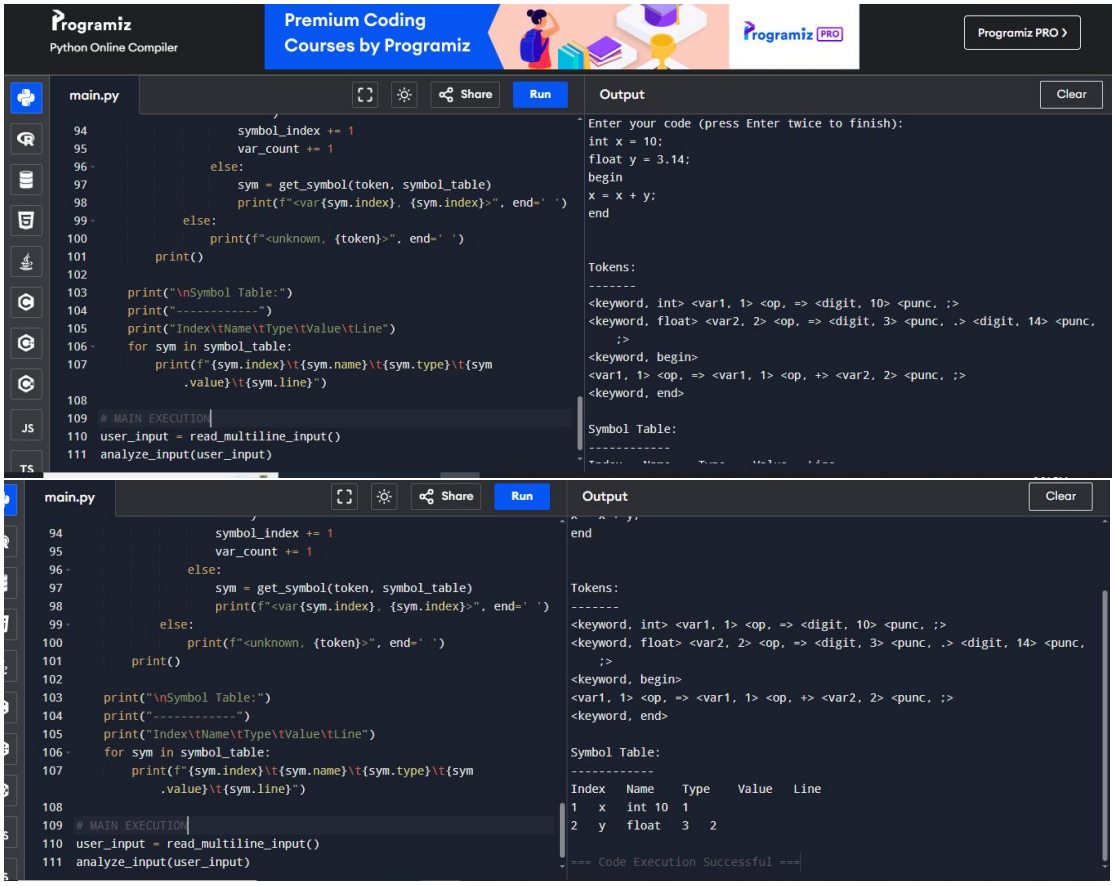
```
import re

class Symbol:
    def __init__(self, index, name, type_, value, line):
        self.index = index
        self.name = name
        self.type = type_
        self.value = value
        self.line = line

def read_multiline_input():
    print("Enter your code (press Enter twice to finish):")
    lines = []
    while True:
        line = input()
        if line == "":
            break
        lines.append(line)
    return "\n".join(lines)

def tokenize(line):
    tokens = []
    current = ''
    operators = "+-*/=<>!&|"
    punctuations = ".,;:{}()[]"

    for c in line:
        if c.isspace():
            if current:
                tokens.append(current)
                current = ''
        elif c in operators or c in punctuations:
            if current:
                tokens.append(current)
                current = ''
            tokens.append(c)
        else:
            current += c

    if current:
        tokens.append(current)

    return tokens

def symbol_exists(name, symbol_table):
    return any(sym.name == name for sym in symbol_table)

def get_symbol(name, symbol_table):
    for sym in symbol_table:
        if sym.name == name:
            return sym
    return None

def analyze_input(user_input):
    keyword_list = ["int", "float", "begin", "end", "print", "if", "else", "while", "main", "new"]

    variable_reg = re.compile(r'^[A-Za-z_][A-Za-z0-9_]*$')
    constants_reg = re.compile(r'^[0-9]+(\.[0-9]+)?([eE][+-]?[0-9]+)?$')
    operators_reg = re.compile(r'^[+\-*/=<>!&|]$')
    punctuations_reg = re.compile(r'^[.,;:{}()\[\]]$')
```

```
    lines = user_input.split('\n')
    line_num = 0
    var_count = 1
    symbol_index = 1
    symbol_table = []

    print("\nTokens:")
    print("-------")

    for raw_line in lines:
        if not raw_line.strip():
            continue

        line_num += 1
        tokens = tokenize(raw_line.strip())

        for i, token in enumerate(tokens):
            if token in keyword_list:
                print(f"<keyword, {token}>", end=' ')
            elif constants_reg.match(token):
                print(f"<digit, {token}>", end=' ')
            elif operators_reg.match(token):
                print(f"<op, {token}>", end=' ')
            elif punctuations_reg.match(token):
                print(f"<punc, {token}>", end=' ')
            elif variable_reg.match(token):
                if not symbol_exists(token, symbol_table):
                    type_ = tokens[i - 1] if i > 0 and tokens[i - 1] in keyword_list else "unknown"
                    value = tokens[i + 2] if (i + 2 < len(tokens) and tokens[i + 1] == "=") else ""
                    sym = Symbol(symbol_index, token, type_, value, line_num)
                    symbol_table.append(sym)
                    print(f"<var{var_count}, {symbol_index}>", end=' ')
                    symbol_index += 1
                    var_count += 1
                else:
                    sym = get_symbol(token, symbol_table)
                    print(f"<var{sym.index}, {sym.index}>", end=' ')
            else:
                print(f"<unknown, {token}>", end=' ')
        print()

    print("\nSymbol Table:")
    print("------------")
    print("Index\tName\tType\tValue\tLine")
    for sym in symbol_table:
        print(f"{sym.index}\t{sym.name}\t{sym.type}\t{sym.value}\t{sym.line}")

# MAIN EXECUTION
user_input = read_multiline_input()
analyze_input(user_input)
```

