Μέλη της Ομάδας:

Αργύρης Μάλλιος Α.Μ.:1051937 Έτος:5^{ο} email: amallios@ceid.upatras.gr

Παναγιώτης Μπαρμπούνης Α.Μ: 1054382 Έτος:4^{ο} email: up1054382@upnet.gr

Κωνσταντίνος Γκαγιάννης Α.Μ.: 1054426 Έτος:4^{ο} email: up1054426@upnet.gr

Αλέξανδρος Νατσολλάρη Α.Μ.:1057769 Έτος:4^{ο} email: up1057769@upnet.gr

Τα **ΤΕΛΙΚΑ** αρχεία περιγραφής της γλώσσας, τα οποία δίνονται ως **είσοδος** στα **Flex και Bison** βρίσκονται στις **τελευταίες σελίδες του pdf**. Παρακάτω παραθέτουμε screenshot εκτέλεσης του parser αλλά όπως θα δείτε και από το αρχείο input.txt έχουμε παραθέσει κάθε δυνατό συνδυασμό εισόδου για να τεστάρουμε τον κώδικα μας. Αν θεωρείτε πως δεν χρειάζεται μπορείτε να αφαιρέσετε το πρώτο κομμάτι μέχρι εκεί που ξεκάνει το <<class ceid:>>. Επίσης για μεγαλύτερη ευκολία έχουμε βάλει μια printf σχεδόν σε κάθε κομμάτι που χρειάζεται δόκιμη έτσι ώστε όταν εντοπίζεται ένα στοιχειό στο αρχείο input.txt να βλέπουμε και τα αντίστοιχα αποτελέσματα. Για παράδειγμα αν εντοπίσουμε for loop έχουμε βάλει να τυπώνεται και το αντίστοιχο μήνυμα κατά την εκτέλεση. Στην λύση μας έχουμε υλοποιήσει το πρώτο ερώτημα καθώς και τα dictionaries από το τρίτο ερώτημα. Γνωρίζουμε ότι κατά το compile εμφανίζονται warnings για shift/reduce και reduce/reduce τα όποια όμως δε μπορούμε να διορθώσουμε καθώς έχουμε χρησιμοποιήσει ιδία tokens σε πολλαπλούς κανόνες με μικρές αλλαγές ωστόσο που έχουν νόημα για την δημιουργία νέου κανόνα διότι θέλαμε σε κάποια σημεία να φτιάξουμε αναδρομικότητα στους κανόνες έτσι ώστε να μπορούν να χρησιμοποιούνται αρκετές φορές στο input μας. Δηλαδή ένα παράδειγμα θα ήταν οι κανόνες :

```
simple_statements: statements {printf("STATEMENT\n");}
        |simple_statements statements {printf("SIMPLESTATMENT\n");}
        |pass_stm
        |del_stmt
        |return_stmt
        |yield_stmt
        |raise_stmt
        |break_stmt
        |continue_stmt
        |import_stmt
```

```
        |global_stmt

        |nonlocal_stmt

        ;
```

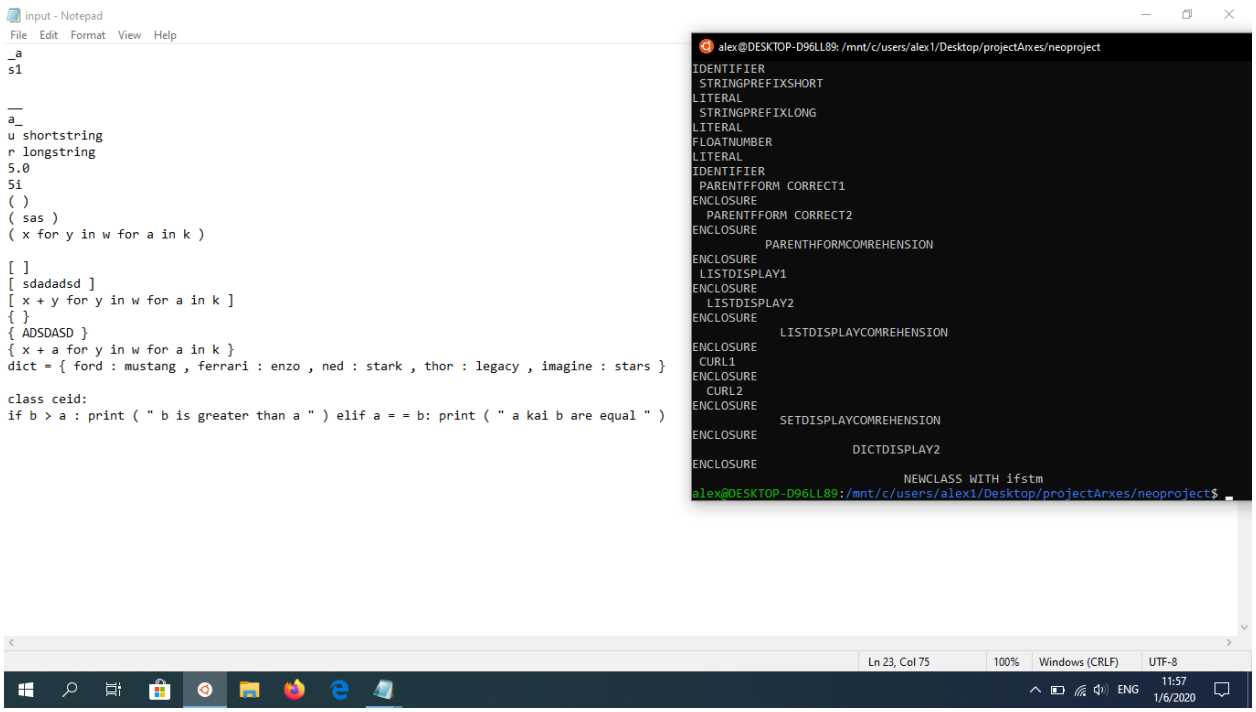
Και όπως θα δείτε και στο αρχείο projbison.y το **|continue_stmt** είναι μόνο μια λέξη. Το όποιο καθιστά τον κανόνα **continue_stmt: CONTINUE** κατά μια έννοια άχρηστο, αφού θα μπορούσαμε να βάλουμε το continue απευθείας στο simple_statements. Ωστόσο αυτό κατά την κρίση μας δεν είναι και πολύ πρακτικό στο να φαίνεται η υλοποίηση του parser όσο το δυνατόν πιο κατανοητή σε κάποιο τρίτο που την διαβάζει και δε ξέρει ότι με την λέξη continue θέλαμε να υλοποιήσουμε το statement continue. Τέλος κάποια από τα shift/reduce και reduce/reduce προέρχονται από κανόνες του τύπου:

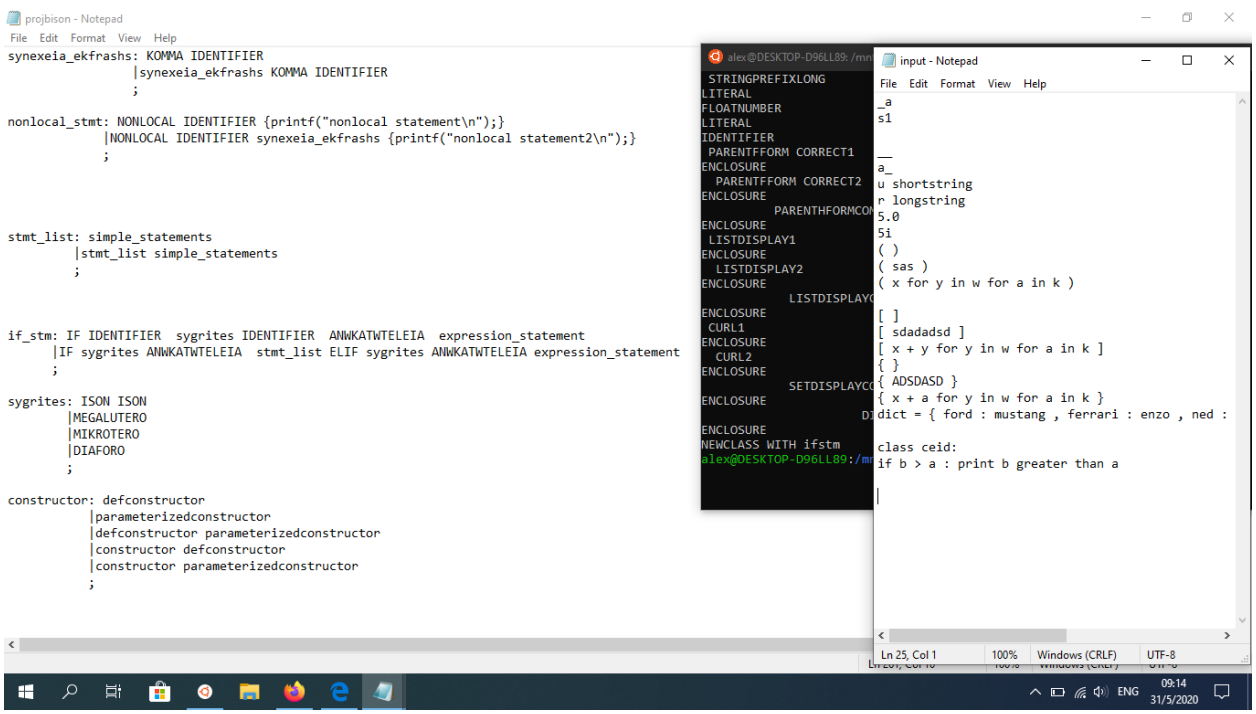
```
simple_statements: statements {printf("STATEMENT\n");}

        |simple_statements statements {printf("SIMPLESTATMENT\n");}
```

Ωστόσο η λογική μας πίσω από κανόνες τέτοιου τύπου ήταν η αναδρομικότητα του κανόνα.

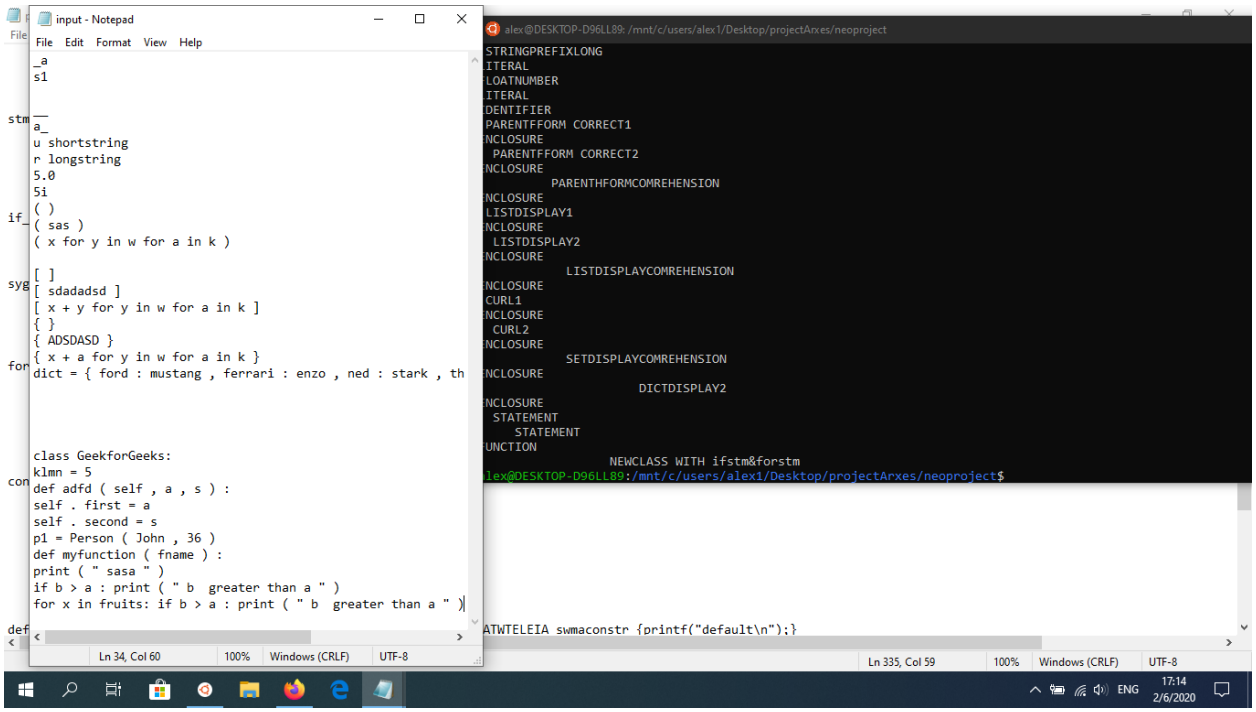
SCREENSHOT ΕΚΤΕΛΕΣΗΣ ΚΩΔΙΚΑ:

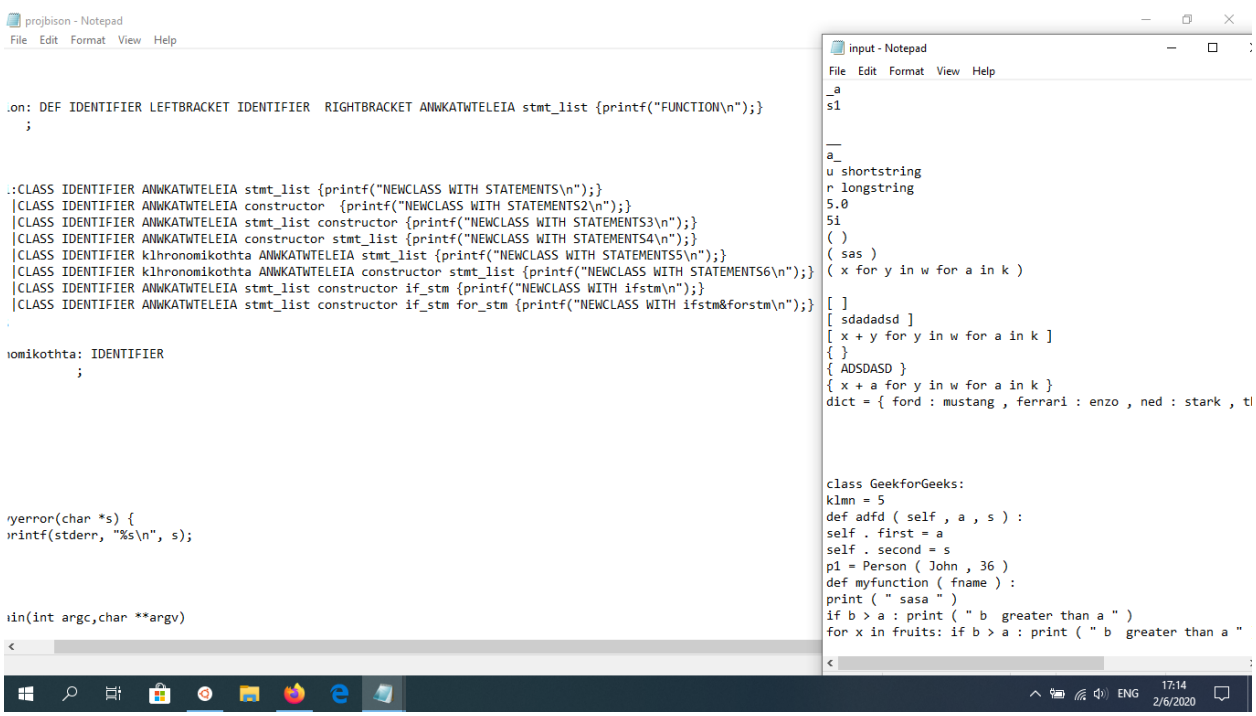
IF STATEMENT:





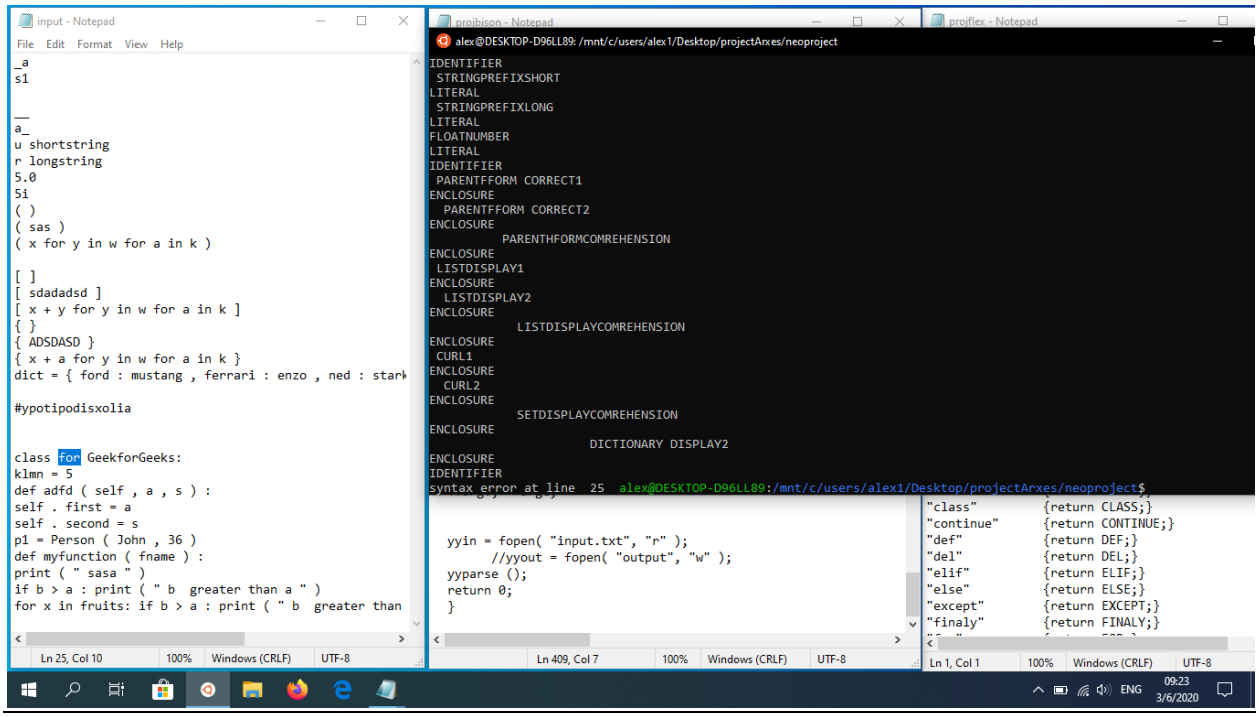
FOR STATEMENT(WITH IF STATEMENT INSIDE):

