Tyson

Мультипарадигмова мова програмування загального призначення

Специфікація та грамматика

Роман Журавльов

2020

1. Вступ

Tyson - мова програмування сімейства C. Це спрощена версія мови Javascript, що реалізує змінні, логіку та цикли. Tyson — мультипарадигмова мова програмування загального призначення, що підтримує імперативний стиль програмування.

1.1 Обробка

Програма, написана мовою Tyson, в вигляді символьної строки розбивається на базові лексеми, після чого масив лексем аналізується на валідність. Результат трансляції виконується у системі часу виконання (runtime system), для чого приймає вхідні дані та надає результат виконання програми. Трансляція передбачає фази лексичного та синтаксичного аналізу, а також фазу перекладу команд в польску нотацію та їх виконання. Фази лексичного та синтаксичного аналізу здійснюються окремими проходами.

Також, окремо Tyson підтримує парсинг та аналіз інструментами ANTLR4, але далі розібрано лише транслятор, написаний спеціально для цієї мови. Результат обробки ANTLR4 можна побачити в консолі

1.2 Нотація

Для опису мови Tyson використовується граматика ANTLR в форматі .g4. Граматика розбита на два файли, TysonLexer та TysonParser. В файлі TysonLexer знаходяться всі базові лексеми мови, а в файлі TysonParser – мовні кострукції та вирази, що з них складаються. Лексеми, що починаються з великої літери є термінальними. Правила ілюструються діаграмами Railroad.

Журавльов Роман
SnelsiThePirate@gmail.com

1.3 Алфавіт

Програма може містити текст з використанням літер англійського алфавіту, цифр та опціонально всіх спеціальних Unicode символів. Повний список приведено в файлі TysonLexer.

2. Лексика

Лексичний аналіз вхідного тексту виконується окремим проходом і не залежить від синтаксичного та семантичного аналізу. Лексичний аналізатор розбиває вихідний текст на масив базових лексем. Базові лексеми складаються з чисел (цілих та десяткових), строкових виразів, ключових слів, ключових символів та ідентифікаторів.

Також, аналізатор підтримує коментарі формату:

```
// Одностроковий коментар
/* Багатостроковий коментар */
```

Знайдені коментарі ігноруються та до масиву лексем не потрапляють.

Журавльов Роман
SnelsiThePirate@gmail.com

2.1 Ключові слова *(KeyWords)*

null	Значення відсутнє
true	Логічний 1 (правда)
false	Логічне 0 (неправда)
if	Оператор умови
else	Логічне інакше
for	Цикл
do	Цикл
while	Цикл
log	Вивести значення
var	Змінна
let	Змінна
const	Константна змінна

Ключові слова зарезервовані в системі, і не можуть бути використані як ім’я змінної. Використання ключових слів в лексемах надано далі.

Журавльов Роман
SnelsiThePirate@gmail.com

2.2 Ключові символи *(KeySymbols)*

(	OpenParen
)	CloseParen
{	OpenBrace
}	CloseBrace
;	Semicolon
,	Comma
=	Assign
:	Colon
.	Dot
++	PlusPlus
--	MinusMinus
+	Plus
-	Minus
!	Not
*	Multiply
/	Divide
%	Modulus
<	LessThan
>	MoreThan
<=	LessThanEquals
>=	GreaterThanEquals
==	Equals
!=	NotEquals
===	StrictEquals
!==	StrictNotEquals
^	Power
&&	And
||	Or

