

**Fatima Jinnah Women University**

**Submitted to: Dr. Bushra Sikander**

**Submitted by: Aimen Jillani (BCS-001)**

**Maryam Siddiqui (BCS-027)**

**Noor Fatima (BCS-035)**

**Wishma Noor (BCS-041)**

**Course: Compiler Construction**

**Semester/Department: BCS (6th SEMESTER)**

**Table of Content**

[**Define your own language** 3](#_Toc162955522)

[**List down the following for your language** 3](#_Toc162955523)

[**Define pattern of the following** 5](#_Toc162955524)

[**LIST OF TOKEN** 6](#_Toc162955525)

[**GRAMMER** 7](#_Toc162955526)

[**Transition Diagram** 7](#_Toc162955526)

## **Define your own language**

**Language name:**

The name of our Language is **TerraCode.**

**General description:**

The term **"Terra"** derives from Latin, meaning Earth or land, while "code" typically refers to instructions or commands used in programming. When combined, "Terra" and "code" suggest a compiler aimed at simulating or modeling natural systems,

It will have a more straightforward syntax that is similar to high-level languages like C, C# C++, which will make it simpler for newcomers to understand fundamental programming ideas. Because TerraCode will be case sensitive, while writing code, special attention must be paid to letter casing. In Axolotl, every sentence will terminate with a question mark (?), serving as a distinct division between several directives or phrases. White spaces, tabs, and new lines are ignored by TerraCode, so programmers can format the code for readability without compromising the program's functionality.

Common control structures found in TerraCode include decision making structures (agar (if)) , (agar, nhi to (if else)), loops (jab tak (while)) , (mqsd k lya (for)) and basic repetitive procedures that may be implemented by amateurs. During the development of a function or procedure the body of the function or procedure will be enclosed in asterisk (\*….\*) and return types will be supported by TerraCode. To make code easier to read, it will support the use of both single- and multi-line comments. Try-catch error codes—basic error handling techniques—will be implemented by our compiler.

# **List down the following for your language**

* Keywords
* Arithmetic Operators
* Relational operators
* Logical operators
* Data types (int, float)
* **KEYWORDS:**
  + agar (known as “if”)
  + nahi to (known as “else”)
  + mqsd k lya (known as “for”)
  + jab tak (known as “while”)
  + ruk jao (known as “break”)
  + jari rkhyn (known as “continue”)
  + wapis (known as “return”)
  + likhyn (known as “print”)
  + prhyn (known as “read”)
  + dikhayn (known as “show”)
  + kryn (known as “do”)
* tay shuda (known as “default”)
* function
* main
* to
* con
* Int
* Fl
* Ch
* St
* **RELATIONAL OPERATORS**

**< :**  less than

> : greater than

**<=**: less than or equal to

**>=** : greater than or equal to

**=** =: equal to

**!=**: not equal to

* **LOGICAL OPERATORS**

**&** : AND

**|** : OR

**!** : NOT

* **ARITHMETIC OPERATORS**

**+** : FOR ADDITION

**-** : FOR SUBTRACTION

**\*** : FOR MULTIPLICATION

**/** : FOR DIVISION

**%** : FOR REMAINDER

++: FOR INCREMENT

--: FOR INCREMENT

* **Assignment Operator**

**=** : equal

* **DATA TYPES:**

In (integer data type for whole numbers)

Fl (floating-point data type for decimal numbers)

Ch (character data type for single character)

St (String data type for more character and numbers)

With these elements defined, beginners can start using the TerraCode programming language to understand basic programming concepts and build their programming skills.

# **Define pattern of the following**

* + **Identifier**
  + **Integer constants**
  + **Real constants**
  + **Comments**
* **IDENTIFIER**

Must start with an alphabet and may contain letters and numbers but it should not contain any special characters.

* **Character constant:** A character enclosed in double quotations.
* **String constant:** Sequence of characters enclosed in single quotations.
* **INTEGER CONSTANTS**

Integer constants is comprised of only whole numbers without decimal points (like 10, 20,40).

* **REAL CONSTANTS**

An optional sequence of digits, followed by a decimal point, and then a mandatory sequence of digits (like 10.982 or 12.00).

* **COMMENTS**

**Single Line Comment:**  Must be start with //

**Multiple Line Comment:**  Must be start with /\* and end with \*/

## **LIST OF TOKEN**

|  |  |
| --- | --- |
| **Symbols** | **Tokens** |
| < | LT |
| <= | LTEQ |
| > | GT |
| >= | GTEQ |
| = = | EQ |
| = | ASOP |
| ! | NOT |
| != | NET |
| + | PLUS |
| - | SUB |
| \* | MUL |
| / | DIV |
| % | REM |
| // | No token generate as it comments for single line |
| \* | FUNC |
| - |  |
| “ ” | char |
| ‘’ | str |
| If statment | If |
| If –else statment | If else |
| while | while |
| /\* \*/ | No token generate as it comments for multiple line |
| break | Break |
| print | Print |
| return | return |
| Continue | Continue |
| For loop | for |
| ++ | Inc |
| -- | Dec |

