ARGS

AKANSHA REDDY ANTHIREDDYGARI

SAIRACHANA PALADUGU

GIRIJA RANI NIMMAGADDA

SASIKANTH POTLURI

SUMMARY OF ARGS

- ARGS is a programming language that has been intentionally created to be easily understandable and usable by people who have little or no experience in programming. We have created this language using Python 3 and it work with a .args extension. Args is capable of performing simple arithmetic operations and expressions including traditional iterations, conditions.

FEATURES SUPPORTED

- **Operators**

1. Arithmetic Operators
2. Comparison Operators
3. Logical Operators

- **Data Types:**

1.Number
2.String
3.Boolean
4.Float
5.List

- **Conditional Statements**

1. IF THEN ELIF ELSE
2. Ternary Operator

- **Looping Statements**

1. FOR loop
2. WHILE loop

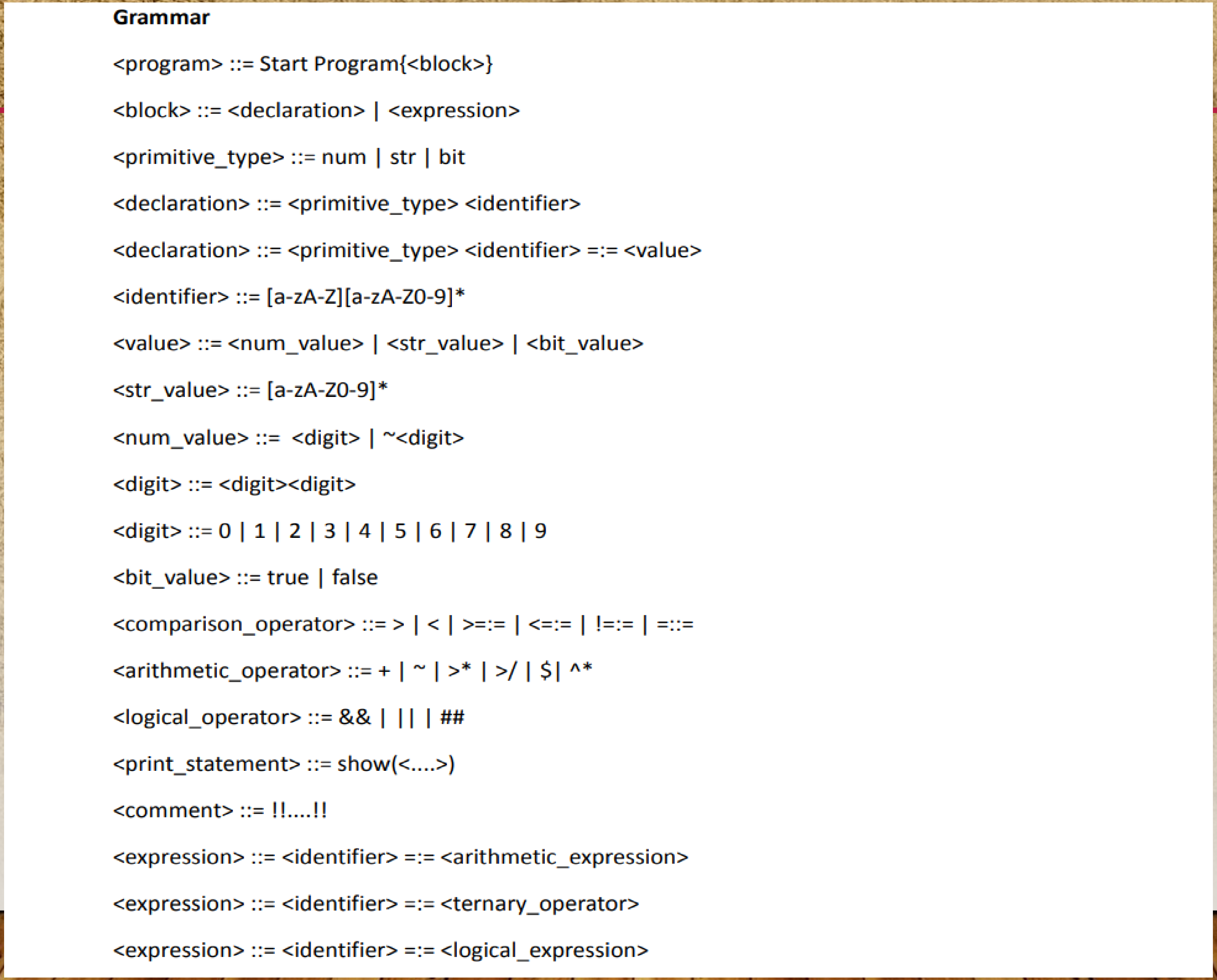
- **Assignment Operator**
- **Print statement**
- **Comments**

