# CS315
# PROJECT 1 REPORT
# TEAM 17

HACI ÇAKIN 21802641 - SECTION 2
MUHAMMED MUSAB OKSAS 21602984 - SECTION 2

## SCOTTY
**—BEAM ME UP SCOTTY—**

# CONTENTS

# BNF of SCOTTY

## 1) Program Definition

**<program>** ::= <main>

**<main>** ::= main {<statements>}

**<statements>** ::= <statement>; |<statement;><statements>

**<statement>** ::= <comment>|<expressions>|<loops>|<conditionals>

**<comment>** ::= <comment_sign><content><comment_sign>

**<content>** ::= <identifier><sentence>|<identifier>

**<expressions>** ::= <IO>|<declarations> |<assignments>

**<loops>** ::= <for_loop>| <for_each> | <while_loop>

**<conditionals>** ::= <if_statement>

**<IO>** ::= <input> | <output>

**<declarations>** ::= < function_declaration>| < function_call> | <bool_declaration> |
<cons_bool_declaration> |<var_string_declaration> |
<cons_string_declaration> | <var_number_declaration> |
<cons_number_declaration> | <var_list_declaration> |
<cons_list_declaration> | <var_object_declaration>

**<assignments>** ::= <bool_assignment> |<string_assignment>|<number_assignment>|
<list_assignment>|<element_of_list_assignment> | <object_assignment> |
<element_of_object_assignment>