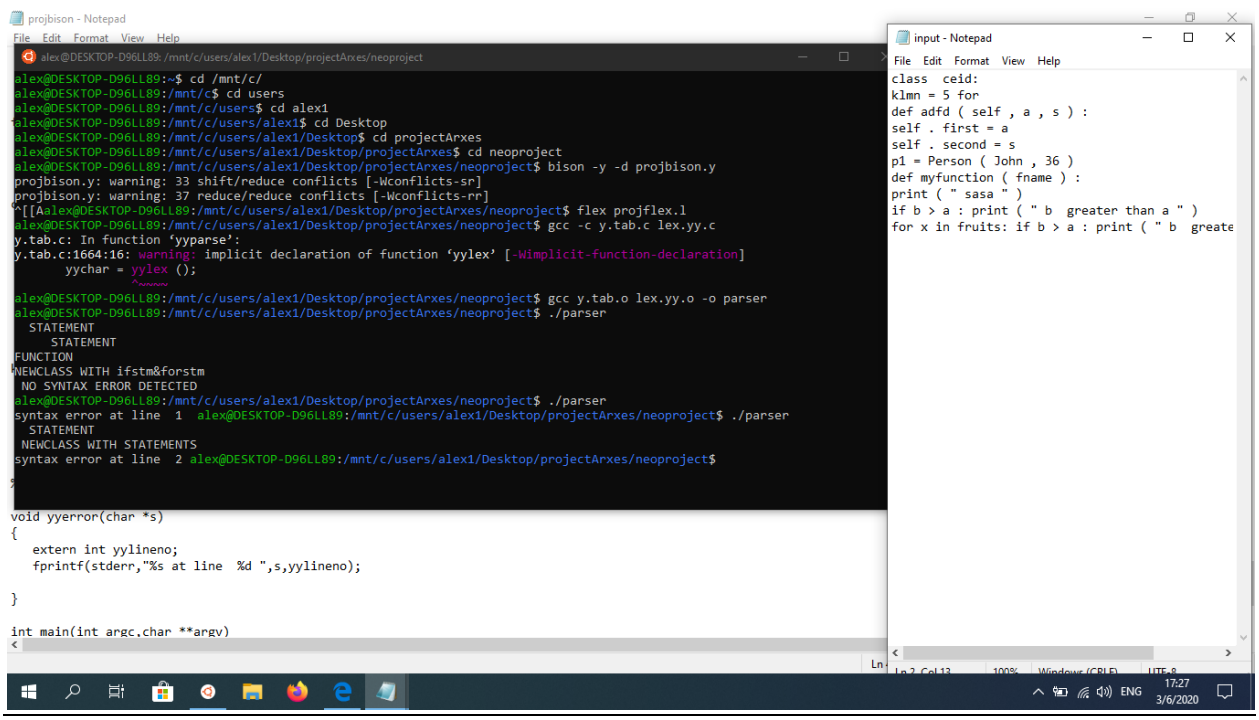




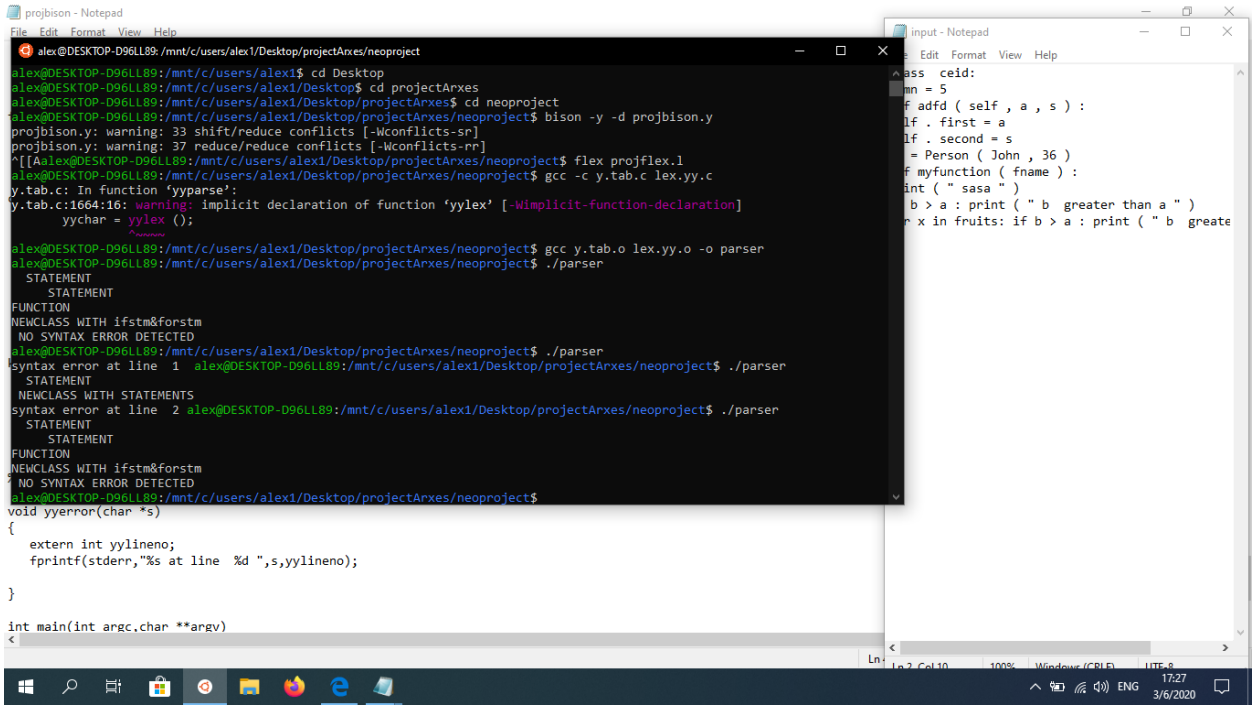
SYNTAX ERROR HANDLING:

Στο επόμενο screenshot φαίνεται τι γίνεται αν προσθέσουμε ένα for εκεί που δεν πρέπει και πως ο parser το χειρίζεται .





Εδώ βλέπουμε τι γίνεται αν δεν υπάρχει κάποιο syntax error :