- **BREAK**
- **CONTINUE**
- **RUN**

- **INPUT**
- **INPUT_INT**
- **Multi-line statements**

- **IS_NUM**
- **IS_STR**
- **IS_LIST**

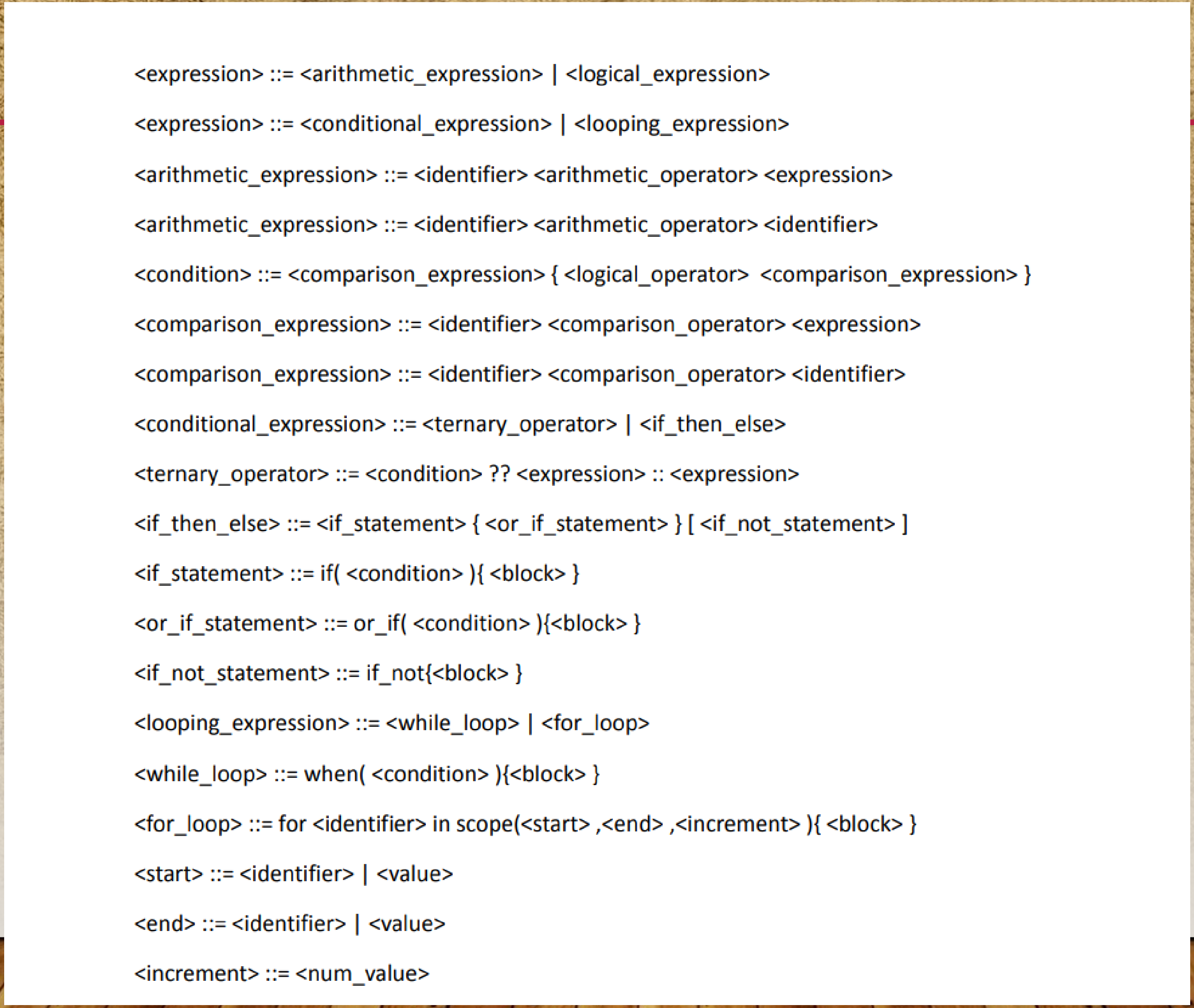
GRAMMAR



Grammar

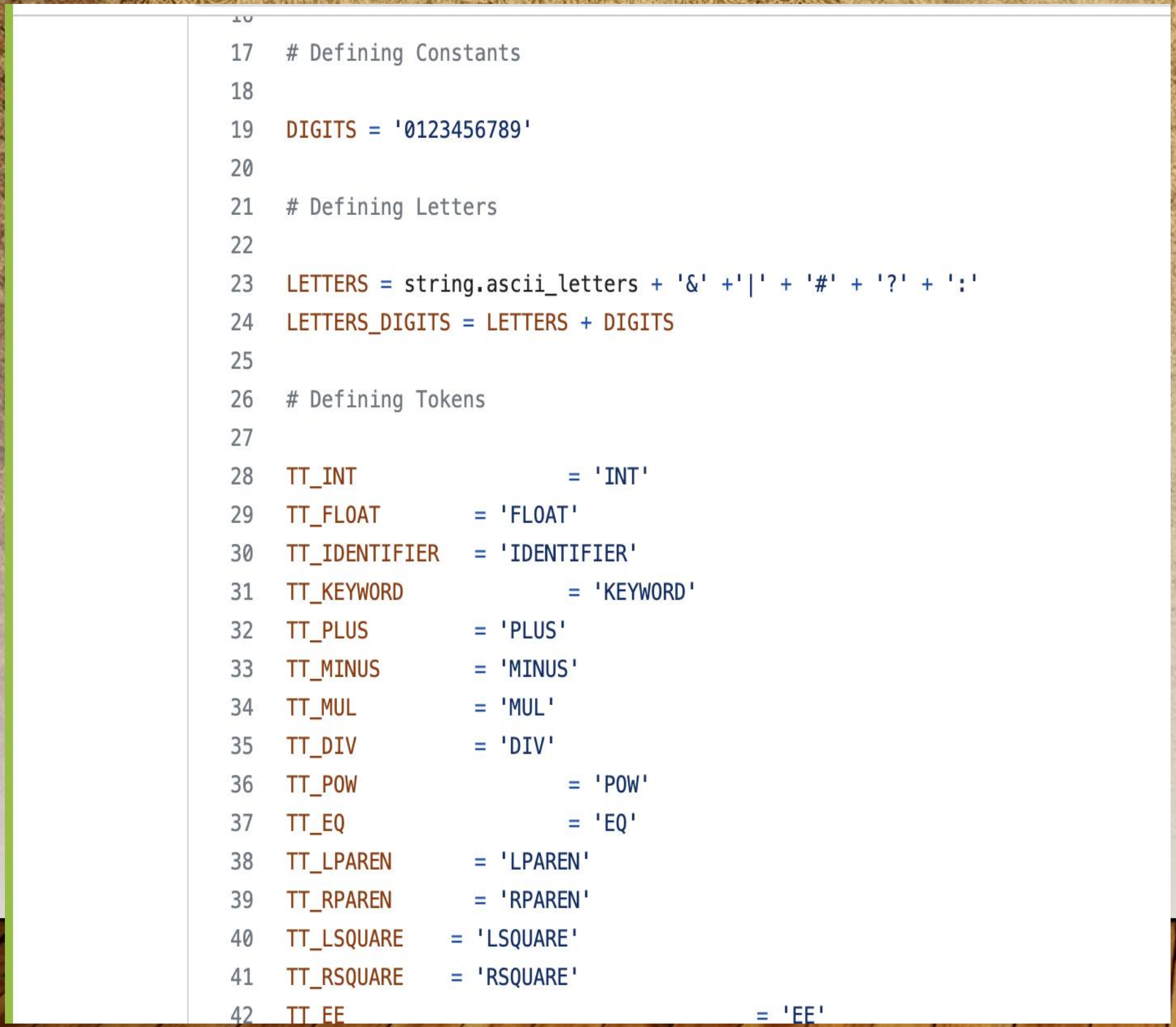
```
<program> ::= Start Program{<block>}

<block> ::= <declaration> | <expression>

<primitive_type> ::= num | str | bit

<declaration> ::= <primitive_type> <identifier>

<declaration> ::= <primitive_type> <identifier> =:= <value>

<identifier> ::= [a-zA-Z][a-zA-Z0-9]*

<value> ::= <num_value> | <str_value> | <bit_value>

<str_value> ::= [a-zA-Z0-9]*

<num_value> ::=  <digit> | ~<digit>

<digit> ::= <digit><digit>

<digit> ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

<bit_value> ::= true | false

<comparison_operator> ::= > | < | >=:= | <=:= | !=:= | =::=

<arithmetic_operator> ::= + | ~ | >* | >/ | $| ^*

<logical_operator> ::= && | || | ##

<print_statement> ::= show(<....>)

<comment> ::= !!....!!

<expression> ::= <identifier> =:= <arithmetic_expression>

<expression> ::= <identifier> =:= <ternary_operator>

<expression> ::= <identifier> =:= <logical_expression>
```

GRAMMAR

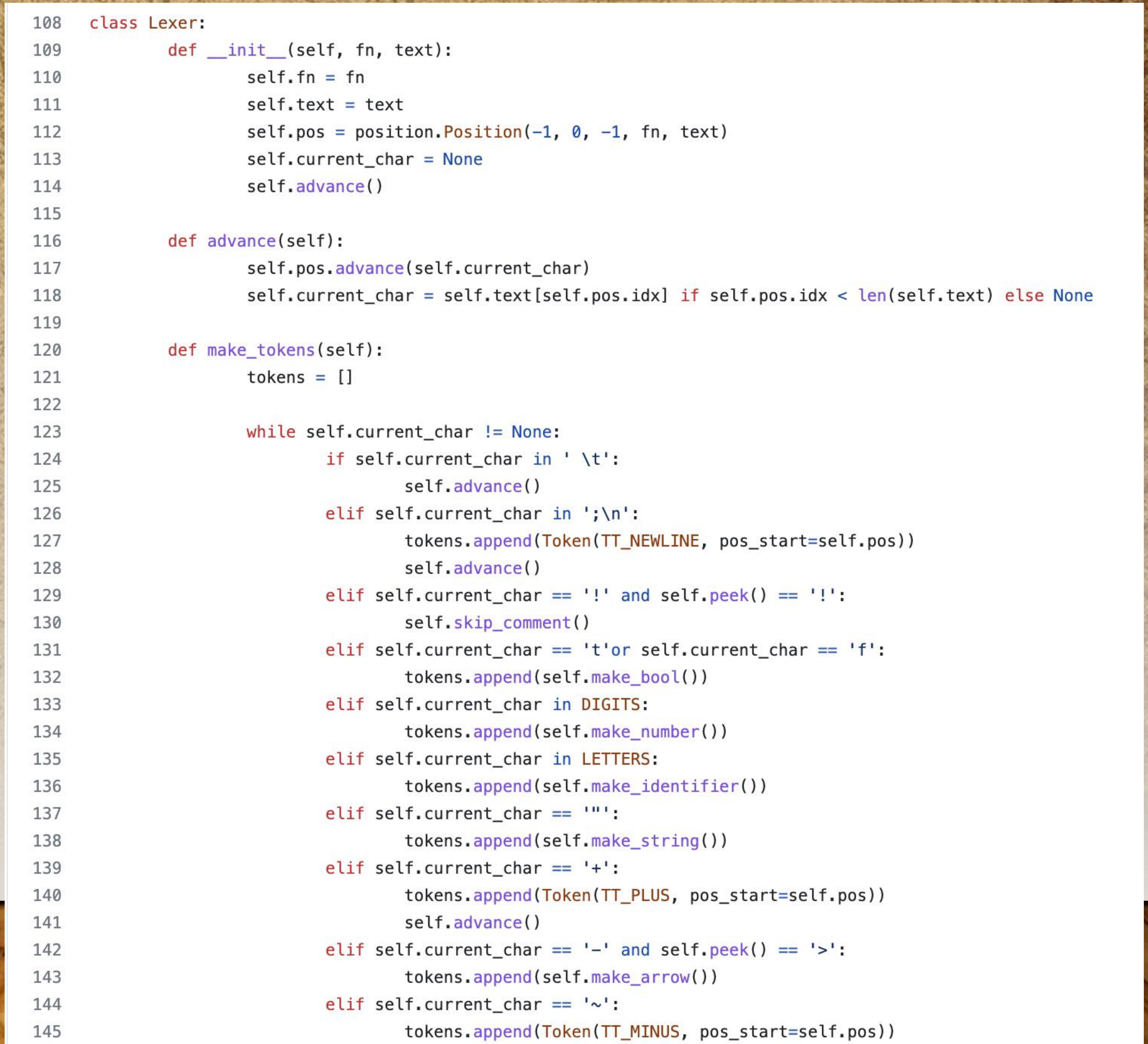


```
<expression> ::= <arithmetic_expression> | <logical_expression>

<expression> ::= <conditional_expression> | <looping_expression>

<arithmetic_expression> ::= <identifier> <arithmetic_operator> <expression>

<arithmetic_expression> ::= <identifier> <arithmetic_operator> <identifier>

<condition> ::= <comparison_expression> { <logical_operator>  <comparison_expression> }

<comparison_expression> ::= <identifier> <comparison_operator> <expression>

<comparison_expression> ::= <identifier> <comparison_operator> <identifier>

<conditional_expression> ::= <ternary_operator> | <if_then_else>

<ternary_operator> ::= <condition> ?? <expression> :: <expression>

<if_then_else> ::= <if_statement> { <or_if_statement> } [ <if_not_statement> ]

<if_statement> ::= if( <condition> ){ <block> }

<or_if_statement> ::= or_if( <condition> ){<block> }

<if_not_statement> ::= if_not{<block> }

<looping_expression> ::= <while_loop> | <for_loop>

<while_loop> ::= when( <condition> ){<block> }

<for_loop> ::= for <identifier> in scope(<start> ,<end> ,<increment> ){ <block> }

<start> ::= <identifier> | <value>

<end> ::= <identifier> | <value>

<increment> ::= <num_value>
```

LEXER

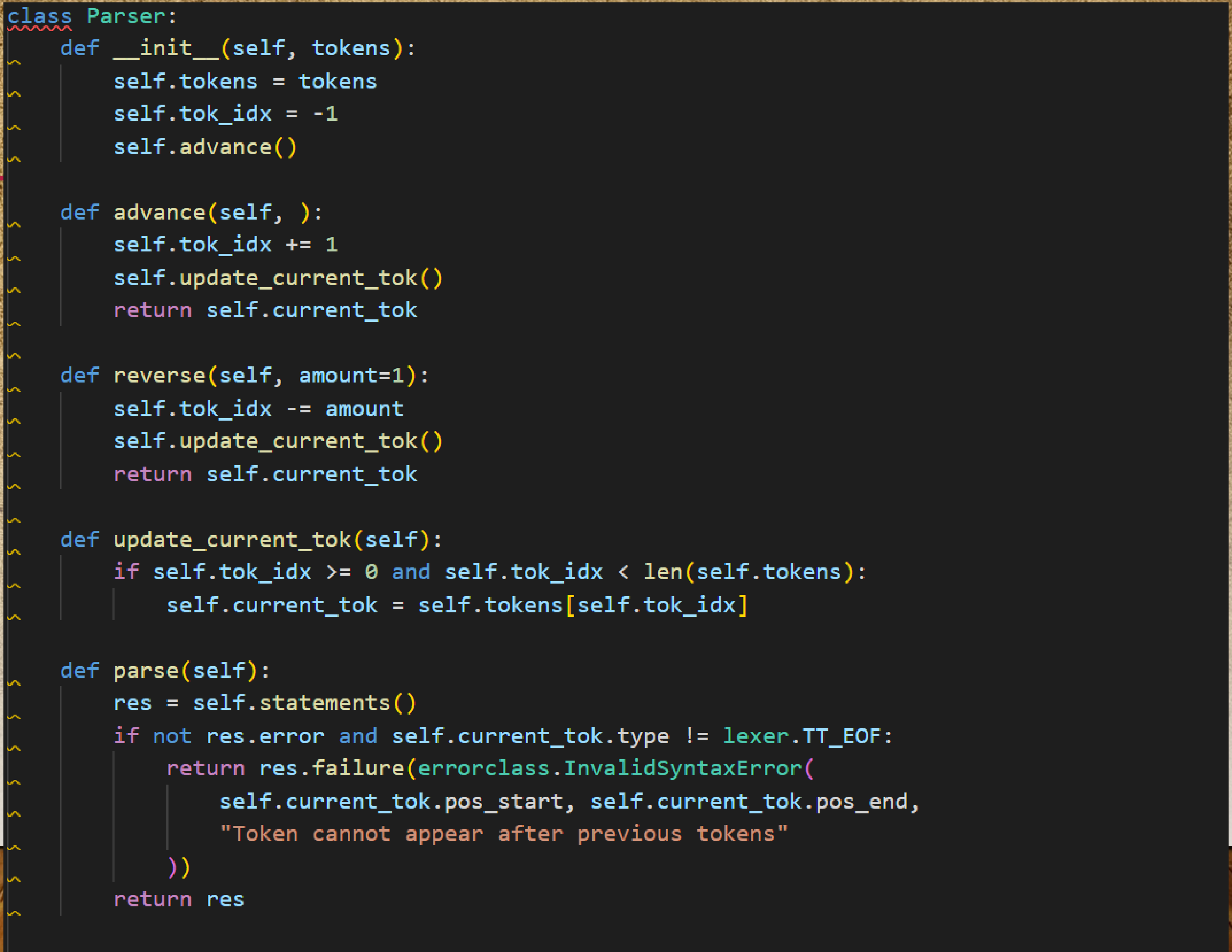


```python
# Defining Constants

DIGITS = '0123456789'

# Defining Letters

LETTERS = string.ascii_letters + '&' +'|' + '#' + '?' + ':'
LETTERS_DIGITS = LETTERS + DIGITS

# Defining Tokens

TT_INT                  = 'INT'
TT_FLOAT        = 'FLOAT'
TT_IDENTIFIER   = 'IDENTIFIER'
TT_KEYWORD              = 'KEYWORD'
TT_PLUS         = 'PLUS'
TT_MINUS        = 'MINUS'
TT_MUL          = 'MUL'
TT_DIV          = 'DIV'
TT_POW                  = 'POW'
TT_EQ                   = 'EQ'
TT_LPAREN       = 'LPAREN'
TT_RPAREN       = 'RPAREN'
TT_LSQUARE    = 'LSQUARE'
TT_RSQUARE    = 'RSQUARE'
TT_EE                                   = 'EE'
```



