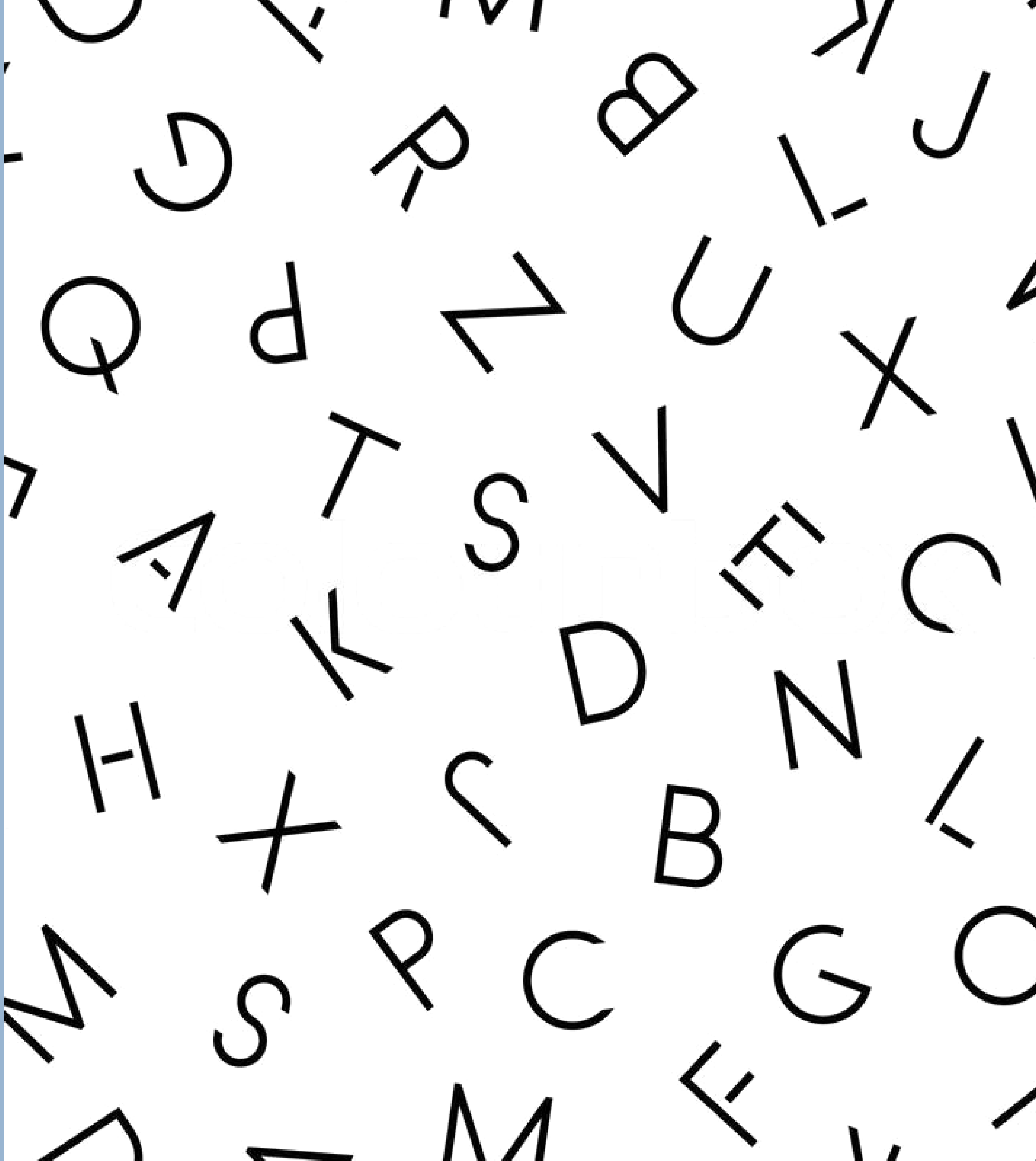
LETTER

TEAM MEMBERS:

- ARUSHI SHAH
- PRADYUMN MOHTA
- PAVAN KALYAN LINGUTLA



Introduction

A language designed for simplicity

Single letter keywords reduce verbosity

Key Features:

- Strong typing
- Comprehensive data structures
- Clear syntax
- Intuitive learning curve

Language Features

- Data Types and Variables: Integer, String, Boolean.
- Arithmetic Operations: Addition, Subtraction, Multiplication, Division.
- Control Structures: If-Else, While, For loops.
- Functions: Declaration, Return values, Parameters.
- Data Structures: Arrays, Stacks, Queues.
- String Operations: Uppercase, Lowercase, Comparison.
- Additional Features: Constants, Input/Output, Ternary operator, Comments.

Grammer Components

Keywords:

- T - Integer type
- S - String type
- B - Boolean type
- P - Print
- M - Method
- A - Array
- I - If statement
- E - Else statement
- W - While loop
- F - For loop
- R - Return
- C - Constant

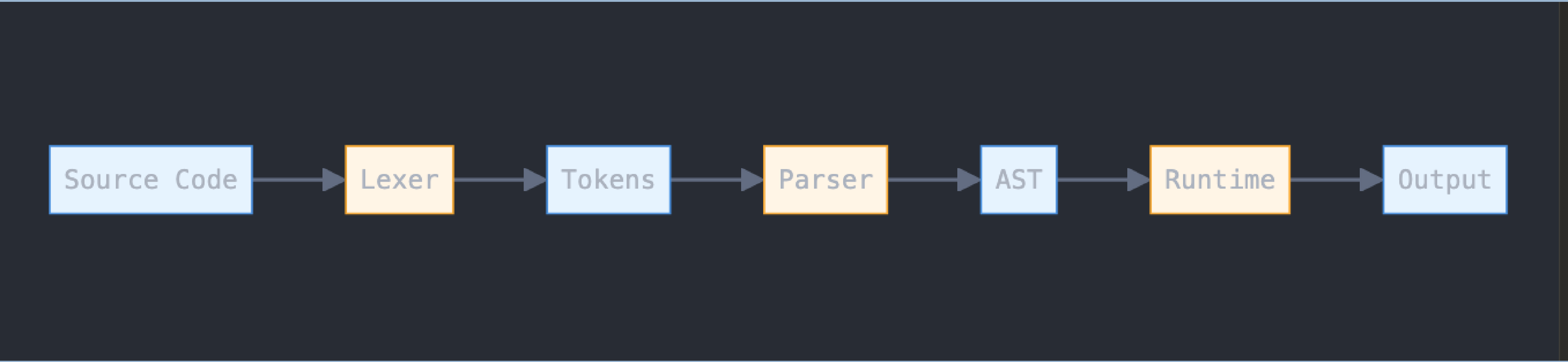
Operators:

- Arithmetic: +, -, *, /
- Relational: <, >, =
- Logical: &, |, !
- Ternary: ? :

Delimiters:

- (), {}, [], ;, ., ,

Architecture



$$\begin{tikzcd}
\boxed{\text{Source Code}} \arrow[r] & \boxed{\text{Lexer}} \arrow[r] & \boxed{\text{Tokens}} \arrow[r] & \boxed{\text{Parser}} \arrow[r] & \boxed{\text{AST}} \arrow[r] & \boxed{\text{Runtime}} \arrow[r] & \boxed{\text{Output}}
\end{tikzcd}$$

Implementation Details

- Lexer:
  1. *Token generation*
  2. *Character processing*
  3. *Comment handling*
- Parser:
  1. *AST creation*
  2. *Grammar validation*
  3. *Error detection*
- Runtime:
  1. *Memory management*
  2. *Scope handling*
  3. *Operation execution*

Lexcial Analysis

Process:

1. Source code → Character stream
2. Character classification
3. Token generation
4. Error detection

Error Handling:

- Invalid characters
- Malformed strings
- Unknown tokens

Example:

Source: T x = 5;

Tokens:

- TYPE_INT "T"
- IDENTIFIER "x"
- EQUAL "="
- INTEGER_LITERAL "5"
- SEMICOLON ";"

Parser

AST Node Types:

- Declaration nodes
- Expression nodes
- Statement nodes
- Function nodes
- Control flow nodes

Example:

Code: T x = a + b;

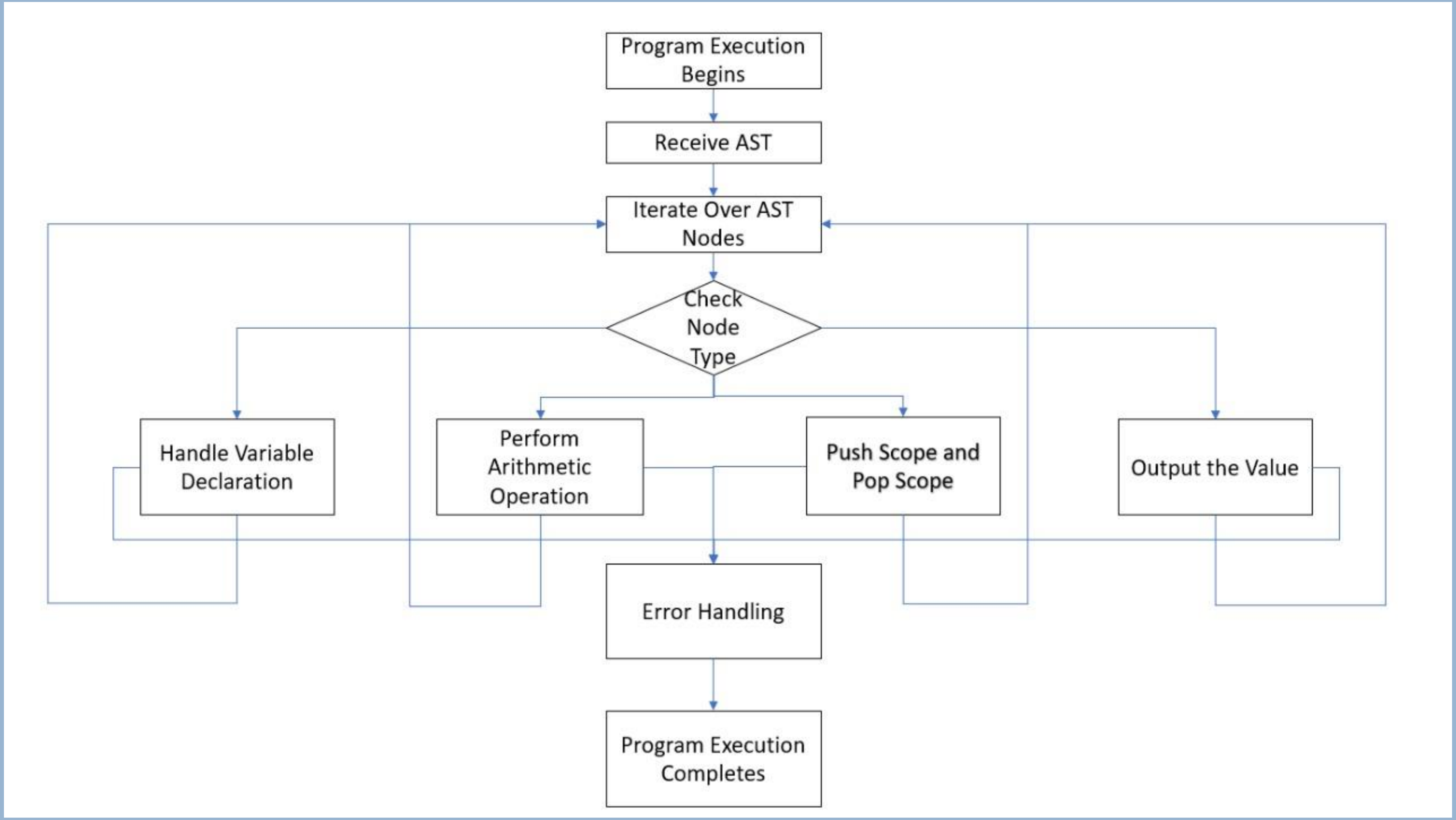
AST Structure:

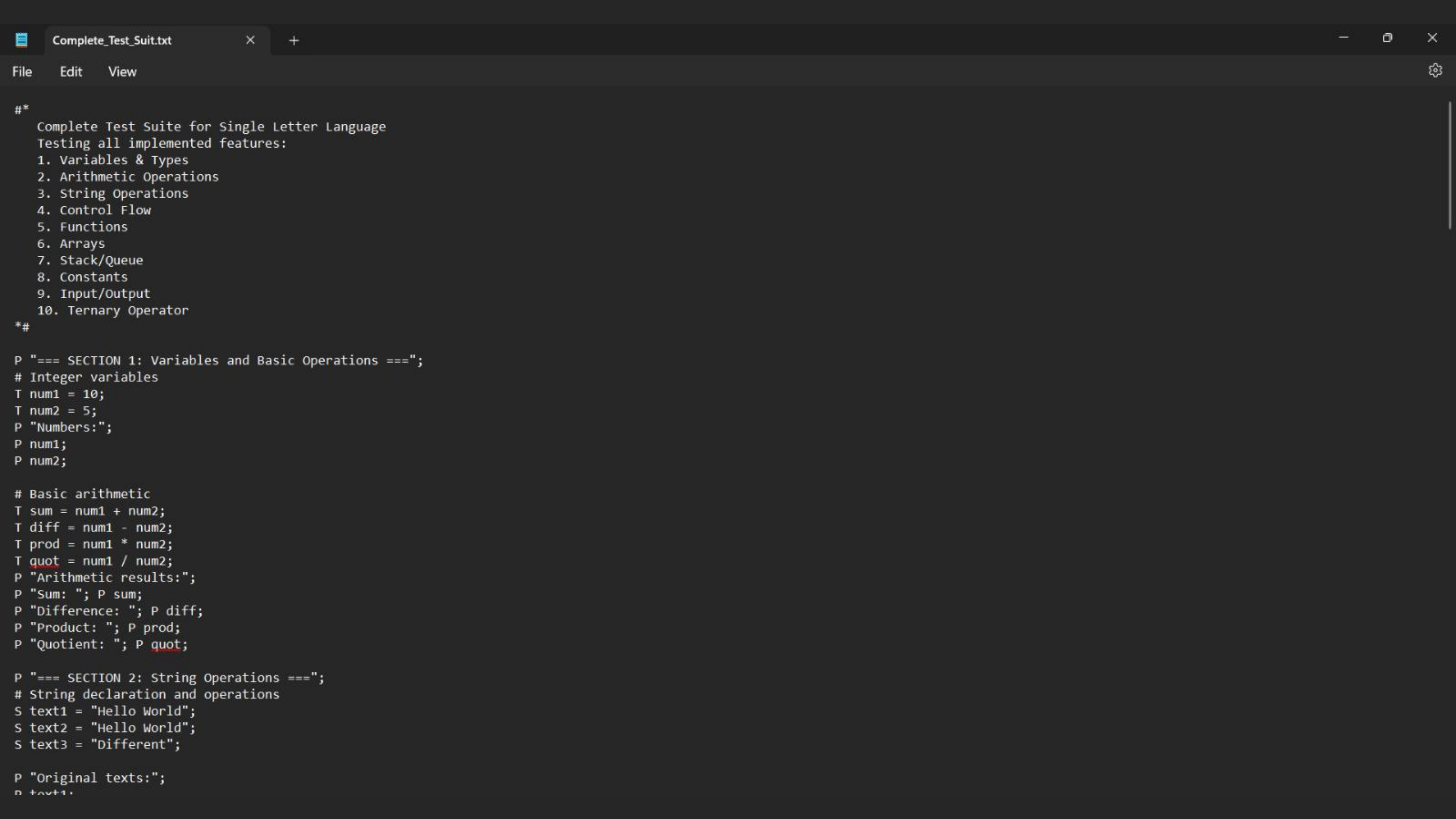
```
- Declaration
  ├── Type: INTEGER
  ├── Identifier: x
  └── Binary Operation
      ├── Left: Identifier(a)
      └── Right: Identifier(b)
```