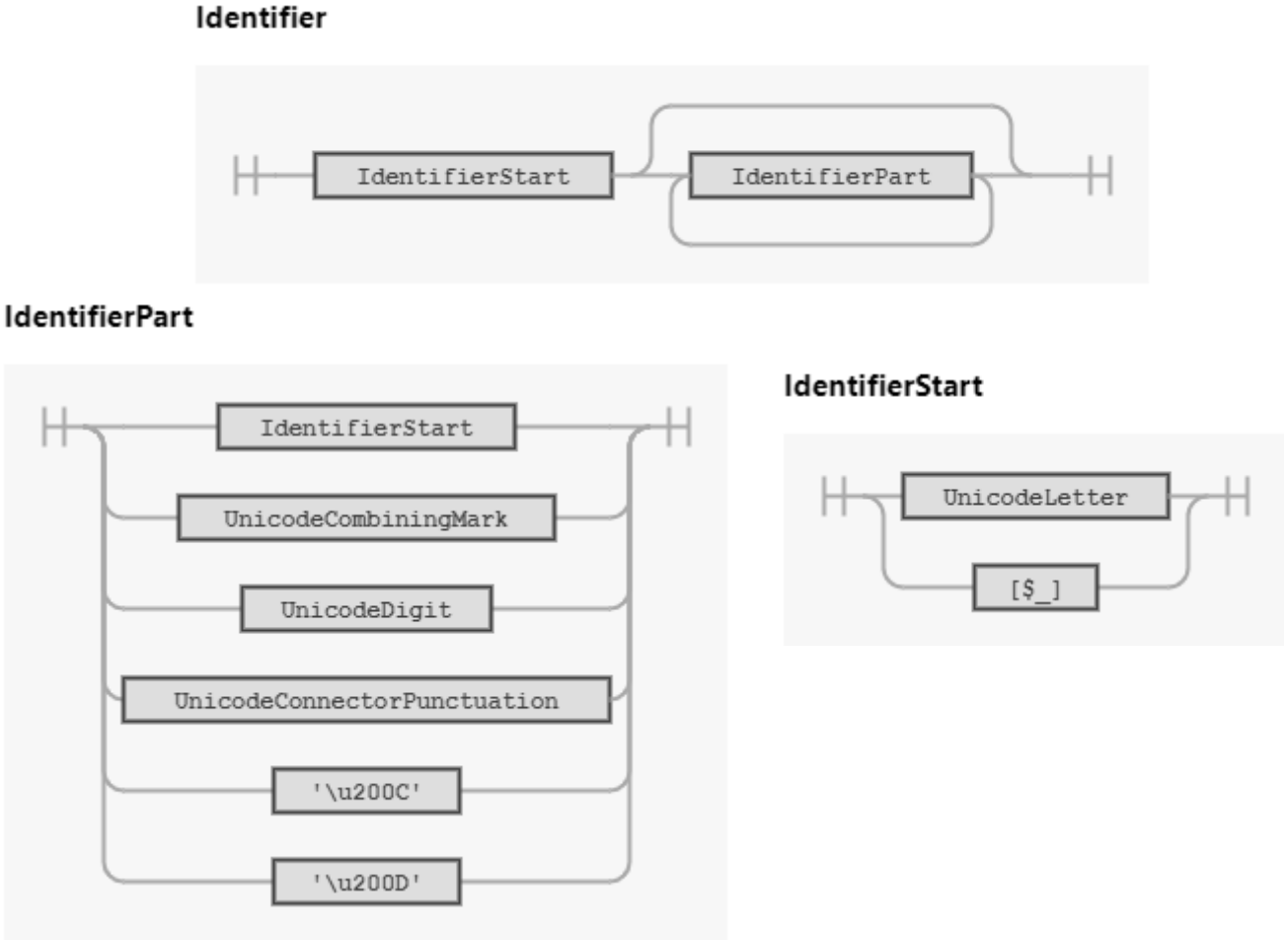
Журавльов Роман
SnelsiThePirate@gmail.com

*=	MultiplyAssign
/=	DivideAssign
&=	ModulusAssign
+=	PlusAssign
-=	MinusAssign
^=	PowerAssign

Ключові символи можуть йти один за одним без пробілів, але якщо з послідовності символів можливо скласти декілька спеціальних символів, обирається найдовший з можливих.

Журавльов Роман
SnelsiThePirate@gmail.com

2.3 Ідентифікатори



Іденифікатор – це змінна, що зберігає в собі значення. Першим символом ідентифікатора може бути літера або нижнє підкреслення, наступні символи, якщо вони є, можуть бути цифрами, нижнім підкресленням або літерами. Довжина ідентифікатора не обмежена.

Приклад ідентифікаторів:

myVariable, _UnderscoreVariable, variableWithNumber4

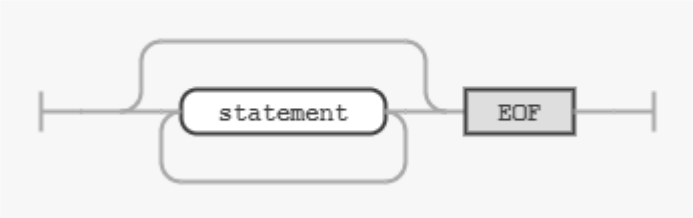
Ідентифікатор не може збігатись з ключовим зарезервованим словом. Елемент, який у фазі лексичного аналізу може бути визначений як ідентифікатор, або як ключове слово, вважається ключовим словом.