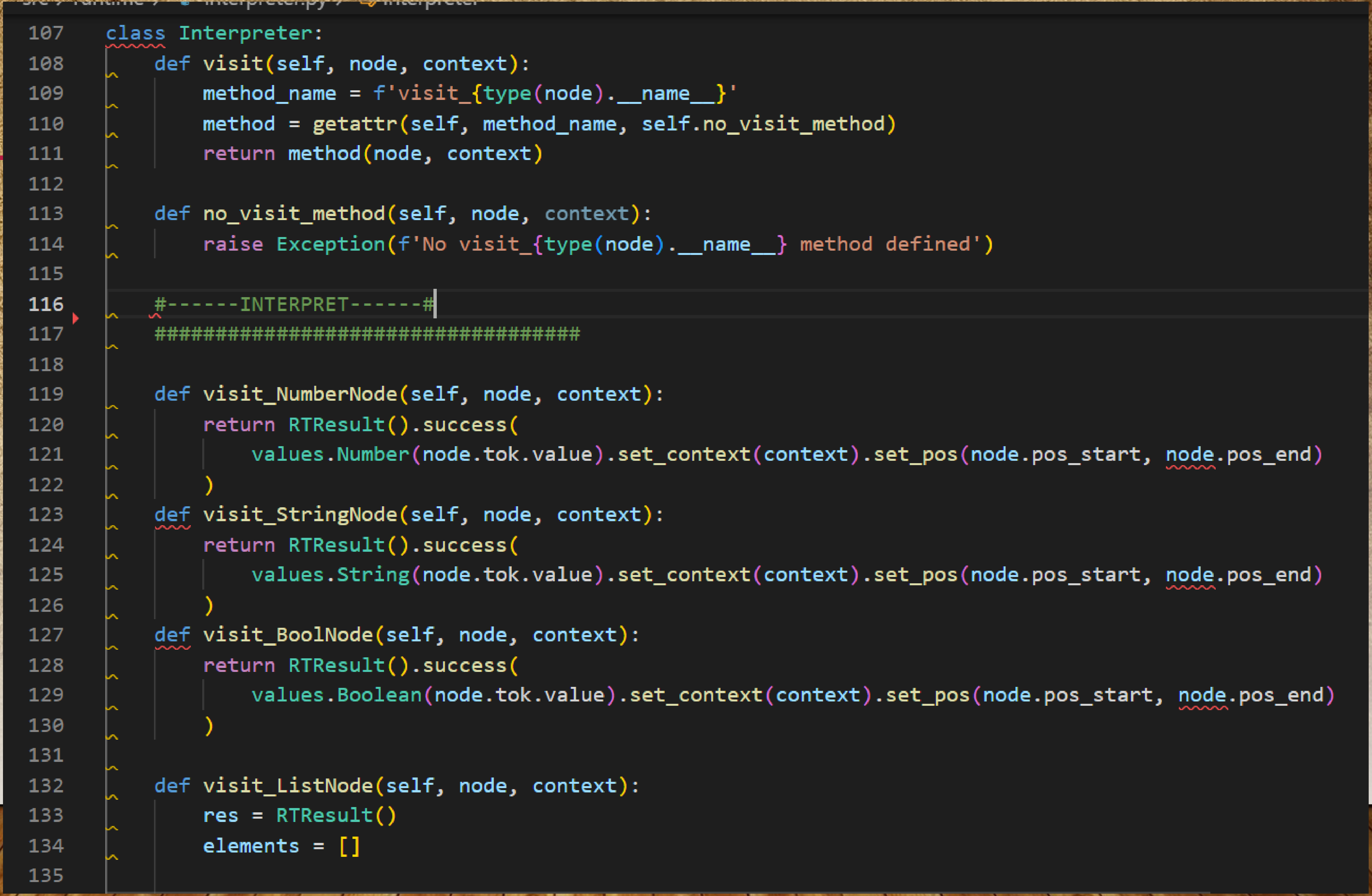
```python
class Lexer:
        def __init__(self, fn, text):
                self.fn = fn
                self.text = text
                self.pos = position.Position(-1, 0, -1, fn, text)
                self.current_char = None
                self.advance()

        def advance(self):
                self.pos.advance(self.current_char)
                self.current_char = self.text[self.pos.idx] if self.pos.idx < len(self.text) else None

        def make_tokens(self):
                tokens = []

                while self.current_char != None:
                        if self.current_char in ' \t':
                                self.advance()
                        elif self.current_char in ';\n':
                                tokens.append(Token(TT_NEWLINE, pos_start=self.pos))
                                self.advance()
                        elif self.current_char == '!' and self.peek() == '!':
                                self.skip_comment()
                        elif self.current_char == 't'or self.current_char == 'f':
                                tokens.append(self.make_bool())
                        elif self.current_char in DIGITS:
                                tokens.append(self.make_number())
                        elif self.current_char in LETTERS:
                                tokens.append(self.make_identifier())
                        elif self.current_char == '"':
                                tokens.append(self.make_string())
                        elif self.current_char == '+':
                                tokens.append(Token(TT_PLUS, pos_start=self.pos))
                                self.advance()
                        elif self.current_char == '-' and self.peek() == '>':
                                tokens.append(self.make_arrow())
                        elif self.current_char == '~':
                                tokens.append(Token(TT_MINUS, pos_start=self.pos))
```

PARSER



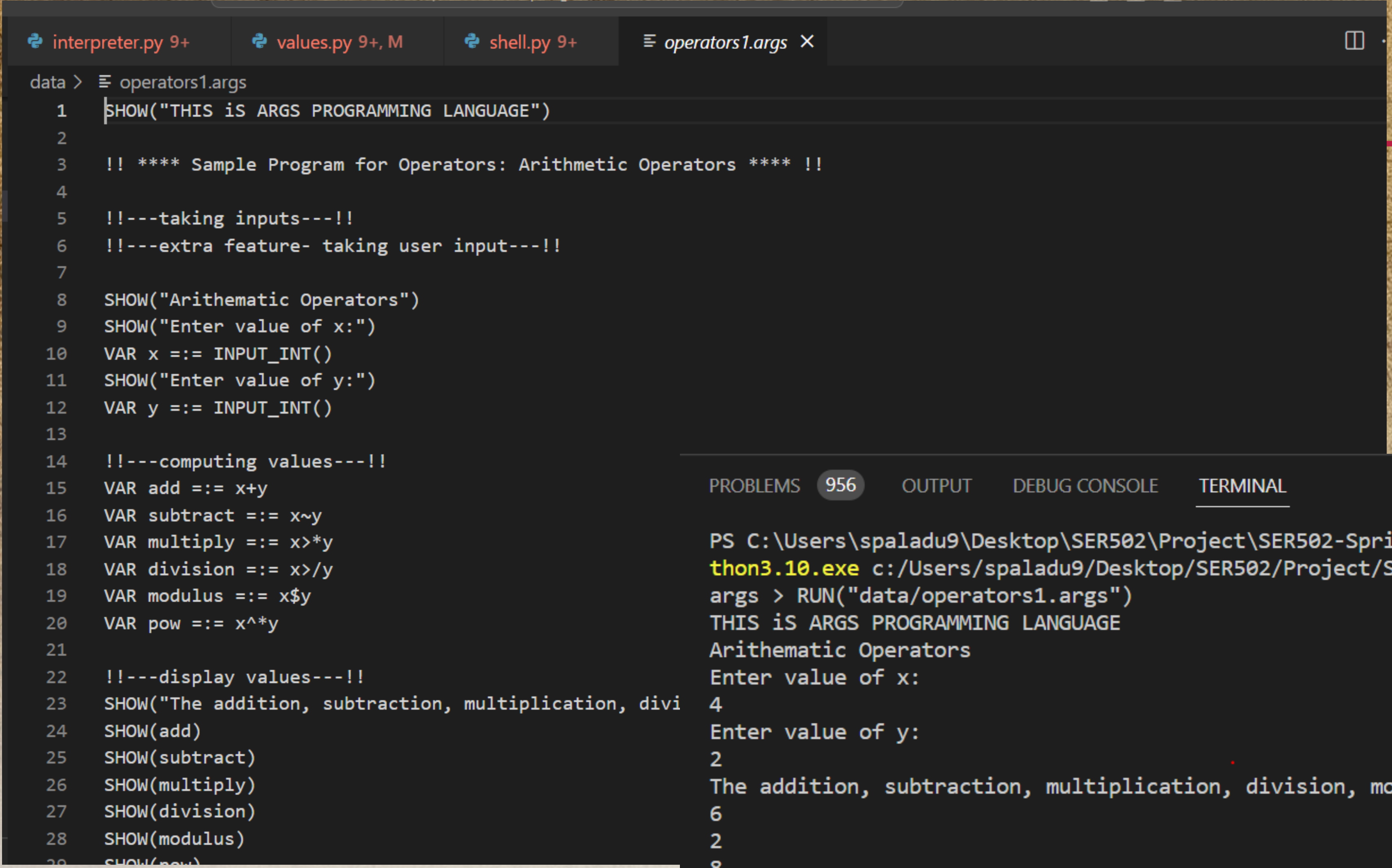
```python
class Parser:
    def __init__(self, tokens):
        self.tokens = tokens
        self.tok_idx = -1
        self.advance()

    def advance(self, ):
        self.tok_idx += 1
        self.update_current_tok()
        return self.current_tok

    def reverse(self, amount=1):
        self.tok_idx -= amount
        self.update_current_tok()
        return self.current_tok

    def update_current_tok(self):
        if self.tok_idx >= 0 and self.tok_idx < len(self.tokens):
            self.current_tok = self.tokens[self.tok_idx]

    def parse(self):
        res = self.statements()
        if not res.error and self.current_tok.type != lexer.TT_EOF:
            return res.failure(errorclass.InvalidSyntaxError(
                self.current_tok.pos_start, self.current_tok.pos_end,
                "Token cannot appear after previous tokens"
            ))
        return res
```

