PORYGON (Team 11)

SER 502

Instructor:
Dr. Ajay Bansal

Team Members

Anmol Mallikarjun Nemagouda

Git Username:
AnmolNemagouda

Chandrakanth Dhanunjai Chintala

Git Username:
Cchinta2

Kaumudi Degekar Gulbarga

Git Username:
KaumudiDG

Rakshilkumar Modi

Git Usernames:
rakshil14
rakshil14-2

Overview

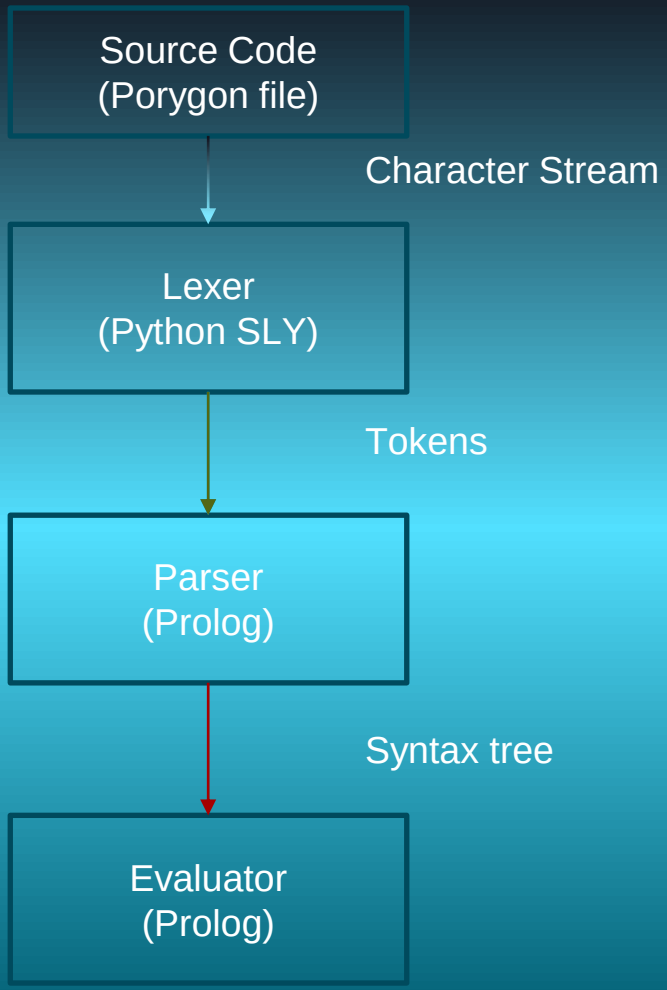
- About the Language.
- Design.
- Design Components.
- Feature list.
- General Rules of the program
- Grammar
- Snapshots of the Programming Language

1 ABOUT THE LANGUAGE

About the Language

- Name: Porygon
- Extension: .pgon
- Paradigm: Imperative
- Programming languages used: Python and Prolog.


2
DESIGN


Source Code
(Porygon file)
Character Stream
Lexer
(Python SLY)
Tokens
Parser
(Prolog)
Syntax tree
Evaluator
(Prolog)

3

DESIGN COMPONENTS

Design Components

- Script: A script that helps to run the Porygon program file.
- Source Code: This is the source code for the newly developed language (Porygon) that needs to be executed.
- Lexer: We will be using Python (SLY) as a Lexer for this project. It accepts the source code as an input and breaks down the given code into tokens. The tokens will be generated as a list which will then be passed on to Prolog for parsing.

Design Components

- Parser: We be using Prolog as a parser for this project. It will take the token list from the lexer and generate a parse tree as per the defined grammar.

- Evaluator: This is used to evaluate the parse tree generated by the parser and execute the instructions.

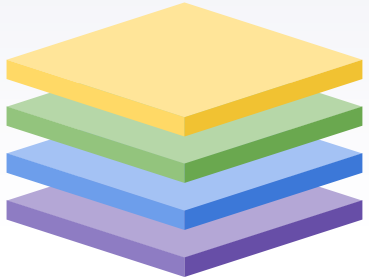
4

FEATURE LIST

Features list

Declarations:

- Constant declaration.
- Plain declaration.
- Variable declaration.



Commands:

- If
- If else
- If else ladder
- For value in range
- Traditional for loop
- While loop
- Assignment
- Print Statement
- Print in newline

Features list

Assignment:

- Initial assignment.
- Declarative assignment.
- Shorthand Assignment.

Operations:

- Arithmetic (addition, subtraction, multiplication, division.).
- Modulus.

