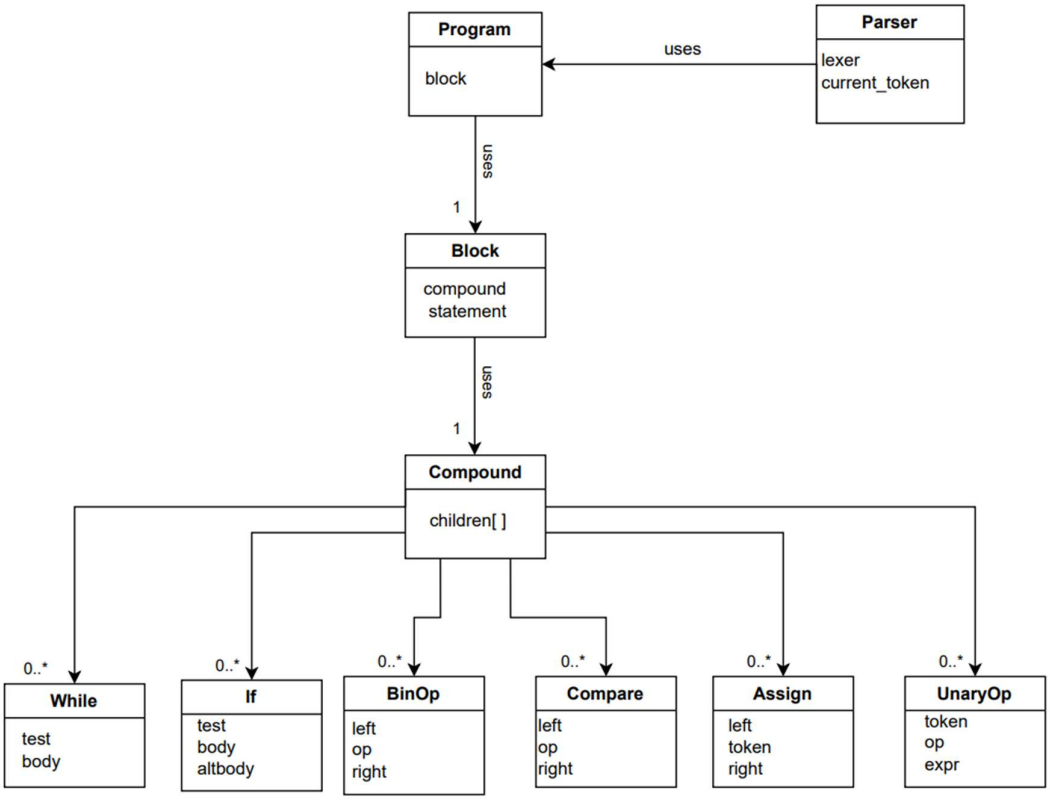
Parser Design Doc:

The code is broken up into unit by the *lexer*.py which the *Parser* class uses to make the an abstract syntax tree for interpretation.

The UML diagram for the AST is as shown below:



When an AST is being made for the code by the *Parser*, it creates a *Program* object which in turn creates a *Block* object which makes a *Compound* object. The Compound object stores all the nodes of all the structures like *If* node, *Assign* node, *BinOp* node, *Compare* node, etc. as a list of nodes in its children[] attribute.

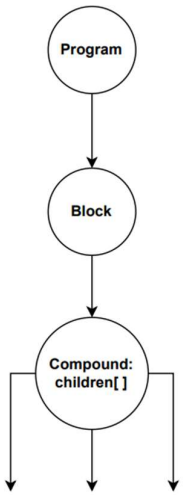
<program> → (<block>) *

<block> → <compound>

<compound> → (<expr>) * | (<parent_exp>) * | (<assign>) * | (<if>) * | (<while>) *

The respective BNFs for <expr>, <parent_exp>, <assign>, <if>, <while> are given below.

The basic top structure which is common to every AST is shown below:



The *Compound* node appends every primitive structure to its children[] attribute.