<u>BNF ΥΠΟΣΥΝΟΛΟΥ ΤΗΣ PYTHON:</u>

atom ::= identifier

| literal

| enclosure

enclosure ::= parenth_form

| list_display

| generator_expression

| dict_display

| string_conversion

| yield_atom

```
literal ::= stringliteral
          | integer
          | longinteger
          | floatnumber
          | imagnumber

stringliteral ::= stringliteralpiece
                | stringliteral stringliteralpiece

parenth_form ::= "(" [expression_list] ")"

list_display ::= "[" [expression_list | list_comprehension] "]"

list_comprehension ::= expression list_for


generator_expression ::= "(" expression genexpr_for ")"

genexpr_for ::= "for" target_list "in" or_test [genexpr_iter]


dict_display ::=  "{" [key_datum_list] "}"
```

```
yield_atom ::= "(" yield_expression ")"

yield_expression ::= "yield" [expression_list]

primary ::=  atom
        | attributeref
        | subscription
        | slicing
        | call

attributeref ::=  primary "." identifier

subscription ::=  primary "[" expression_list "]"

slicing ::= simple_slicing | extended_slicing

simple_slicing ::=  primary "[" short_slice "]"

extended_slicing ::= primary "[" slice_list "]"

slice_list ::=  slice_item ("," slice_item)* [","]

slice_item ::= expression | proper_slice | ellipsis

proper_slice ::= short_slice | long_slice
```

```
short_slice ::= [lower_bound] ":" [upper_bound]

long_slice ::= short_slice ":" [stride]

lower_bound ::=  expression

upper_bound ::= expression

stride ::= expression

ellipsis ::=  "..."

call ::=  primary "(" [argument_list [","]
      | expression genexpr_for] ")"

argument_list ::=  positional_arguments ["," keyword_arguments]["," "*" expression] ["," "**"
expression]
                | keyword_arguments ["," "*" expression] ["," "**" expression]
                | "*" expression ["," "**" expression]
                | "**" expression

positional_arguments ::= expression ("," expression)*

keyword_arguments ::=  keyword_item ("," keyword_item)*
```

```
keyword_item ::= identifier "=" expression

power ::= primary ["**" u_expr]

u_expr ::=  power
        | "-" u_expr
        | "+" u_expr
        | "~" u_expr

m_expr ::= u_expr
        | m_expr "*" u_expr
        | m_expr "//" u_expr
        | m_expr "/" u_expr
        | m_expr "%" u_expr

a_expr ::=  m_expr
        | a_expr "+" m_expr
        | a_expr "-" m_expr

shift_expr ::= a_expr
            | shift_expr ( "<<" | ">>" ) a_expr

and_expr ::= shift_expr | and_expr "&" shift_expr

xor_expr ::= and_expr
```

```
        | xor_expr "^" and_expr

or_expr ::= xor_expr
       | or_expr "|" xor_expr

comparison ::=
           or_expr ( comp_operator or_expr )*

comp_operator ::= "<" | ">" | "==" | ">=" | "<=" | "<>" | "!="
              | "is" ["not"] | ["not"] "in"

expression ::=  conditional_expression | lambda_form

old_expression ::=  or_test | old_lambda_form

conditional_expression ::= or_test ["if" or_test "else" expression]

or_test ::= and_test
       | or_test "or" and_test

and_test ::= not_test
        | and_test "and" not_test

not_test ::= comparison
        | "not" not_test
```

```
lambda_form ::= "lambda" [parameter_list] ":" expression

old_lambda_form ::= "lambda" [parameter_list] ":" old_expression

expression_list ::= expression ( "," expression )* [","]

simple_stmt ::= expression_stmt
            | assert_stmt
            | assignment_stmt
            | augmented_assignment_stmt
            | pass_stmt
            | del_stmt
            | print_stmt
            | return_stmt
            | yield_stmt
            | raise_stmt
            | break_stmt
            | continue_stmt
            | import_stmt
            | global_stmt
            | exec_stmt

expression_stmt ::=  expression_list
```

```
assert_stmt ::=  "assert" expression ["," expression]

assignment_stmt ::= (target_list "=")+(expression_list | yield_expression)

target_list ::=  target ("," target)* [","]

target ::= identifier
           | "(" target_list ")"
           | "[" target_list "]"
           | attributeref
           | subscription
           | slicing

augmented_assignment_stmt ::= target augop(expression_list | yield_expression)

augop ::= "+=" | "-=" | "*=" | "/=" | "//=" | "%=" | "**="| ">>=" | "<<=" | "&=" | "^=" | "|="

pass_stmt ::= "pass"

del_stmt ::= "del" target_list

print_stmt ::= "print" (  [expression ("," expression)* [","]]| ">>" expression [("," expression)+ [","]] )

return_stmt ::= "return" [expression_list]
```

```
yield_stmt ::=  yield_expression

raise_stmt ::= "raise" [expression ["," expression["," expression]]]

break_stmt ::= "break"

continue_stmt ::= "continue"

import_stmt ::= "import" module ["as" name]( "," module ["as" name] )*
            | "from" relative_module "import" identifier["as" name]( "," identifier ["as" name] )*
            | "from" relative_module "import" "(" identifier ["as" name]( "," identifier ["as" name] )* [","] ")"
            | "from" module "import" "*"

module ::= (identifier ".")* identifier

relative_module ::= "."* module | "."+

name ::= identifier

global_stmt ::= "global" identifier ("," identifier)*

exec_stmt ::= "exec" or_expr["in" expression ["," expression]]

compound_stmt ::= if_stmt
              | while_stmt
```

```
            | for_stmt
            | try_stmt
            | with_stmt
            | funcdef
            | classdef

suite ::= stmt_list NEWLINE
      | NEWLINE INDENT statement+ DEDENT

statement ::= stmt_list NEWLINE
          | compound_stmt

stmt_list ::= simple_stmt (";" simple_stmt)* [";"]

if_stmt ::= "if" expression ":" suite( "elif" expression ":" suite )*["else" ":" suite]

while_stmt ::= "while" expression ":" suite["else" ":" suite]

for_stmt ::= "for" target_list "in" expression_list":" suite["else" ":" suite]

try_stmt ::=  try1_stmt
         | try2_stmt

try1_stmt ::= "try" ":" suite("except" [expression["," target]] ":" suite)+["else" ":" suite]["finally" ":" suite]
```

```
try2_stmt ::= "try" ":" suite"finally" ":" suite

with_stmt ::= "with" expression ["as" target] ":" suite

funcdef ::= [decorators] "def" funcname "(" [parameter_list] ")"":" suite

decorators ::= decorator+

decorator ::= "@" dotted_name ["(" [argument_list [","]] ")"] NEWLINE

dotted_name ::=
        identifier ("." identifier)*

parameter_list ::=
            (defparameter ",")*
           ("*" identifier [, "**" identifier]
            | "**" identifier
            | defparameter [","] )

defparameter ::= parameter ["=" expression]

sublist ::= parameter ("," parameter)* [","]

parameter ::= identifier | "(" sublist ")"
```

funcname ::= identifier

classdef ::= "class" classname [inheritance] ":"suite

inheritance ::= "(" [expression_list] ")"

classname ::= identifier

file_input ::= (NEWLINE | statement)*

interactive_input ::= [stmt_list] NEWLINE

| compound_stmt NEWLINE

eval_input ::= expression_list NEWLINE*

input_input ::= expression_list NEWLINE

<u>BNF ΤΗΣ ΓΡΑΜΜΑΤΙΚΗΣ ΠΟΥ ΦΤΙΑΞΑΜΕ ΣΤΟ BISON:</u>

```
programm:
        |atom programm
        |classi programm
        |function programm
        ;


atom::= IDENTIFIER
    |literal
    | enclosure
    ;


literal::= stringliteral
        |INTEGER
        |FLOATNUMBER
        |IMAGINUMBER
        ;

stringliteral::= stringprefixShort
```

```
        |stringprefixLong
        ;

stringprefixShort::= r SHORTSTRING
        |u SHORTSTRING
        |R SHORTSTRING
        |U SHORTSTRING
        |f SHORTSTRING
        |F SHORTSTRING
        |fr SHORTSTRING
        |Fr SHORTSTRING
        |fR SHORTSTRING
        |FR SHORTSTRING
        |rf SHORTSTRING
        |rF SHORTSTRING
        |Rf SHORTSTRING
        |RF SHORTSTRING
        ;

stringprefixLong::=  r LONGSTRING
        |u LONGSTRING
        |R LONGSTRING
        |U LONGSTRING
        |f LONGSTRING
        |F LONGSTRING
        |fr LONGSTRING
        |Fr LONGSTRING
        |fR LONGSTRING
        |FR LONGSTRING
```

```
        |rf LONGSTRING
        |rF LONGSTRING
        |Rf LONGSTRING
        |RF LONGSTRING
        ;


enclosure::= parenth_form
      |list_display
      |set_display
      |dict_display
      //|string_conversion
      //|yield_atom
      ;


parenth_form::= LEFTBRACKET  RIGHTBRACKET
        |LEFTBRACKET IDENTIFIER  RIGHTBRACKET
        |LEFTBRACKET comprehension RIGHTBRACKET
        ;


list_display::= LSQUAREBRACKET RSQUAREBRACKET
        |LSQUAREBRACKET IDENTIFIER RSQUAREBRACKET
        |LSQUAREBRACKET comprehension RSQUAREBRACKET
        ;

set_display::= CURLLEFT CURLRIGHT
       |CURLLEFT IDENTIFIER CURLRIGHT
```

```
        |CURLLEFT comprehension CURLRIGHT
        ;

dict_display::=IDENTIFIER ISON CURLLEFT CURLRIGHT
        |IDENTIFIER ISON CURLLEFT dictindex CURLRIGHT
        ;


dictindex::=IDENTIFIER ANWKATWTELEIA IDENTIFIER
       |dictindex KOMMA IDENTIFIER ANWKATWTELEIA IDENTIFIER
       ;
comprehension::= IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
          |IDENTIFIER PLUS IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
          |IDENTIFIER MINUS IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
          |IDENTIFIER MUL IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
          |IDENTIFIER DIVISION IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
          ;


simple_statements::= statements
              |simple_statements statements
              |pass_stm
              |del_stmt
              |return_stmt
              |yield_stmt
              |raise_stmt
              |break_stmt
```

```
            |continue_stmt
            |import_stmt
            |global_stmt
            |nonlocal_stmt
            ;


statements::= expression_statement
        |assignment_statment
        ;


expression_statement::= PRINT LEFTBRACKET DOUBLEQUOTE protash DOUBLEQUOTE RIGHTBRACKET
                |PRINT FLOATNUMBER
                ;

protash::= IDENTIFIER
      |protash IDENTIFIER
      ;


assignment_statment::= IDENTIFIER ISON FLOATNUMBER
                |IDENTIFIER ISON IMAGINUMBER
                |IDENTIFIER ISON INTEGER
                |IDENTIFIER ISON IDENTIFIER
                |augmented_assignment_stmt
                ;
```