# **GRAMMER**

START → BLOCK

BLOCK → { STMT\_LIST }

STMT\_LIST → STMT ( STMT\_LIST | ε )

STMT → DEC\_STMT | CONST\_DEC\_STMT | ASSIGN\_STMT | WHILE\_STMT | IF\_STMT | READ\_STMT | SHOW\_STMT | BLOCK

DEC\_STMT → TYPE ID\_LIST ;

TYPE → Int | Fl | St | Ch |

ID\_LIST → IDENTIFIER ( , ID\_LIST | ε )

CONST\_DEC\_STMT → con TYPE ID\_EXPR\_LIST ;

ID\_EXPR\_LIST → IDENTIFIER = EXPR ( , ID\_EXPR\_LIST | ε )

ASSIGN\_STMT → IDENTIFIER = EXPR ;

WHILE\_STMT → jab tak ( EXPR ) BLOCK

IF\_STMT → agar ( EXPR ) BLOCK ( nahi to BLOCK | ε )

READ\_STMT → prhyn IDENTIFIER ;

SHOW\_STMT → dikhayn EXPR ;

EXPR → PRIMARY | UNARY\_OPR EXPR | EXPR BIN\_OPR EXPR

PRIMARY → IDENTIFIER | INTEGER\_CONSTANT | FLOAT\_CONSTANT | CHARACTER\_CONSTANT | STRING\_CONSTANT | ( EXPR )

UNARY\_OPR → - | ~

BIN\_OPR → + | - | \*. | / | =. | < | > | <= | >= | ~= | & | |

IDENTIFIER → LETTER (LETTER | DIGIT)\*

INTEGER\_CONSTANT → [0-9]+

FLOAT\_CONSTANT → [0-9]\*\.[0-9]+

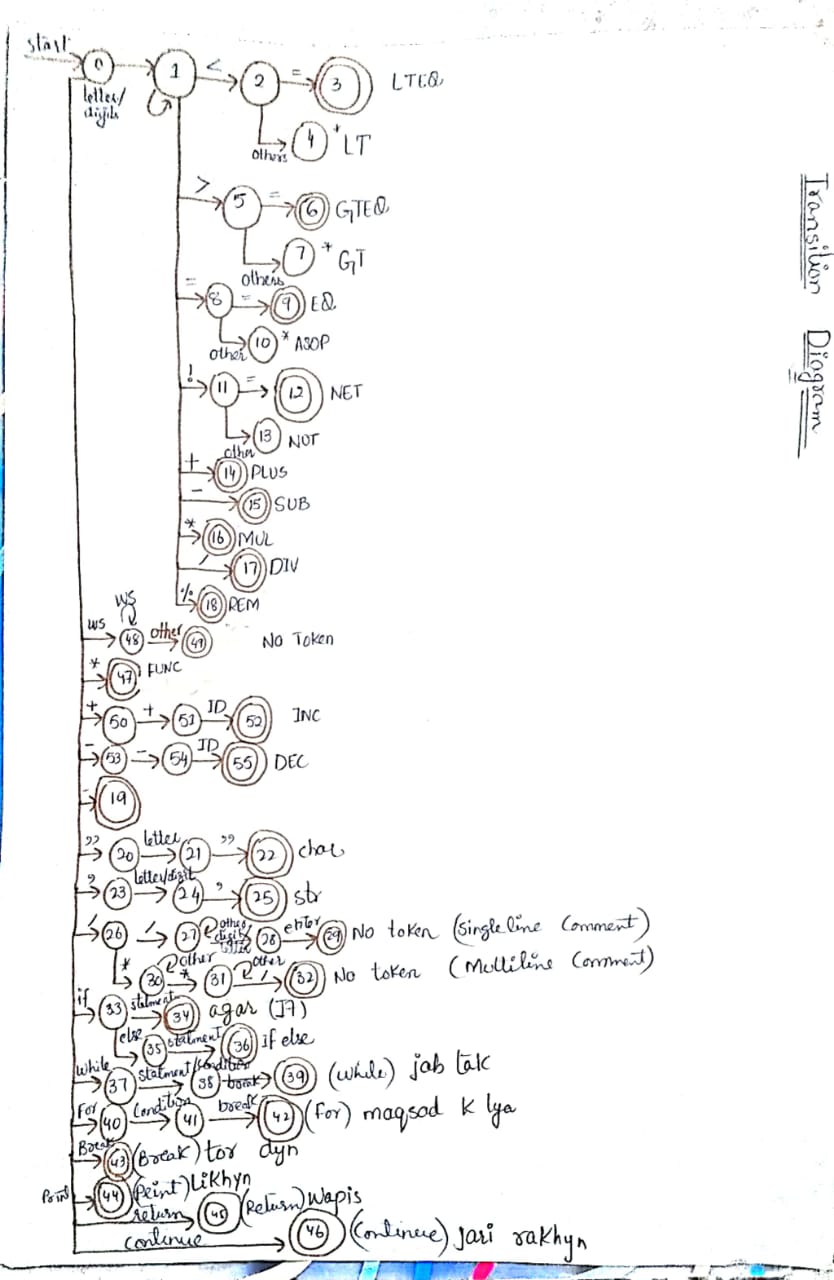
CHARACTER\_CONSTANT → " [a-zA-Z0-9] "

STRING\_CONSTANT → ' [a-zA-Z0-9]+ '

COMMENT → $ COMMENT\_BODY .