AST for basic structures is shown below:

Mathematical Expressions:

BNF for Mathematical expressions:

<expr> → <term> ((PLUS | MINUS) <term>)*

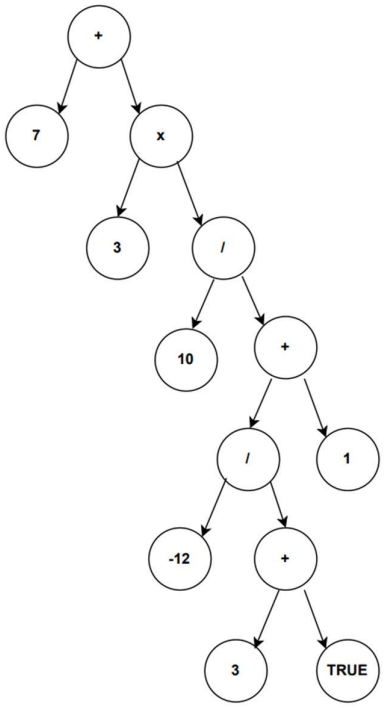
<term> → <factor> ((MUL | DIV) <factor>)*

<factor> → constant | TRUE | FALSE | PLUS factor | MINUS factor | variable | LPAREN <expr> RPAREN

The AST is made using the objects of class *BinOp* liked together based on the expression.

Attributes of class *BinOP* are: left, token, right.

For example, the AST for the expression "7 + 3 * (10 / (-12 / (3 + TRUE) - 1))" is as follows:



Comparison Statements:

BNF for comparison statement is:

<parent_exp> → <expr> (comp_op <expr>) *

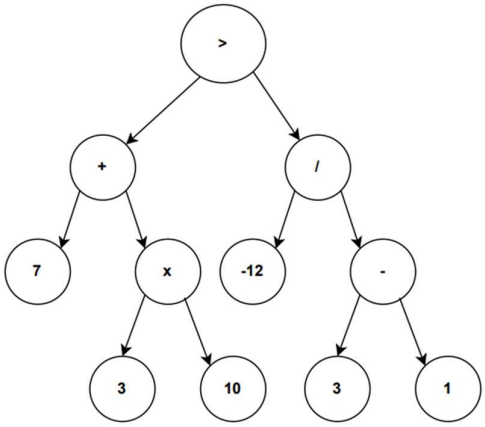
comp_op → == | ~= |<= | >= | < | >

The AST is made using the objects of class *Compare* liked together based on the expression.

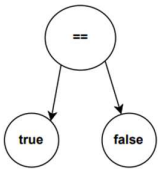
Attributes of class *Compare* are: left, token, right.

AST for the expressions look like:

a) "7 + 3 * 10 > -12 / (3 - 1)"



b) "true == flase"



Assignment Statements:

<assign> → <var> = <parent_exp>

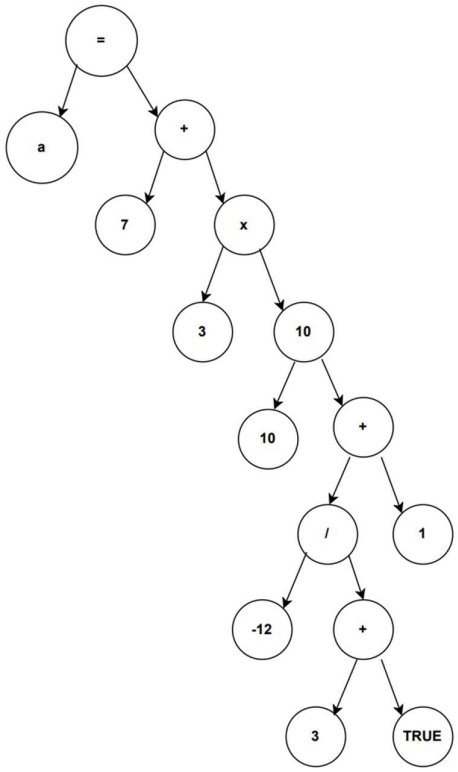
<var> → <identifier>

The AST is made using the objects of class *Assign* liked together based on the given equation.

