ΑΡΧΕΣ ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ

Συμμετέχοντες

Άγγελος Μαργκάς 1059684
Ιάσων-Γεώργιος Παυλάκης 1059688
Κωνσταντίνος Τσάκωνας 1059666
Ιωάννης Χριστοδουλάκος 1062664

Πίνακας περιεχομένων

1.

Περιγραφή του υποσυνόλου της γλώσσας Python σε BNF

```
<input> ::=
          <newlines>
          | <statements>

<suite>::=
            <stmt_list> NEWLINE
   |      NEWLINE INDENT <statements> DEDENT

<statement>::=
   <stmt_list> NEWLINE
   | <compound_stmt>

<statements>::=
   <statement>
   | <statements> <statement>

<stmt_list>::=
   <simple_stmt>
   | <simple_stmt> ';'
   | <simple_stmt> <simple_stmts>
   | <simple_stmt> <simple_stmts> ';'

<simple_stmts>::=
   ';' <simple_stmt>
   | <simple_stmts> ';' <simple_stmt>

<newlines>::=
          NEWLINE
          | <newlines> NEWLINE

<simple_stmt>::=
          <expression_stmt>
          | <assignment_stmt>
          | <print_stmt>
          | <return_stmt>
          | <break_stmt>
          | <import_stmt>

<expression_stmt>::=
          <expression_list>
```

```
<assignment_stmt>::=
<assignment_stmt_targer_list> <expression_list>

<assignment_stmt_targer_list>::=
   <target_list> '='
   | <assignment_stmt_targer_list> <target_list> '='


<print_stmt>::=
PRINT
| PRINT <expression>
| PRINT <expression> ','
| PRINT <expression> <expressions>
| PRINT <expression> <expressions> ','
| PRINT RIGHT_OP <expression>
| PRINT RIGHT_OP <expression> <expressions>
| PRINT RIGHT_OP <expression> <expressions> ','


<return_stmt>::=
   RETURN
   | RETURN <expression_list>

<break_stmt>::=
   BREAK

<compound_stmt>::=
   <if_stmt>
   | <for_stmt>
   | <funcdef>
   | <classdef>

<if_stmt>::=
   IF <expression> ':' <suite>
   | IF <expression> ':' <suite> ELSE ':' <suite>
   | IF <expression> ':' <suite> <elif_stmt>
   | IF <expression> ':' <suite> <elif_stmt> ELSE ':' <suite>

<elif_stmt>::=
         ELIF <expression> ':' <suite>
   <elif_stmt> ELIF <expression> ':' <suite>

<for_stmt>:
   FOR <target_list> IN <expression_list> ':' <suite>
   | FOR <target_list> IN <expression_list> ':' <suite> ELSE ':' <suite>
```

```
<funcdef>::=
        DEF <funcname> '(' ')' ':' <suite>
        | <decorators> DEF <funcname> '(' ')' ':' <suite>
        | DEF <funcname> '(' <parameter_list> ')' ':' <suite>
        | <decorators> DEF <funcname> '(' <parameter_list> ')' ':' <suite>

<decorators>::=
   <decorator>
   | <decorators> <decorator>

<decorator>::=
   '@' <dotted_name> NEWLINE
   | '@' <dotted_name> '(' ')' NEWLINE
   | '@' <dotted_name> '(' <argument_list> ')' NEWLINE
   | '@' <dotted_name> '(' <argument_list> ',' ')' NEWLINE

<dotted_name>::=
   <identifier>
   | <identifier> <dot_identifiers>

<dot_identifiers>::=
   '.' <identifier>
   | <dot_identifiers> '.' <identifier>

<parameter_list>::=
   STAR <identifier>
   | STAR <identifier> ',' DOUBLESTAR <identifier>
   | DOUBLESTAR <identifier>
   | <defparameter>
   | <defparameter>','
   | <defparameters> STAR <identifier>
   | <defparameters> STAR <identifier> ',' DOUBLESTAR <identifier>
   | <defparameters> DOUBLESTAR <identifier>
   | <defparameters> defparameter
   | <defparameters> def<parameter >','

<defparameter>::=
   <parameter>
   | <parameter >'=' <expression>

<defparameters>::=
   defparameter >','
   | <defparameters> <defparameter >','

<sublist>::=
   <parameter>
   | <parameter >','
   | <parameter ><parameters>
   | <parameter ><parameters> ','
```

```
<parameter>::=
   <identifier>
   | '(' <sublist> ')'

<parameters>::=
   ',' <parameter >
   | <parameters> ',' <parameter>

<funcname>::=
   identifier

<classdef>::=
   CLASS <classname> ':' <suite>
   | CLASS <classname> <inheritance> ':' <suite>