Key Functions in Runtime Class:

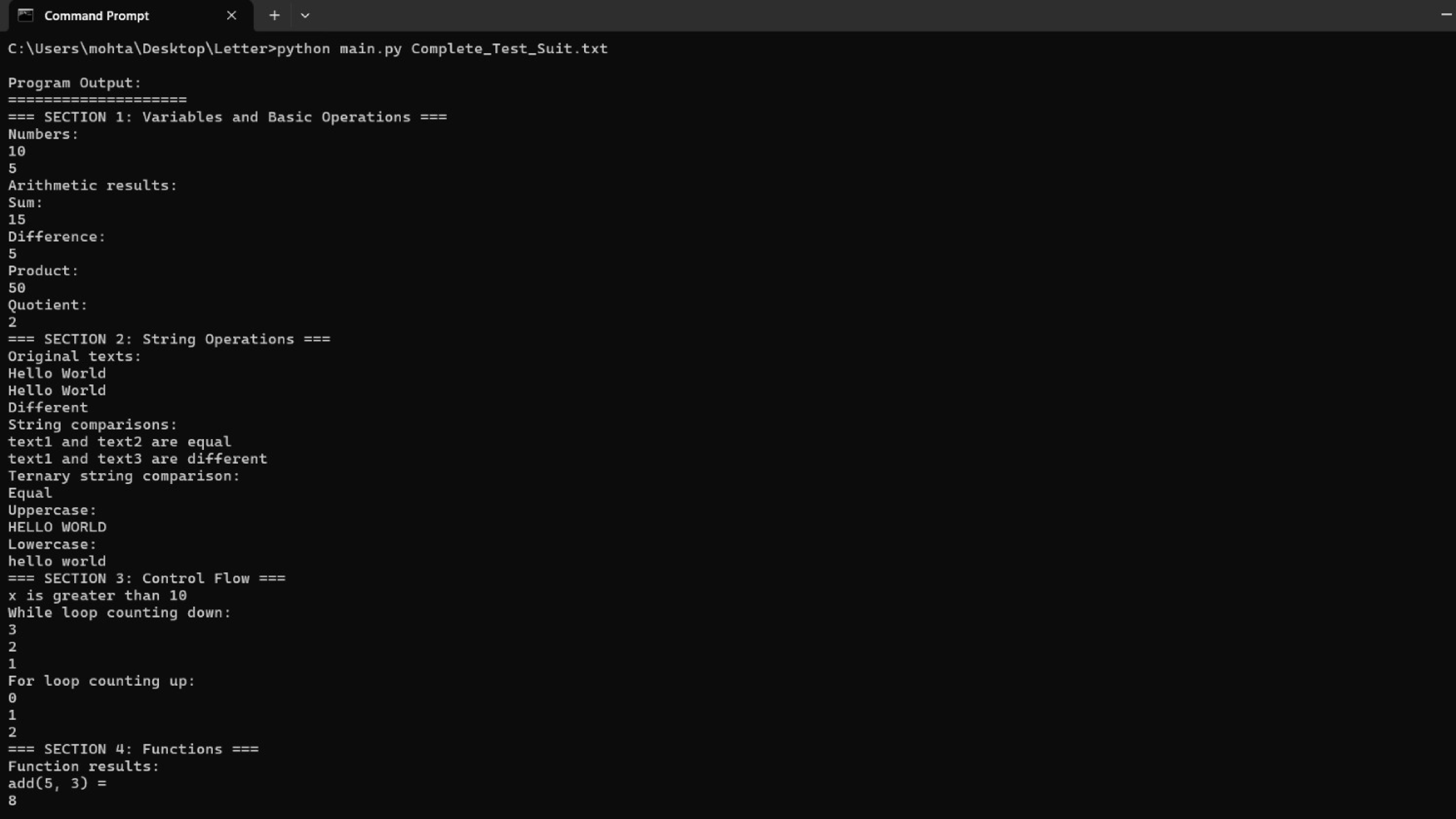
1. Memory Management:
   - *Variables are stored in a dictionary (self.variables).*
   - *Constants are stored in self.constants.*
   - *Functions are stored in self.functions, where each function has parameters and a body.*
2. Node Execution:
   - *Each AST node is handled by a specific method, like execute_assignment, execute_print, execute_function_call, etc.*
   - *Operations like binary operations (execute_binary_op), control structures (if-else, loops), and function calls are supported.*