Operations(cont.):

- Increment.
- Decrement.
- Square of a number.
- Square root.
- Cube of a number.
- Cube root.
- Exponent.
- String length.
- Sting Concatenation.

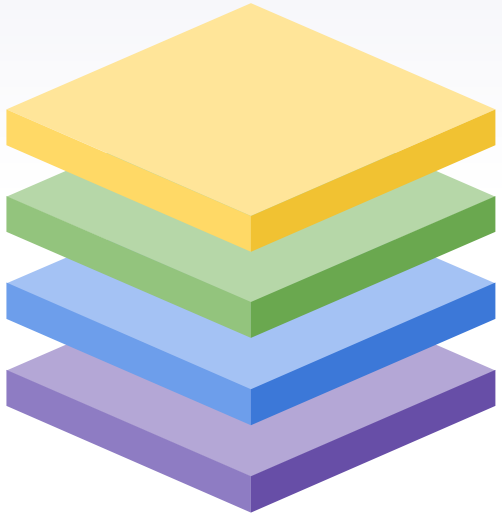
Features list

Operations(Cont.):

- Ternary
- Boolean operations

Additional features:

- Error Handling.
- Type Checking.



5

GENERAL RULES OF THE PROGRAM

General Rules of the Program

Data Types:

- Integers
- Strings
- Floating point numbers
- Boolean expressions.

Keywords:

- const
- if
- else

Keywords(cont.):

- elif
- for
- while
- and
- or
- not
- printnl
- print
- in

General Rules of the Program

Keywords (cont.):

- true
- false
- sq
- sqrt
- cb
- cbrt
- mod
- range
- int

Keywords(cont.):

- float
- bool
- string
- strlen

General Rules of the Program

Basic rules:

- Each program begins and ends with curly braces ‘{}’.
- Each declaration and command line must end with a semi colon ‘;’.
- String Values must be enclosed within double-quotes (“ “).
- Comments should start with ‘#’.

General Rules of the Program

Variable rules:

- Variable names should always begin with a lowercase letter.
- Variable names can be alphanumeric and can contain '_' , however they cannot end with '_'.
- Variable names can also contain uppercase letters.
- Initial declaration should always be done before commands.

Data types(cont.):

- Default values for integer and float are '0' and '0.0' respectively.
- Only integer operations will be performed with 'int' keyword and float operations with 'float' keyword.
- Float values will always be returned for square root and cube root operations.

