to a circular node containing ';', with an exit arrow pointing away from it.</p>

3.3 Semantic Characteristics

- 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/>>