## 2) Declarations

**<var_number_declaration>** ::= var_number $identifier | var_number
<number_assignment>

**<cons_number_declaration>** ::= cons_number <number_assignment>

**<var_list_declaration>** ::= var_list identifier | var_list <list_assignment>

**<cons_list_declaration>** ::= cons_list <list_assignment>

**<var_string_declaration>** ::= var_string identifier | var_string <string_assignment>

**<cons_string_declaration>** ::= cons_string <string_assignment>

**<var_bool_declaration>** ::= var_bool identifier | var_bool <bool_assignment>

**<cons_bool_declaration>** ::= cons_bool <bool_assignment>

**<var_object_declaration>** ::= var_object identifier | var_object identifier <object> |var_object <object_assignment>

**3) Assignments**

**<number_assignment>** ::= $identifier = <number_operation> | $identifier <increment_decrement>

**<list_assignment>** ::= identifier = <list>

**<string_assignment>** ::= identifier = string;

**<bool_assignment>** ::= identifier = <bool_operation>

**<object_assignment>** ::= identifier = <object>

**4) Loops**

**<for_each>** ::= forEach( identifier in <list>) {statements} | forEach( identifier in identifier) {<statements>}

**<for_loop>** ::= for(<declarations>; <bool_expression>; <assignments>){<statements>}

**<while>** ::= while(<bool_expression>){<statements>}

**5) Conditionals**

**<if_statement>** ::= if(<bool_expression>){<statements>} | if(<logic_expression>) {<statements>}else{<statements>}

**6) Function**

**<function_declaration>** :: function identifier (<params>) =>{<function_body>}

**<function_body>** ::= < function_statements> return <type>;

**<params>** ::= <empty> | <param> | <param>, <params>

**<param>** ::= identifier | $identifier

**<function_statements>** ::= <statements> | <empty>

**<function_call>** ::= identifier (<types>)

**7) List**

**<list>** ::= [<types>]

**<element_of_list>** ::= identifier[<number_operation>]

**<element_of_list_assignment>** ::= <element_of_list> = <type>

## 8) Object

**<object>** ::= {<object_types>}

**<object_types>** ::= <declarations> | <declarations> , <object_types> | <object>

**<element_of_object>** ::= identifier[<number_operation>] | identifier->identifier

**<element_of_object_assignment>** ::= <element_of_object> = <type>

## 9) Operations

**<number_operation>** ::= <number_operation>
<substitution_sum_operator><number_term>|<number_term>

**<number_term>** ::= <number_term><multiplication_division_operator><number_factor> |
<number_factor>

**<number_factor>** ::= ( <number_operation>)|<number_type>

**<bool_operation>** ::= <bool_operation><bool_operator><bool_term> | <bool_term>

**<bool_term>** ::= (<bool_expression>) | <bool_type> | ~<bool_type>

**<type>**::= <number_operation> | <bool_operation> | string | <list> | <object>

**<number_type>** ::= integer | double | $identifier | $<element_of_list>

**<bool_type>** ::= true | false | < function_call> | identifier | <element_of_list>

**<types>** ::= <empty> | <type> | <type>, <types>

**<bool_operator>** ::= && | '||' | ~& | <===> | ==> | '~|'

**<comparison_operator>** ::= < | > | <= | >= | == | !=

**<increment_decrement>** ::= \+\+ | \-\-

**<multiplication_division_operator>** ::= * | /

**<substitution_sum_operator>** ::= + | -

**<bool_expression>** ::= <bool_expression_with_number> | <bool_operation>

**<bool_expression_with_number>** ::=
    <number_operation><comparison_operator><number_operation>

**<input>** ::= input( identifier ) | input( $identifier )

**<print>** ::= print(<types>) | printLine(<types>)

**<empty>** ::= null

## LANGUAGE CONSTRUCTS

**<program> :** In Scotty, program is constructed on “main”. This is the very beginning of the program.

**<main> :** Main is fundamental part of program which is runnable. The main format is like,

```
main{
  //example main
}
```

**<statements> :** Statement are to determine the statement in the code blocks.

**<statement> :** There are four main types of statements in Scotty which are expressions, loops, conditionals and comments.

**<comment> :** In Scotty, comment are similar to the comments in C family languages.

```
/* This is a comment example */
```

**<content> :** It is the content of the comment.

**<expressions> :** In Scotty, main three expressions are IO, declaration and assignments.

**<loops> :** There are 3 types of loop expression which are for, forEach and while.

**<conditionals> :** There is only one conditional statement in Scotty which is if statement.

**<IO> :** In Scotty; input is the stdin, print and printLine is the stdout.

**<declarations> :** To declare string, boolean, number and list types.

**<assignments> :** To assign string, boolean, number, list and its elements.

**<var_number_declaration> :** It is for declaration of the number variable.

```
var_number $exampleNumber = 1;
var_number $exampleNumber2;
var_number $exampleNumber3 = $exampleNumber;
var_number $exampleNumber4 = $exampleNumber * $exampleNumber3;
var_number $exampleNumber5 = $exampleList[2];
```

**<cons_number_declaration> :** It is for declaration of the constant number variable.

```
cons_number $exampleNumber = 1;
```

**<var_list_declaration> :** It is for declaration of the list variable.

```
var_list exampleList = [
        item,
        item,
];
var_list exampleList2;
var_list exampleList3 = [ 20,”example”, 2021 , true];
```

**<cons_list_declaration> :** It is for declaration of the constant list variable.

```
cons_list exampleList = [
        item,
        item,
];
```

**<var_string_declaration> :** It is for declaration of the string variable.

```
var_string exampleString = “This is an example string”;

var_string exampleString2;

var_string exampleString3 = exampleString;
```

**<cons_string_declaration> :** It is for declaration of the constant string variable.

```
cons_string exampleString = “This is an example string”;
```

**<var_bool_declaration> :** It is for declaration of the boolean variable.

```
var_bool exampleBool = true;

var_bool exampleBool2;

var_bool exampleBool3 = false || exampleBool;
```

**<cons_bool_declaration> :** It is for declaration of the constant boolean variable.

```
cons_bool exampleBool = true;
```

**<var_object_declaration> :** It is for declaration of objects.

```
var_object example;
var_object exampleObject{
        var_number $health = 100,
        var_string name = “Hacı Çakın”
};

var_object exampleObject2 = {
        var_number $health = 100,
        var_string name = “Hacı Çakın”
};
```

**<number_assignment> :** To assign a number to identifier.

```
$exampleNumber = 5;
$exampleNumber = $exampleNumber1;
$exampleNumber = $exampleList[2];
```

**<list_assignment> :** To assign element to list.

```
exampleList = [
        item,
        item
];
exampleList2 = exampleList;
```