Журавльов Роман
SnelsiThePirate@gmail.com

3. Мовні конструкції

Програма складається з нуля або більше виразів:

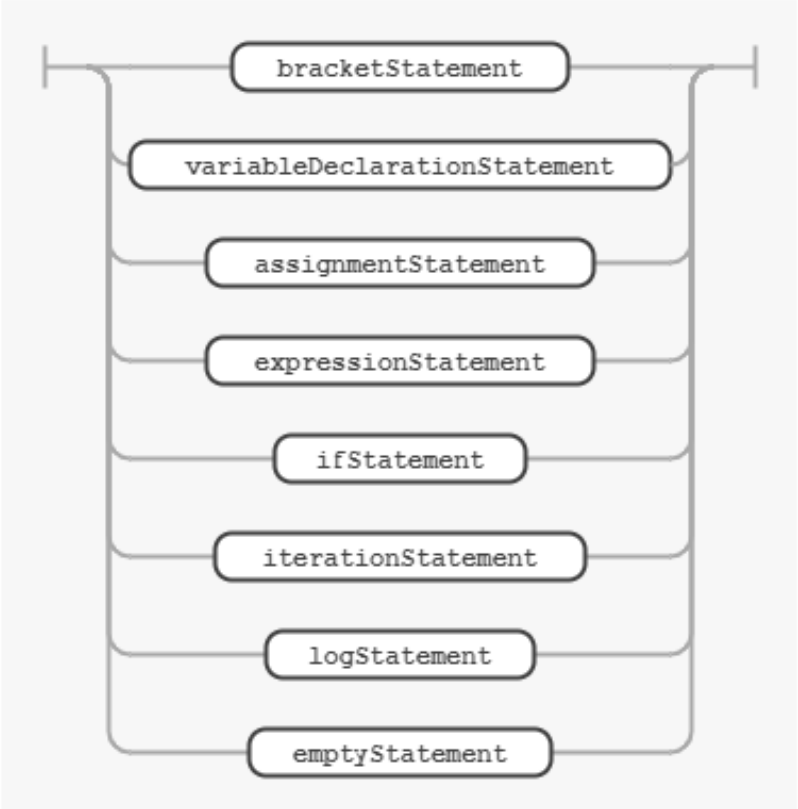
program



3.1.Вираз *(statement)*

Tyson підтримує 8 типів виразів. Вони включають створення змінних, присвоєння значення змінним, вивід значень на екран, логіку та цикли. Граматику кожного виразу надано далі.

statement



Журавльов Роман
SnelsiThePirate@gmail.com

3.2.1 bracketStatement

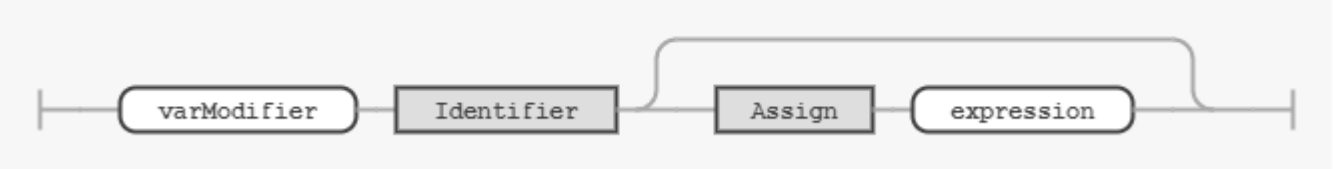
bracketStatement

Набір виразів, оточений дужками ‘{’ та ‘}’. Вирази всередині оброблюються послідовно, як єдиний вираз.