INTERPRETER

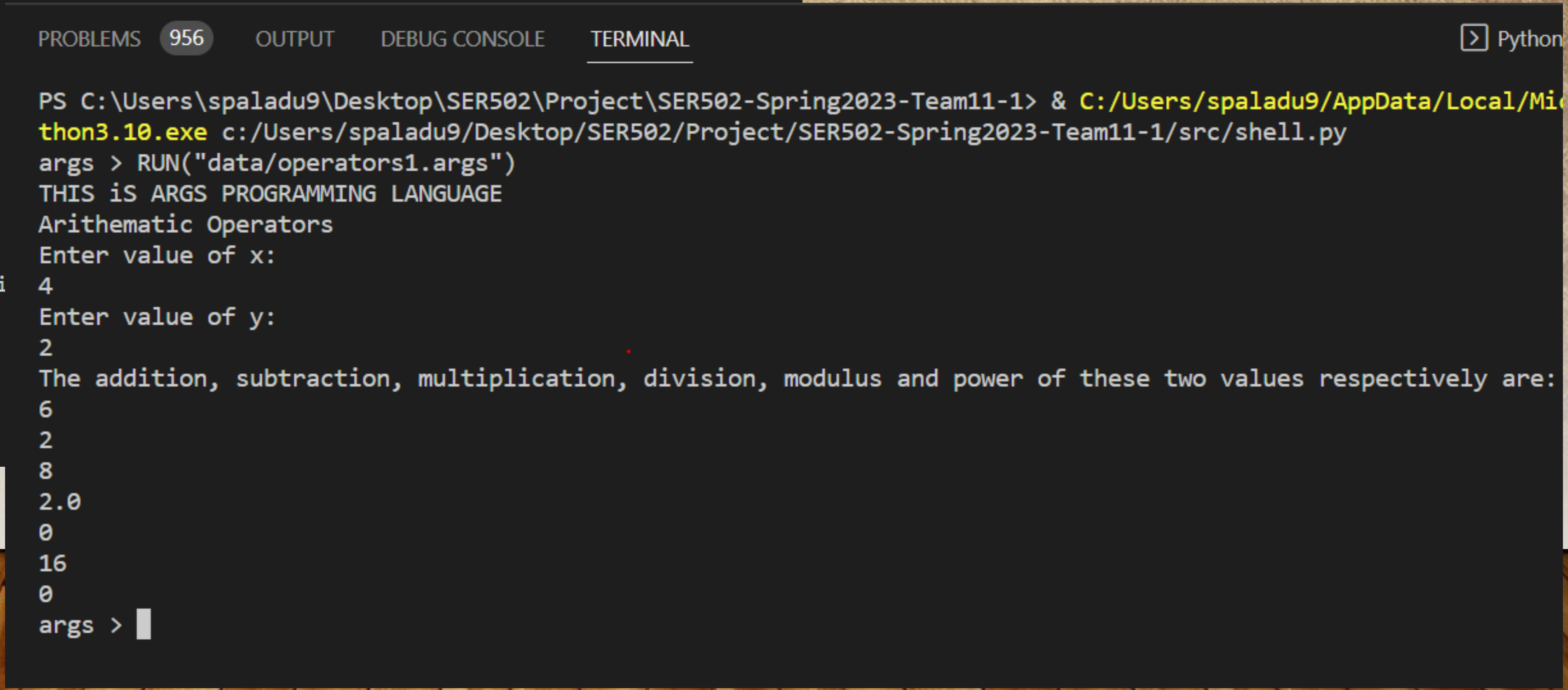


```
class Interpreter:
    def visit(self, node, context):
        method_name = f'visit_{type(node).__name__}'
        method = getattr(self, method_name, self.no_visit_method)
        return method(node, context)

    def no_visit_method(self, node, context):
        raise Exception(f'No visit_{type(node).__name__} method defined')

    #------INTERPRET------#
    ####################################

    def visit_NumberNode(self, node, context):
        return RTResult().success(
            values.Number(node.tok.value).set_context(context).set_pos(node.pos_start, node.pos_end)
        )
    def visit_StringNode(self, node, context):
        return RTResult().success(
            values.String(node.tok.value).set_context(context).set_pos(node.pos_start, node.pos_end)
        )
    def visit_BoolNode(self, node, context):
        return RTResult().success(
            values.Boolean(node.tok.value).set_context(context).set_pos(node.pos_start, node.pos_end)
        )

    def visit_ListNode(self, node, context):
        res = RTResult()
        elements = []

```

SAMPLE PROGRAM - 1

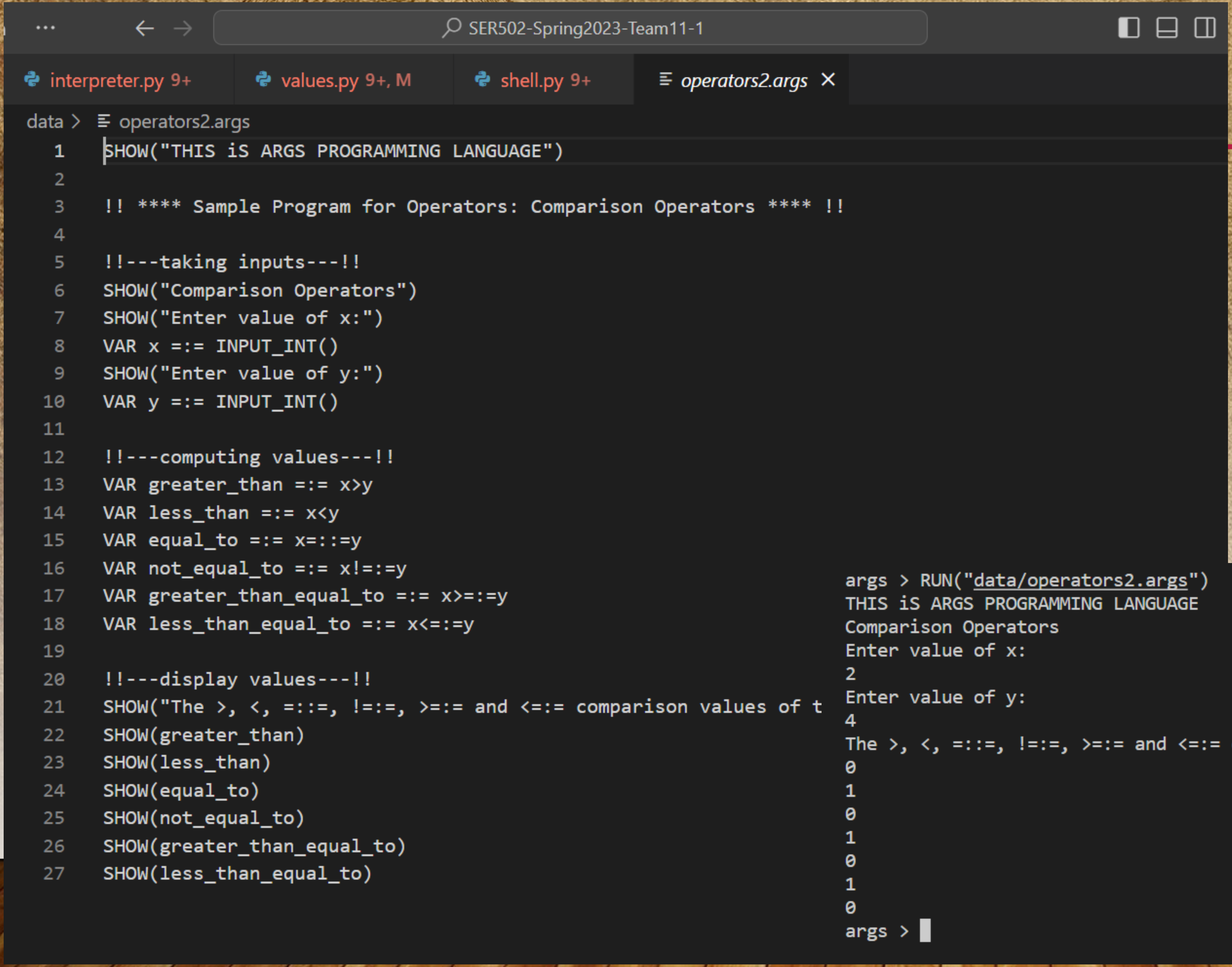


```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for Operators: Arithmetic Operators **** !!

!!---taking inputs---!!
!!---extra feature- taking user input---!!

SHOW("Arithematic Operators")
SHOW("Enter value of x:")
VAR x =:= INPUT_INT()
SHOW("Enter value of y:")
VAR y =:= INPUT_INT()

!!---computing values---!!
VAR add =:= x+y
VAR subtract =:= x~y
VAR multiply =:= x>*y
VAR division =:= x>/y
VAR modulus =:= x$y
VAR pow =:= x^*y

!!---display values---!!
SHOW("The addition, subtraction, multiplication, divi
SHOW(add)
SHOW(subtract)
SHOW(multiply)
SHOW(division)
SHOW(modulus)
```

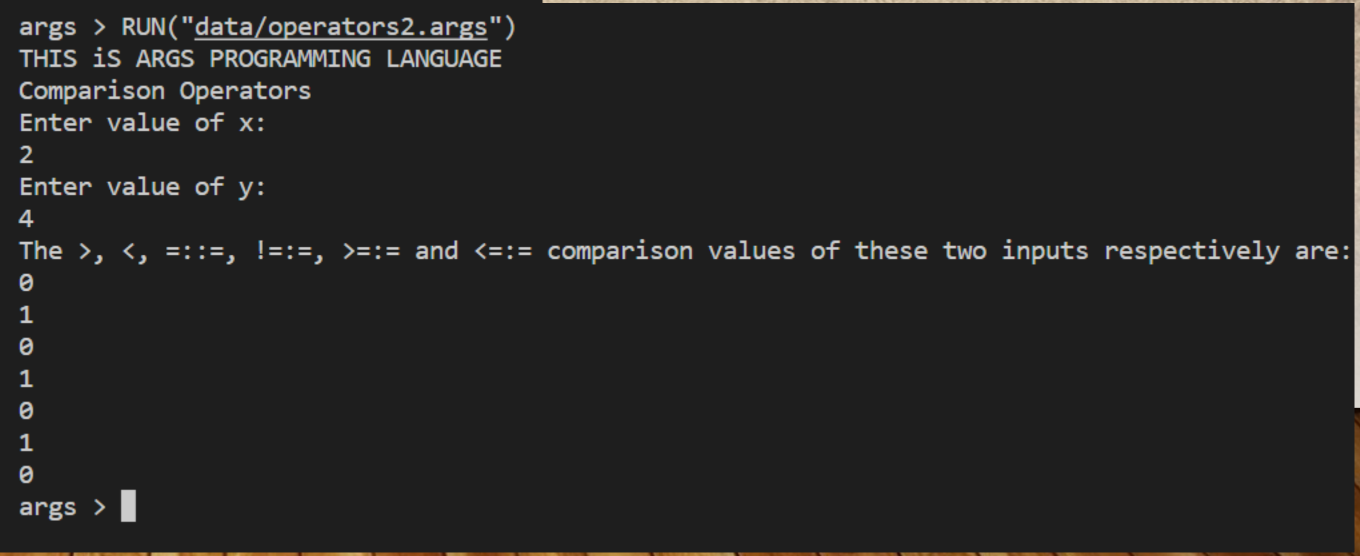


```
PS C:\Users\spaladu9\Desktop\SER502\Project\SER502-Spring2023-Team11-1> & C:/Users/spaladu9/AppData/Local/Mi
thon3.10.exe c:/Users/spaladu9/Desktop/SER502/Project/SER502-Spring2023-Team11-1/src/shell.py
args > RUN("data/operators1.args")
THIS iS ARGS PROGRAMMING LANGUAGE
Arithematic Operators
Enter value of x:
4
Enter value of y:
2
The addition, subtraction, multiplication, division, modulus and power of these two values respectively are:
6
2
8
2.0
0
16
0
args >
```