**<string_assignment> :** To assign string to identifier.

```
exampleString = “This is an example of string assignment”;
exampleString2 = exampleString3;
```

**<bool_assignment> :** To assign logical value to identifier.

```
exampleBool = false;
exampleBool2 = exampleBool && true;
```

**<object_assignment> :** It is for assign a value to object.

```
exampleObject  = {
        var_number $health = 100,
        var_string name = “Hacı Çakın”
};
```

**<for_each> :** This is representation of the foreach loop in scotty.

```
forEach(element in exampleList){
  // codes
};
```

**<for_loop> :** Scotty has the same for structure with C family.

```
var_number $exampleNumber = 2;
for( var_number $forNumber = 0; $forNumber < $exampleNumber; $forNumber = $forNumber + 1
){
  //codes
};
```

**<while> :** As in for statement, Scotty has also same structure in while statement with C family

```
var_number $exampleNumber = 2;
var_number $exampleNumber2 = 0;

while($exampleNumber > $exampleNumber2){
  //codes
};
```

**<if_statement> :** As similar to the if statement of C family, Scotty has same conditional structure.

```
var_bool exampleBool = true;
if( exampleBool ){
  //codes
};
exampleBool = false;
if( exampleBool){
   //codes
}else{
   //codes
};
```

**<function_declaration> :** Scotty has the function structure has function keyword, function name, parameters and function body. " => " assignment is used to indicate that this is a function declaration.

```
function exampleFunction($firstValue, $secondValue, exampleString) => {
        printLine(exampleString);
        return $firstValue > $secondValue;
};
```

**<function_body> :** It is body of function. The reason of this is that function can have a return statement.

**<params> :** It consist of the params(can be empty).

**<param> :** Param can be string, number, bool, list or function.

**<function_statements> :** It consist of statements(can be empty).

**<function_call> :** It invokes the functions.

```
var_bool resultOfFunction = exampleFunction(5,3, “exampleString");
```

**<list> :** In Scotty, list can have various types of variables simultaneously.

**<element_of_list> :** It gets the element of selected index. Indexes starts from 0 as C family has. If wanted index is a number variable, before the call, $ should be written.

```
var_number $exampleNumber = $exampleList[2];
var_string exampleString = exampleList[0];
```

**<element_of_list_assignment> :** It same with other assignment process. First get the element from the list with a call, then put the new value.

```
exampleList[2] = “This is a new value”;
```

**<object> :** It is consist of the declarations of types. The reason to discrete them from the types is in objects there is an identifier also, which lists do not have.

**<object_types> :** It is how types should use.

```
var_object exampleObject = {
        var_number $health = 100,
        var_string name = “Hacı Çakın”
};
```

**<element_of_object> :** :It is for reaching to object properties. There is two possibilities. First one similar to the element reaching same with the lists have. Second one is special for the objects.

```
$exampleObject[0];
exampleObject->name;
```

**<element_of_object_assignment> :** After calling element of the object, assign a value to it.

```
$exampleObject[0] = 200;
exampleObject->name = “Musab Oksas”;
```

**<number_operation> :** This describes how the numeral operations should be done. It is suitable for the presedence rules. It also follows leftmost recursion. To be suitable them, it has <number_term> and <number_factor>

**<number_term> :** It is a helper that makes possible to perform numerical operations. This one provides the presedence rule which means this gives importance for multiplication and division over the addition and subtraction.

**<number_factor> :** It's purpose is nearly same with <number_term>. It is again helper part that give change to give presedence to parentheses. <number_type> is the end of the recursive algorithm.

**<bool_operation> :** Similar to calculate numeral values, in calculating logical values, Scotty follow a recursive way to provide presedence rule. In order to that again it has parts that are <bool_operator> and <bool_term> which are triggered recursively.

**<bool_term> :** It provides presedence rule for parentheses in logical expressions which means again it is helper to find result of logical expression while following priority rules. <bool_type> is the end of the logical recursion.

