LESL : LAZY EVALUATION SIMPLE LANGUAGE

TEAM 8

ABHISHEK DUTTA

APOORV KHAIRNAR

DHANANJAY PANDIT

AKIL KUMAR THOTA

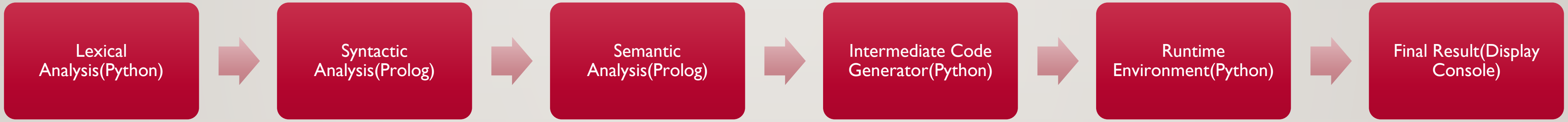
OVERVIEW

- Features of LESL
- Working of LESL
- Language Design
- Lesl Grammar
- Intermediate Code Sample
- Lazy Evaluation
- Tools used
- Instructions to build and run the language

FEATURES

- No data type declaration
- Arithmetic operations
- Relational operations including ternary operators
- Conditionals
- Loops
- **Lazy Evaluation**

WORKING OF LESL



Lexical Analysis(Python)

Syntactic Analysis(Prolog)

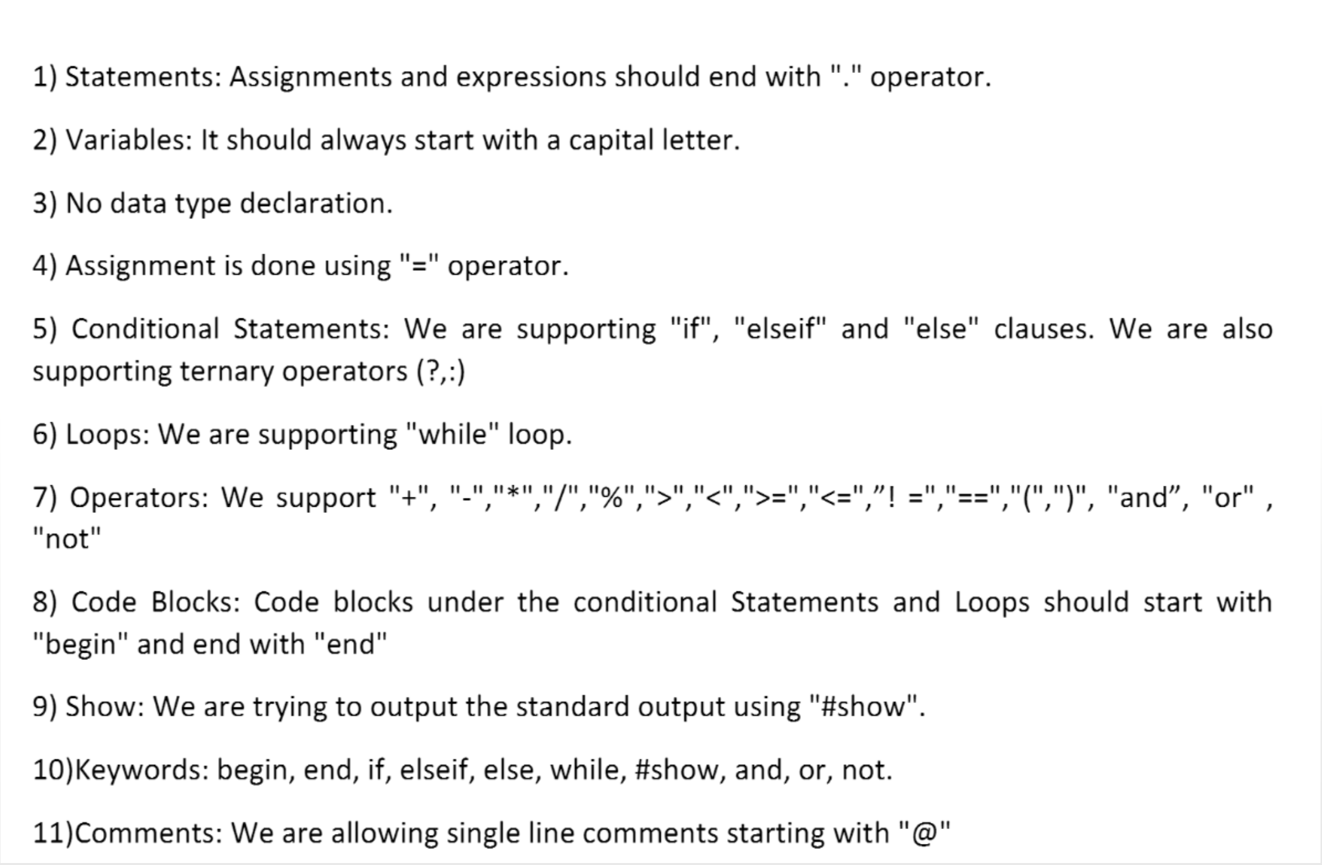
Semantic Analysis(Prolog)