<inheritance>::=
   '(' ')'
   | '(' <expression_list> ')'

<classname>::=
   <identifier>

<suite>::=
   <stmt_list> NEWLINE
   | NEWLINE INDENT <statements> DEDENT



<import_stmt>::=
   IMPORT< module >
   | IMPORT< module >AS <name>
   | IMPORT< module ><modules>
   | IMPORT< module >AS <name> modules
   | FROM <relative_module> IMPORT <identifier>
   | FROM <relative_module> IMPORT <identifier> AS <name>
   | FROM <relative_module> IMPORT <identifier> <import_stmt_identifiers>
   | FROM <relative_module> IMPORT <identifier> AS <name> <import_stmt_identifiers>
   | FROM <relative_module> IMPORT '(' <identifier> ')'
   | FROM <relative_module> IMPORT '(' <identifier> AS <name> ')'
   | FROM <relative_module> IMPORT '(' <identifier> <import_stmt_identifiers> ')'
   | FROM <relative_module> IMPORT '(' <identifier> AS <name> <import_stmt_identifiers> ')'
   | FROM <relative_module> IMPORT '(' <identifier> ',' ')'
   | FROM <relative_module> IMPORT '(' <identifier> AS <name> ',' ')'
   | FROM <relative_module> IMPORT '(' <identifier> <import_stmt_identifiers> ',' ')'
   | FROM <relative_module> IMPORT '(' <identifier> AS <name> <import_stmt_identifiers> ',' ')'
   | FROM< module >IMPORT STAR

<module>::=
   <identifier>
   |< module >'.' <module>

<relative_module>::=
   <module>
   | <dot_modules>
   | <dots>

<dot_modules>::=
   '.' <module>
   | <dot_modules> '.' module

<dots>::=
   '.'
   | <dots> '.'

<modules>::=
   ',' <module>
   | ','< module >AS <name>
   | <modules> ',' <module>
   | <modules> ','< module >AS <name>

<import_stmt_identifiers>::=
   ',' <identifier>
   | ',' <identifier> AS <name>
   | <import_stmt_identifiers> ',' <identifier>
   | <import_stmt_identifiers> ',' <identifier> AS <name>

<name>::=
   <identifier>


<primary>::=
                  <atom>
                  | <attributeref>
                  | <call>

<call>::=
                  <primary> '(' ')'
                  | <primary> '(' <argument_list> ')'
                  | <primary> '(' <argument_list> ',' ')'


<argument_list>::=
                  <positional_arguments>
                  | <positional_arguments> ',' <keyword_arguments>
                  | <positional_arguments> ',' STAR <expression>
                  | <positional_arguments> ',' DOUBLESTAR <expression>
                  | <positional_arguments> ',' <keyword_arguments> ',' STAR <expression>
                  | <positional_arguments> ',' <keyword_arguments> ',' DOUBLESTAR <expression>
```

```
                    | <positional_arguments> ',' STAR <expression> ',' DOUBLESTAR <expression>
                    | <positional_arguments> ',' <keyword_arguments> ',' STAR <expression> ',' DOUBLESTAR
<expression>
                    | <keyword_arguments>
                    | <keyword_arguments> ',' STAR <expression>
                    | <keyword_arguments> ',' DOUBLESTAR <expression>
                    | <keyword_arguments> ',' STAR <expression> ',' DOUBLESTAR <expression>
                    | STAR <expression>
                    | STAR <expression> ',' DOUBLESTAR <expression>
                    | DOUBLESTAR <expression>

<positional_arguments>::=
   <expression>
   | <expression> <expressions>

<keyword_arguments>::=
   <keyword_item>
   | <keyword_item> <keyword_items>

<keyword_item>::=
   <identifier> '=' <expression>

<keyword_items>::=
   ',' <keyword_item>
   | <keyword_items> ',' <keyword_item>


<expression_list>::=
   <expression>
   | <expression> ','
   | <expression> <expressions>
   | <expression> <expressions> ','

<expressions>::=
   ',' <expression>
   | <expressions> ',' <expression>


<expression>::=
   <conditional_expression>
   | <lambda_form>



<conditional_expression>::=