COMMENT\_BODY → COMMENT\_BODY | ε

## **Transition Diagram**



## **Code**

## import re

## # Define token classes

## class Token:

## def \_\_init\_\_(self, tok\_type, value):

## self.type = tok\_type

## self.value = value

## # Define token types

## class TokenType:

## # Keywords

## IF = "IF"

## ELSE = "ELSE"

## FOR = "FOR"

## WHILE = "WHILE"

## BREAK = "BREAK"

## CONTINUE = "CONTINUE"

## RETURN = "RETURN"

## PRINT = "PRINT"

## READ = "READ"

## SHOW = "SHOW"

## DO = "DO"

## DEFAULT = "DEFAULT"

## FUNCTION = "FUNCTION"

## MAIN = "MAIN"

## TO = "TO"

## CON = "CON"

## INT = "INT"

## FL = "FL"

## CH = "CH"

## ST = "ST"

## 

## # Relational operators

## LT = "LT"

## LTEQ = "LTEQ"

## GT = "GT"

## GTEQ = "GTEQ"

## EQ = "EQ"

## NET = "NET"

## 

## # Logical operators

## AND = "AND"

## OR = "OR"

## NOT = "NOT"

## 

## # Arithmetic operators

## PLUS = "PLUS"

## SUB = "SUB"

## MUL = "MUL"

## DIV = "DIV"

## REM = "REM"

## INC = "INC"

## DEC = "DEC"

## 

## # Assignment operator

## ASOP = "ASOP"

## 

## # Data types

## IDENTIFIER = "IDENTIFIER"

## INTEGER\_CONSTANT = "INTEGER\_CONSTANT"

## REAL\_CONSTANT = "REAL\_CONSTANT"

## CHAR\_CONSTANT = "CHAR\_CONSTANT"

## STRING\_CONSTANT = "STRING\_CONSTANT"

## 

## # Comments

## SINGLE\_LINE\_COMMENT = "SINGLE\_LINE\_COMMENT"

## MULTI\_LINE\_COMMENT = "MULTI\_LINE\_COMMENT"

## # Define patterns

## patterns = {

## # Keywords

## TokenType.IF: r'agar',

## TokenType.ELSE: r'nahi to',

## TokenType.FOR: r'mqsd k lya',

## TokenType.WHILE: r'jab tak',

## TokenType.BREAK: r'ruk jao',

## TokenType.CONTINUE: r'jari rkhyn',

## TokenType.RETURN: r'wapis',

## TokenType.PRINT: r'likhyn',

## TokenType.READ: r'prhyn',

## TokenType.SHOW: r'dikhayn',

## TokenType.DO: r'kryn',

## TokenType.DEFAULT: r'tay shuda',

## TokenType.FUNCTION: r'function',

## TokenType.MAIN: r'main',

## TokenType.TO: r'to',

## TokenType.CON: r'con',

## TokenType.INT: r'Int',

## TokenType.FL: r'Fl',

## TokenType.CH: r'Ch',

## TokenType.ST: r'St',

## 

## # Relational operators

## TokenType.LT: r'<',

## TokenType.LTEQ: r'<=',

## TokenType.GT: r'>',

## TokenType.GTEQ: r'>=',

## TokenType.EQ: r'==',

## TokenType.NET: r'!=',

## 

## # Logical operators

## TokenType.AND: r'&',

## TokenType.OR: r'\|',

## TokenType.NOT: r'!',

## 

## # Arithmetic operators

## TokenType.INC: r'\+\+', # Match ++ before +

## TokenType.DEC: r'--', # Match -- before -

## TokenType.PLUS: r'\+',

## TokenType.SUB: r'-',

## TokenType.MUL: r'\\*',

## TokenType.DIV: r'/',

## TokenType.REM: r'%',

## 

## # Assignment operator

## TokenType.ASOP: r'=',

## 

## # Data types

## TokenType.IDENTIFIER: r'[a-zA-Z][a-zA-Z0-9]\*',

## TokenType.INTEGER\_CONSTANT: r'\b\d+\b',

## TokenType.REAL\_CONSTANT: r'\b\d+\.\d+\b',

## TokenType.CHAR\_CONSTANT: r"'.'",

## TokenType.STRING\_CONSTANT: r'"[^"]\*"',

## 

## # Comments

## TokenType.SINGLE\_LINE\_COMMENT: r'//.\*',

## TokenType.MULTI\_LINE\_COMMENT: r'/\\*(.|\n)\*?\\*/'

## }

## # Define lexer

## class Lexer:

## def \_\_init\_\_(self, text):

## self.text = text

## self.tokens = []

## 

## def tokenize(self):

## token\_regex = '|'.join(f'(?P<{tok}>{pat})' for tok, pat in patterns.items())

## for match in re.finditer(token\_regex, self.text, re.MULTILINE):

## for name, value in match.groupdict().items():

## if value is not None:

## self.tokens.append(Token(name, value))

## break

## # Sample usage

## if \_\_name\_\_ == "\_\_main\_\_":

## # Ask user for file path and extension

## file\_path = input("Please enter the full path of the file including its extension (e.g., C:\\path\\to\\your\\file.txt): ")

