```
Lexic.txt

Alphabet: all letters of the English alphabet (A-Z, a-z), digits and the
underscore

Operators:
+
-
*
/
%
=
<
>
<=
>=
+=
-=
++ (instead of x += 1)
-- (same as above)
** (power operator)


Separators:
space
;  -> after each statement
[] -> for indexing
{} -> for scope blocks


Reserved words:
defvar -> for defining a variable
deflist -> for defining a list/array
if, else, else if     -> for conditionals
and
or
not
loop   -> for a "while" type of loop
input, print


Identifiers:
letter = "a" | ... | "z" | "A" | ... | "Z"
digit = "0" | "1" | ... | "9"
nonzerodigit = "1" | "2" | ... | "9"
identifier = letter | letter {(letter | digit)}
sign = "+" | "-"

Types:
int = [sign] nonzerodigit {digit}
char = "'" (letter | digit) "'"
str = """ {(letter|digit)} """
bool = "true" | "false"
double = [sign] ("0" | nonzerodigit {digit}) "." {digit}
```

Token.in

```
+
-
*
/
%
"
'
=
<
>
<=
>=
+=
-=
++
--
**
space
;
{
}
[
]
defvar
deflist
and
or
not
if
else
loop
input
print
int
str
char
bool
double
```

Syntax.in

```
math_operator = "+" | "-" | "*" | "/" | "%" | "**"
relational_operator = "==" | "!=" | "<" | "<=" | ">" | ">="
boolean_operator = "and" | "or" | "not"
type = int | str | char | double | bool
list = "deflist" identifier "[]" ":" type ";"
variable = "defvar" identifier [":" type] {"," identifier [":" type]} ";"
number = int | double
mathematical_expression = number {math_operator number}
relational_operand = identifier | int | double | mathematical_expression
relational_expression = relational_operand relational_operator
relational_operand
expression = (mathematical_expression|relational_expression) {boolean_operator
expression}
condition = expression relation expression

assignment = identifier "=" expression ";"
input_output_statement = "input" "(" (identifier|type) {"," (identifier|type) }
")" ";" | "print" "(" (identifier|type) {"," (identifier|type) } ")" ";"
simple_statement = assignment | input_output_statement
compound_statement = simple_statement {";" compound_statement}
statement = compound_statement | if_statement | loop_statement
if_statement = "if" condition "{" statement "}" ["else" "{" statement "}"]
loop_statement = "loop" expression "{" {statement} "}"
```