SER502 Spring 2024 Team 4

Dev Jani (djani4)
Kalyani Joshi (kjoshi29)
Kirtan Soni (kgsoni)
Lakshmi Kruthi Hosamane Keshava Raman (lhosaman)

Overview

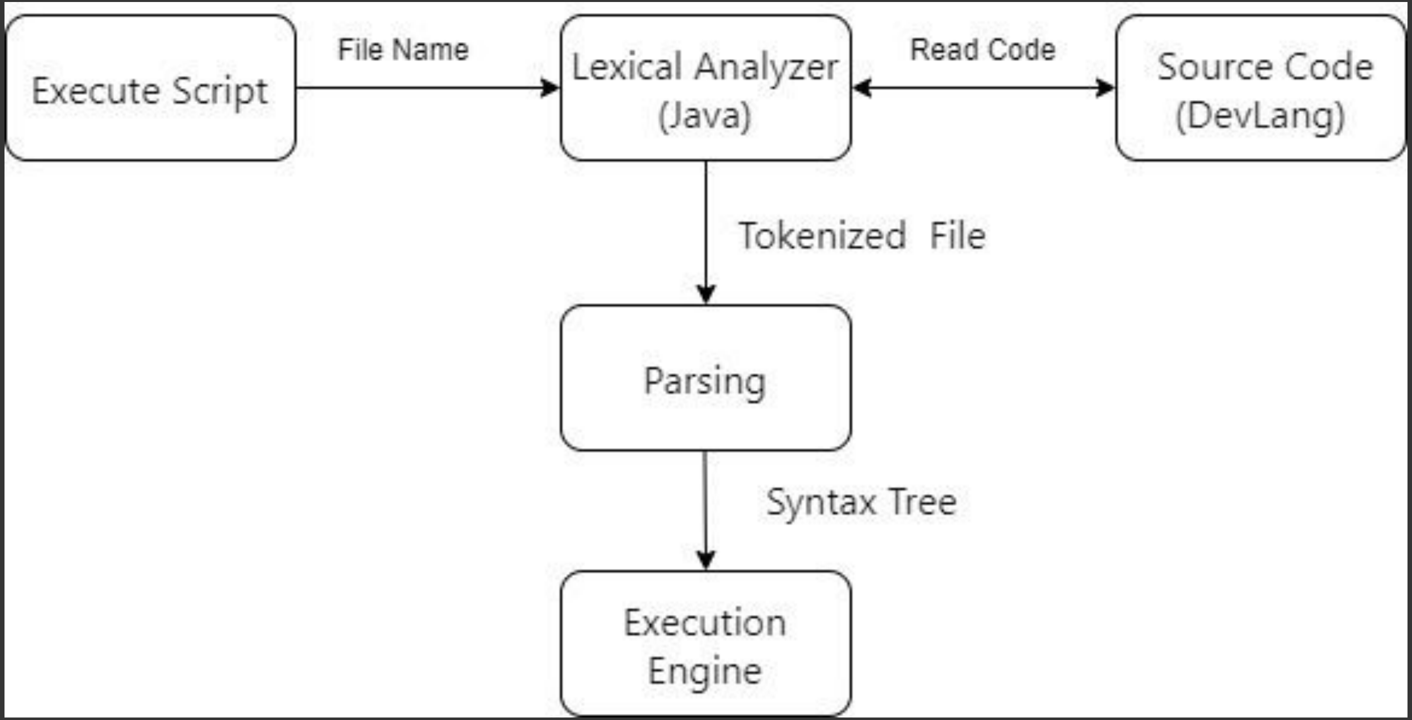
- About the Language
- Project Pipeline
- Grammar
- Lexical Analyzer and Parser
- Runtime Execution Snapshots

1. About the Language

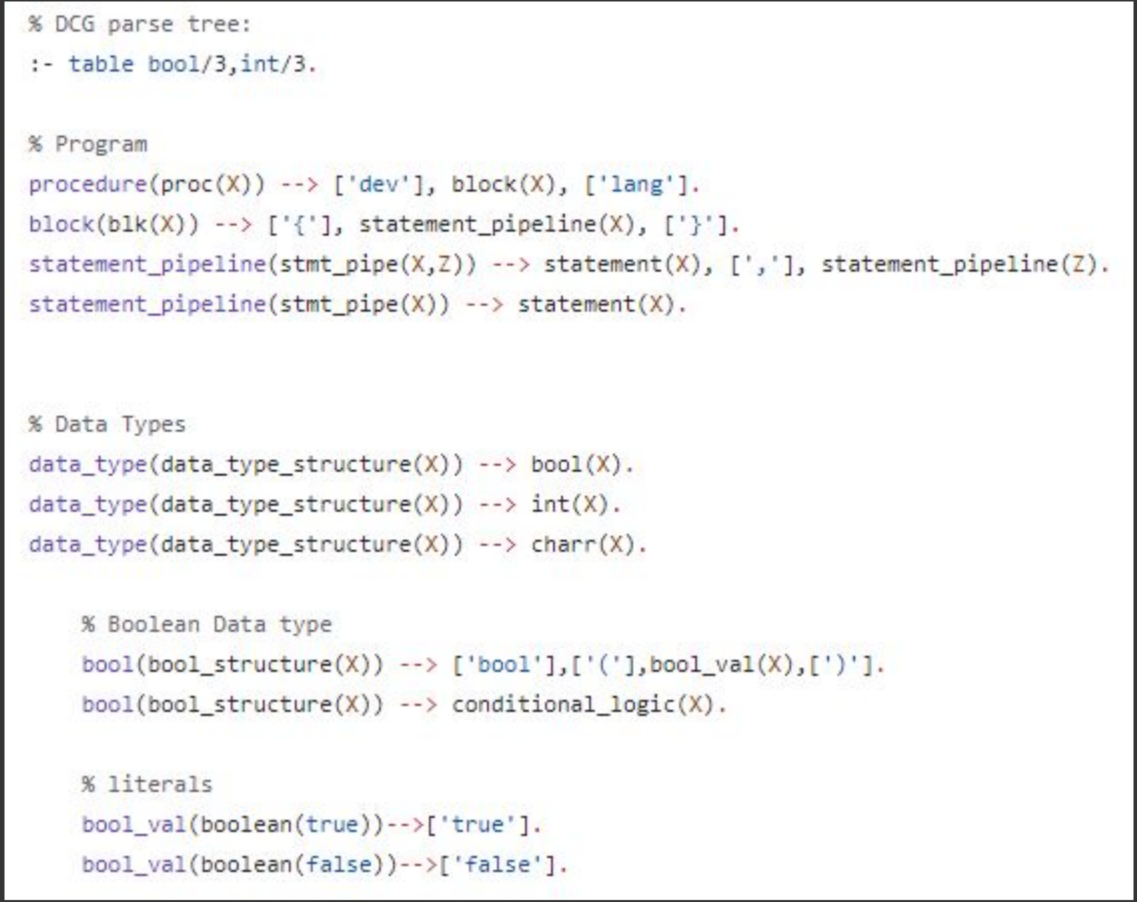
For this project, we created a programming language called DevLang. Some of the details of our programming language are as follows:-

- **The file extension of DevLang is “.dl”.**
- **The Lexer is made using Java Programming Language and the Parser and Evaluator is created using Prolog.**
- **The Lexer is developed using the tool “ANTLR”.**