augmented_assignment_stmt::= IDENTIFIER PLUS ISON IDENTIFIER

|IDENTIFIER MINUS ISON IDENTIFIER

;

pass_stm::= PASS

;

del_stmt::= DEL IDENTIFIER

|DEL LEFTBRACKET IDENTIFIER RIGHTBRACKET

|DEL LSQUAREBRACKET IDENTIFIER RSQUAREBRACKET

;

return_stmt::= RETURN IDENTIFIER

;

yield_stmt::= YIELD IDENTIFIER

|YIELD FROM IDENTIFIER

;

raise_stmt::= RAISE IDENTIFIER FROM IDENTIFIER

|RAISE IDENTIFIER LEFTBRACKET DOUBLEQUOTE IDENTIFIER DOUBLEQUOTE RIGHTBRACKET FROM IDENTIFIER

;

break_stmt::= BREAK

```
        ;

continue_stmt::= CONTINUE
          ;

import_stmt::= IMPORT IDENTIFIER
        |IMPORT module
        |FROM module IMPORT IDENTIFIER
        |IMPORT module AS IDENTIFIER
        ;


module::= IDENTIFIER
     |IDENTIFIER TELEIA IDENTIFIER
     |module TELEIA IDENTIFIER
     ;

global_stmt::= GLOBAL IDENTIFIER
        |GLOBAL IDENTIFIER synexeia_ekfrashs
        ;

synexeia_ekfrashs::= KOMMA IDENTIFIER
              |synexeia_ekfrashs KOMMA IDENTIFIER
              ;

nonlocal_stmt::= NONLOCAL IDENTIFIER
          |NONLOCAL IDENTIFIER synexeia_ekfrashs
          ;
```

stmt_list::= simple_statements

|stmt_list simple_statements

;

if_stm::= IF IDENTIFIER sygrites IDENTIFIER ANWKATWTELEIA expression_statement

|IF IDENTIFIER sygrites IDENTIFIER ANWKATWTELEIA expression_statement ELIF IDENTIFIER sygrites IDENTIFIER ANWKATWTELEIA expression_statement

;

sygrites::= ISON ISON

|MEGALUTERO

|MIKROTERO

|DIAFORO

;

for_stm::= FOR IDENTIFIER IN IDENTIFIER ANWKATWTELEIA expression_statement

|FOR IDENTIFIER IN IDENTIFIER ANWKATWTELEIA if_stm

;

constructor::= defconstructor

|parameterizedconstructor

|defconstructor parameterizedconstructor

|constructor defconstructor

|constructor parameterizedconstructor

;

defconstructor::= DEF INIT LEFTBRACKET IDENTIFIER RIGHTBRACKET ANWKATWTELEIA swmaconstr

;

parameterizedconstructor::= DEF IDENTIFIER LEFTBRACKET IDENTIFIER RIGHTBRACKET swmaconstr

|DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA recursive_swmaconstr

|DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA recursive_swmaconstr object

|DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA recursive_swmaconstr function

|DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA recursive_swmaconstr object function

;

swmaconstr::= module ISON DOUBLEQUOTE IDENTIFIER DOUBLEQUOTE

|module ISON IDENTIFIER

;

recursive_swmaconstr::= swmaconstr

|recursive_swmaconstr swmaconstr

;

object::= IDENTIFIER ISON IDENTIFIER LEFTBRACKET IDENTIFIER RIGHTBRACKET

|IDENTIFIER ISON IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET

;

function::= DEF IDENTIFIER LEFTBRACKET IDENTIFIER RIGHTBRACKET ANWKATWTELEIA stmt_list

;

classi::=CLASS IDENTIFIER ANWKATWTELEIA stmt_list

|CLASS IDENTIFIER ANWKATWTELEIA constructor

|CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor

```
    |CLASS IDENTIFIER ANWKATWTELEIA constructor stmt_list

    |CLASS IDENTIFIER klhronomikothta ANWKATWTELEIA stmt_list

    |CLASS IDENTIFIER klhronomikothta ANWKATWTELEIA constructor stmt_list

    |CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor if_stm

    |CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor if_stm for_stm

   ;


klhronomikothta::= IDENTIFIER

            ;
```

Τελικά αρχεία περιγραφής της γλώσσας, τα οποία δίνονται ως είσοδος στα Flex και Bison:

- **Bison:**

```
%{
#include <stdio.h>
#include <string.h>
#include <stdio.h>
void yyerror(char *);
extern FILE *yyin;
extern FILE *yyout;

char tmplist_expression[100];
char list_expression[100];



%}


%locations


%union
{
   char    name;

}

%token FALSE
%token TRUE
%token AND
%token ASSERT
%token ASYNC
%token AWAIT
%token BREAK
%token CLASS
%token CONTINUE
%token DEF
%token DEL
%token ELIF
%token ELSE
%token EXCEPT
%token FINALY
%token FOR
%token FROM
%token GLOBAL
%token IF
```

```
%token IMPORT
%token IN
%token IS
%token LAMBDA
%token NONLOCAL
%token NOT
%token NONE
%token OR
%token PASS
%token RAISE
%token RETURN
%token TRY
%token WHILE
%token WITH
%token YIELD
%token LBRACKET
%token RBRACKET
%token LBRACE
%token RBRACE
%token SEMICOLON
%token LOWERCASE
%token <ystr1> NUM
%token LETTER
%token KATWPAVLA
%token QUOTE
%token DOUBLEQUOTE
%token DIGIT
%token r
%token u
%token R
%token U
%token f
%token F
%token fr
%token Fr
%token fR
%token FR
%token rf
%token Rf
%token rF
%token RF
%token SHORTSTRING
%token LONGSTRING
%token IDENTIFIER
%token INTEGER
%token FLOATNUMBER
%token IMAGINUMBER
%token LEFTBRACKET
%token RIGHTBRACKET
%token LSQUAREBRACKET
%token RSQUAREBRACKET
```

```
%token CURLLEFT
%token CURLRIGHT
%token PLUS
%token MUL
%token DIVISION
%token MINUS
%token ISON
%token KOMMA
%token ANWKATWTELEIA
%token PRINT
%token SINISON
%token PLINISON
%token TELEIA
%token AS
%token INIT
%token MEGALUTERO
%token MIKROTERO
%token DIAFORO



%%


programm:
        |atom programm
        |classi  programm
        |function programm
        ;





atom: IDENTIFIER{printf("IDENTIFIER\n");}
    |literal {printf("LITERAL\n");}
    | enclosure {printf("ENCLOSURE\n");}
    ;





literal: stringliteral
        |INTEGER
        |FLOATNUMBER {printf("FLOATNUMBER\n");}
        |IMAGINUMBER {printf("IMAGINERYNUMBER\n");}
        ;

stringliteral: stringprefixShort {printf("STRINGPREFIXSHORT\n");}
            |stringprefixLong   {printf("STRINGPREFIXLONG\n");}
```

```
            ;

stringprefixShort: r SHORTSTRING
          |u SHORTSTRING
          |R SHORTSTRING
          |U SHORTSTRING
          |f SHORTSTRING
          |F SHORTSTRING
          |fr SHORTSTRING
          |Fr SHORTSTRING
          |fR SHORTSTRING
          |FR SHORTSTRING
          |rf SHORTSTRING
          |rF SHORTSTRING
          |Rf SHORTSTRING
          |RF SHORTSTRING
          ;

stringprefixLong:  r LONGSTRING
          |u LONGSTRING
          |R LONGSTRING
          |U LONGSTRING
          |f LONGSTRING
          |F LONGSTRING
          |fr LONGSTRING
          |Fr LONGSTRING
          |fR LONGSTRING
          |FR LONGSTRING
          |rf LONGSTRING
          |rF LONGSTRING
          |Rf LONGSTRING
          |RF LONGSTRING
          ;


enclosure: parenth_form
        |list_display
        |set_display
        |dict_display
        //|string_conversion
        //|yield_atom
        ;


parenth_form: LEFTBRACKET  RIGHTBRACKET {printf("PARENTFFORM CORRECT1\n");}
          |LEFTBRACKET IDENTIFIER  RIGHTBRACKET {printf("PARENTFFORM CORRECT2\n");}
          |LEFTBRACKET comprehension RIGHTBRACKET {printf("PARENTHFORMCOMREHENSION\n");}
          ;


list_display: LSQUAREBRACKET RSQUAREBRACKET {printf("LISTDISPLAY1\n");}
```

```
        |LSQUAREBRACKET IDENTIFIER RSQUAREBRACKET {printf("LISTDISPLAY2\n");}
        |LSQUAREBRACKET comprehension RSQUAREBRACKET {printf("LISTDISPLAYCOMREHENSION\n");}
        ;

set_display: CURLLEFT CURLRIGHT {printf("CURL1\n");}
       |CURLLEFT IDENTIFIER CURLRIGHT  {printf("CURL2\n");}
       |CURLLEFT comprehension CURLRIGHT {printf("SETDISPLAYCOMREHENSION\n");}
       ;

dict_display:IDENTIFIER ISON CURLLEFT CURLRIGHT {printf("DICTIONARY DISPLAY\n");}
        |IDENTIFIER ISON CURLLEFT dictindex CURLRIGHT {printf("DICTIONARY DISPLAY2\n");}
        ;


dictindex:IDENTIFIER ANWKATWTELEIA IDENTIFIER
      |dictindex KOMMA IDENTIFIER ANWKATWTELEIA IDENTIFIER
      ;
comprehension: IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
        |IDENTIFIER PLUS IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
        |IDENTIFIER MINUS IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
        |IDENTIFIER MUL IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
        |IDENTIFIER DIVISION IDENTIFIER FOR IDENTIFIER IN IDENTIFIER FOR IDENTIFIER IN IDENTIFIER
        ;




simple_statements: statements {printf("STATEMENT\n");}
            |simple_statements statements {printf("SIMPLESTATMENT\n");}
            |pass_stm
            |del_stmt
            |return_stmt
            |yield_stmt
            |raise_stmt
            |break_stmt
            |continue_stmt
            |import_stmt
            |global_stmt
            |nonlocal_stmt
            ;


statements: expression_statement
      |assignment_statment
      ;



expression_statement: PRINT LEFTBRACKET DOUBLEQUOTE protash DOUBLEQUOTE RIGHTBRACKET
             |PRINT FLOATNUMBER
             ;
```

```
protash: IDENTIFIER
      |protash IDENTIFIER
      ;



assignment_statment: IDENTIFIER ISON FLOATNUMBER
                |IDENTIFIER ISON IMAGINUMBER
                |IDENTIFIER ISON INTEGER
                |IDENTIFIER ISON IDENTIFIER
                |augmented_assignment_stmt {printf("AUGUMENTEDSTATMENT\n");}
                ;

augmented_assignment_stmt: IDENTIFIER  PLUS ISON  IDENTIFIER
                      |IDENTIFIER MINUS ISON IDENTIFIER
                      ;


pass_stm: PASS
        ;

del_stmt: DEL IDENTIFIER
       |DEL LEFTBRACKET IDENTIFIER RIGHTBRACKET
       |DEL LSQUAREBRACKET IDENTIFIER RSQUAREBRACKET
       ;


return_stmt: RETURN IDENTIFIER {printf("return statement\n");}
          ;


yield_stmt: YIELD IDENTIFIER {printf("yield statement\n");}
         |YIELD FROM IDENTIFIER
         ;


raise_stmt: RAISE IDENTIFIER FROM IDENTIFIER {printf("raisestm\n");}
         |RAISE IDENTIFIER LEFTBRACKET DOUBLEQUOTE IDENTIFIER DOUBLEQUOTE RIGHTBRACKET FROM IDENTIFIER
{printf("raisestm2\n");}
         ;

break_stmt: BREAK
         ;

continue_stmt: CONTINUE
            ;

import_stmt: IMPORT IDENTIFIER {printf("IMPORT IDENTIFIER\n");}
          |IMPORT module {printf("IMPORT MODULE\n");}
          |FROM module IMPORT IDENTIFIER {printf("FROM IMPORT MODULE\n");}
```

```
        |IMPORT module AS IDENTIFIER {printf(" IMPORT MODULE AS\n");}
        ;


module: IDENTIFIER
    |IDENTIFIER TELEIA IDENTIFIER
    |module TELEIA IDENTIFIER
    ;

global_stmt: GLOBAL IDENTIFIER {printf("global statement\n");}
        |GLOBAL IDENTIFIER synexeia_ekfrashs {printf("global statement2\n");}
        ;

synexeia_ekfrashs: KOMMA IDENTIFIER
            |synexeia_ekfrashs KOMMA IDENTIFIER
            ;

nonlocal_stmt: NONLOCAL IDENTIFIER {printf("nonlocal statement\n");}
          |NONLOCAL IDENTIFIER synexeia_ekfrashs {printf("nonlocal statement2\n");}
          ;




stmt_list: simple_statements
       |stmt_list simple_statements
       ;



if_stm: IF IDENTIFIER  sygrites IDENTIFIER  ANWKATWTELEIA  expression_statement
    |IF IDENTIFIER sygrites IDENTIFIER ANWKATWTELEIA  expression_statement ELIF IDENTIFIER sygrites IDENTIFIER
ANWKATWTELEIA  expression_statement
    ;

sygrites: ISON ISON
      |MEGALUTERO
      |MIKROTERO
      |DIAFORO
      ;
for_stm: FOR IDENTIFIER IN IDENTIFIER ANWKATWTELEIA expression_statement
      |FOR IDENTIFIER IN IDENTIFIER ANWKATWTELEIA if_stm
      ;




constructor: defconstructor
        |parameterizedconstructor
        |defconstructor parameterizedconstructor
        |constructor defconstructor
```

```
        |constructor parameterizedconstructor
        ;


defconstructor: DEF INIT LEFTBRACKET IDENTIFIER RIGHTBRACKET ANWKATWTELEIA swmaconstr
{printf("default\n");}
          ;


parameterizedconstructor: DEF IDENTIFIER LEFTBRACKET IDENTIFIER RIGHTBRACKET swmaconstr
                  |DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA
recursive_swmaconstr {printf("parameterizedconstructor2\n");}
                  |DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA
recursive_swmaconstr object  {printf("parameterizedconstructor with object creation\n");}
                  |DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA
recursive_swmaconstr function
                  |DEF IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET ANWKATWTELEIA
recursive_swmaconstr object function
                  ;


swmaconstr: module ISON DOUBLEQUOTE IDENTIFIER DOUBLEQUOTE
        |module ISON IDENTIFIER
        ;


recursive_swmaconstr: swmaconstr
                |recursive_swmaconstr swmaconstr
                ;


object: IDENTIFIER ISON IDENTIFIER LEFTBRACKET IDENTIFIER RIGHTBRACKET
     |IDENTIFIER ISON IDENTIFIER LEFTBRACKET IDENTIFIER synexeia_ekfrashs RIGHTBRACKET
     ;
```

```
function: DEF IDENTIFIER LEFTBRACKET IDENTIFIER  RIGHTBRACKET ANWKATWTELEIA stmt_list
{printf("FUNCTION\n");}
       ;



classi:CLASS IDENTIFIER ANWKATWTELEIA stmt_list {printf("NEWCLASS WITH STATEMENTS\n");}
     |CLASS IDENTIFIER ANWKATWTELEIA constructor  {printf("NEWCLASS WITH STATEMENTS2\n");}
     |CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor {printf("NEWCLASS WITH STATEMENTS3\n");}
     |CLASS IDENTIFIER ANWKATWTELEIA constructor stmt_list {printf("NEWCLASS WITH STATEMENTS4\n");}
     |CLASS IDENTIFIER klhronomikothta ANWKATWTELEIA stmt_list {printf("NEWCLASS WITH STATEMENTS5\n");}
     |CLASS IDENTIFIER klhronomikothta ANWKATWTELEIA constructor stmt_list {printf("NEWCLASS WITH
STATEMENTS6\n");}
     |CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor if_stm {printf("NEWCLASS WITH ifstm\n");}
     |CLASS IDENTIFIER ANWKATWTELEIA stmt_list constructor if_stm for_stm  {printf("NEWCLASS WITH
ifstm&forstm\n NO SYNTAX ERROR DETECTED\n");}
    ;

klhronomikothta: IDENTIFIER
             ;




%%

void yyerror(char *s)
{
   extern int yylineno;
   fprintf(stderr,"%s at line  %d ",s,yylineno);

}

int main(int argc,char **argv)
{

  ++argv; --argc;


  yyin = fopen( "input.txt", "r" );
              //yyout = fopen( "output", "w" );
  yyparse ();
  return 0;
  }
```