## # Read the file

## try:

## with open(file\_path, "r") as file:

## text = file.read()

## 

## # Tokenize

## lexer = Lexer(text)

## lexer.tokenize()

## 

## # Print tokens in table format

## print("+---------------------+----------------------+")

## print("| Lexeme | Token |")

## print("+---------------------+----------------------+")

## for token in lexer.tokens:

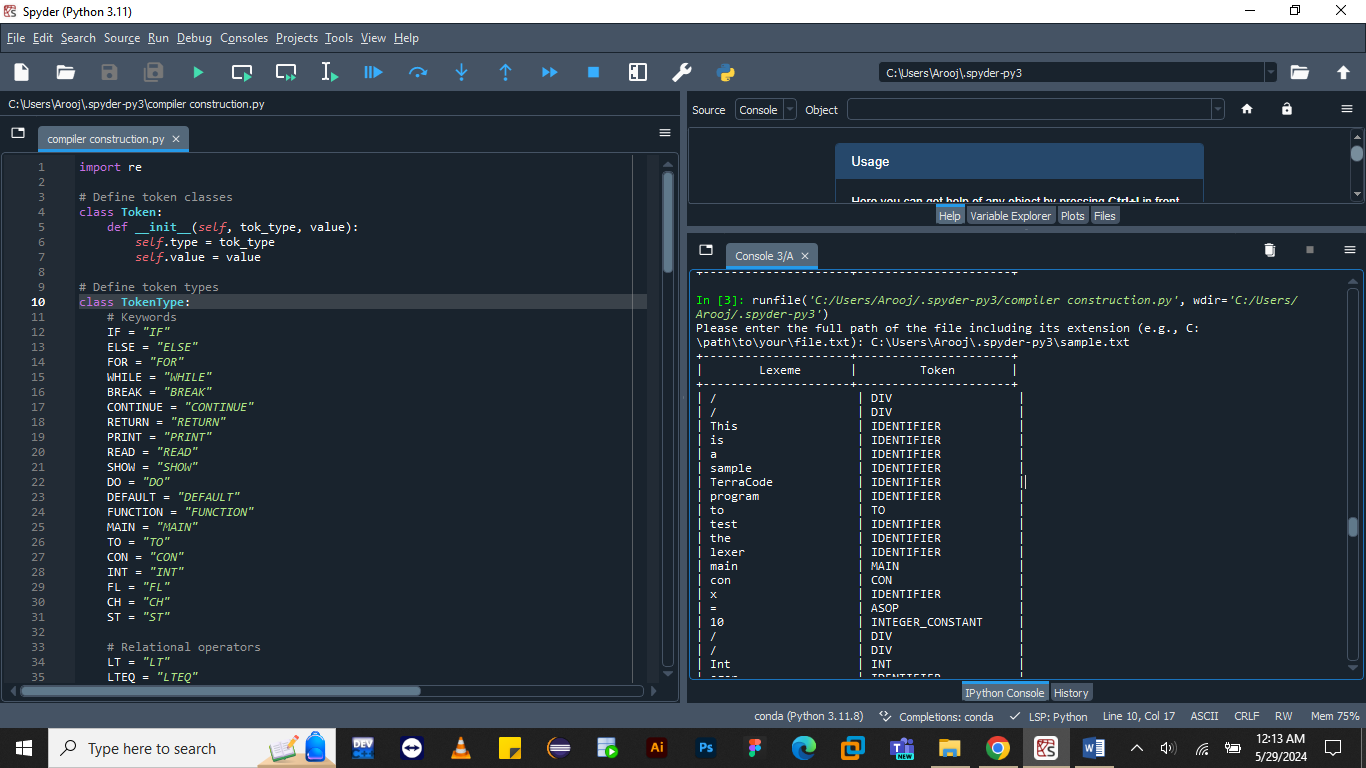
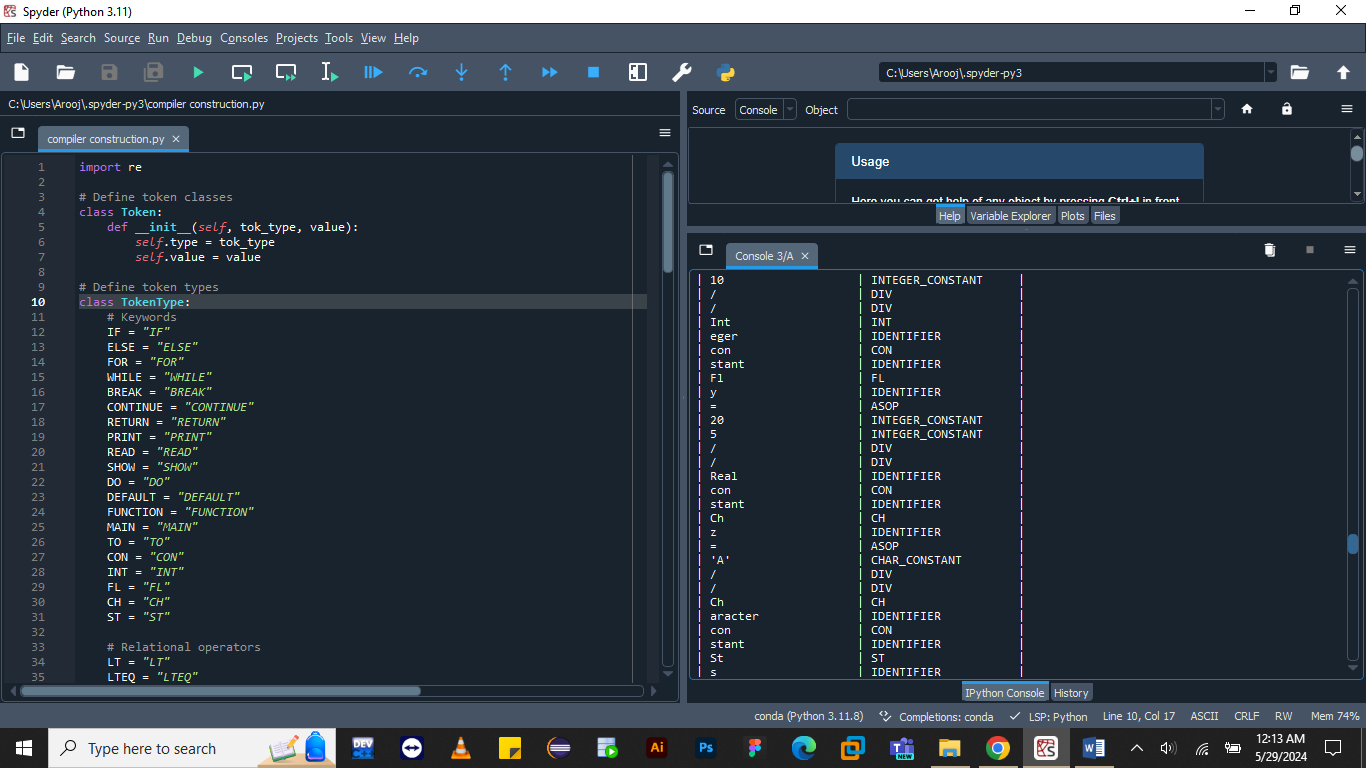
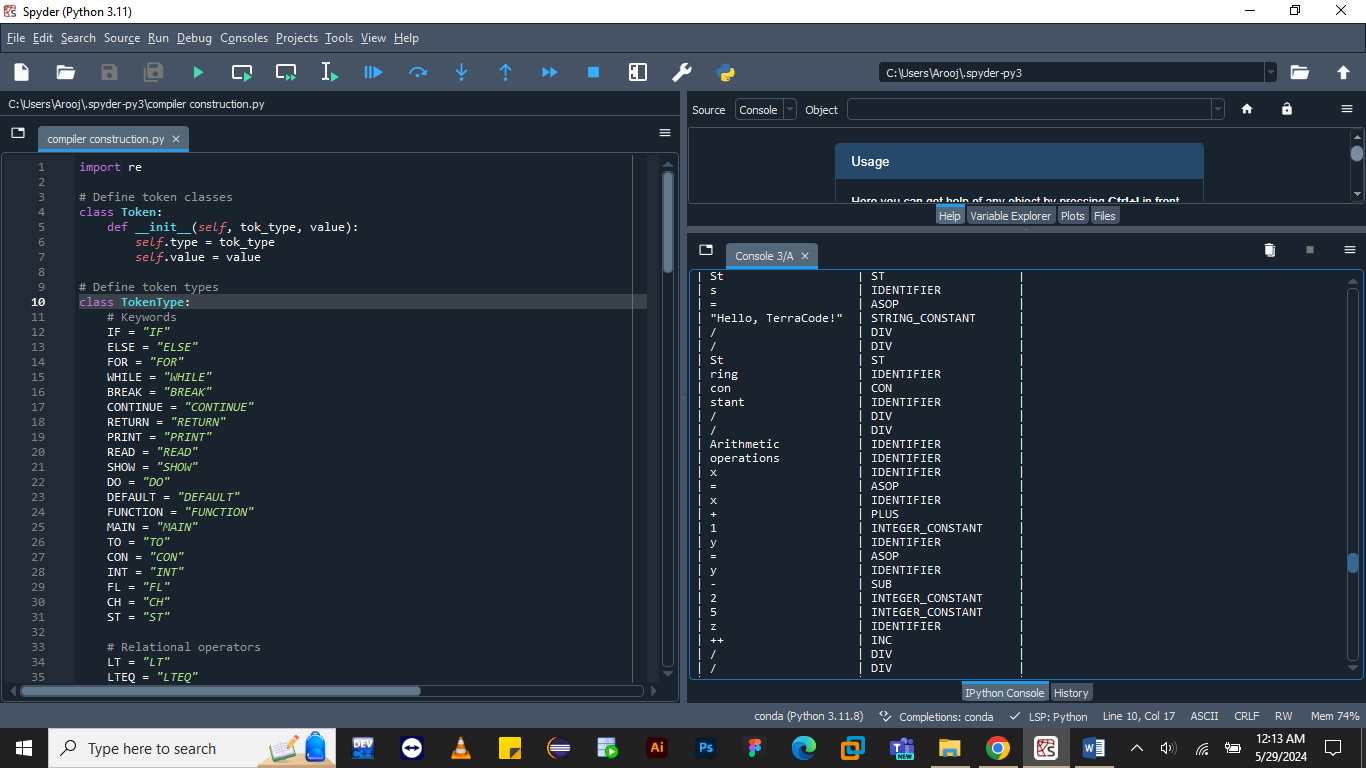
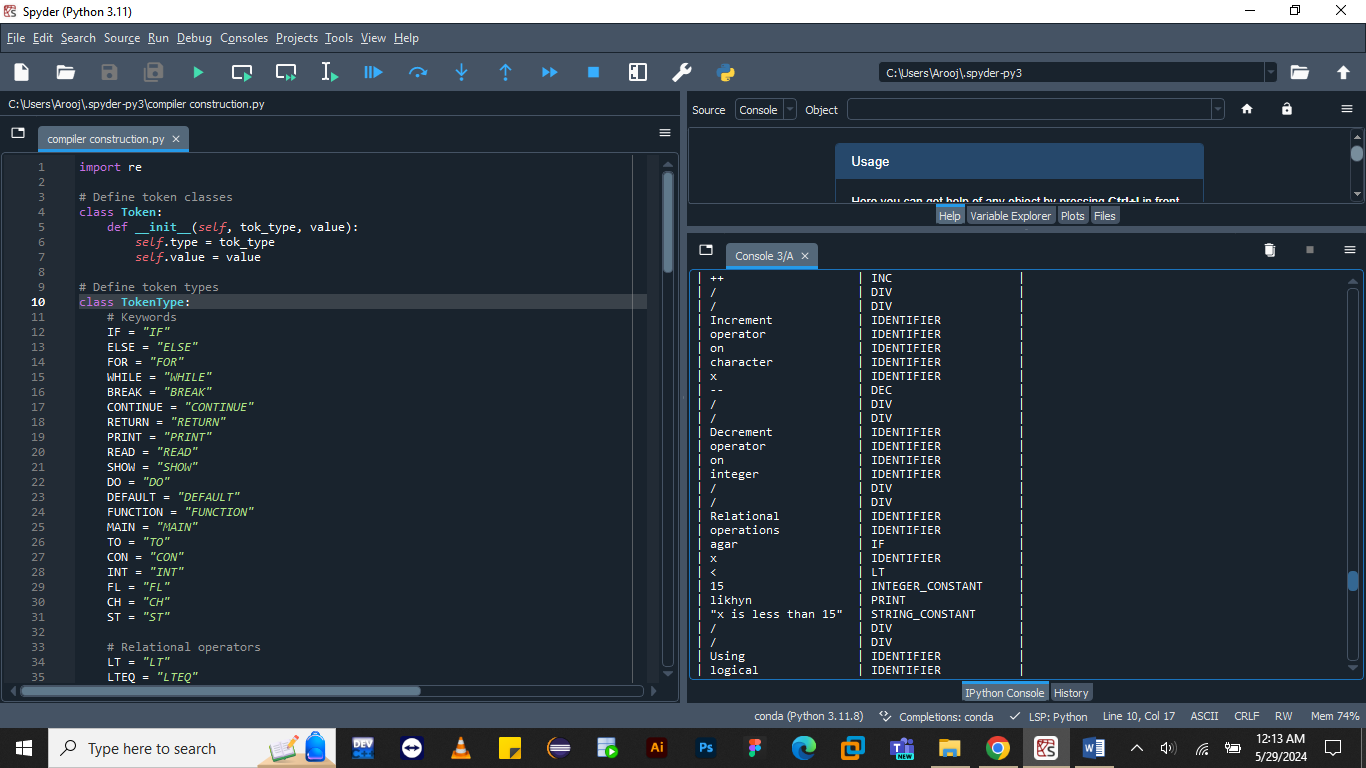