Program Pipeline

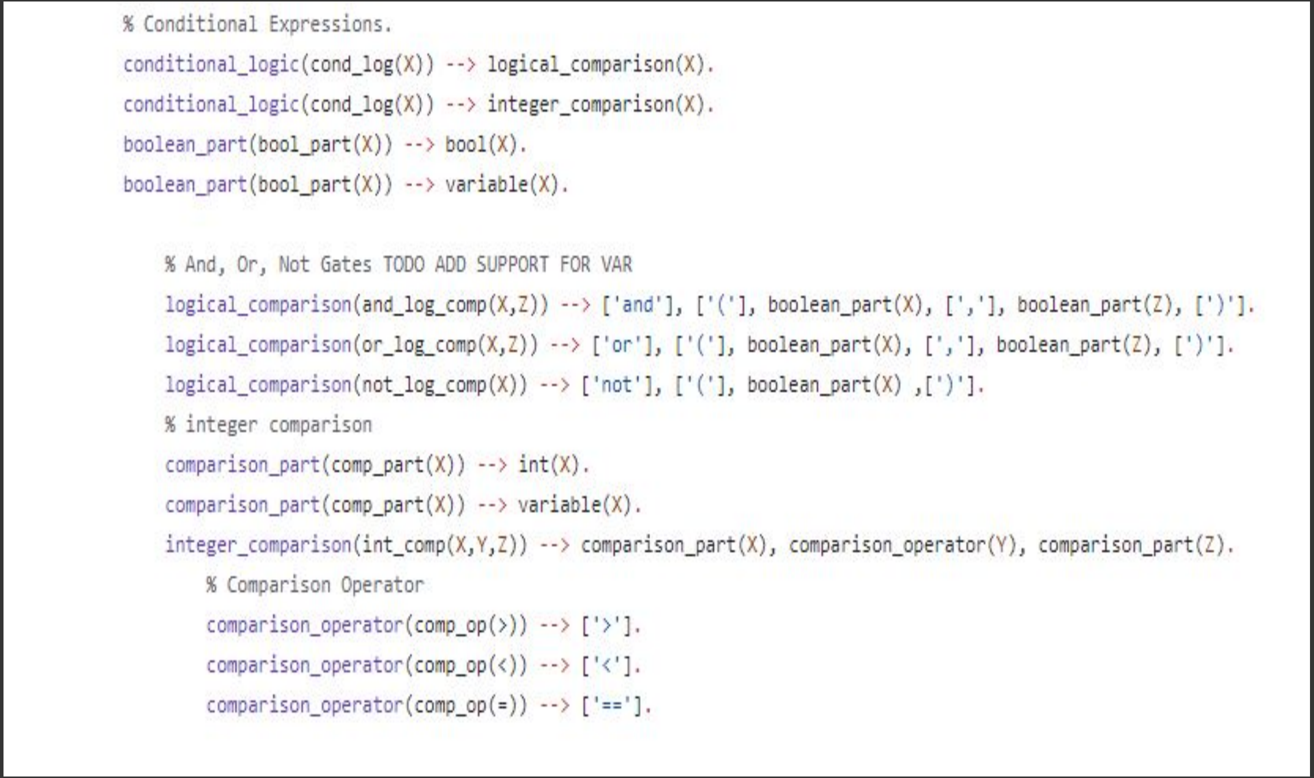


Grammar



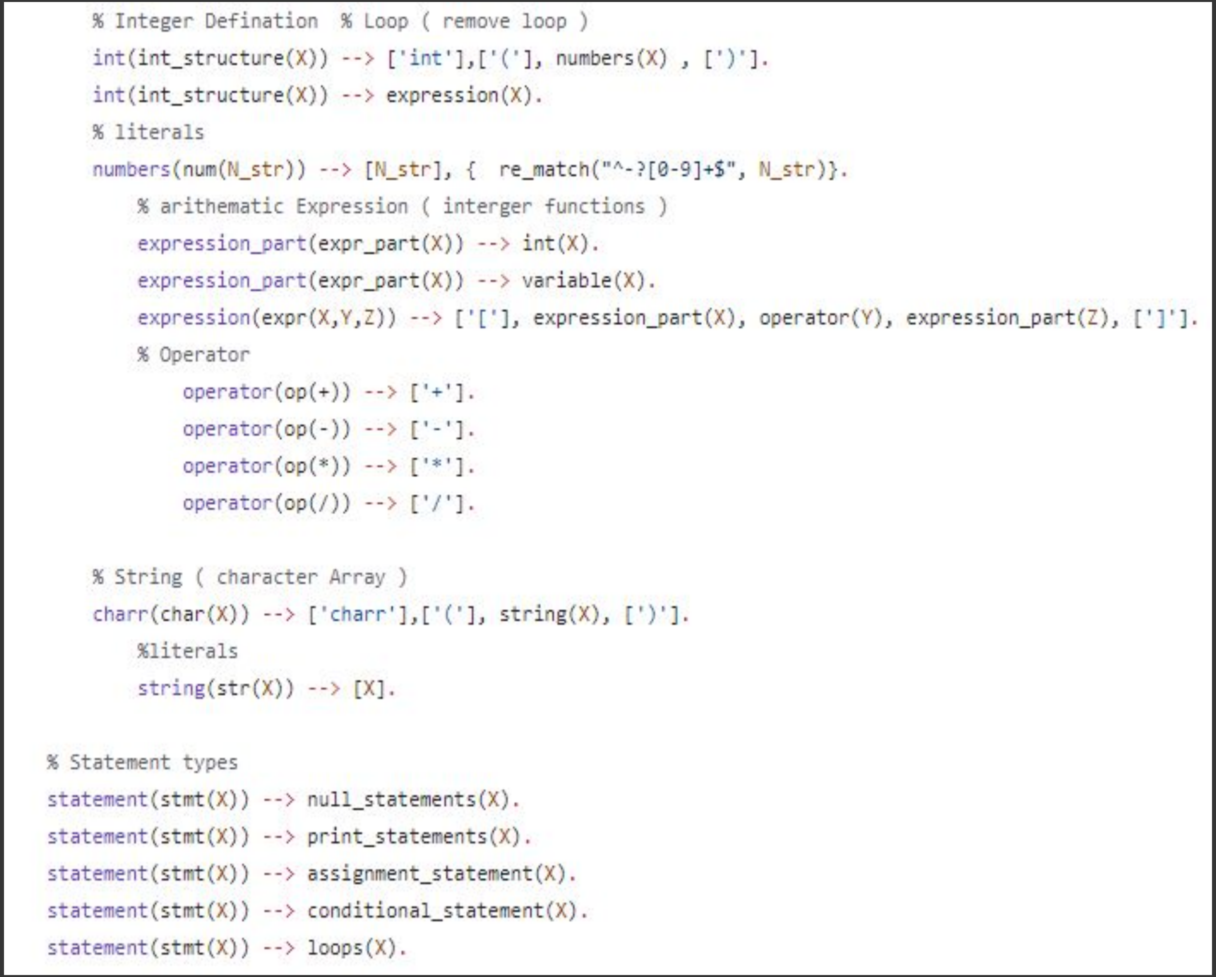
```
% DCG parse tree:
:- table bool/3,int/3.

% Program
procedure(proc(X)) --> ['dev'], block(X), ['lang'].
block(blk(X)) --> ['{'], statement_pipeline(X), ['}'].
statement_pipeline(stmt_pipe(X,Z)) --> statement(X), [','], statement_pipeline(Z).
statement_pipeline(stmt_pipe(X)) --> statement(X).



% Data Types
data_type(data_type_structure(X)) --> bool(X).
data_type(data_type_structure(X)) --> int(X).
data_type(data_type_structure(X)) --> charr(X).

    % Boolean Data type
    bool(bool_structure(X)) --> ['bool'],['('],bool_val(X),[')'].
    bool(bool_structure(X)) --> conditional_logic(X).

    % literals
    bool_val(boolean(true))-->['true'].
    bool_val(boolean(false))-->['false'].
```