Execution


Program Execution Begins
Receive AST
Iterate Over AST Nodes
Check Node Type
Handle Variable Declaration
Perform Arithmetic Operation
Push Scope and Pop Scope
Output the Value
Error Handling
Program Execution Completes

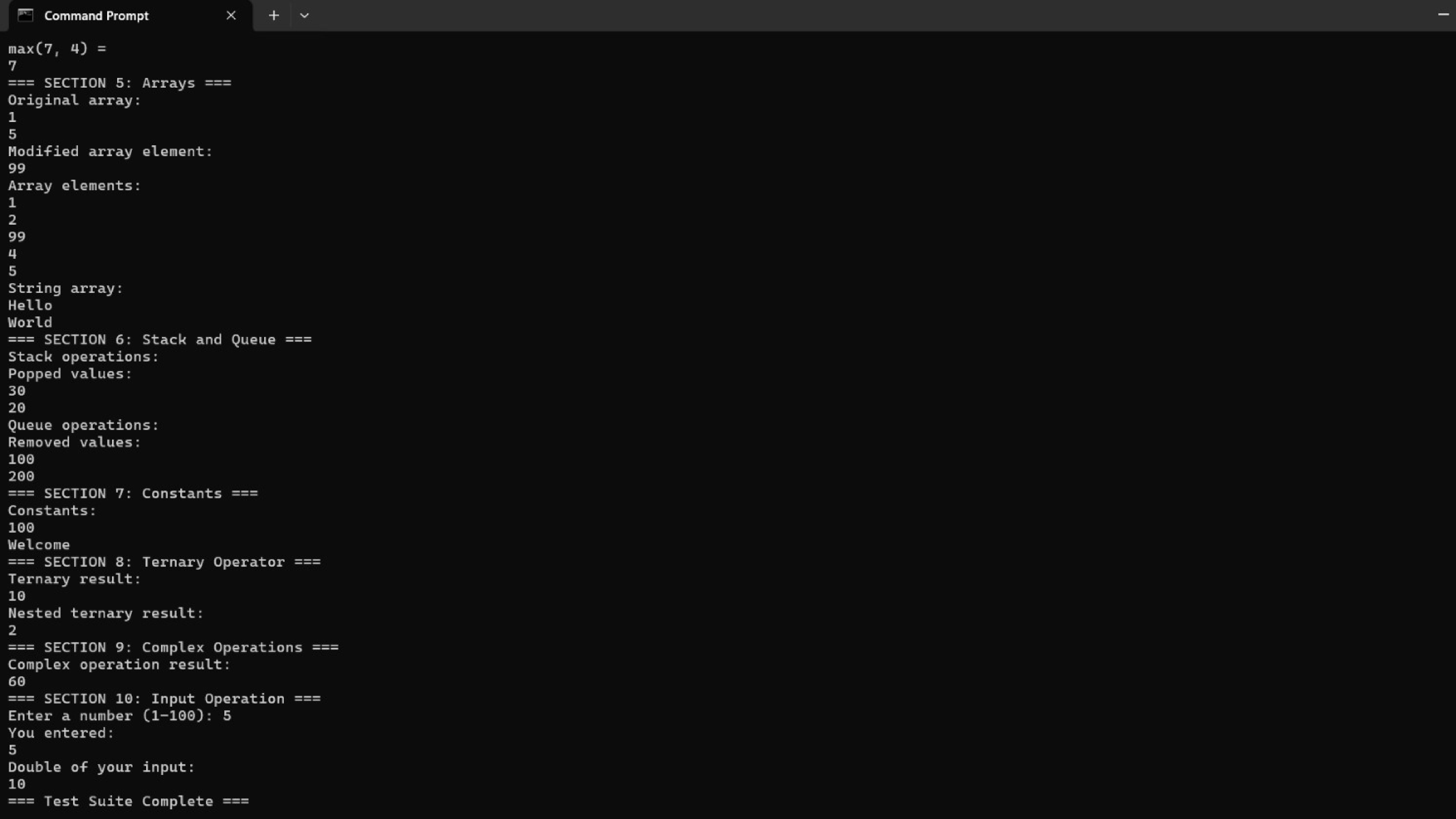


```
#*
   Complete Test Suite for Single Letter Language
   Testing all implemented features:
   1. Variables & Types
   2. Arithmetic Operations
   3. String Operations
   4. Control Flow
   5. Functions
   6. Arrays
   7. Stack/Queue
   8. Constants
   9. Input/Output
   10. Ternary Operator
*#

P "=== SECTION 1: Variables and Basic Operations ===";
# Integer variables
T num1 = 10;
T num2 = 5;
P "Numbers:";
P num1;
P num2;

# Basic arithmetic
T sum = num1 + num2;
T diff = num1 - num2;
T prod = num1 * num2;
T quot = num1 / num2;
P "Arithmetic results:";
P "Sum: "; P sum;
P "Difference: "; P diff;
P "Product: "; P prod;
P "Quotient: "; P quot;

P "=== SECTION 2: String Operations ===";
# String declaration and operations
S text1 = "Hello World";
S text2 = "Hello World";
S text3 = "Different";

P "Original texts:";
P text1;
```