SAMPLE PROGRAM - 2

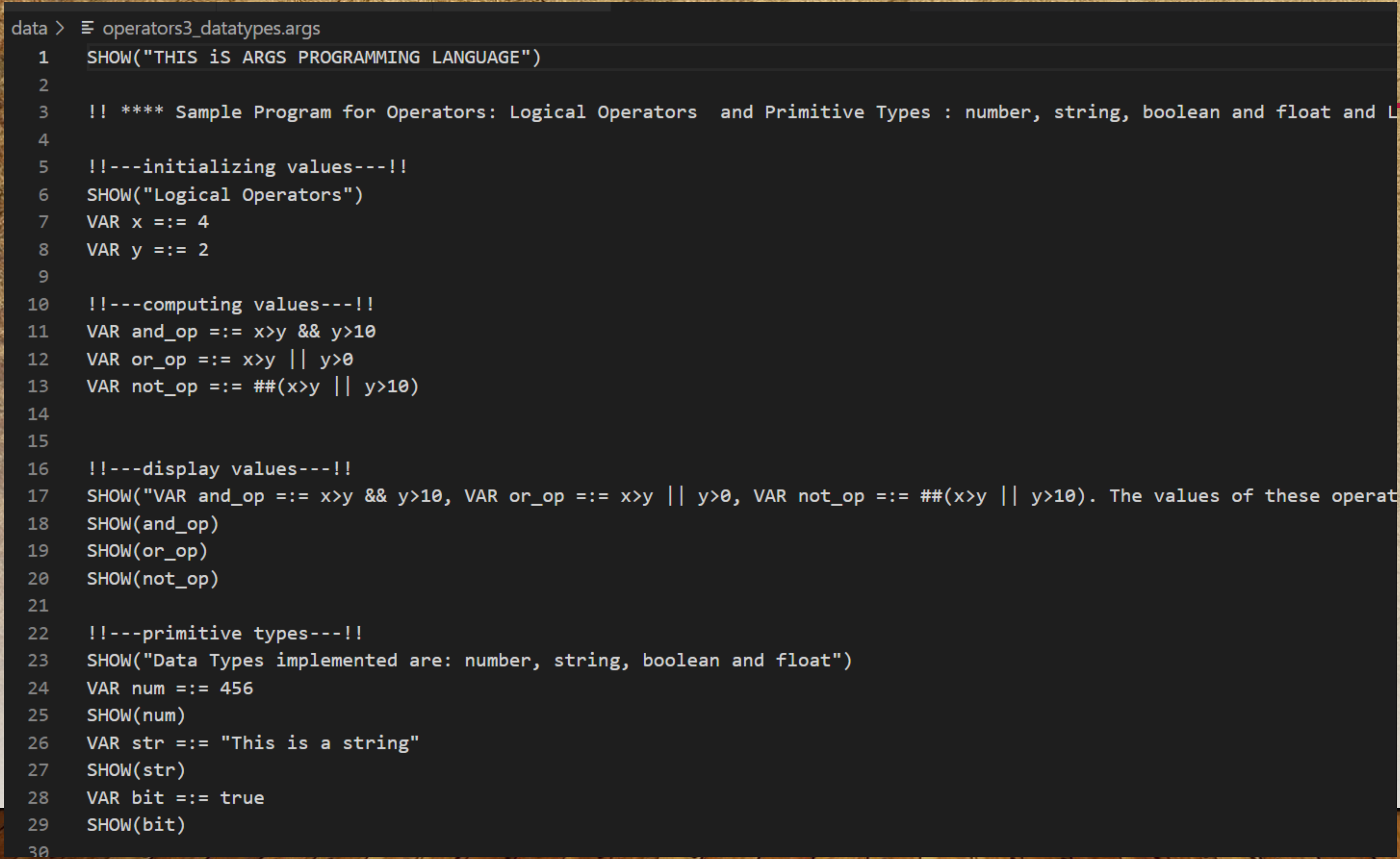


```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for Operators: Comparison Operators **** !!

!!---taking inputs---!!
SHOW("Comparison Operators")
SHOW("Enter value of x:")
VAR x =:= INPUT_INT()
SHOW("Enter value of y:")
VAR y =:= INPUT_INT()

!!---computing values---!!
VAR greater_than =:= x>y
VAR less_than =:= x<y
VAR equal_to =:= x=::=y
VAR not_equal_to =:= x!=:=y
VAR greater_than_equal_to =:= x>=:=y
VAR less_than_equal_to =:= x<=:=y

!!---display values---!!
SHOW("The >, <, =::=, !=:=, >=:= and <=:= comparison values of t
SHOW(greater_than)
SHOW(less_than)
SHOW(equal_to)
SHOW(not_equal_to)
SHOW(greater_than_equal_to)
SHOW(less_than_equal_to)
```



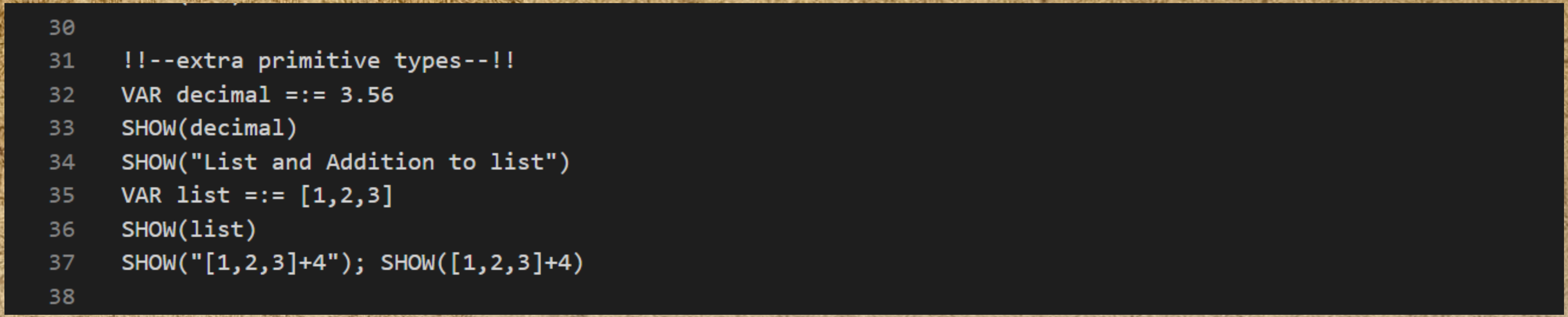
```
args > RUN("data/operators2.args")
THIS iS ARGS PROGRAMMING LANGUAGE
Comparison Operators
Enter value of x:
2
Enter value of y:
4
The >, <, =::=, !=:=, >=:= and <=:= comparison values of these two inputs respectively are:
0
1
0
1
0
1
0
args >
```

SAMPLE PROGRAM - 3

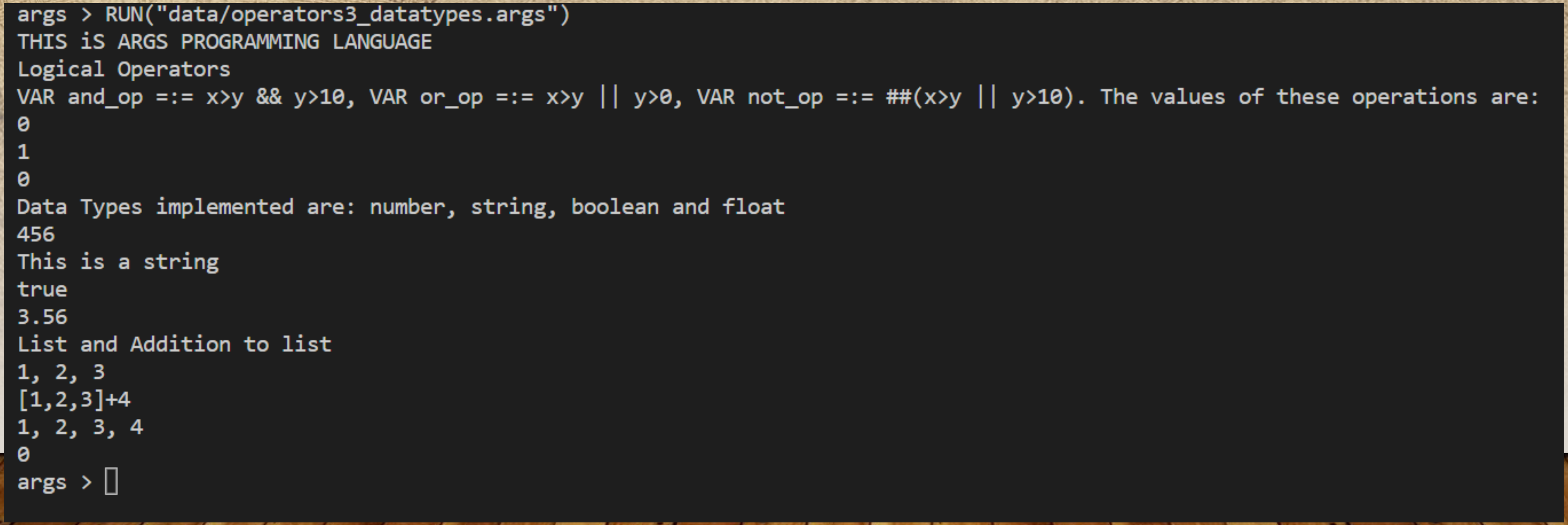

data > operators3_datatypes.args

```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for Operators: Logical Operators  and Primitive Types : number, string, boolean and float and L

!!---initializing values---!!
SHOW("Logical Operators")
VAR x =:= 4
VAR y =:= 2

!!---computing values---!!
VAR and_op =:= x>y && y>10
VAR or_op =:= x>y || y>0
VAR not_op =:= ##(x>y || y>10)


!!---display values---!!
SHOW("VAR and_op =:= x>y && y>10, VAR or_op =:= x>y || y>0, VAR not_op =:= ##(x>y || y>10). The values of these operat
SHOW(and_op)
SHOW(or_op)
SHOW(not_op)

!!---primitive types---!!
SHOW("Data Types implemented are: number, string, boolean and float")
VAR num =:= 456
SHOW(num)
VAR str =:= "This is a string"
SHOW(str)
VAR bit =:= true
SHOW(bit)
```