Grammar (Cont'd)



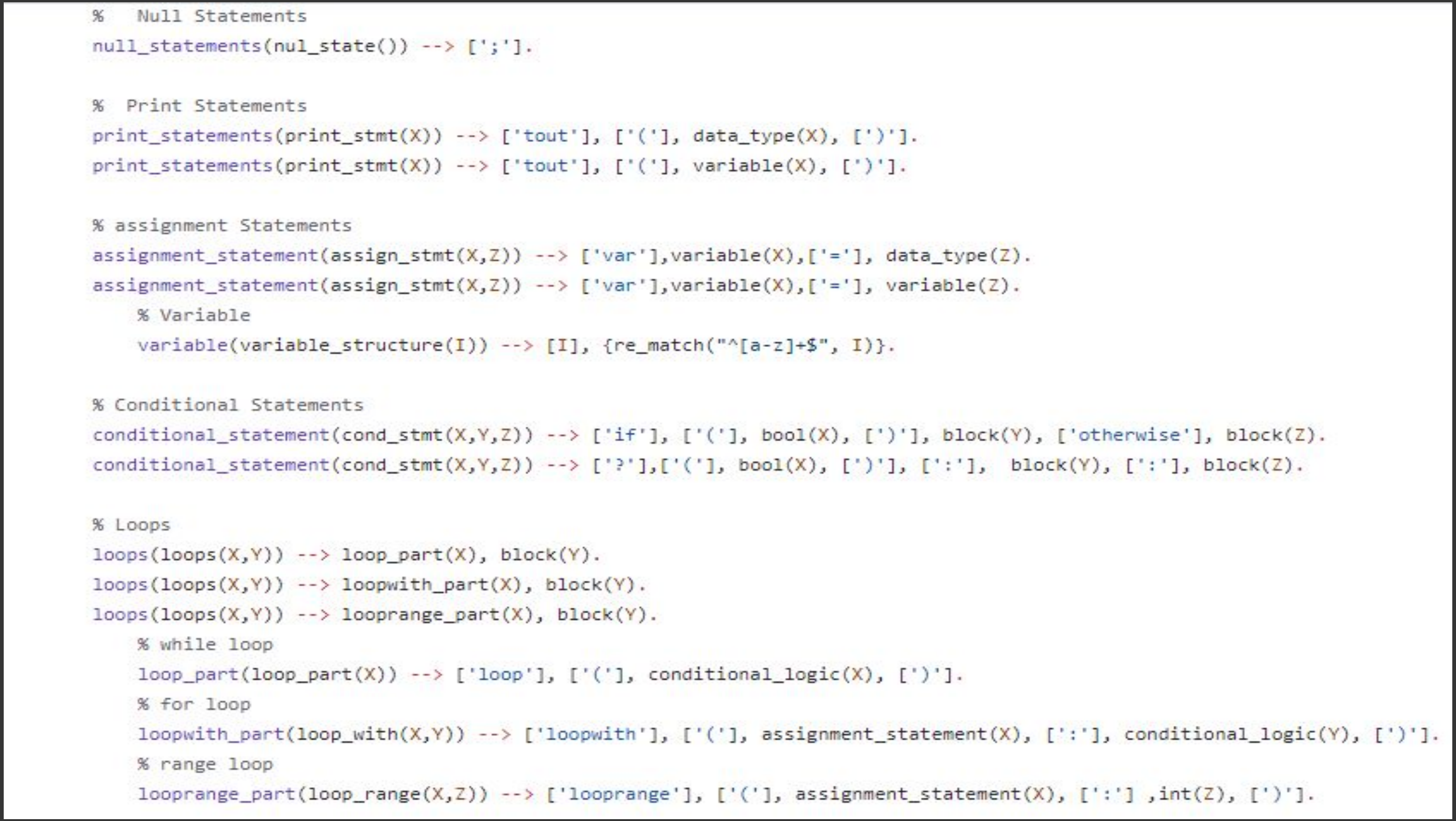
```
% Conditional Expressions.
conditional_logic(cond_log(X)) --> logical_comparison(X).
conditional_logic(cond_log(X)) --> integer_comparison(X).
boolean_part(bool_part(X)) --> bool(X).
boolean_part(bool_part(X)) --> variable(X).

    % And, Or, Not Gates TODO ADD SUPPORT FOR VAR
    logical_comparison(and_log_comp(X,Z)) --> ['and'], ['('], boolean_part(X), [','], boolean_part(Z), [')'].
    logical_comparison(or_log_comp(X,Z)) --> ['or'], ['('], boolean_part(X), [','], boolean_part(Z), [')'].
    logical_comparison(not_log_comp(X)) --> ['not'], ['('], boolean_part(X) ,[')'].
    % integer comparison
    comparison_part(comp_part(X)) --> int(X).
    comparison_part(comp_part(X)) --> variable(X).
    integer_comparison(int_comp(X,Y,Z)) --> comparison_part(X), comparison_operator(Y), comparison_part(Z).
        % Comparison Operator
        comparison_operator(comp_op(>)) --> ['>'].
        comparison_operator(comp_op(<)) --> ['<'].
        comparison_operator(comp_op(=)) --> ['=='].
```

Grammar (Cont’d)



```
  % Integer Defination  % Loop ( remove loop )
  int(int_structure(X)) --> ['int'],['('], numbers(X) , [')'].
  int(int_structure(X)) --> expression(X).
  % literals
  numbers(num(N_str)) --> [N_str], {  re_match("^-?[0-9]+$", N_str)}.
    % arithematic Expression ( interger functions )
    expression_part(expr_part(X)) --> int(X).
    expression_part(expr_part(X)) --> variable(X).
    expression(expr(X,Y,Z)) --> ['['], expression_part(X), operator(Y), expression_part(Z), [']'].
    % Operator
      operator(op(+)) --> ['+'].
      operator(op(-)) --> ['-'].
      operator(op(*)) --> ['*'].
      operator(op(/)) --> ['/'].

  % String ( character Array )
  charr(char(X)) --> ['charr'],['('], string(X), [')'].
    %literals
    string(str(X)) --> [X].

% Statement types
statement(stmt(X)) --> null_statements(X).
statement(stmt(X)) --> print_statements(X).
statement(stmt(X)) --> assignment_statement(X).
statement(stmt(X)) --> conditional_statement(X).
statement(stmt(X)) --> loops(X).
```

Grammar (Cont'd)