Intermediate Code Generator(Python)

Runtime Environment(Python)

Final Result(Display Console)

LANGUAGE DESIGN



1) Statements: Assignments and expressions should end with "." operator.

2) Variables: It should always start with a capital letter.

3) No data type declaration.

4) Assignment is done using "=" operator.

5) Conditional Statements: We are supporting "if", "elseif" and "else" clauses. We are also supporting ternary operators (?,:)

6) Loops: We are supporting "while" loop.

7) Operators: We support "+", "-","*","/","%",">","<",">=","<=",”! =","==","(",")", "and”, "or" , "not"

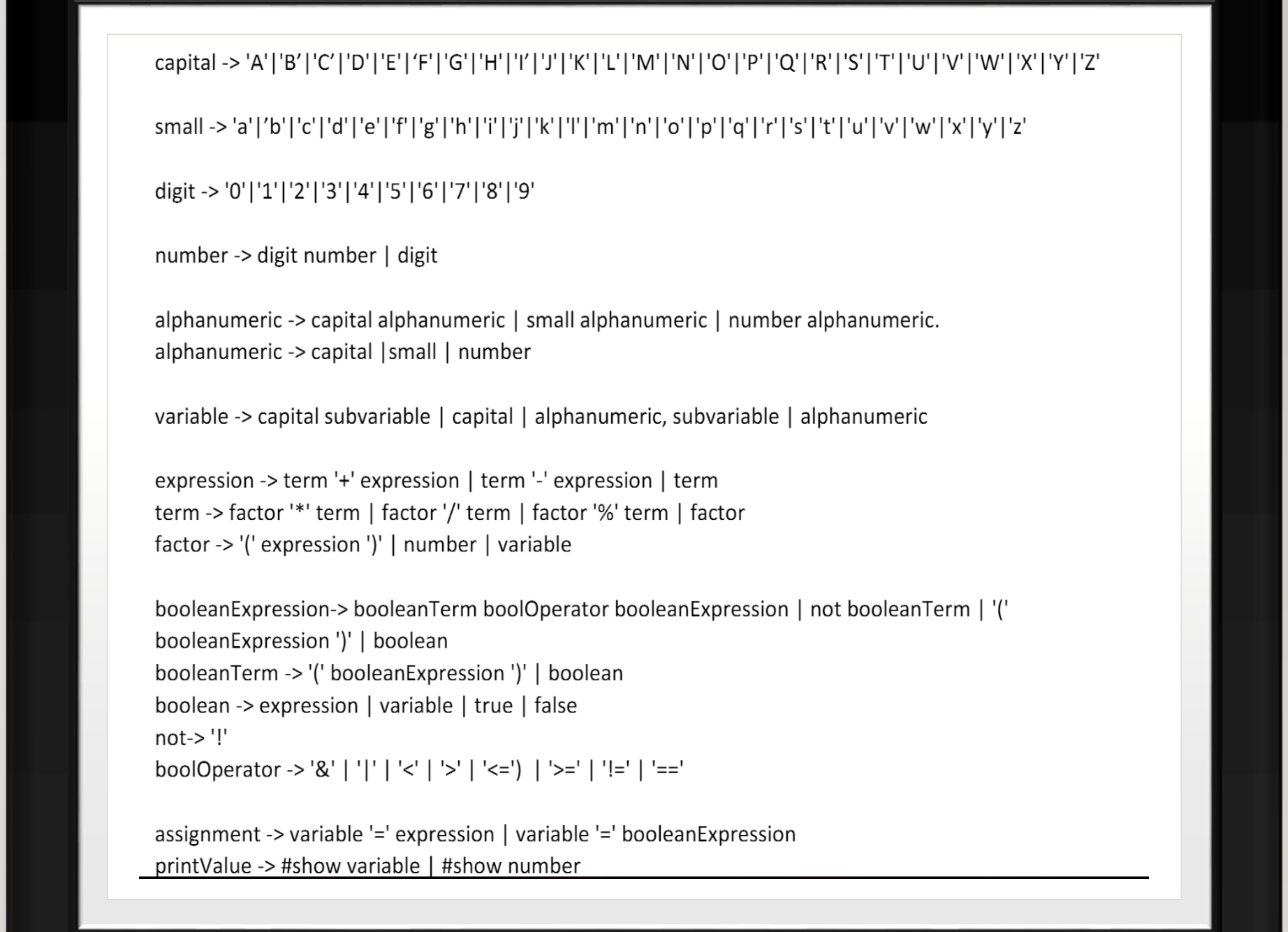
8) Code Blocks: Code blocks under the conditional Statements and Loops should start with "begin" and end with "end"