SAMPLE PROGRAM – 3 OUTPUT

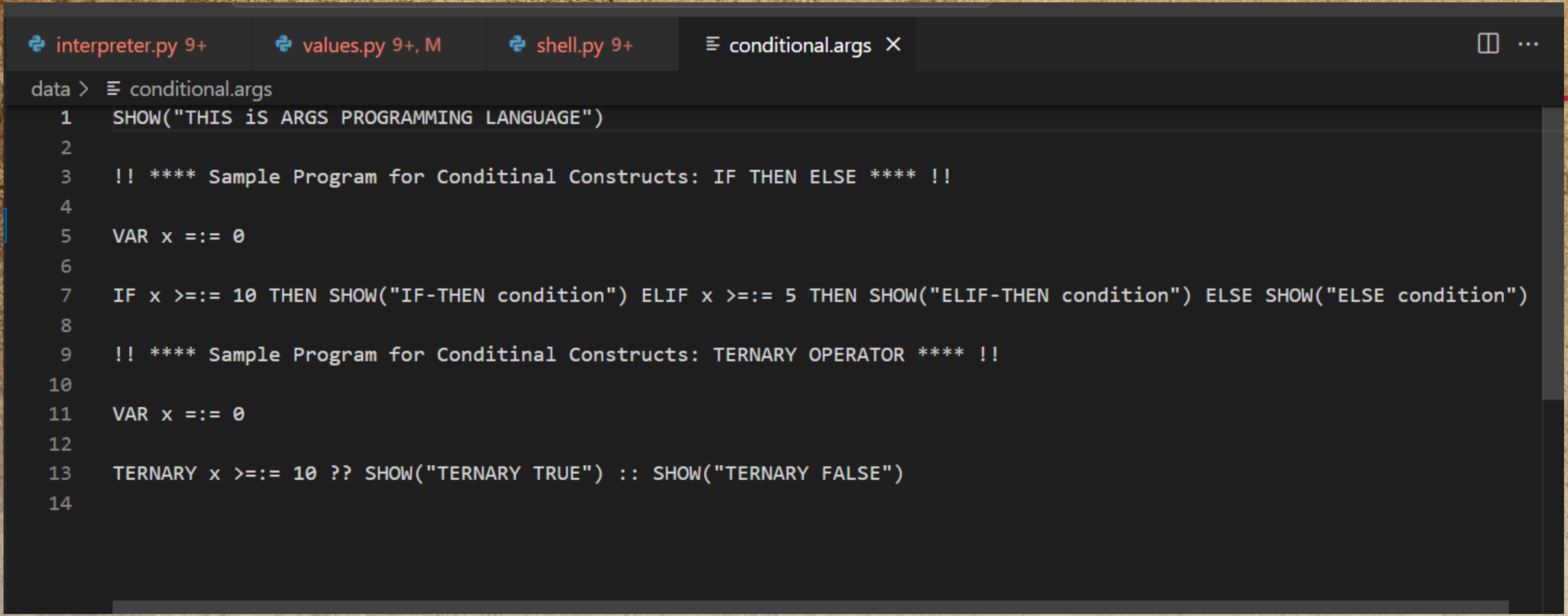


```

!!--extra primitive types--!!
VAR decimal =:= 3.56
SHOW(decimal)
SHOW("List and Addition to list")
VAR list =:= [1,2,3]
SHOW(list)
SHOW("[1,2,3]+4"); SHOW([1,2,3]+4)

```

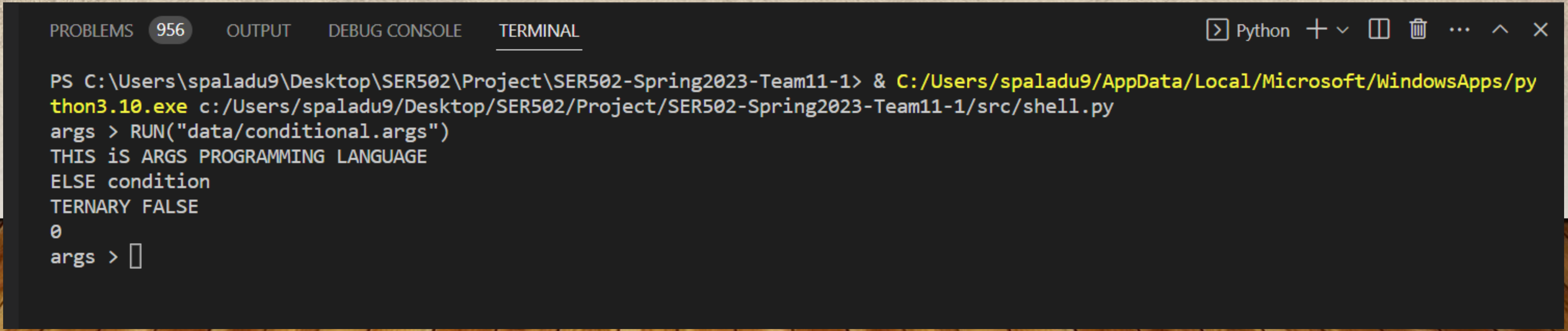


```
args > RUN("data/operators3_datatypes.args")
THIS iS ARGS PROGRAMMING LANGUAGE
Logical Operators
VAR and_op =:= x>y && y>10, VAR or_op =:= x>y || y>0, VAR not_op =:= ##(x>y || y>10). The values of these operations are:
0
1
0
Data Types implemented are: number, string, boolean and float
456
This is a string
true
3.56
List and Addition to list
1, 2, 3
[1,2,3]+4
1, 2, 3, 4
0
args > 
```

SAMPLE PROGRAM - 4

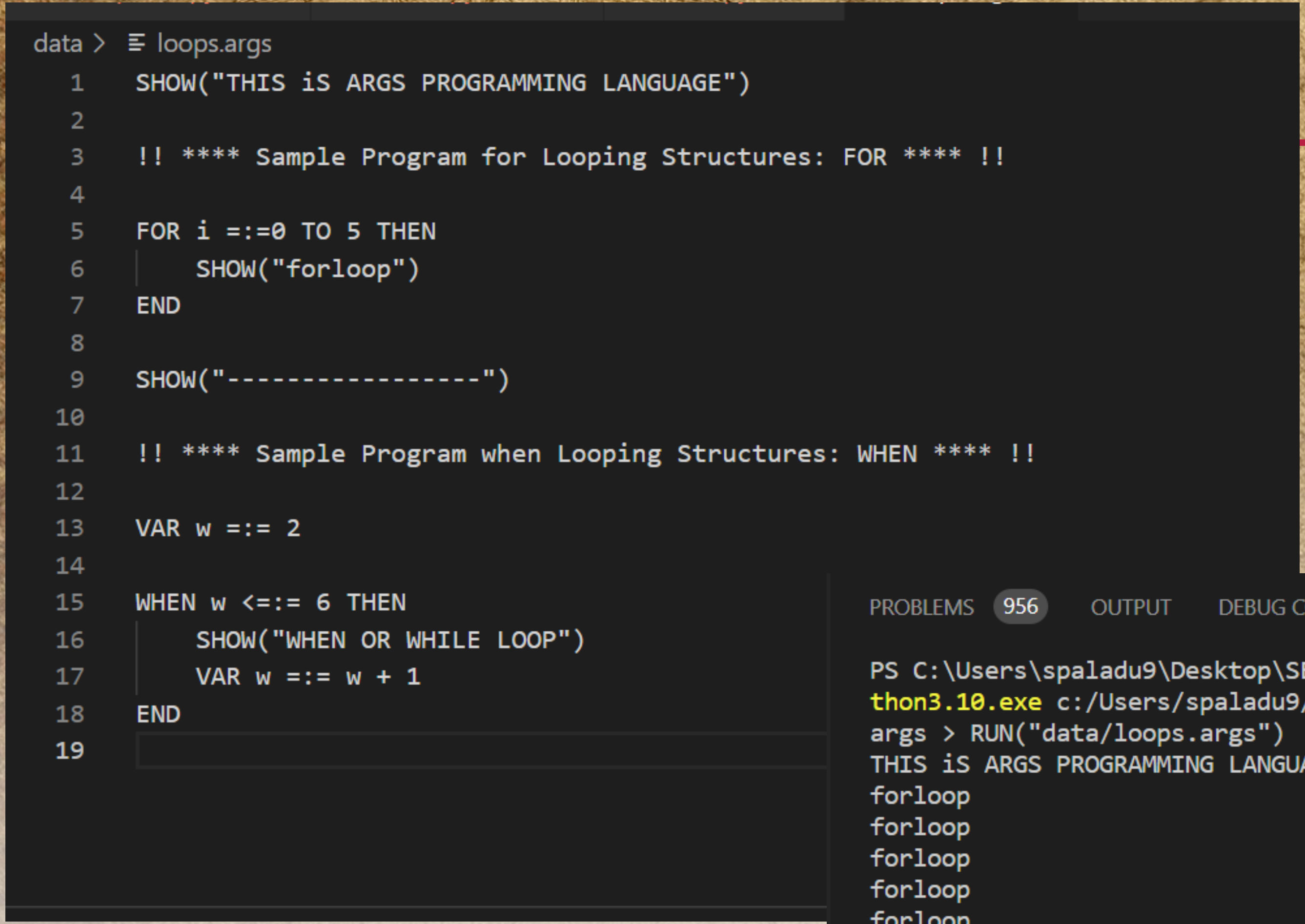


```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for Conditinal Constructs: IF THEN ELSE **** !!

VAR x =:= 0

IF x >=:= 10 THEN SHOW("IF-THEN condition") ELIF x >=:= 5 THEN SHOW("ELIF-THEN condition") ELSE SHOW("ELSE condition")

!! **** Sample Program for Conditinal Constructs: TERNARY OPERATOR **** !!

VAR x =:= 0

TERNARY x >=:= 10 ?? SHOW("TERNARY TRUE") :: SHOW("TERNARY FALSE")