                    <or_test>
                    | <or_test> IF <or_test> ELSE expression

<power>::=
   <primary>
```

```
    | DOUBLESTAR <u_expr>



<u_expr>::=
    power
    | '-' <u_expr>
    | '+' <u_expr>
    | '~' <u_expr>

<m_expr>::=
    <u_expr>
    | <m_expr> STAR <u_expr>
    | <m_expr> DOUBLESLASH <u_expr>
    | <m_expr> SLASH <u_expr>
    | <m_expr> '%' <u_expr>

<a_expr>::=
    <m_expr>
    | <a_expr> '+' <m_expr>
    | <a_expr> '-' <m_expr>

<shift_expr>::=
    <a_expr>
    | <shift_expr> RIGHT_OP <a_expr>
    | <shift_expr> LEFT_OP <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>
    | <comparison_operators_or_exprs>

<comparison_operators_or_exprs>::=
    <comp_operator>< or_expr>
    | <comparison_operators_or_exprs> <comp_operator> < or_expr>

<comp_operator>::=
    "<" | ">" | "==" | ">=" | "<=" | "<>" | "!="
    | IS | IS NOT | IN | NOT IN
```

```
<target_list>::=
   <target>
   | <target_list> ',' <target>
   | <target_list> ','

<target>::=
   <identifier>
   | '(' <target_list> ')'
   | '[' <target_list> ']'
   | <attributeref>

<attributeref>::=
                  <primary> '.' <identifier>

<atom>::=
                  <identifier>
                  | <literal>
                  | <enclosure>


<enclosure>::=
                  <parenth_form>
                  | <dict_display>

<parenth_form>::=
   '(' ')'
   | '(' <expression_list> ')'


<dict_display>::=
   '{' '}'
   | '{' <key_datum_list> '}'


<key_datum_list>::=
   <key_datum>
   | <key_datum >','
   | <key_datum ><key_datums>
   | <key_datum ><key_datums> ','


<key_datums>::=
   ',' <key_datum>
   | <key_datums> ',' <key_datum>


<key_datum >::=
   <expression >':' <expression>
```

```
<identifier>::=
   IDENTIFIER

<stringliteral>::=
   SHORTSTRING | LONGSTRING

<longinteger>::=
   <integer> 'l' | <integer> 'L'

<integer>::=
   DECINTEGER | OCTINTEGER | HEXINTEGER

<floatnumber>::=
   POINTFLOAT | EXPONENTFLOAT

<imagnumber>::=
   IMAGNUMBER
```

2. Περιγραφή της υλοποιημένης γλώσσας σε BNF

```
<program> ::=

        //empty

        | <statement_list>

<statement_list>  ::=

        <statement_list> <statement>

        | <statement>

<statement> ::=

        <import_stmt>

        | <assignment_stmt>

        | <if_stmt>

        | <for_stmt>

        | <print_stmt>

        | <funcdef>

        | <classdef>

        | <call>

        | <return_stmt>

        | <lambda_form

        | <dict_setdefault>

        |  <dict_items>
```

```
<return_stmt> ::=
                RETURN
                | RETURN <expression_list>

<call> ::=
        <primary> LPAR RPAR
        | <primary> LPAR <expression_list> RPAR
        | <identifier> EQUAL <primary> LPAR  RPAR
        | <identifier> EQUAL <primary> LPAR <expression_list> RPAR

<primary> ::=
        <identifier>
        |<attr_identifier>



<lambda_form> ::=
        LAMBDA COLON <expression>
        | LAMBDA <parameter_list> COLON <expression>


<print_stmt> ::=
        PRINT
        | PRINT <expression>
        | PRINT <expression_list>
        | PRINT RIGHT_OP <expression>
        | PRINT RIGHT_OP <expression_list>
        | PRINT LPAR <call> RPAR
```

```
<assignment_stmt> ::=
        <assignment_stmt_targer_list> <expression_list>
        |<assignment_stmt_targer_list> <call>

<assignment_stmt_targer_list> ::=
        <target_list> EQUAL
        | <assignment_stmt_targer_list> <target_list> EQUAL

<target_list> ::=
        <target>
        | <target_list> COMMA <target>
        | <target_list> COMMA

<target> ::=
        <identifier>
        |<attr_identifier>
        |LPAR <target>_list> RPAR

<import_stmt> ::=
        IMPORT <module>
        | IMPORT <module> AS <name>