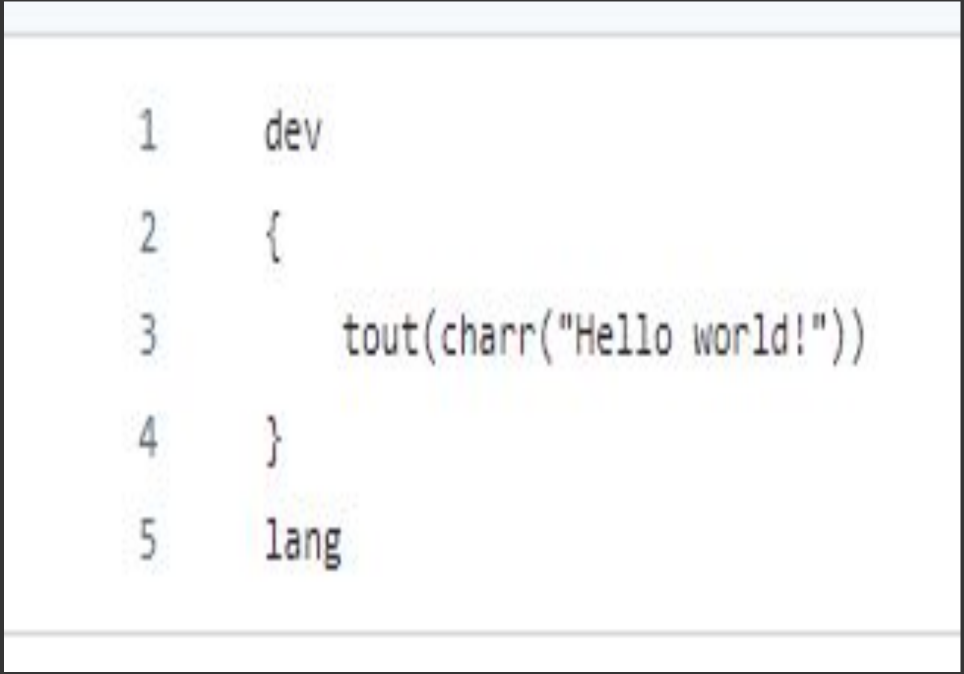


```
%   Null Statements
null_statements(nul_state()) --> [';'].

%  Print Statements
print_statements(print_stmt(X)) --> ['tout'], ['('], data_type(X), [')'].
print_statements(print_stmt(X)) --> ['tout'], ['('], variable(X), [')'].

% assignment Statements
assignment_statement(assign_stmt(X,Z)) --> ['var'],variable(X),['='], data_type(Z).
assignment_statement(assign_stmt(X,Z)) --> ['var'],variable(X),['='], variable(Z).
    % Variable
    variable(variable_structure(I)) --> [I], {re_match("^[a-z]+$", I)}.

% Conditional Statements
conditional_statement(cond_stmt(X,Y,Z)) --> ['if'], ['('], bool(X), [')'], block(Y), ['otherwise'], block(Z).
conditional_statement(cond_stmt(X,Y,Z)) --> ['?'],['('], bool(X), [')'], [':'],  block(Y), [':'], block(Z).

% Loops
loops(loops(X,Y)) --> loop_part(X), block(Y).
loops(loops(X,Y)) --> loopwith_part(X), block(Y).
loops(loops(X,Y)) --> looprange_part(X), block(Y).
    % while loop
    loop_part(loop_part(X)) --> ['loop'], ['('], conditional_logic(X), [')'].
    % for loop
    loopwith_part(loop_with(X,Y)) --> ['loopwith'], ['('], assignment_statement(X), [':'], conditional_logic(Y), [')'].
    % range loop
    looprange_part(loop_range(X,Z)) --> ['looprange'], ['('], assignment_statement(X), [':'] ,int(Z), [')'].
```

Lexical Analyzer and Parser

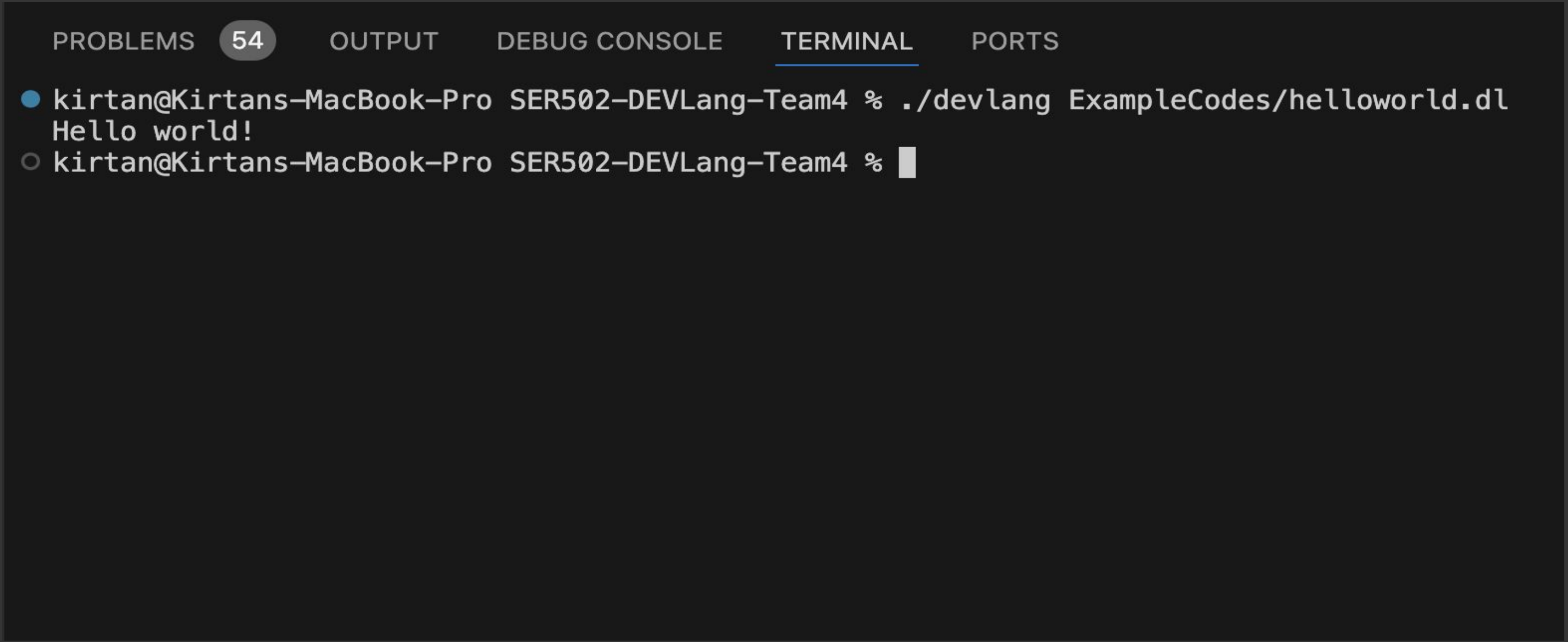
- **The lexical analyzer takes the input program file with the .dl extension and generates a list of tokens by removing spaces, tabs and newlines.**
- **The list of tokens which we get from the terminal are given to the parser.**
- **Taking the defined grammar rules as reference, the parser converts the list of tokens and generates the syntax tree.**