```



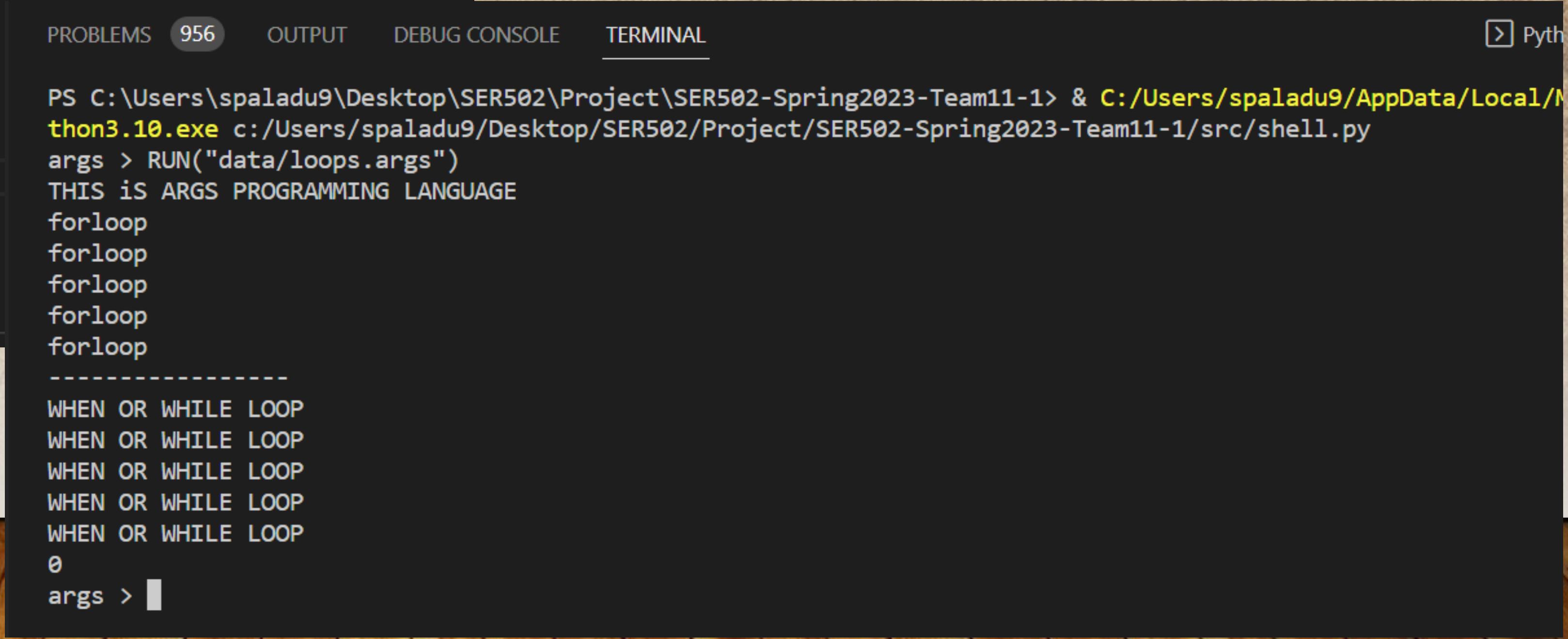
```
PS C:\Users\spaladu9\Desktop\SER502\Project\SER502-Spring2023-Team11-1> & C:/Users/spaladu9/AppData/Local/Microsoft/WindowsApps/py
thon3.10.exe c:/Users/spaladu9/Desktop/SER502/Project/SER502-Spring2023-Team11-1/src/shell.py
args > RUN("data/conditional.args")
THIS iS ARGS PROGRAMMING LANGUAGE
ELSE condition
TERNARY FALSE
0
args > 
```

SAMPLE PROGRAM - 5



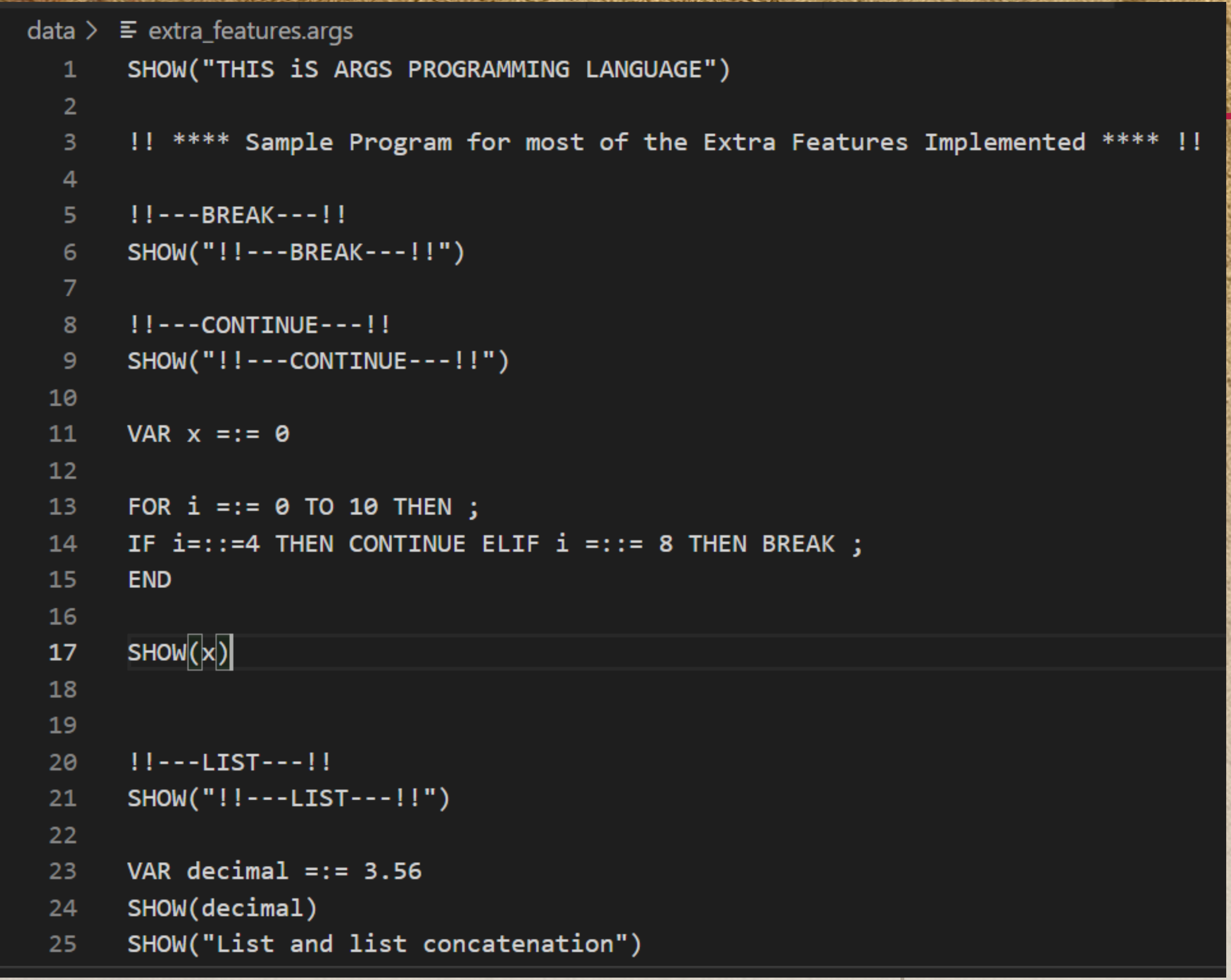
data > loops.args

```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for Looping Structures: FOR **** !!

FOR i =:=0 TO 5 THEN
    SHOW("forloop")
END

SHOW("-----------------")

!! **** Sample Program when Looping Structures: WHEN **** !!

VAR w =:= 2

WHEN w <=:= 6 THEN
    SHOW("WHEN OR WHILE LOOP")
    VAR w =:= w + 1
END

```



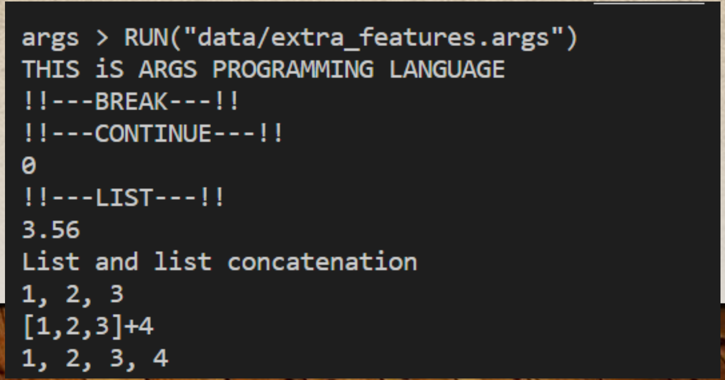
```
PS C:\Users\spaladu9\Desktop\SER502\Project\SER502-Spring2023-Team11-1> & C:/Users/spaladu9/AppData/Local/N
thon3.10.exe c:/Users/spaladu9/Desktop/SER502/Project/SER502-Spring2023-Team11-1/src/shell.py
args > RUN("data/loops.args")
THIS iS ARGS PROGRAMMING LANGUAGE
forloop
forloop
forloop
forloop
forloop
-----------------
WHEN OR WHILE LOOP
WHEN OR WHILE LOOP
WHEN OR WHILE LOOP
WHEN OR WHILE LOOP
WHEN OR WHILE LOOP
0
args >
```