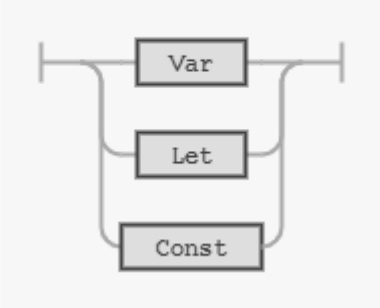
3.2.2 variableDeclarationStatement

variableDeclarationStatement

variableDeclaration



varModifier



Вираз оголошує змінну, та опціонально присвоює їй значення. Змінна може бути перезаписуєма (var, let) або константа (const).

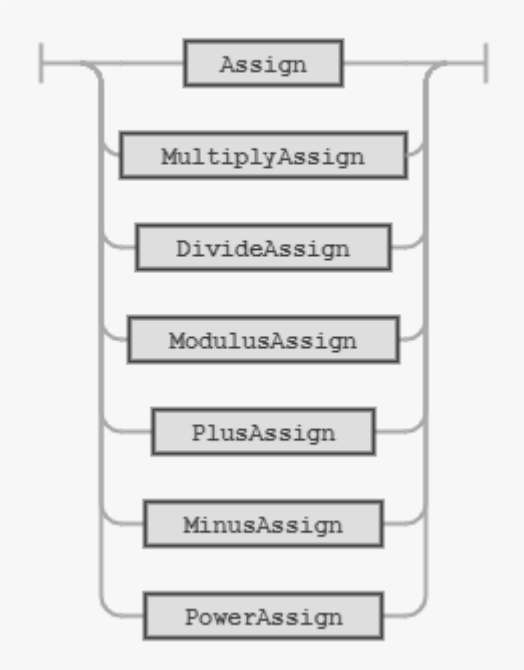
Журавльов Роман
SnelsiThePirate@gmail.com