Sample Program - Print Statement



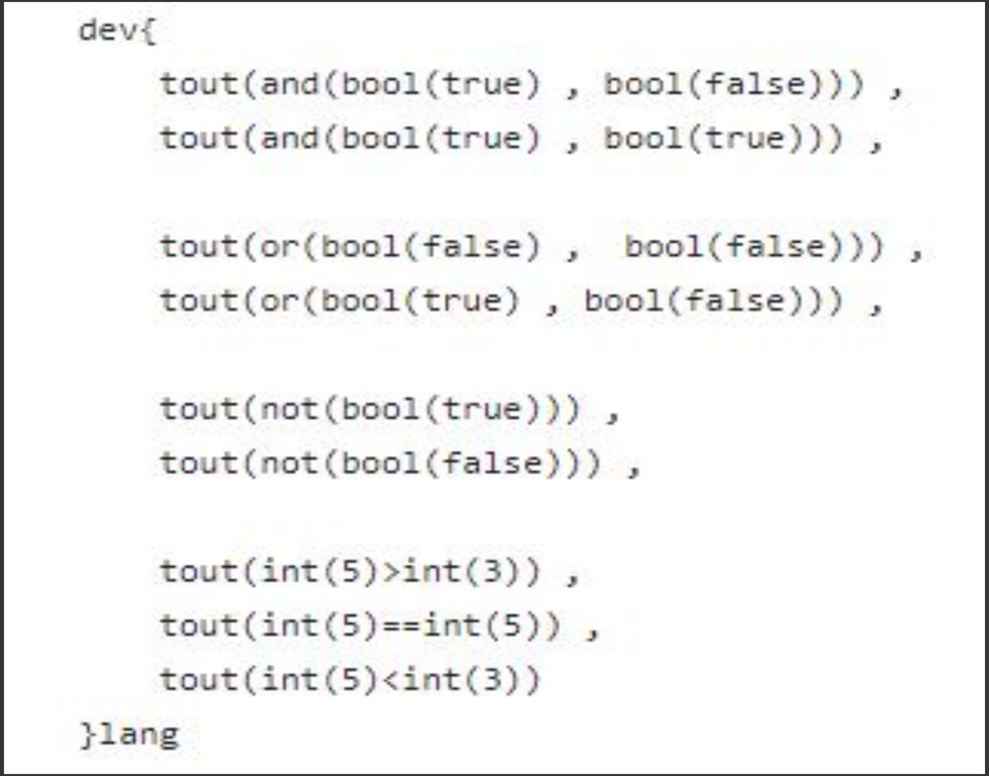
```
dev
{
    tout(charr("Hello world!"))
}
lang
```

Runtime Execution - Print Statement



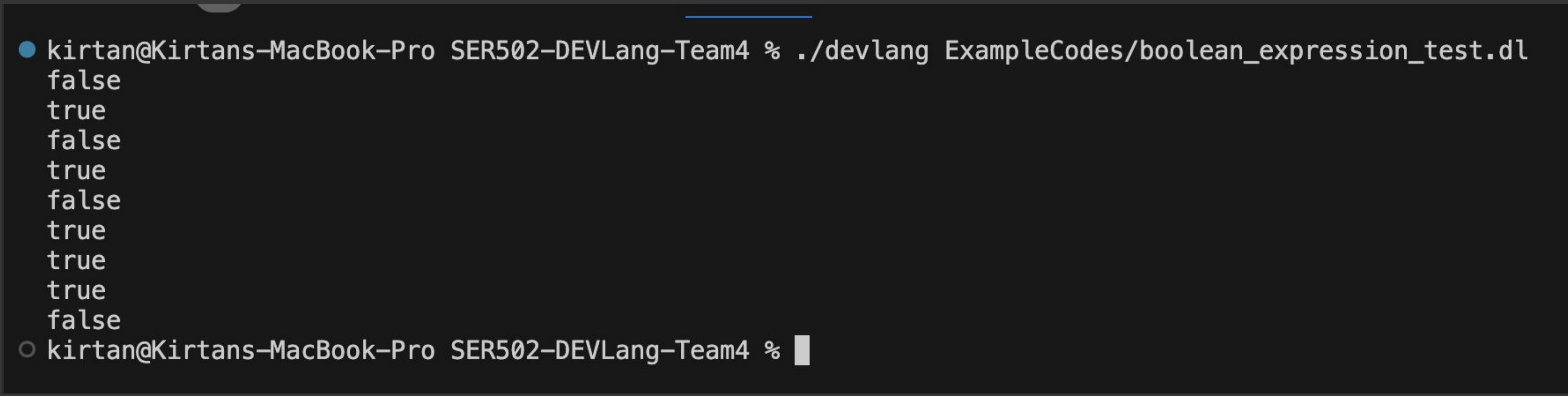
```
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/helloworld.dl
Hello world!
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 %
```

Sample Program - Boolean Statements (AND, OR, NOT, Greater Than, Less Than, Equal To)



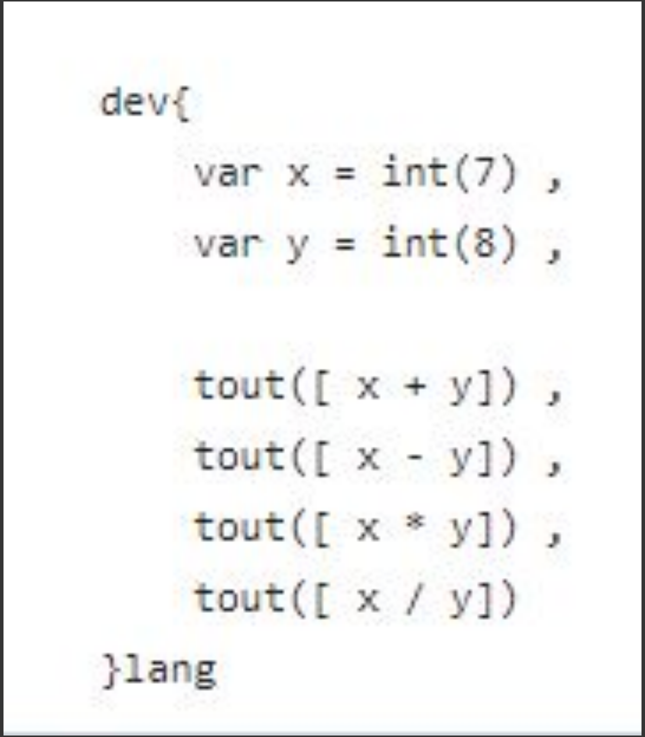
```
dev{
    tout(and(bool(true) , bool(false))) ,
    tout(and(bool(true) , bool(true))) ,

    tout(or(bool(false) ,  bool(false))) ,
    tout(or(bool(true) , bool(false))) ,

    tout(not(bool(true))) ,
    tout(not(bool(false))) ,

    tout(int(5)>int(3)) ,
    tout(int(5)==int(5)) ,
    tout(int(5)<int(3))
}lang
```

Runtime Execution - Boolean Statements (AND, OR, NOT, Greater Than, Less Than, Equal To)