6

GRAMMAR

```
%:-use_rendering(svgtree).
:-table expr/3,factor/3,term/3,boolcondition/3, and_condition/3.

%Programming block begins here
block(t_blk(D,C))--> ['{'],decl(D),commandlist(C),['}'].

% Declaration statments here for constants,variables etc.

decl(t_decl(D,DL))--> decls(D),[';'],decl(DL).
decl(t_decl(D)) --> decls(D),[';'].

decls(D)--> constassign(D).
decls(D)--> declassign(D).
decls(D)--> plainassign(D).

constassign(t_const_flt(C,F)) --> ['const'], ['float'], variablename(C),['='],floatvalue(F).
constassign(t_const_int_e(C,Expr)) --> ['const'], ['int'], variablename(C),['='],expr(Expr).
constassign(t_const_str_e(C,Expr)) --> ['const'], ['string'], variablename(C),['='],expr(Expr).
constassign(t_const_bool_e(C,Expr)) --> ['const'], ['bool'], variablename(C),['='],expr(Expr).
constassign(t_const_flt_e(C,Expr)) --> ['const'], ['float'], variablename(C),['='],expr(Expr).

declassign(t_int(Var,Value)) --> ['int'], variablename(Var),['='],expr(Value).
declassign(t_str(Var,Value)) --> ['string'], variablename(Var),['='],expr(Value).
declassign(t_bool(Var,Value)) --> ['bool'], variablename(Var),['='],expr(Value).
declassign(t_flt(Var,Value)) --> ['float'], variablename(Var),['='],expr(Value).
```

```
plainassign(t_int_decl(Var)) --> ['int'], variablename(Var).
plainassign(t_str_decl(Var)) --> ['string'], variablename(Var).
plainassign(t_bool_decl(Var)) --> ['bool'], variablename(Var).
plainassign(t_flt_decl(Var)) --> ['float'], variablename(Var).

% Commands List start here
commandlist(t_cmd(PlainCmnd,CmndList)) --> plaincommand(PlainCmnd),[;],commandlist(CmndList).
commandlist(t_cmd(PlainCmnd,CmndList)) --> blockcommand(PlainCmnd),commandlist(CmndList).
commandlist(t_cmd(PlainCmnd)) --> plaincommand(PlainCmnd), [;].
commandlist(t_cmd(BlkCmnd)) --> blockcommand(BlkCmnd).

% Declaration of plain commands
plaincommand(plain_assign(Assign)) --> assignment(Assign).
plaincommand(plain_ternary(Ternary)) --> ternary(Ternary).
plaincommand(plain_print(Print)) --> printStmt(Print).
plaincommand(plain_strlen(StrLen)) --> strlen(StrLen).
plaincommand(plain_increment(Value)) -->increment_operation(Value).
plaincommand(plain_decrement(Value)) -->decrement_operation(Value).

%Separeted syntax structures which need {} so that they do not need ; too.
blockcommand(blkcmd(If)) --> ifcommand(If).
blockcommand(blkcmd(IfElse)) --> ifelsecommand(IfElse).
blockcommand(blkcmd(IfELseLadder)) --> ifELseLaddercommand(IfELseLadder).
blockcommand(blkcmd(While)) --> whilecommand(While).
blockcommand(blkcmd(For)) --> forcommand(For).
blockcommand(blkcmd(ForInRange)) --> forinrangecommand(ForInRange).
```

```
% Declaration of types of assignment
assignment(assign(A))--> initialassignment(A).
%assignment(assign(A))--> declassign(A).
assignment(assign(A))--> shorthandAssign(A).

shorthandAssign(shassignadd(Var,Expr))--> variablename(Var),['+='],expr(Expr).
shorthandAssign(shassignsub(Var,Expr))--> variablename(Var),['-='],expr(Expr).
shorthandAssign(shassignmul(Var,Expr))--> variablename(Var),['*='],expr(Expr).
shorthandAssign(shassigndiv(Var,Expr))--> variablename(Var),['/='],expr(Expr).
shorthandAssign(shassignexpo(Var,Expr))--> variablename(Var),['^='],expr(Expr).

initialassignment(iassign(Var,Expr))--> variablename(Var),['='],expr(Expr).

% EXPRESSIONS STARTS HERE

expr(addition(X,Y))--> expr(X),['+'],term(Y).
expr(subtraction(X,Y))--> expr(X),['-'],term(Y).
expr(X)--> term(X).

term(multiplication(X,Y))--> term(X),['*'],factor(Y).
term(division(X,Y))--> term(X),['/'],factor(Y).
term(modulus(X,Y))--> term(X),['mod'],factor(Y).
term(X)--> factor(X).
```

```
factor(exponent(X,Y))--> factor(X),['^'],exponent(Y).
factor(X)--> exponent(X).

exponent(S)--> square(S).
exponent(Sr)--> squareRoot(Sr).
exponent(C)--> cube(C).
exponent(Cr)--> cubeRoot(Cr).
exponent(X)--> ['('],expr(X),[')'].
exponent(A)--> initialassignment(A).
exponent(Var)--> variablename(Var).
exponent(N)--> num(N).
exponent(T) --> ternary(T).
exponent(B) --> boolvalue(B).
exponent(S) -->stringvalue(S).
exponent(I)--> increment_operation(I).
exponent(Dec)--> decrement_operation(Dec).
exponent(B) --> boolcondition(B).
exponent(S) --> strlen(S).

square(square(S))--> ['sq'],['('],expr(S),[')'].
squareRoot(squareroot(Sr))--> ['sqrt'],['('],expr(Sr),[')'].
cube(cube(C))--> ['cube'],['('],expr(C),[')'].
cubeRoot(cuberoot(S))--> ['cbrt'],['('],expr(S),[')'].
```

```
%Boolean Expressions start here including logical operators 'and', 'or', 'not'.

boolcondition(or(X,Y))--> boolcondition(X),['or'],and_condition(Y).
boolcondition(X) --> and_condition(X).
and_condition(and(X,Y))--> and_condition(X),['and'],condition(Y).
and_condition(X) --> condition(X).
condition(not(X))--> ['not'],condition(X).
condition(X)--> ['('],boolcondition(X),[')'].
condition(InitAssign) --> initialassignment(InitAssign).
condition(equivalance(Expr1,Expr2))--> expr(Expr1),['=='],expr(Expr2).