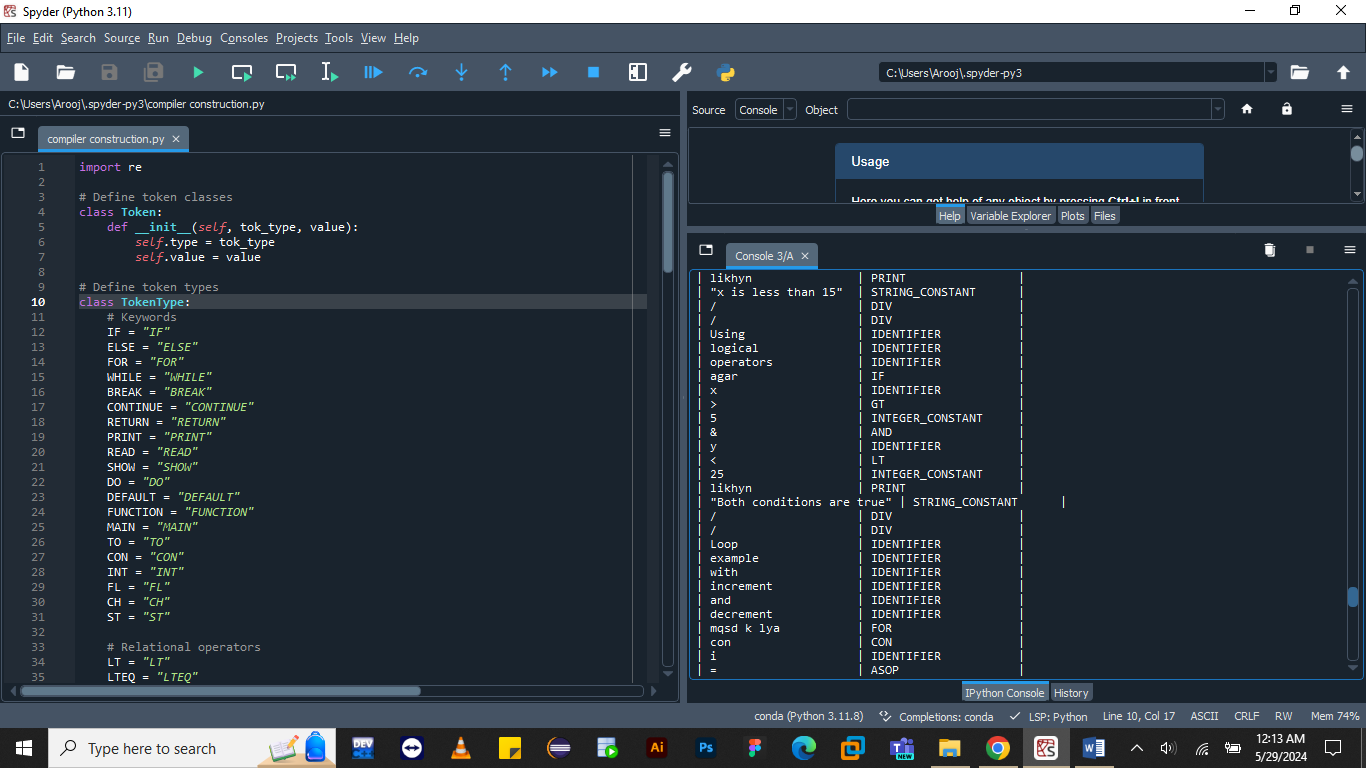
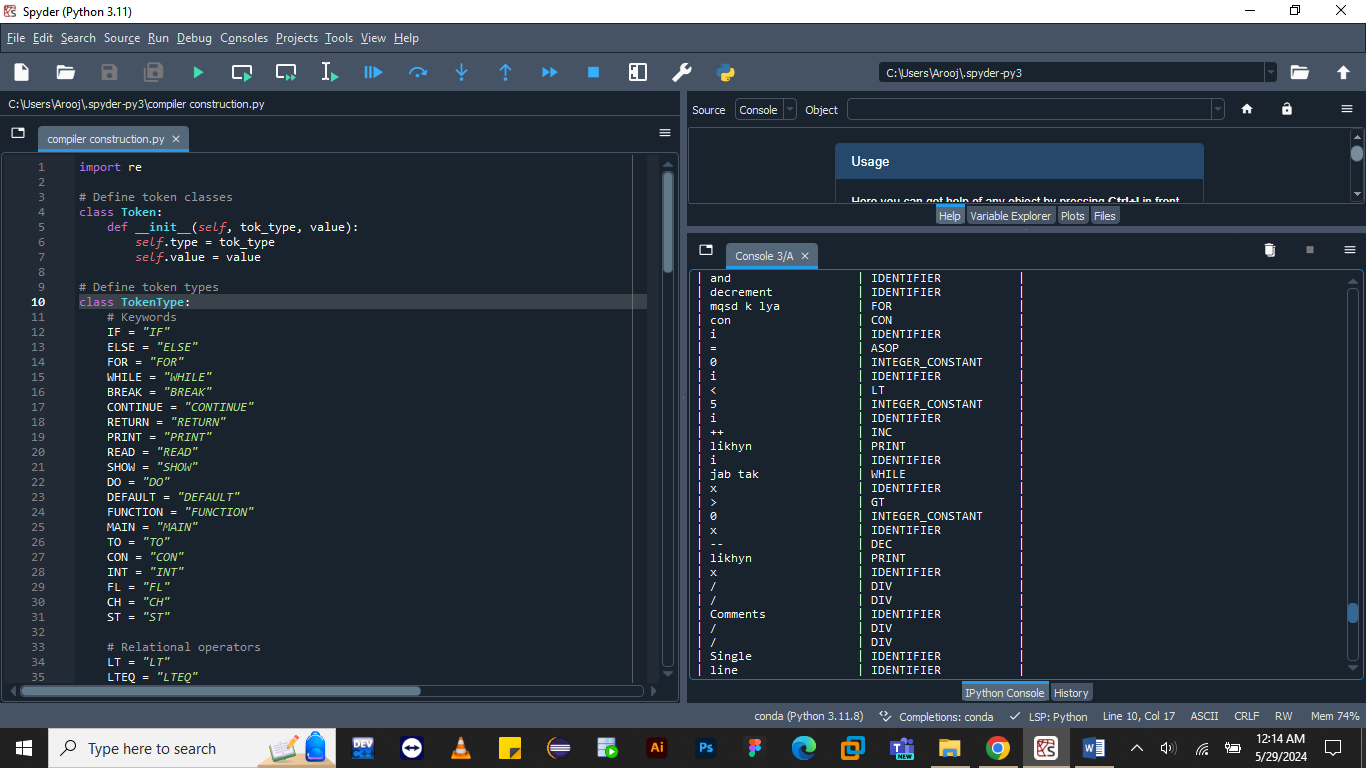
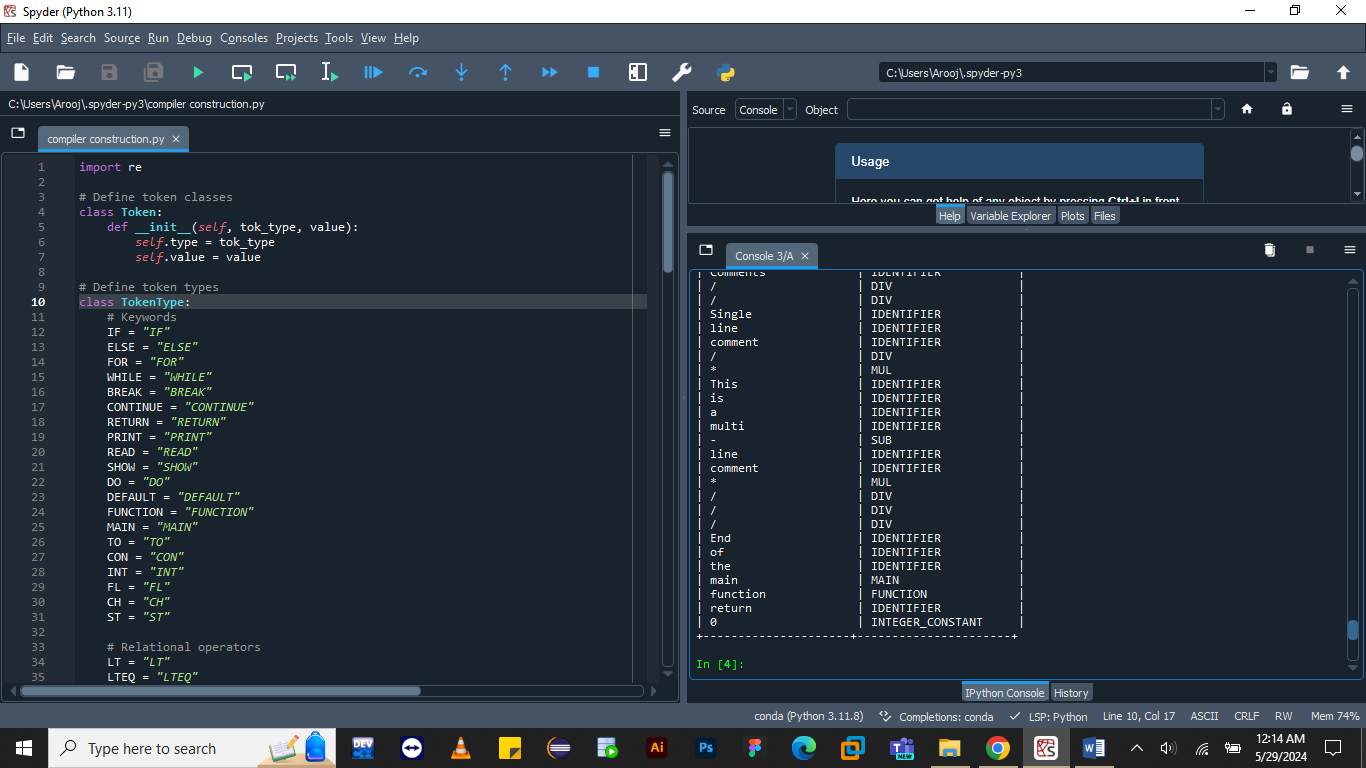
## print(f"| {token.value.ljust(20)} | {token.type.ljust(20)} |")

## print("+---------------------+----------------------+")

## except Exception as e:

## print(f"Error: {e}")

## **Output**

## **Input file**