3.3 assignmentStatement

assignmentStatement

assignment

assignmentOperator

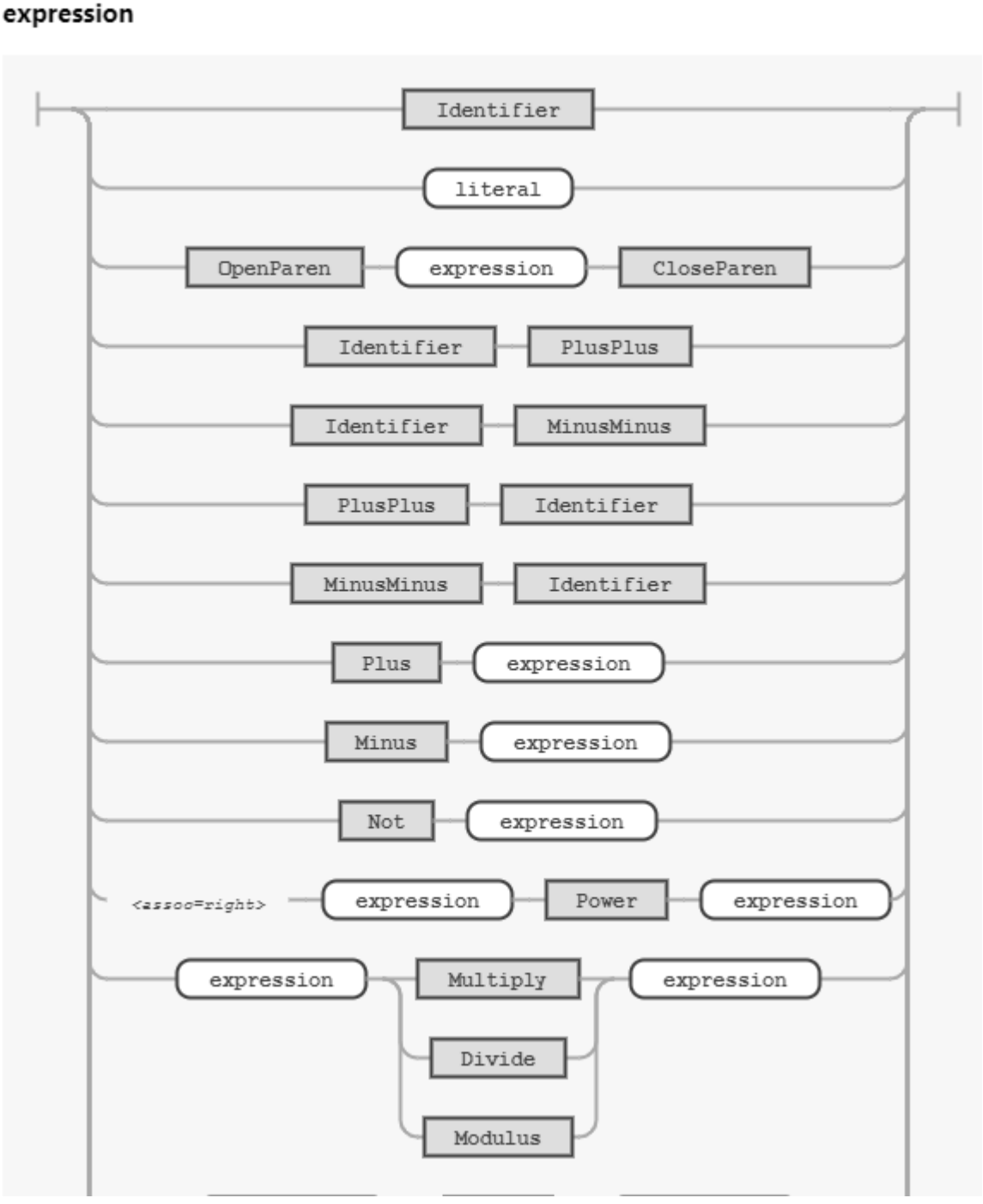


Цей вираз присвоює змінній нове значення. При цьому вираз підтримує формат поточного присвоєння, коли шаблон “змінна += значення” інтерпретується як “змінна = змінна + значення” для всіх операцій поточного присвоєння.

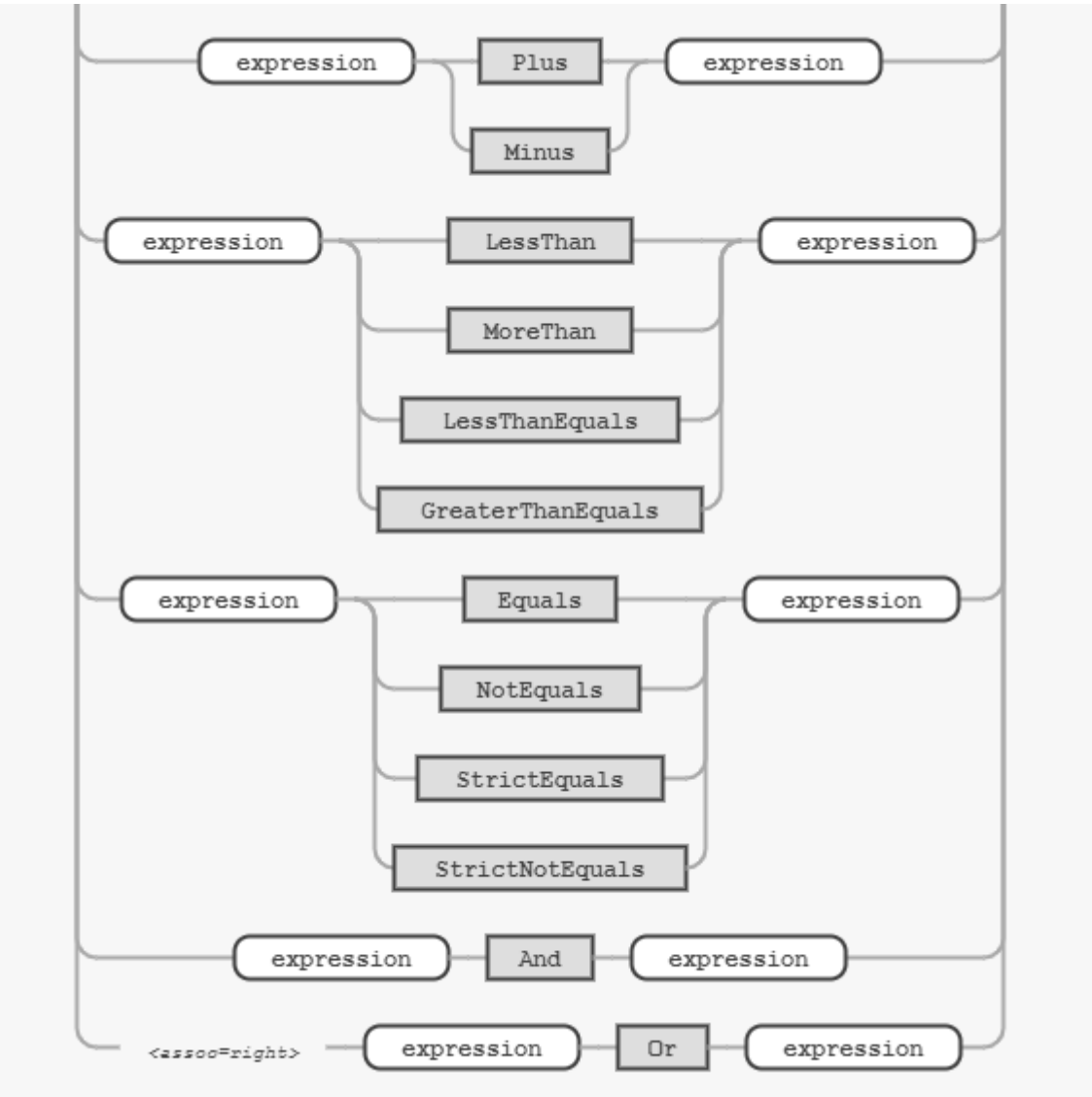
Журавльов Роман
SnelsiThePirate@gmail.com