Output:


Programiz
Python Online Compiler
Premium Coding
Courses by Programiz
Programiz PRO
Programiz PRO ›
main.py
Share
Run
Output
Clear
94 symbol_index += 1
95 var_count += 1
96 else:
97 sym = get_symbol(token, symbol_table)
98 print(f"<var{sym.index}, {sym.index}>", end=' ')
99 else:
100 print(f"<unknown, {token}>", end=' ')
101 print()
102
103 print("\nSymbol Table:")
104 print("------------")
105 print("Index\tName\tType\tValue\tLine")
106 for sym in symbol_table:
107 print(f"{sym.index}\t{sym.name}\t{sym.type}\t{sym
.value}\t{sym.line}")
108
109 # MAIN EXECUTION
110 user_input = read_multiline_input()
111 analyze_input(user_input)
Enter your code (press Enter twice to finish):
int x = 10;
float y = 3.14;
begin
x = x + y;
end
Tokens:
<keyword, int> <var1, 1> <op, => <digit, 10> <punc, ;>
<keyword, float> <var2, 2> <op, => <digit, 3> <punc, .> <digit, 14> <punc,
;>
<keyword, begin>
<var1, 1> <op, => <var1, 1> <op, +> <var2, 2> <punc, ;>
<keyword, end>
Symbol Table:
----
main.py
Share
Run
Output
Clear
94 symbol_index += 1
95 var_count += 1
96 else:
97 sym = get_symbol(token, symbol_table)
98 print(f"<var{sym.index}, {sym.index}>", end=' ')
99 else:
100 print(f"<unknown, {token}>", end=' ')
101 print()
102
103 print("\nSymbol Table:")
104 print("------------")
105 print("Index\tName\tType\tValue\tLine")
106 for sym in symbol_table:
107 print(f"{sym.index}\t{sym.name}\t{sym.type}\t{sym
.value}\t{sym.line}")
108
109 # MAIN EXECUTION
110 user_input = read_multiline_input()
111 analyze_input(user_input)
end
Tokens:
<keyword, int> <var1, 1> <op, => <digit, 10> <punc, ;>
<keyword, float> <var2, 2> <op, => <digit, 3> <punc, .> <digit, 14> <punc,
;>
<keyword, begin>
<var1, 1> <op, => <var1, 1> <op, +> <var2, 2> <punc, ;>
<keyword, end>
Symbol Table:

Index Name Type Value Line
1 x int 10 1
2 y float 3 2
=== Code Execution Successful ===