SAMPLE PROGRAM - 6



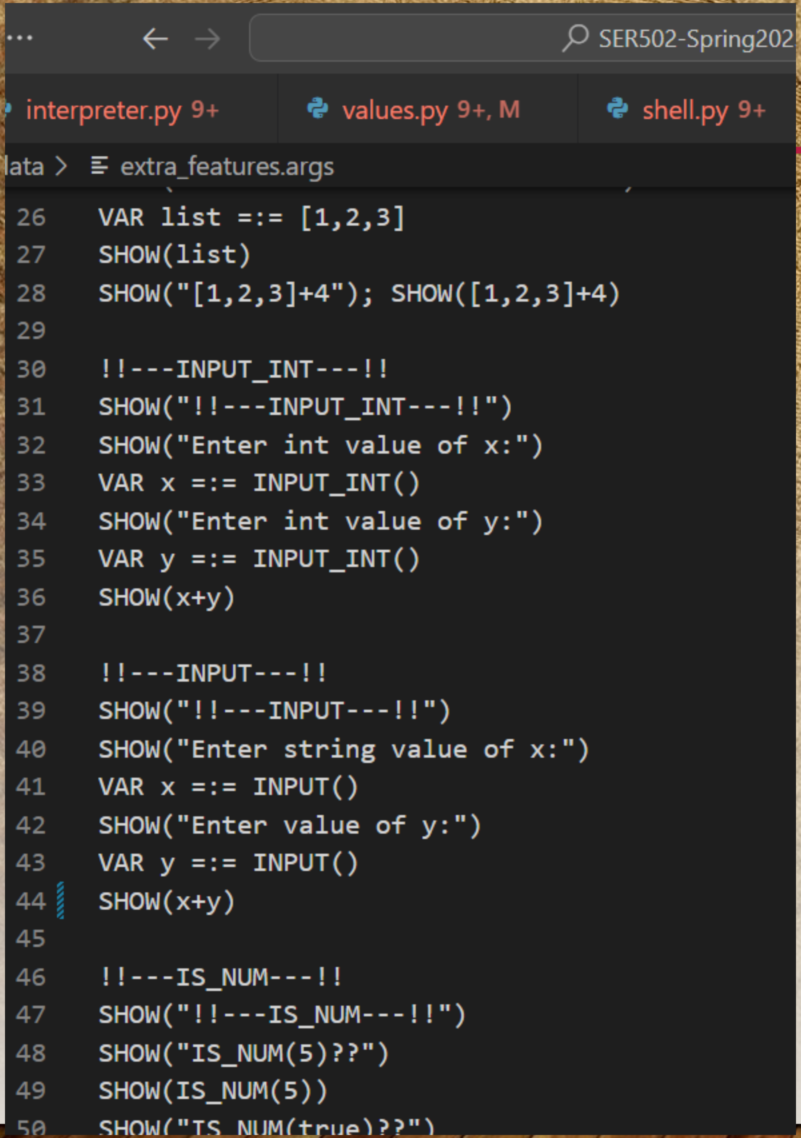
data > extra_features.args

```
SHOW("THIS iS ARGS PROGRAMMING LANGUAGE")

!! **** Sample Program for most of the Extra Features Implemented **** !!

!!---BREAK---!!
SHOW("!!---BREAK---!!")

!!---CONTINUE---!!
SHOW("!!---CONTINUE---!!")

VAR x =:= 0

FOR i =:= 0 TO 10 THEN ;
IF i=::=4 THEN CONTINUE ELIF i =::= 8 THEN BREAK ;
END

SHOW(x)


!!---LIST---!!
SHOW("!!---LIST---!!")

VAR decimal =:= 3.56
SHOW(decimal)
SHOW("List and list concatenation")
```



```
args > RUN("data/extra_features.args")
THIS iS ARGS PROGRAMMING LANGUAGE
!!---BREAK---!!
!!---CONTINUE---!!
0
!!---LIST---!!
3.56
List and list concatenation
1, 2, 3
[1,2,3]+4
1, 2, 3, 4
```

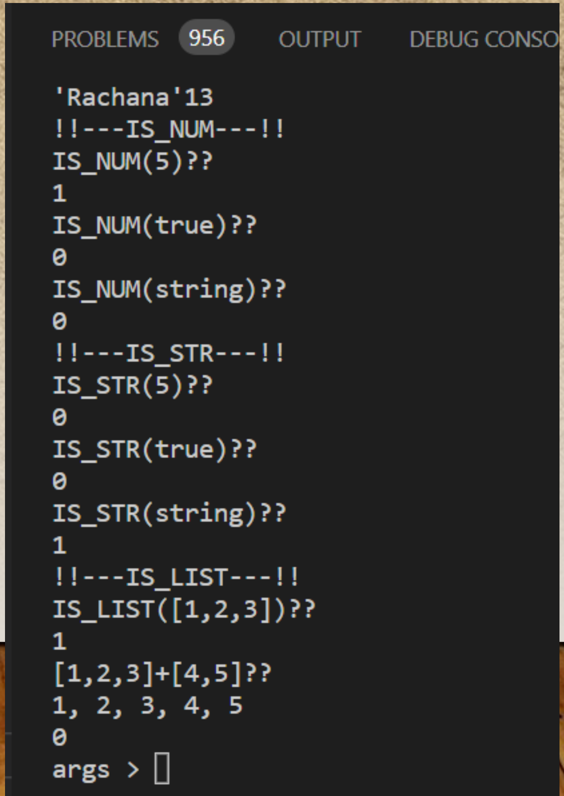
SAMPLE PROGRAM - 7



interpreter.py 9+ values.py 9+, M shell.py 9+

lata > extra_features.args

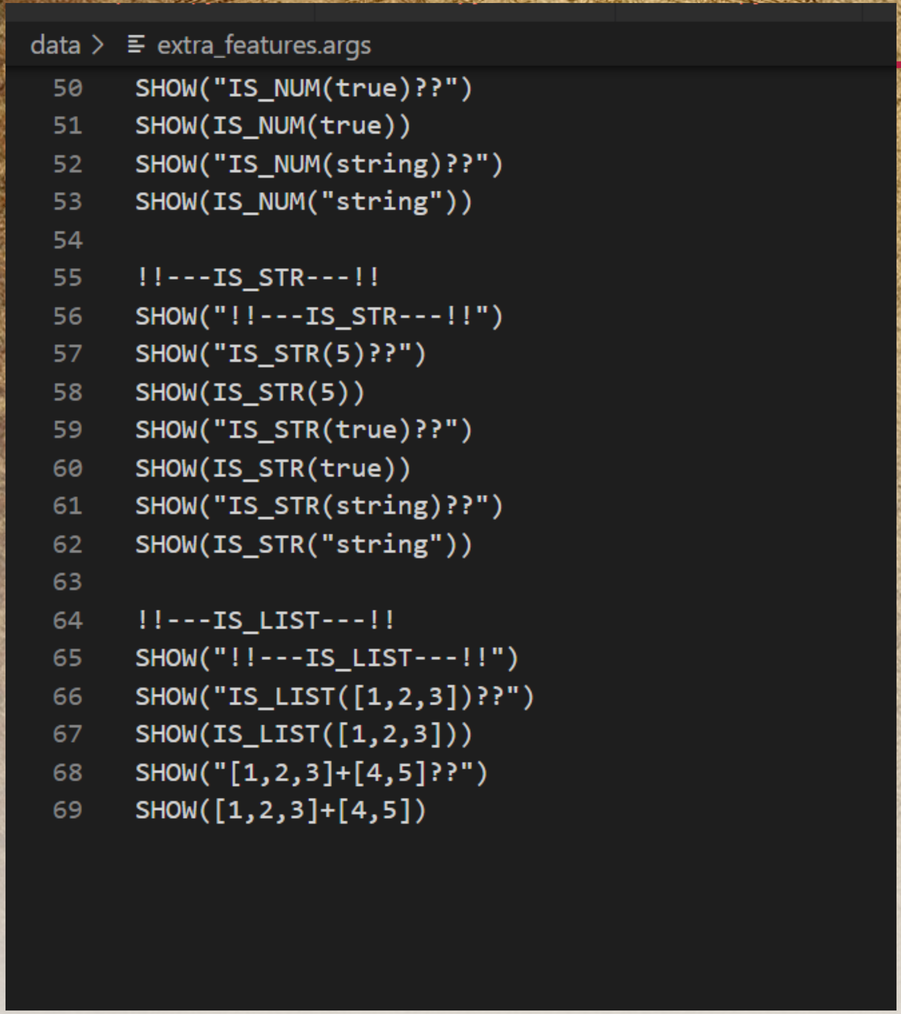
```
VAR list =:= [1,2,3]
SHOW(list)
SHOW("[1,2,3]+4"); SHOW([1,2,3]+4)

!!---INPUT_INT---!!
SHOW("!!---INPUT_INT---!!")
SHOW("Enter int value of x:")
VAR x =:= INPUT_INT()
SHOW("Enter int value of y:")
VAR y =:= INPUT_INT()
SHOW(x+y)

!!---INPUT---!!
SHOW("!!---INPUT---!!")
SHOW("Enter string value of x:")
VAR x =:= INPUT()
SHOW("Enter value of y:")
VAR y =:= INPUT()
SHOW(x+y)

!!---IS_NUM---!!
SHOW("!!---IS_NUM---!!")
SHOW("IS_NUM(5)??")
SHOW(IS_NUM(5))
SHOW("IS_NUM(true)??")
```



PROBLEMS 956 OUTPUT DEBUG CONSO

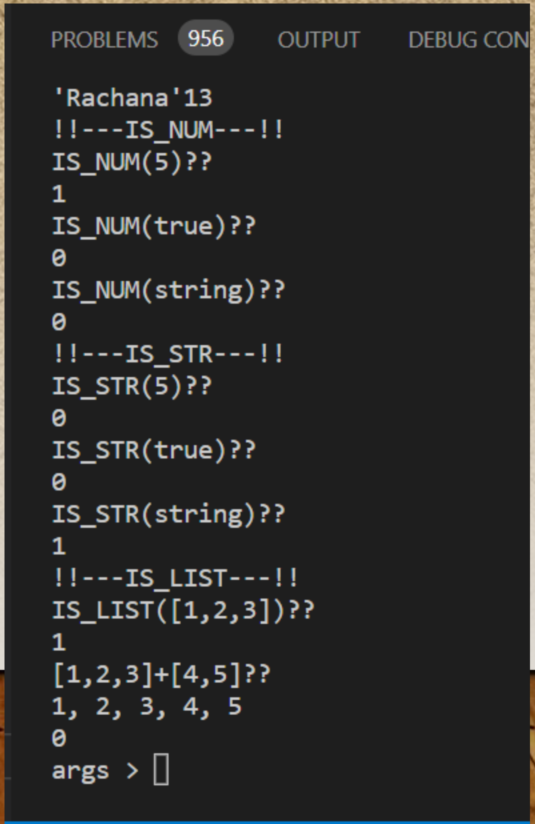
```
'Rachana'13
!!---IS_NUM---!!
IS_NUM(5)??
1
IS_NUM(true)??
0
IS_NUM(string)??
0
!!---IS_STR---!!
IS_STR(5)??
0
IS_STR(true)??
0
IS_STR(string)??
1
!!---IS_LIST---!!
IS_LIST([1,2,3])??
1
[1,2,3]+[4,5]??
1, 2, 3, 4, 5
0
args > 
```

SAMPLE PROGRAM - 8



data > extra_features.args

```
SHOW("IS_NUM(true)??")
SHOW(IS_NUM(true))
SHOW("IS_NUM(string)??")
SHOW(IS_NUM("string"))

!!---IS_STR---!!
SHOW("!!---IS_STR---!!")
SHOW("IS_STR(5)??")
SHOW(IS_STR(5))
SHOW("IS_STR(true)??")
SHOW(IS_STR(true))
SHOW("IS_STR(string)??")
SHOW(IS_STR("string"))

!!---IS_LIST---!!
SHOW("!!---IS_LIST---!!")
SHOW("IS_LIST([1,2,3])??")
SHOW(IS_LIST([1,2,3]))
SHOW("[1,2,3]+[4,5]??")
SHOW([1,2,3]+[4,5])
```



```
'Rachana'13
!!---IS_NUM---!!
IS_NUM(5)??
1
IS_NUM(true)??
0
IS_NUM(string)??
0
!!---IS_STR---!!
IS_STR(5)??
0
IS_STR(true)??
0
IS_STR(string)??
1
!!---IS_LIST---!!
IS_LIST([1,2,3])??
1
[1,2,3]+[4,5]??
1, 2, 3, 4, 5
0
args > 
```

ARGS

Version 1

Youtube Link : https://youtu.be/cem-eW2uvN8