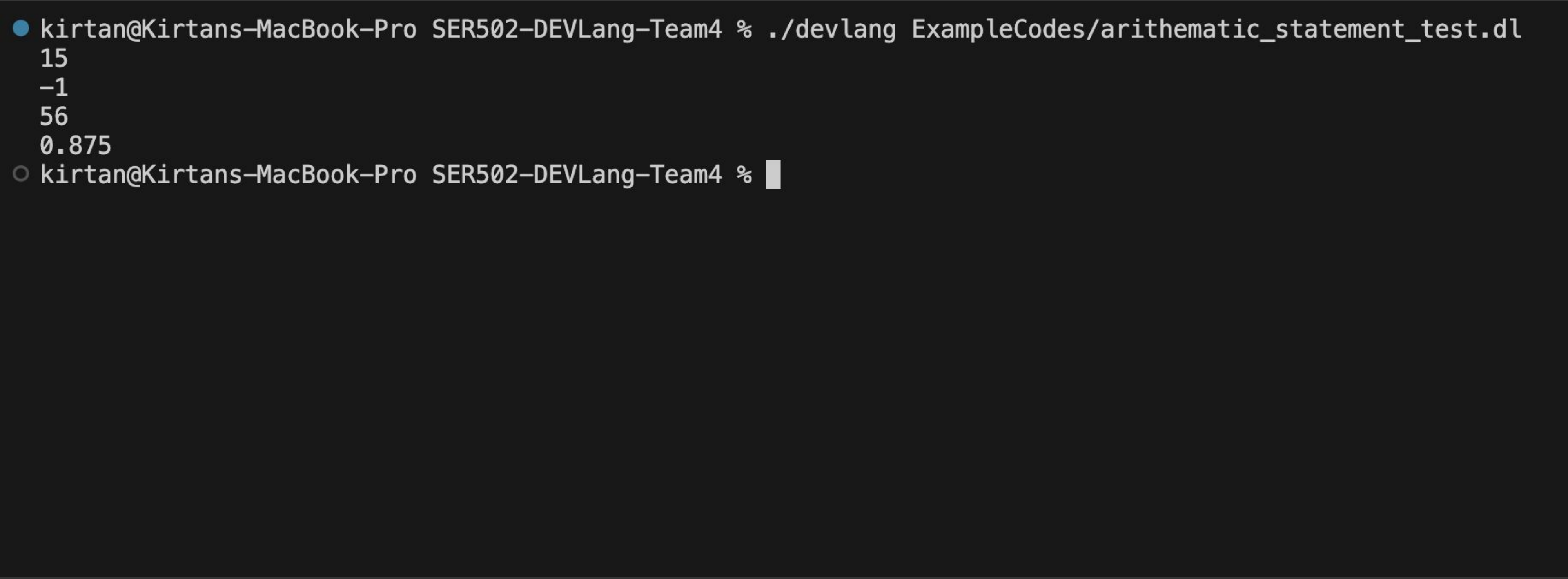
```
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/boolean_expression_test.dl
false
true
false
true
false
true
true
true
false
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % 
```

Sample Program - Arithmetic Statements



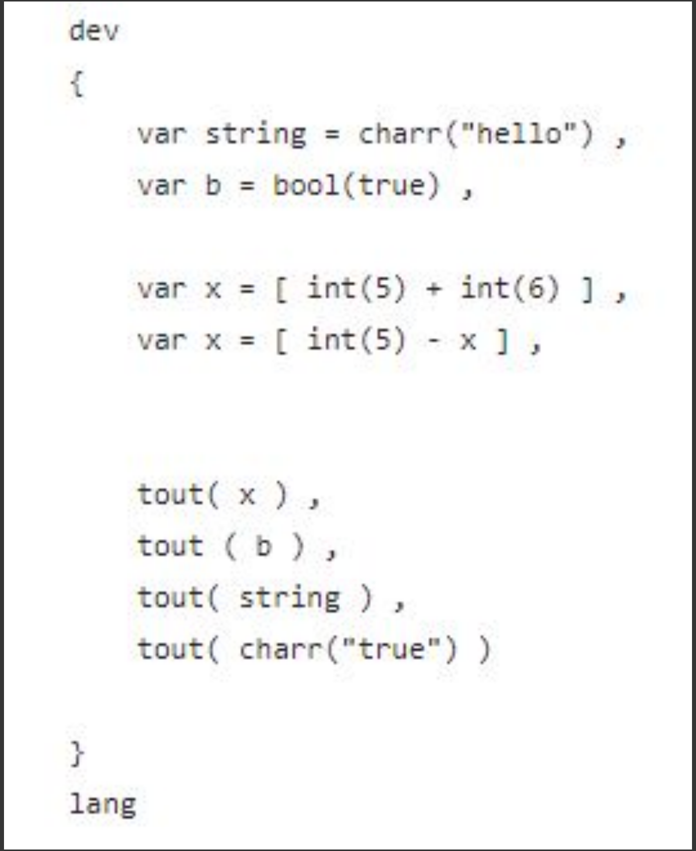
```
dev{
    var x = int(7) ,
    var y = int(8) ,

    tout([ x + y]) ,
    tout([ x - y]) ,
    tout([ x * y]) ,
    tout([ x / y])
}lang
```

Runtime Execution - Arithmetic Statements



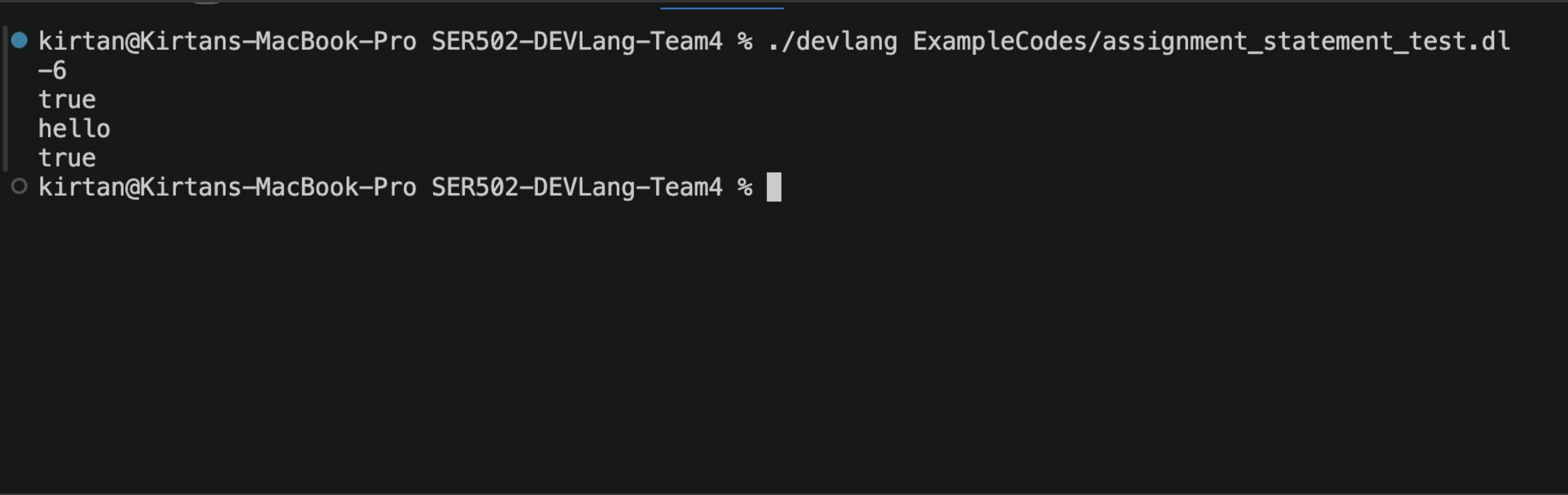
```
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/arithematic_statement_test.dl
15
-1
56
0.875
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 %
```