3.4 expressionStatement

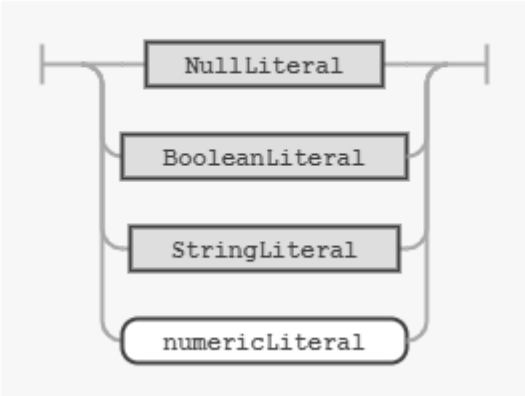
Expression – це будь-що, що повертає значення: число, логічний вираз, змінна, математичний вираз. Це базова логіка операторів програми, та інструмент їх взаємодії між собою. Expression визначає набір базових операцій, що можуть бути виконані над одним чи двома значеннями.



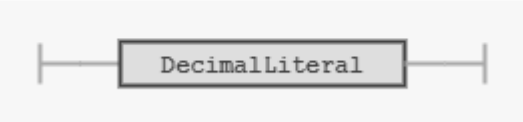
Журавльов Роман
SnelsiThePirate@gmail.com



literal



numericLiteral



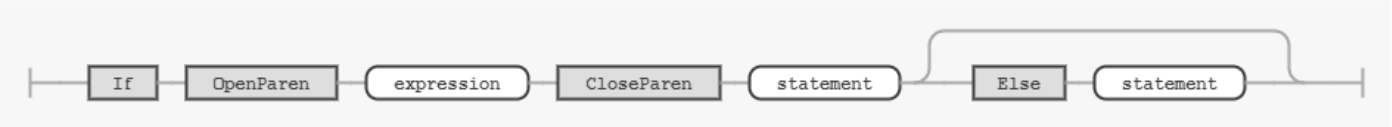
Приклад expression:

1, true, “String characters”, 1 + 2, 3 > 2, myVariable, temp > 0 || temp < 4.