9) Show: We are trying to output the standard output using "#show".

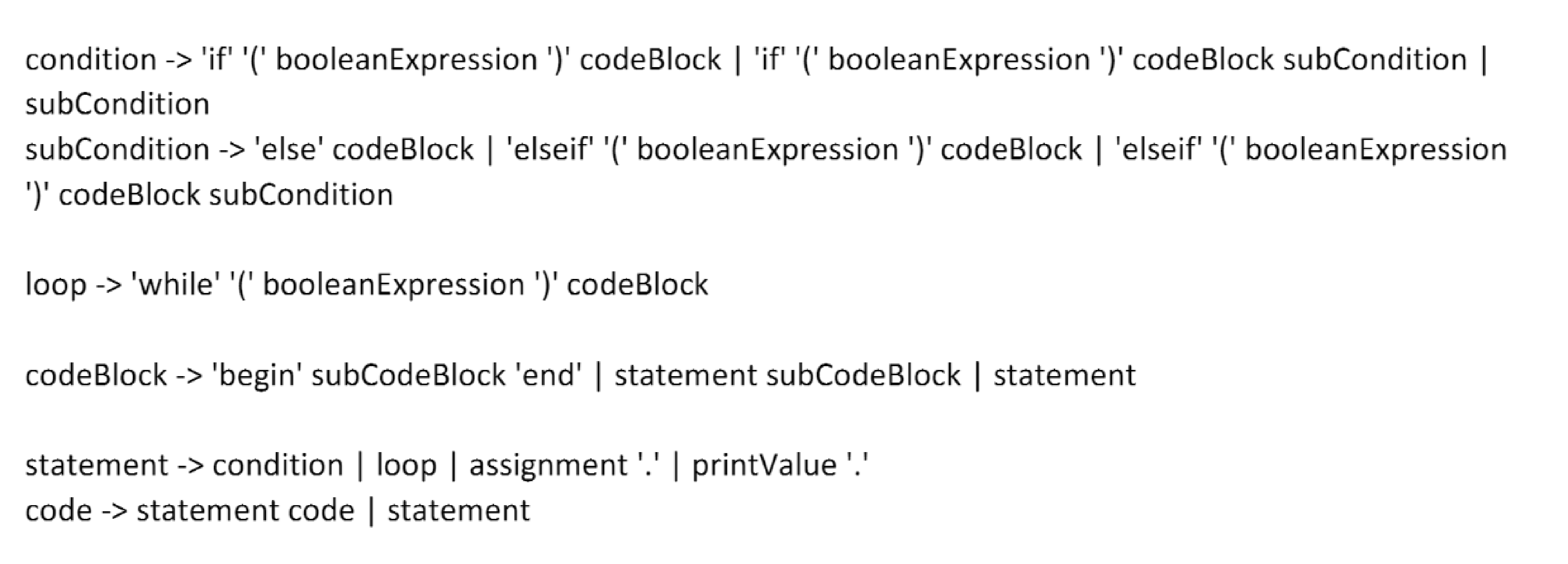
10)Keywords: begin, end, if, elseif, else, while, #show, and, or, not.