Command Prompt

```
C:\Users\mohta\Desktop\Letter>python main.py Complete_Test_Suit.txt

Program Output:
====================
=== SECTION 1: Variables and Basic Operations ===
Numbers:
10
5
Arithmetic results:
Sum:
15
Difference:
5
Product:
50
Quotient:
2
=== SECTION 2: String Operations ===
Original texts:
Hello World
Hello World
Different
String comparisons:
text1 and text2 are equal
text1 and text3 are different
Ternary string comparison:
Equal
Uppercase:
HELLO WORLD
Lowercase:
hello world
=== SECTION 3: Control Flow ===
x is greater than 10
While loop counting down:
3
2
1
For loop counting up:
0
1
2
=== SECTION 4: Functions ===
Function results:
add(5, 3) =
8
```



```
max(7, 4) =
7
=== SECTION 5: Arrays ===
Original array:
1
5
Modified array element:
99
Array elements:
1
2
99
4
5
String array:
Hello
World
=== SECTION 6: Stack and Queue ===
Stack operations:
Popped values:
30
20
Queue operations:
Removed values:
100
200
=== SECTION 7: Constants ===
Constants:
100
Welcome
=== SECTION 8: Ternary Operator ===
Ternary result:
10
Nested ternary result:
2
=== SECTION 9: Complex Operations ===
Complex operation result:
60
=== SECTION 10: Input Operation ===
Enter a number (1-100): 5
You entered:
5
Double of your input:
10
=== Test Suite Complete ===
```

Testing Strategy

- Individual components
- Integration testing
- Error cases
- Edge cases

Error Handling

The runtime checks for common errors during execution:

1. Syntax Errors: Handled during parsing (not runtime).
2. Runtime Errors:
   - Undefined variables.
   - Invalid operations (e.g., division by zero).
   - Array index out of bounds.

Questions

- Github repository: https://github.com/pavankalyan9564/SER502-Letter-Team3
- Emails:
  - ashsh223@asu.edu
  - pmohta@asu.edu
  - plingutl@asu.edu