**<type> :** It defines all the types which are list, string , numeral types and logical types.

**<number_type> :** It defines double, integer, identifier which have $(indicator) as a number.

**<bool_type> :** It defines logical values whether true or false.

**<types> :** Consists of all types.

**<bool_operator> :** It is for constructing logical operators such as and, or, nor, nand.

**<comparison_operator> :** It is for constructing comparison operators such as less than, greater than, less than equal, greater than equal, equality and inequality.

**<multiplication_division_operator> :** It is for constructing multiplication operator ("*") and division operator ("/").

**<substitution_sum_operator> :** It is for constructing sum operator("+") and subtraction operator("-").

**<bool_expression> :** It is for finding result of bool expression in <bool_expression_with_number> or direct logical comparison.

```
var_bool temp = false;
print(( temp || false) );
```

**<bool_expression_with_number> :** It is for just deriving the numeral logical comparison
```
print( 3>5 );
```

**<increment_decrement> :** It is adding or subtracting 1 from the number

**<input> :** It is described for the taking input from the user. Input takes just one param and this param takes the value that user enter.

```
var_string exampleString;
input(exampleString);
```

**<print> :** In Scotty, there are two types of printing method. One of them is printing results in same line. Other one passes the next line after printing.

```
print(“This is same line printing”);
printLine(“This passes next line after printing process is done");
```

**<empty> :** It is empty indicator.

## 1) Terminals of SCOTTY

**identifier :** this terminal creates tokens that are given to variables, functions etc.

**var :** this terminal for the declaration of the variables

**cons :** this terminal for the declaration of the constants

**function :** this terminal for the declaration of the function

**in :** this terminal is used in the foreach declaration

**print :** this terminal is for printing without passing next line

**printLine :** this terminal is for printing with before passing next line

**input :** this terminal is for getting input from the user

**“+” :** this terminal is addition operator

**“-” :** this terminal is subtraction operator

**“*” :** this terminal is multiplication operator

**“/“ :** this terminal is division operator

**“=” :** this terminal is for assignment
**“=>” :** this terminal is used in function declaration to trace the declaration of function

**“(” :** this terminal is left parentheses

**“)” :** this terminal is right parentheses

**“{” :** this terminal is left curly brackets

**“}” :** this terminal is right curly brackets

**“[” :** this terminal is left cornered brackets

**“]” :** this terminal is right cornered brackets

“,” : this terminal is for comma

“!” : this terminal is for exclamation

“~” : this terminal is for logical not

**“&&” :** this terminal is for logical and

“||” : this terminal is for logical or

**“~&” :** this terminal is for logical and

“->” : this terminal is for to reaching element of the object

“~|” : this terminal is for logical nor

“<===>” : this terminal is for if and only if(iff)

“==>” : this terminal is for logical implication

“>” : this terminal is for greater than

“<” : this terminal is for less than

“<=” : this terminal is for less than or equal to

“>=” : this terminal is for greater than or equal to

“==” : this terminal is for equality

“!=” : this terminal is for inequality

“\+\+” : this terminal is for one increment

“\-\-” : this terminal is for one decrement

## NONTRIVIAL TOKENS

**1) Literals**

In Scotty;

- Numeric literals are described by whether double and integer values. As in same with in most of the programming languages, integers are represented like 21 and doubles are represented like 2.1 . We do not want to change this one because nearly all the programming languages have same structure.
- String literals are described between the quotation marks. As we do not change numeric literals, we save the common structure of strings to increase writability and readability.
- Boolean literals are consist of true and false as a logical truth.

### 2) Reserved Words

Because these are reserved words in Scotty, they can not be used except their own definition. While we were determining reserved words of Scotty, we try to follow common rules but we had tried to increase readability. Therefore, we added the “var” word to show it is not a constant. Other words like “number”, “string”, “bool”, “list”, “if”, “for” etc. are common most of the programming languages.