Sample Program - Assignment Statements



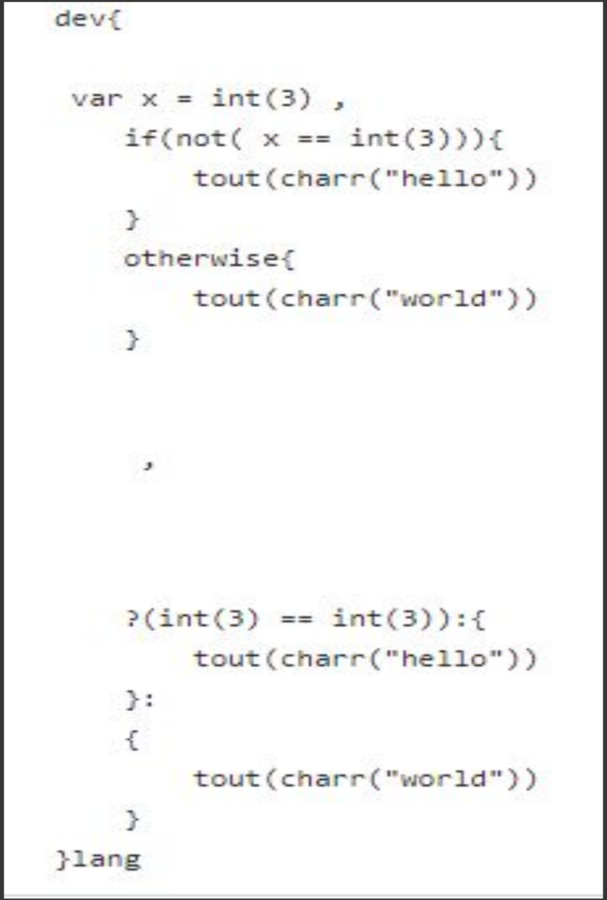
```
dev
{
    var string = charr("hello") ,
    var b = bool(true) ,

    var x = [ int(5) + int(6) ] ,
    var x = [ int(5) - x ] ,


    tout( x ) ,
    tout ( b ) ,
    tout( string ) ,
    tout( charr("true") )

}
lang
```

Runtime Execution - Assignment Statements



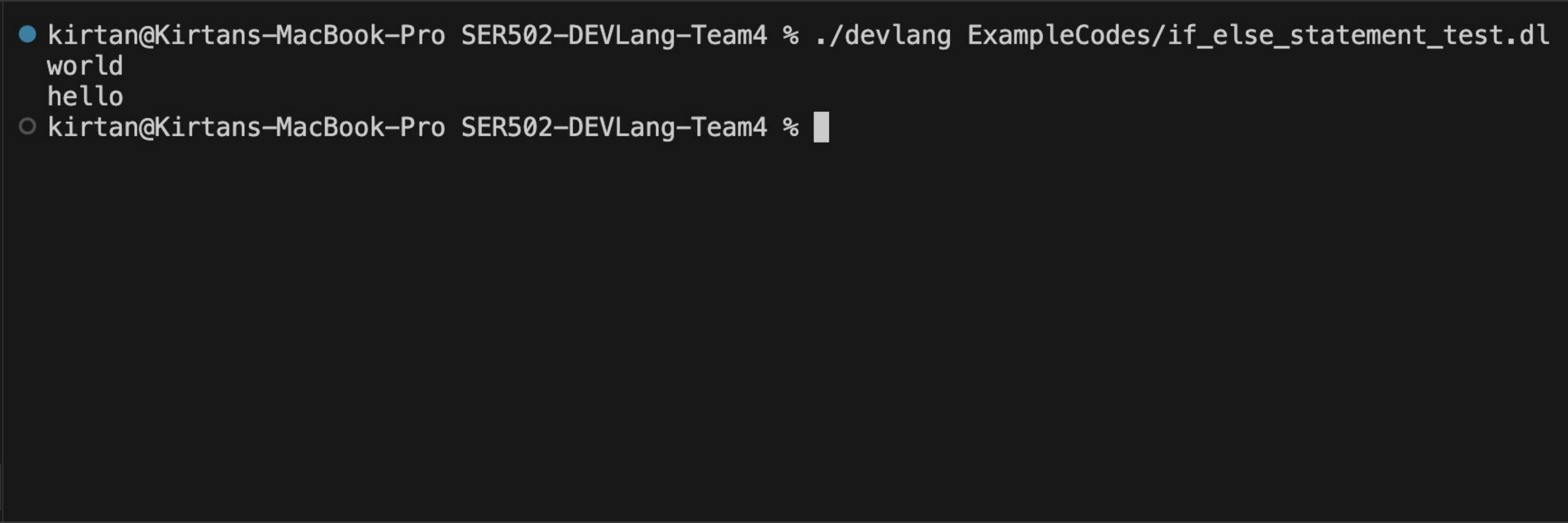
```
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/assignment_statement_test.dl
-6
true
hello
true
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % 
```

Sample Program - If Else and Ternary Statements



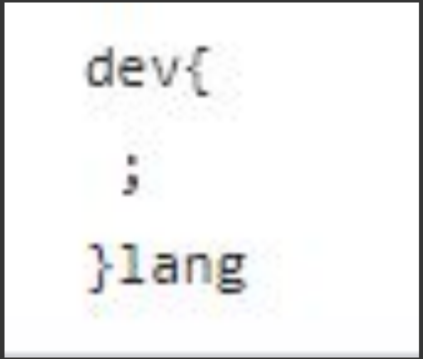
```
dev{

 var x = int(3) ,
    if(not( x == int(3))){
        tout(charr("hello"))
    }
    otherwise{
        tout(charr("world"))
    }



     ,




    ?(int(3) == int(3)):{
        tout(charr("hello"))
    }:
    {
        tout(charr("world"))
    }
}lang
```

Runtime Execution - If Else and Ternary Statements



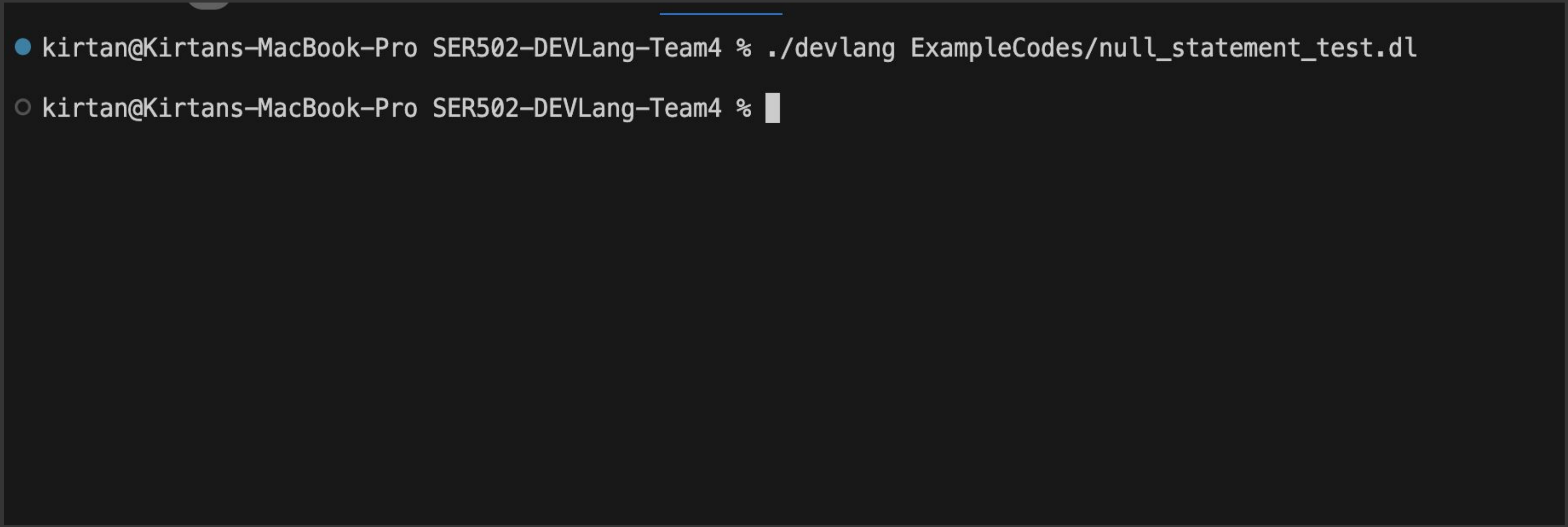
```
● kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/if_else_statement_test.dl
  world
  hello
○ kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % 
```