Журавльов Роман
SnelsiThePirate@gmail.com

3.5 ifStatement

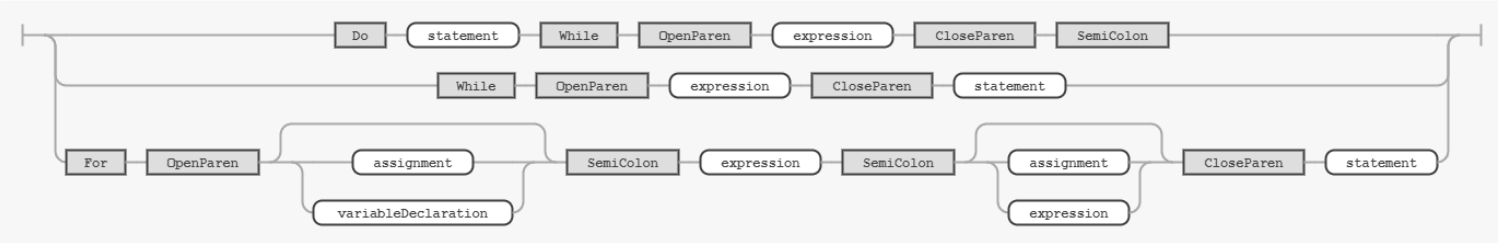
ifStatement



Вираз if виконує statement після дужок, якщо виконується умова в дужках. Опціонально оператор підтримує логічну вітку “Інакше”, яка виконується у випадку, якщо значення в дужках хибне. Інакше ця вітка ігнорується.

3.6 iterationStatement

iterationStatement



Існує три вирази, що ініціюють циклову обробку, це for, doWhile та While.

While – цикл з передумовою. Він виконує перевірку, і якщо значення в дужках правдиве, виконує вираз, що йде за дужками, а потім повертається назад до умови.

Do while – цикл з постумовою. Він спочатку виконує вираз після do, а потім перевіряє на правдивість умову після while, та повертається назад до виразу, якщо ця умова виконується.

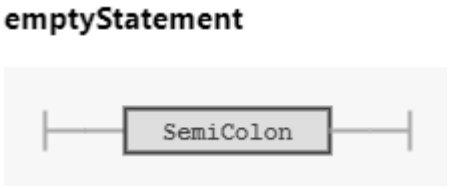
For – це варіація циклу While, але з опціональними блоком ініціювання та ітерації.

Журавльов Роман
SnelsiThePirate@gmail.com

3.7 logStatement

logStatement виводить передане значення на екран. Ця команда використовується для дебагінгу та демонстрації роботи програми чи змінних користувачу.

3.8 emptyStatement



Пусті вирази також підтримуються. Такий вираз складається лише з крапки з комою. Ніяких команд при цьому не виконується.

Журавльов Роман
SnelsiThePirate@gmail.com

4 Повна ANTLR граматика мови Tyson

Lexer

```
lexer grammar TysonLexer;

channels { ERROR }

MultiLineComment:               '/*' .*? '*/'              -> channel(HIDDEN);
SingleLineComment:              '//' ~[\r\n\u2028\u2029]* -> channel(HIDDEN);

OpenParen:                      '(';
CloseParen:                     ')';
OpenBrace:                      '{';
CloseBrace:                     '}';
SemiColon:                      ';';
Comma:                          ',';
Assign:                         '=';
Colon:                          ':';
Dot:                            '.';
PlusPlus:                       '++';
MinusMinus:                     '--';
Plus:                           '+';
Minus:                          '-';
Not:                            '!';
Multiply:                       '*';
Divide:                         '/';
Modulus:                        '%';
LessThan:                       '<';
MoreThan:                       '>';
LessThanEquals:                 '<=';
GreaterThanEquals:              '>=';
Equals:                         '==';
NotEquals:                      '!=';
StrictEquals:                   '===';
StrictNotEquals:                '!==';
Power:                          '^';
And:                            '&&';
Or:                             '||';
MultiplyAssign:                 '*=';
DivideAssign:                   '/=';
ModulusAssign:                  '%=';
PlusAssign:                     '+=';
MinusAssign:                    '-=';
PowerAssign:                    '^=';

NullLiteral:                    'null';

BooleanLiteral:                 'true'
              |                 'false';
```

Журавльов Роман
SnelsiThePirate@gmail.com

```
DecimalLiteral:                 DecimalIntegerLiteral
              |                 DecimalIntegerLiteral '.' [0-9]*
              |                 '.' [0-9]+
              ;

If:                             'if';
Else:                           'else';
For:                            'for';
Do:                             'do';
While:                          'while';

Log:                            'log';

Var:                            'var';
Let:                            'let';
Const:                          'const';

Identifier:                     IdentifierStart IdentifierPart*;

StringLiteral:                 ('"' DoubleStringCharacter* '"'
             |                  '\'' SingleStringCharacter* '\'')
             ;

WhiteSpaces:                    [\t\u000B\u000C\u0020\u00A0]+ -> channel(HIDDEN);

LineTerminator:                 [\r\n\u2028\u2029] -> channel(HIDDEN);

UnexpectedCharacter:            . -> channel(ERROR);

// Fragment rules

fragment DoubleStringCharacter
    : ~["\\\r\n]
    | '\\' EscapeSequence
    ;
fragment SingleStringCharacter
    : ~['\\\r\n]
    | '\\' EscapeSequence
    ;
fragment EscapeSequence
    : CharacterEscapeSequence
    ;
fragment CharacterEscapeSequence
    : SingleEscapeCharacter
    | NonEscapeCharacter
    ;

fragment SingleEscapeCharacter
    : ['"\\bfnrtv]
    ;
```

