Group: CompileOpt (12)

- **Basic Types:**

Variable declaration format: *"assign variable_type: variable_name = initial value"*

assign int: x = 42;
assign string: name = "John";
prnt(name.at(3)); $ prints value n, which is at index 3 of the string "John"
name.replace(ind,"k"); $ replace value at index ind with k.
assign bool: flag = True ;

string1.slice(start,end); $ gives from string1.at(start) to string1.at(end-1) both included.

string1.concat(string2); $ Gives string1 + string2

=: Assignment operator.

!=,==, <, >: Comparison operators.
+,-,*,/, **,%, and, or : Binary operators.
++,--: Unary operators.
Comments:
Single-line: $ commented text
Multi-line: $* commented text *$

Identifiers: Letters, digits, and underscores, starting with a letter (case-sensitive)
Keywords:

- **Compound Types:**

tuple: let tuple: name = ( )

array: assign int: name(max_size) = [1,2,3,4] ;

name.append(5) ;

size = name.size() ;

var x = name.at(2) ; $ accessing the elements from 2nd index .

name.head() $ gives 1st element in name array.

name.tail() $ gives the last element in the array.

$ daya_ type x is equal to data-type of array

lists: assign list: name1 = [];

name1.append(5) ;

size = name1.size() ;

- **Conditionals**

```
agar condition  {
   $ code to execute if the condition is true
}
baki condition  {
   $ code to execute if this condition is true
}
nahito {
   $ code to execute if the condition is false }
```

- **Loops**

```
jabtak condition  {
   $ code to execute if the condition is true (while loop)
}
```

- **Functions**

Define:

```
func:int/void/bool  myFunction (int: x, bool: y)  {
   $ function body

   respond result;
}
```

Call:

```
myFunction(x, y);
```

- **Closures**

```
func:int/void/bool  myFunction (int: x, bool: y)  {
   assign int: x = 5;
   assign int: result = 0;
   func:int/void/bool  myFunction (int: a, bool: b)  {
          $ function body
         assign k = x+a;
         respond k;
   }
   respond result;
}
```

- **Mutable variables**

```
assign int/bool/string : name = —;
```

- **Exceptions**

```
try{
        $ continue
            throw exception
}
catch (exception){
        prnt(3, " blabla \n");      $ \n → new line.
        prnt(variable_name);
}
```

```
program ::= statement*

statement ::= variable_declaration
        | assignment
        | print_statement
        | if_else_statement
        | while_loop
        | function_definition
        | closure_definition
        | mutable_variable_declaration
        | exception_handling_statement

variable_declaration ::= 'assign' variable_type ':' identifier '=' expression ';'

assignment ::= identifier '=' expression ';'

print_statement ::= 'prnt' '(' expression ');'

if_else_statement ::= 'agar' expression '{' program '}' baki expression '{' program '}' nahito '{'
program '}'
```

while_loop ::= 'jabtak' expression '{' program '}'

function_definition ::= 'func' ':' (variable_type | 'void') identifier '(' parameter_list? ')' '{' program '}' 'respond' expression ';'

closure_definition ::= 'func' ':' (variable_type | 'void') identifier '(' parameter_list? ')' '{' program '}' 'respond' expression ';'

parameter_list ::= (variable_type ':' identifier) (',' variable_type ':' identifier)*

mutable_variable_declaration ::= 'assign' variable_type identifier ':' identifier '=' expression ';'

exception_handling_statement ::= 'try' '{' program 'throw' identifier ';' '}' 'catch' '(' identifier ')' '{' program '}'

expression ::= literal
| identifier
| unary_operation
| binary_operation
| '(' expression ')'
| function_call
| list_operation
| array_operation
| member_access

literal ::= number
| boolean
| string

unary_operation ::= '++' expression
| '--' expression

```
binary_operation ::= expression '+' expression
                   | expression '-' expression
                   | expression '*' expression
                   | expression '/' expression
                   | expression '**' expression
                   | expression '%' expression
                   | expression '<' expression
                   | expression '>' expression
                   | expression '<=' expression
                   | expression '>=' expression
                   | expression '==' expression
                   | expression '!=' expression
                   | expression 'and' expression
                   | expression 'or' expression

function_call ::= identifier '(' argument_list? ')'

list_operation ::= identifier '.' ('append' '(' expression ')' | 'size' '(' ')' | 'at' '(' expression ')' | 'head' '(' ')' | 'tail' '(' ')')

array_operation ::= identifier '.' ('append' '(' expression ')' | 'size' '(' ')' | 'at' '(' expression ')' | 'head' '(' ')' | 'tail' '(' ')')

member_access ::= identifier '.' identifier

argument_list ::= expression (',' expression)*
```