Sample Program - Null Statement



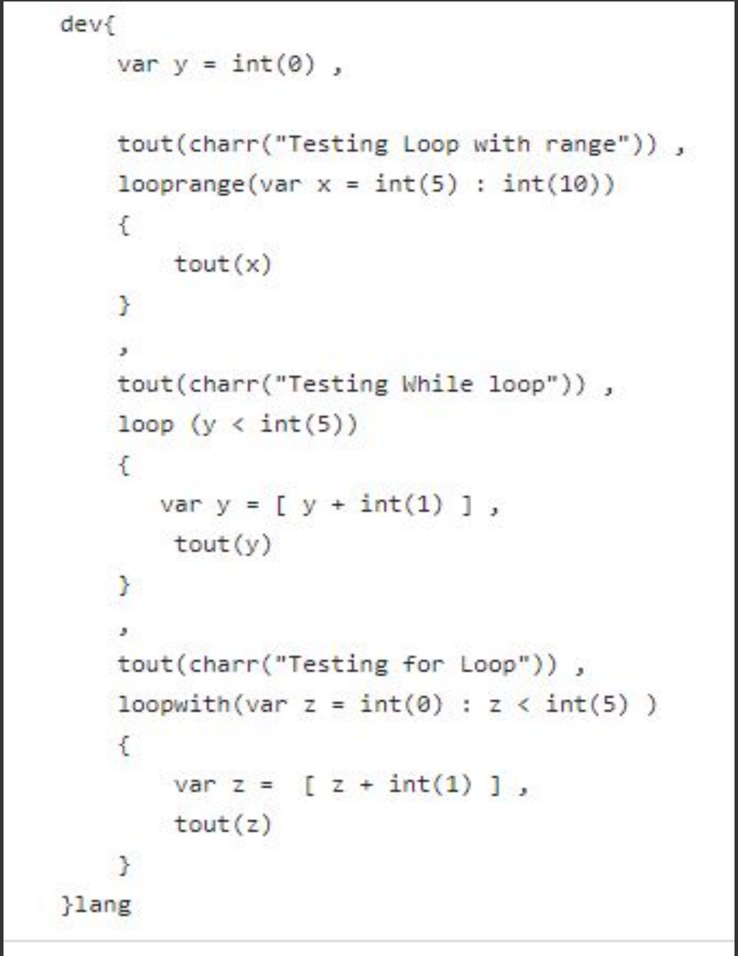
```
dev{
 ;
}lang
```

Runtime Execution - Null Statement



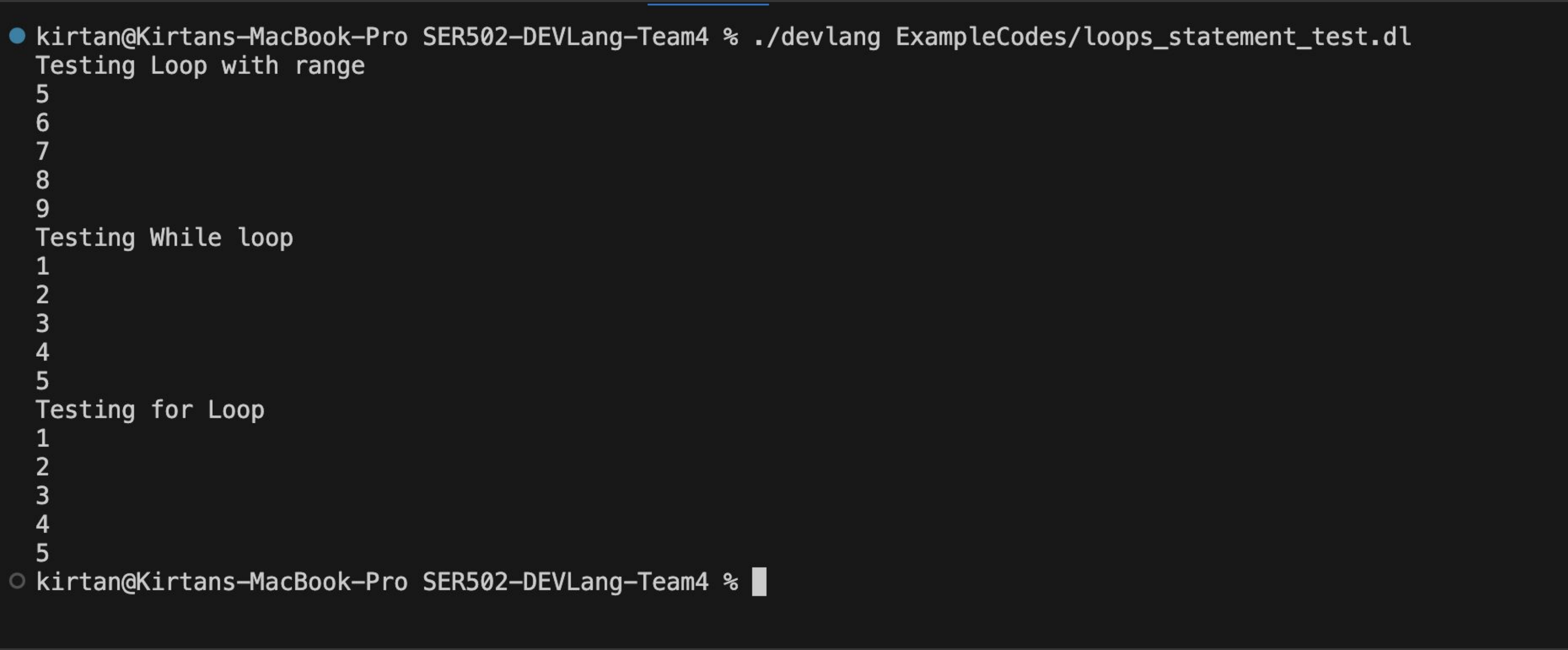
```
● kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/null_statement_test.dl

○ kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % 
```

Sample Program - For Loop, While Loop, For Range Loop



```
dev{
    var y = int(0) ,

    tout(charr("Testing Loop with range")) ,
    looprange(var x = int(5) : int(10))
    {
        tout(x)
    }
    ,
    tout(charr("Testing While loop")) ,
    loop (y < int(5))
    {
       var y = [ y + int(1) ] ,
        tout(y)
    }
    ,
    tout(charr("Testing for Loop")) ,
    loopwith(var z = int(0) : z < int(5) )
    {
        var z =  [ z + int(1) ] ,
        tout(z)
    }
}lang
```

Runtime Execution



```
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 % ./devlang ExampleCodes/loops_statement_test.dl
Testing Loop with range
5
6
7
8
9
Testing While loop
1
2
3
4
5
Testing for Loop
1
2
3
4
5
kirtan@Kirtans-MacBook-Pro SER502-DEVLang-Team4 %
```

Thank You!