Журавльов Роман
SnelsiThePirate@gmail.com

```
fragment NonEscapeCharacter
    : ~['"\\bfnrtv0-9xu\r\n]
    ;
fragment EscapeCharacter
    : SingleEscapeCharacter
    | [0-9]
    | [xu]
    ;

fragment DecimalIntegerLiteral
    : '0'
    | [1-9] [0-9]*
    ;

fragment IdentifierPart
    : IdentifierStart
    | UnicodeCombiningMark
    | UnicodeDigit
    | UnicodeConnectorPunctuation
    | '\u200C'
    | '\u200D'
    ;
fragment IdentifierStart
    : UnicodeLetter
    | [$_]
    ;
```

Журавльов Роман
SnelsiThePirate@gmail.com

Parser

```
parser grammar TysonParser;

options {
    tokenVocab=TysonLexer;
}

program
    : statement* EOF
    ;

statement
    : bracketStatement
    | variableDeclarationStatement
    | assignmentStatement
    | expressionStatement
    | ifStatement
    | iterationStatement
    | logStatement
    | emptyStatement
    ;

bracketStatement
    : OpenBrace statement* CloseBrace
    ;

logStatement
    : Log OpenParen expression CloseParen
    ;

emptyStatement
    : SemiColon
    ;

variableDeclarationStatement
    : variableDeclaration SemiColon
    ;

variableDeclaration
    : varModifier Identifier (Assign expression)?
    ;

varModifier
    : Var
    | Let
    | Const
    ;
```

Журавльов Роман
SnelsiThePirate@gmail.com

```
assignmentStatement
    : assignment SemiColon
    ;

assignment
    : Identifier assignmentOperator expression
    ;

assignmentOperator
    : Assign
    | MultiplyAssign
    | DivideAssign
    | ModulusAssign
    | PlusAssign
    | MinusAssign
    | PowerAssign
    ;

expressionStatement
    : expression SemiColon
    ;

ifStatement
    : If OpenParen expression CloseParen statement (Else statement)?
    ;

iterationStatement
    : Do statement While OpenParen expression CloseParen SemiColon
                                                        # DoStatement
    | While OpenParen expression CloseParen statement
                                                        # WhileStatement
    | For OpenParen (assignment | variableDeclaration)? SemiColon expression SemiColon (
assignment | expression)? CloseParen statement          # ForStatement
    ;

expression
    : Identifier                                            # IdentifierExpression
    | literal                                               # LiteralExpression
    | OpenParen expression CloseParen                       # ParenthesizedExpression
    | Identifier PlusPlus                                   # PostIncrementExpression
    | Identifier MinusMinus                                 # PostDecreaseExpression
    | PlusPlus Identifier                                   # PreIncrementExpression
    | MinusMinus Identifier                                 # PreDecreaseExpression
    | Plus expression                                       # UnaryPlusExpression
    | Minus expression                                      # UnaryMinusExpression
    | Not expression                                        # NotExpression
    | <assoc=right> expression Power expression             # PowerExpression
    | expression (Multiply | Divide | Modulus) expression   # MultiplicativeExpression
    | expression (Plus | Minus) expression                  # AdditiveExpression
    | expression (LessThan | MoreThan | LessThanEquals | GreaterThanEquals) expression
    # RelationalExpression
```

```
    | expression (Equals | NotEquals | StrictEquals | StrictNotEquals) expression
    # EqualityExpression
    | expression And expression                                    # LogicalAndExpression
    | <assoc=right> expression Or expression                       # LogicalOrExpression
    ;

literal
    : NullLiteral
    | BooleanLiteral
    | StringLiteral
    | numericLiteral
    ;

numericLiteral
    : DecimalLiteral
    ;
```