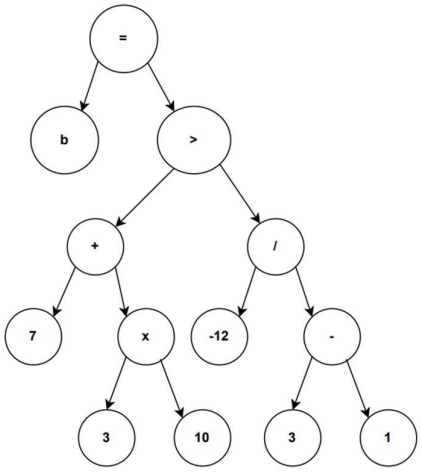
Attributes of class *Assign* are: left, token, right.

AST for the following assignment statements is going to look like:

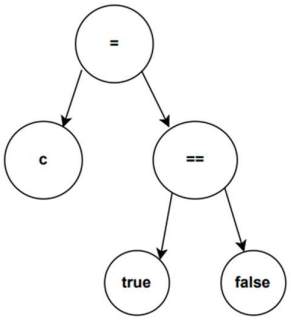
a) "a = 7 + 3 * (10 / (-12 / (3 + TRUE) - 1))"



b) "b = 7 + 3 * 10 > -12 / (3 - 1)"



c) "c = (true == false)"



Conditional Statements:

BNF for conditional statements using if:

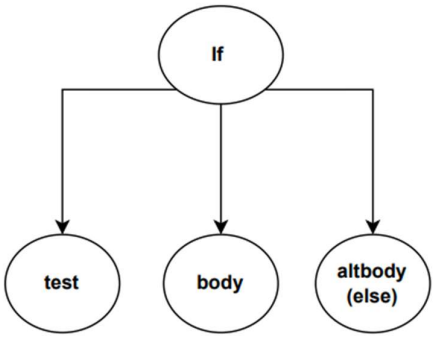
<if> → if <expr> then <block> (elseif <expr> then <block>) * [else <block>]

The AST is made using the objects of class *If* liked together based on the code.

Attributes of class *If* are: test, body, alt.

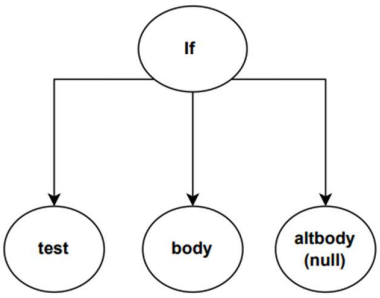
AST for different cases of If blocks is shown below:

i)
```
if (test)
    then
    //body
else
    //altbody
end
```



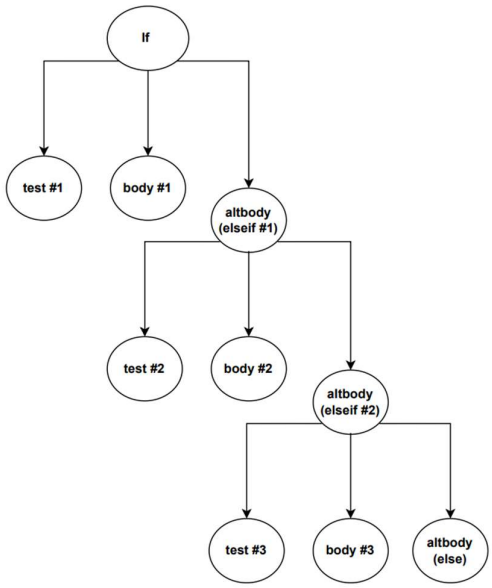
ii)

```
if (test)
    then
    //body
end
```



iii)

```
if (test #1)
    then
    //body #1
elseif (test #2)
    //body #2
elseif (test #3)
    //body #3
else
    //altbody
end
```



In this manner, *if* nodes are made in a recursive manner for interpretation.

Loop Structure:

BNF for repetitional statements using while:

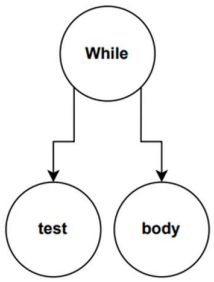
<while> → while <expr> do <block> end

The AST is made using the objects of class *Compare* liked together based on the expression.

Attributes of class *Compare* are: left, token, right.

AST for the following code looks like:

```
while(condition)
        //body
end
```



The *body* is a *block* of code whose BNF <block> is mentioned above.
Hence the body can also include while loops within itself creating nested loops.