11)Comments: We are allowing single line comments starting with "@"

LESL GRAMMAR



```
capital -> 'A'|'B’|'C’|'D'|'E'|‘F'|'G'|'H'|'I’|'J'|'K'|'L'|'M'|'N'|'O'|'P'|'Q'|'R'|'S'|'T'|'U'|'V'|'W'|'X'|'Y'|'Z'

small -> 'a'|’b'|'c'|'d'|'e'|'f'|'g'|'h'|'i'|'j'|'k'|'l'|'m'|'n'|'o'|'p'|'q'|'r'|'s'|'t'|'u'|'v'|'w'|'x'|'y'|'z'

digit -> '0'|'1'|'2'|'3'|'4'|'5'|'6'|'7'|'8'|'9'

number -> digit number | digit

alphanumeric -> capital alphanumeric | small alphanumeric | number alphanumeric.
alphanumeric -> capital |small | number

variable -> capital subvariable | capital | alphanumeric, subvariable | alphanumeric

expression -> term '+' expression | term '-' expression | term
term -> factor '*' term | factor '/' term | factor '%' term | factor
factor -> '(' expression ')' | number | variable

booleanExpression-> booleanTerm boolOperator booleanExpression | not booleanTerm | '('
booleanExpression ')' | boolean
booleanTerm -> '(' booleanExpression ')' | boolean
boolean -> expression | variable | true | false
not-> '!'
boolOperator -> '&' | '|' | '<' | '>' | '<=')  | '>=' | '!=' | '=='

assignment -> variable '=' expression | variable '=' booleanExpression
printValue -> #show variable | #show number
```



```
condition -> 'if' '(' booleanExpression ')' codeBlock | 'if' '(' booleanExpression ')' codeBlock subCondition | subCondition
subCondition -> 'else' codeBlock | 'elseif' '(' booleanExpression ')' codeBlock | 'elseif' '(' booleanExpression ')' codeBlock subCondition

loop -> 'while' '(' booleanExpression ')' codeBlock

codeBlock -> 'begin' subCodeBlock 'end' | statement subCodeBlock | statement

statement -> condition | loop | assignment '.' | printValue '.'
code -> statement code | statement
```

INTERMEDIATE CODE SAMPLE

- mov 2, t2. (assignment operation)
- add t1, 2, t3 (addition operation)
- L3: jle t4, t5, L5 (Label and jump if less than operation)
- jmp L3 (jump to Label L3)
- L5: div t5,4,t6 (Label L5 with division operation)
- shw A (print operation)

LAZY EVALUATION

- In programming language theory, lazy evaluation, or call-by-need is an evaluation strategy which delays the evaluation of an expression until its value is needed (non-strict evaluation) and which also avoids repeated evaluations (sharing).
- The sharing can reduce the running time of certain functions by an exponential factor over other non-strict evaluation strategies, such as call-by-name.
- Performance increases by avoiding needless calculations, and error conditions in evaluating compound expressions.

LAZY EVALUATION IN LESL

- Scope of LESL is pretty limited as it does not support functions.
- However, we still support complex expression in accordance with operator precedence.
- Thus, we implemented LESL in such a way that none of the complex expressions will be evaluated until we explicitly ask for its value.
- Just to give you a simple example, if your program has 100 lines but there is no "print" statement, then none of the expressions will be calculated, those saving a lot of time.

EXAMPLE

- A = 5.
- B = A * 2.
- C = 15 / 3
- D = A + B - C.
- A = True.
- #show D.
- #show A.

Conversion from code to run-time environment

- A = 5
- B = 5 * 2
- C = 15 / 3
- D = 5 + (5 * 2) – (15 / 3)
- A = True
- Calculate and print D = 10
- Print A = True.

TOOLS USED

- SWI-Prolog
- Python

THANK YOU!!