- **var_number :** Token reserved for declaration of numeric values
- **var_string :** Token reserved for declaration of string values
- **var_bool :** Token reserved for declaration of logical values
- **var_list :** Token reserved for declaration of lists
- **cons_num :** Token reserved for declaration of numeric constants
- **cons_string :** Token reserved for declaration of constant strings
- **cons_bool :** Token reserved for declaration of constant logical values
- **cons_list :** Token reserved for declaration of constant lists
- **function :** Token reserved for declaration of the function
- **return :** Token reserved for declaration for returning a value
- **if :** Token reserved for if statements
- **else :** Token reserved for for else part of the if statements
- **for :** Token reserved for for loops construction
- **forEach :** Token reserved forEach loops construction
- **while :** Token reserved for while loops construction
- **print :** Token reserved for printing values without passes next line
- **printLine :** Token reserved for printing values while passing next line
- **input :** Token reserved for reading values from the user
- **null :** Token reserved for indicate the empty values
- **true :** Token reserved for true logic
- **false :** Token reserved for false logic
- **in :** Token reserved for constructing iteration in the forEach loops

### 3) Identifiers

In Scotty, identifiers for any type except numeral ones starts with letter and continue with characters. In numeral identifiers, they starts with “$” sign to indicate that is number, and followed by any character combinations. The another reason of using “$” is to decrease type mismatch.

### 4) Comments

In Scotty, comments are designed as a structure to write between “/*” and “*/” in the single line. The reason and main purpose is that in C family, same structure is in usage. There could be added other alternatives like “/**” and “**/” or “//” but we want to make it simple because one of them is enough to perform writing comment.

## EVALUATION of SCOTTY

### 1) Readability

The readability of language also can be thought of as familiarity of a language. Therefore, users have to be familiar with the syntax of the language. In order to succeed that, the syntax of the language should follow the trends and popular expressions. What Scotty tries to do is this. Our motivation for readability is following the trend and constructing a language that has simple and common parts. In order to that, as it can be realized, our conditional statements( if and else ) and loops ( for, while, forEach ) follow today's trends(as most of the programming languages have similar structure). Moreover, for variables syntax of the Scotty is following very understandable ways. For constants, it uses the “cons” expression to show it is constant. For changeable variables, it uses “var” to determine it is variable. Also discrete the numeric values, it uses the “$” sign to indicate it is numeric. In order to identify variable type, it uses keywords like “number”, “string”, “bool”, “function” which are common in C family languages. In function description, Scotty uses arrow “=>” sign to indicate the new function description as node.js framework which is one of the popular server-side framework has same notation. Lastly, Scotty has a main method which is also common and has curly brackets for code blocks which increase the readability. After each statement in Scotty, there is a semi-colon notation”;” which eases to read. As we think about all these syntax simplicity for people who have at least one programming language knowledge can read Scotty’s code very easily.

## 2) Writability

Our motivation to increase writability is nearly similar to the as we have in readability. The reason of this is that a person who has programming background search for common syntax. Therefore, we try not to break the writability and continue using common syntax while improving the readability as much as possible. In order to protect writability, we use the alternative for structure which is similar to python. Moreover, we shorten IO syntaxes without disrupting meaning and readability. We use “input”, “print” and “printLine” reserved words to do this. As you can compare, input and output processes are much more easier compare to c++(because in c++ << parts can really get longer) and java(without using with scanner parts). Other parts are following common syntax which not decreases the writability. In addition, Scotty has the syntax “object” which give change to look at the problems from the object oriented perspective. The “function” syntax is also giving chance to perform algorithms in meaningful and portable ways. Last factor that increase the writability is Scotty has simple increment “++” and decrement “- -” notations. As a result, with respect to writability, Scotty provides lots of alternatives for syntax.

## 3) Reliability

In Scotty, first factor that increase the reliability is that Scotty is adjusted to perform logical and numerical operations without disrupting the precedence rules. For the numerical operations, it gives priority according priority order of parentheses , multiplication, division, summation and subtraction. As it happens in numerical operations, the precedence rules are followed for the logical operation too. Another factor that may increase the reliability of the language is defining each variable according to their name. In some languages like javascript and dart, all the variables can be defined with using “var” word. However, this way decreases the reliability. Unlike javascript and dart, Scotty has different reserved words for numbers(double, integer), strings, bools, lists and functions which have a positive impact on the reliability.