condition(notequalsto(Expr1,Expr2))--> expr(Expr1),['!='],expr(Expr2).
condition(lessthan(Expr1,Expr2))--> expr(Expr1),['<'],expr(Expr2).
condition(lessthan_orequalto(Expr1,Expr2))--> expr(Expr1),['<='],expr(Expr2).
condition(greaterthan(Expr1,Expr2))--> expr(Expr1),['>'],expr(Expr2).
condition(greaterthan_orequalto(Expr1,Expr2))--> expr(Expr1),['>='],expr(Expr2).
condition(Bool)--> boolvalue(Bool).
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%% IF STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%

ifcommand((If))--> ifpart(If).
ifelsecommand(ifelse(If,Else))--> ifpart(If),elsepart(Else).
ifELseLaddercommand(ifelseladder(If ,Elif ,Else))--> ifpart(If),elseifpart(Elif),elsepart(Else).

ifpart(if(B,X))--> ['if'],['('],boolcondition(B), [')'],['{'],commandlist(X),['}'].
elseifpart(elseif(B,E1,E2))--> ['elif'],['('],boolcondition(B), [')'],['{'],commandlist(E1),['}'],elseifpart(E2).
elseifpart(elseif(B,E))--> ['elif'],['('],boolcondition(B), [')'],['{'],commandlist(E),['}'].
elsepart(else(C))--> ['else'],['{'],commandlist(C),['}'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%% WHILE STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%
whilecommand(while(Condition,C))--> ['while'],['('],boolcondition(Condition), [')'],['{'],commandlist(C),['}'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%% FOR STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%
forcommand(for(Assign,BoolCondition,Valupdation,C))--> ['for'],['('],assignment(Assign),[';'],boolcondition(BoolCondition),[';'],variableupdation(Valupdation),[')'],['{'],commandlist(C),['}'].
%%%%%%%%%%%%%%%%%%%%%%%%%%%% variable updation%%%%%%%%%%%%%%%%%%%%%%%%%%%%
variableupdation((Ops))--> increment_operation(Ops).
variableupdation((Ops))--> decrement_operation(Ops).
variableupdation((Assign))--> assignment(Assign).

%%%%%%%%%%%%%%%%%%%%%%%%%%%% increment and decrement operations%%%%%%%%%%%%%%%%%%%%%%%%%%%%
increment_operation(increment(Var)) --> variablename(Var),['++'].
increment_operation(increment(Var)) --> ['++'],variablename(Var).

decrement_operation(decrement(Var)) --> variablename(Var),['--'].
decrement_operation(decrement(Var)) --> ['--'],variablename(Var).

%%%%%%%%%%%%%%%%%%%%%%%%%%%% FOR IN RANGE STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%
forinrangecommand(forinrange(Var,SR,ER,C))--> ['for'],variablename(Var),['in'],['range'],['('], range(SR),[','],range(ER),[')'],['{'],commandlist(C),['}'].
range(Range)--> variablename(Range).
range(Range)--> num(Range).
```

```
%%%%%%%%%%%%check before pushing%%%%%%%%%%%%%% Terenary Statement%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
ternary(ternary(Bool,Expr1,Expr2))-->['('], boolcondition(Bool),[')'],['?'],expr(Expr1),[':'],expr(Expr2).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% PRINT STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
printStmt(print(Print))--> ['print'],['('],expr(Print),[')'].
printStmt(printnl(Print))--> ['printnl'],['('],expr(Print),[')'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% STRING LENGTH STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
strlen(stringlength(Strlen)) --> ['strlen'],['('],stringvalue(Strlen),[')'].

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% STRING VALUE DEFINITION STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
stringvalue(str(Str))--> ['\"'],[Str],['\"'].
stringvalue((Str))--> variablename(Str).
% LEXER is handling it
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% FLOAT VALUE DEFINITION STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

floatvalue(N)--> num(N).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%% BOOLEAN VALUE DEFINITION STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
boolvalue(bool(true))--> ['true'].
boolvalue(bool(false))--> ['false'].
```

```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VARIABLE NAME DEFINITION STATEMENT%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% VAR NAME SHOULD NOT START WITH A LOWERCASE LETTER%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% VAR NAME SHOULD NOT START WITH A SPECIAL CHARACTER%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%% VAR NAME CAN BE ALPHANUMERIC AND CAN CONTAIN UNDERSCORE%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%% VAR NAME SHOULD NOT END WITH UNDERSCORE %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
variablename(var(Atom)) -->
    [Atom],
    {\+ member(Atom, ['const','int','string','float','bool','mod','and','or','not','print','println','strlen',
        'sqrt','cbrt','sq','cube','true','false','if','elif','else', 'for', 'in','range','while'])},
    {\+number(Atom)},
    { atom_chars(Atom, [First|RestChars]) },
    { code_type(First, lower) },
    {restOfVariableName(RestChars)}.

restOfVariableName([Char|Rest]):-
    code_type(Char, alnum); Char == '_',
    restOfVariableName(Rest).
restOfVariableName([]).



num(num(Num)) --> [Num],{number(Num)}.
```