- **Flex:**

```
%{
```

```
#include "y.tab.h"
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
void yyerror (char *s);
int yylex();

%}

%option noyywrap
%option yylineno


integer [0-9]
%%


"false"         {return FALSE;}
"none"          {return NONE;}
"true"          {return TRUE;}
"and"           {return AND;}
"assert"   {return ASSERT;}
"async"         {return ASYNC;}
"await"         {return AWAIT;}
"break"         {return BREAK;}
"class"         {return CLASS;}
"continue"      {return CONTINUE;}
"def"           {return DEF;}
"del"           {return DEL;}
"elif"          {return ELIF;}
"else"          {return ELSE;}
"except"  {return EXCEPT;}
"finaly"   {return FINALY;}
"for"           {return FOR;}
"from"          {return FROM;}
"global"   {return GLOBAL;}
"if"            {return IF;}
"import"  {return IMPORT;}
"in"            {return IN;}

"lambda"{return LAMBDA;}
"nonlocal"      {return NONLOCAL;}
"not"           {return NOT;}
"or"            {return OR;}
"pass"          {return PASS;}
"raise"         {return RAISE;}
"return"  {return RETURN;}
"try"           {return TRY;}
"while"         {return WHILE;}
"with"          {return WITH;}
"yield"         {return YIELD;}
```

```
"_"            {return KATWPAVLA;}
"'"            {return QUOTE;}
"\""           {return DOUBLEQUOTE;}
"r"            {return r;}
"u"            {return u;}
"R"            {return R;}
"U"            {return U;}
"f"            {return f;}
"F"            {return F;}
"fr"           {return fr;}
"Fr"           {return Fr;}
"fR"           {return fR;}
"FR"           {return FR;}
"rf"           {return rf;}
"Rr"           {return Rf;}
"rF"           {return rF;}
"RF"           {return RF;}
"shortstring"   {return SHORTSTRING;}
"longstring"    {return LONGSTRING;}
"("            {return LEFTBRACKET;}
")"            {return RIGHTBRACKET;}
"["            {return LSQUAREBRACKET;}
"]"            {return RSQUAREBRACKET;}
"{"            {return CURLLEFT;}
"}"            {return CURLRIGHT;}
"+"            {return PLUS;}
"*"            {return MUL;}
"/"            {return DIVISION;}
"-"            {return MINUS;}
"="            {return ISON;}
","            {return KOMMA;}
":"            {return ANWKATWTELEIA;}
"print"        {return PRINT;}
"+="           {return SINISON;}
"-="           {return PLINISON;}
"."            {return TELEIA;}
"as"           {return AS;}
"init"         {return INIT;}
">"            {return MEGALUTERO;}
"<"            {return MIKROTERO;}
"diaforo"      {return DIAFORO;}


\n             {}
```

```
{integer}+                  {return INTEGER;}
[_a-zA-z0-9_]*              {return IDENTIFIER;}
[-+]?([0-9]*\.[0-9]+|[0-9]+)  {return FLOATNUMBER;}
[-+]?([0-9][i,j]*)          {return IMAGINUMBER;}
.;
%%
```

- **Input:**
  programm
  u shortstring

r longstring
5.0
5i
( )
(sas)
(x for y in w for a in k)

[ ]
[sdadadsd]
[x + y for y in w for a in k]
{ }
{ ADSDASD }
{ x + a for y in w for a in k }
dict = { ford : mustang , ferrari : enzo , ned : stark , thor : legacy , imagine : stars }
#randomsxolia

class ceid:
klmn = 5
def adfd (self , a , s) :
self . first = a
self . second = s
p1 = Person (John , 36)
def myfunction (fname) :
print (" sasa ")
if b > a : print (" b greater than a ")
for x in fruits: if b > a : print (" b greater than a ")