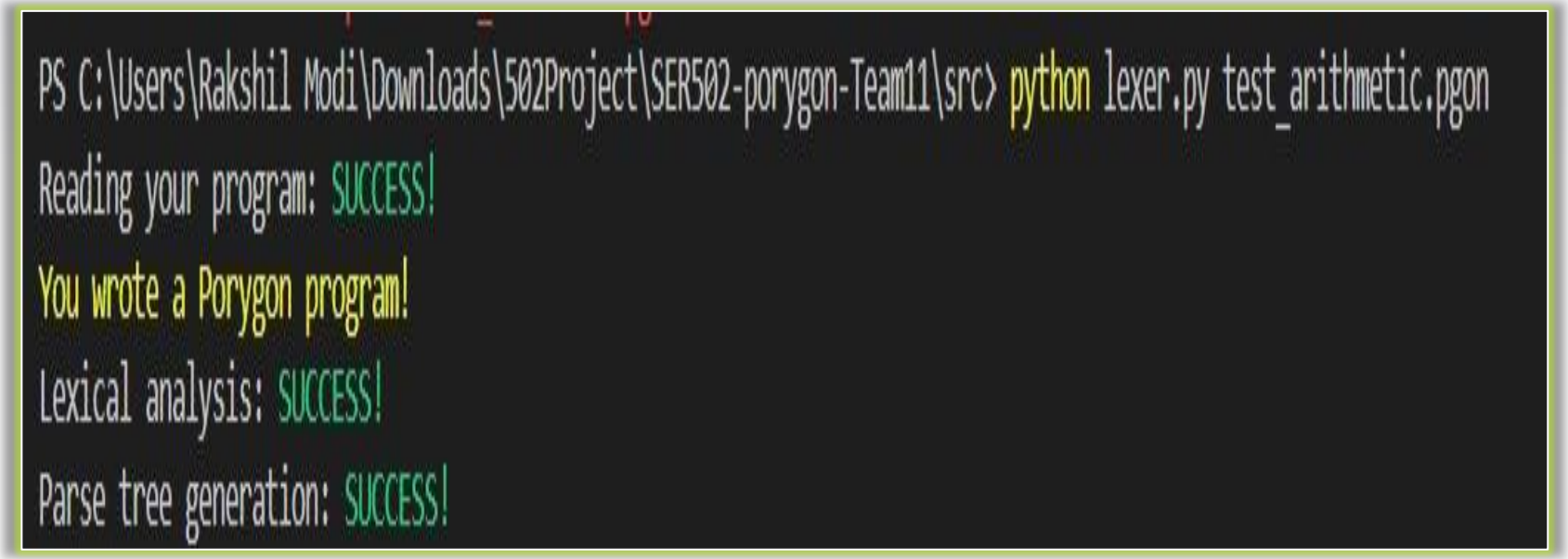
EVALUATOR

```
boolean(true).
boolean(false).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% hard look up to check variables presence and return it %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
hard_look_up(X,Env,Val):-
    flatten(Env, FEnv), %to include constant variables in search
    hlu(X, FEnv, Val).

hlu(X,[],_Val):-
    concat('Uninitialized variable: ',X,Str),
    print_message(Str),
    halt.

hlu(HVar,[(HVar,HVal,_HType)|_T],HVal).

hlu(X,[(HVar,_HVal,_HType)|T],Val):-
    X \= HVar,
    hlu(X,T,Val).

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%% soft look up to check variables presence  %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
soft_look_up(X,Env,Val):-
    flatten(Env, FEnv), %to include constant variables in search
    slu(X, FEnv, Val).

slu(_X,[],_Val):-
    fail.

slu(HVar,[(HVar,HVal,_HType)|_T],HVal).
```

7

SNAPSHOTS OF THE PROGRAMMING LANGUAGE

PROGRAM RUNNING



PROGRAM EXAMPLE

```
{
        int c;
        int b   = 25;
        int e = 35;
        int d = c+b+e;
        int f = 100-e;
        int mul = f*25;
        int div = mul/5;
        float a;
        float mult = a*2.0;
        int all_arthmetic1 = (b*e)/5+100-25 mod 5;

         mult+= 15.0;

        c = b mod 2;



        println(d);
        println(f);
        println(mul);
        println(div);
        println(a);
        println(mult);
        println(c);
        print(all_arthmetic1);

}
```

OUTPUT

```
Output:
60
65
1625
325
0.0
15.0
1
Execution: SUCCESS!
```

THANKS!