        | IMPORT <modules> <modules>
        | IMPORT <modules> AS <name> <modules>
        | FROM <relative_module> IMPORT <identifier>
```

```
    | FROM <relative_module> IMPORT <identifier> AS <name>

    | FROM <relative_module> IMPORT <identifier> <import_stmt_identifiers>

    | FROM <relative_module> IMPORT <identifier> AS <name> <import_stmt_identifiers>

    | FROM <relative_module> IMPORT LPAR <identifier> RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> AS <name> RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> <import_stmt_identifiers> RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> AS <name> <import_stmt_identifiers> RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> COMMA RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> AS <name> COMMA RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> <import_stmt_identifiers> COMMA RPAR

    | FROM <relative_module> IMPORT LPAR <identifier> AS <name> <import_stmt_identifiers> COMMA RPAR

    | FROM <relative_module> IMPORT STAR


<module> ::=

    <module> DOT <identifier>

    | <identifier>

<relative_module> ::=

    <module>

    | <dots> <module>

    | <dots>


<dots> ::=

    DOT

    | <dots> DOT

<modules> ::=

    <modules> COMMA <module>
```

```
        | <modules> COMMA <module> AS <name>

        | COMMA<module>

        | COMMA<module> AS <name>

<import_stmt_identifiers> ::=

        COMMA <identifier>

        | COMMA <identifier> AS <name>

        | <import_stmt_identifiers> COMMA <identifier>

        | <import_stmt_identifiers> COMMA <identifier> AS <name>

<name> ::=

         <identifier>

<if_stmt> ::=

        IF <expression> COLON <statement_list>

        | IF <expression> COLON <statement_list> ELSE COLON<statement_list>

        | IF <expression> COLON <statement_list> <elif_stmt>

        | IF <expression> COLON <statement_list> <elif_stmt> ELSE COLON <statement_list>

<elif_stmt> ::=

        ELIF <expression> COLON  <statement_list>

        | elif_stmt ELIF <expression> COLON  <statement_list>

<for_stmt> ::=

        FOR <for_target_list> IN <expression_list> COLON <statement_list>
        |FOR <for_target_list> IN RANGE LPAR <expression_list> RPAR COLON  <statement_list>

        | FOR <for_target_list> IN <expression_list> COLON <statement_list> ELSE COLON <statement_list>


<for_target_list> ::=

        <for_target>

        | <for_target_list> COMMA <target>

        | <for_target_list> COMMA


<for_target> ::=

        <identifier>

        |LPAR <for_target_list> RPAR



<funcdef> ::=

        DEF <funcname> LPAR RPAR COLON <statement_list>

        |<decorators> DEF <funcname> LPAR RPAR COLON <statement_list>

        | DEF <funcname> LPAR <parameter>_list RPAR COLON <statement_list>

        | <decorators> DEF <funcname> LPAR <parameter_list> RPAR COLON <statement_list>


<decorators> ::=

        <decorator>

        | <decorators> <decorator>

<decorator> ::=

        PAPAKI <dotted_name> NEWLINE

        | PAPAKI <dotted_name> LPAR RPAR NEWLINE


<dotted_name> ::=
        <identifier>

        | <identifier> <dot_identifiers>


<dot_identifiers> ::=

        DOT <identifier>

        | <dot_identifier> DOT <identifier>


<parameter_list> ::=

        STAR <identifier>

        | STAR <identifier> COMMA DOUBLESTAR <identifier>

        | DOUBLESTAR <identifier>

        | <defparameter>

        | <defparameter> COMMA

        | <defparameters> STAR <identifier>

        | <defparameters> STAR <identifier> COMMA DOUBLESTAR <identifier>

        | <defparameters> DOUBLESTAR <identifier>

        | <defparameters> <defparameter>

        | <defparameters> <defparameter> COMMA

<defparameter> ::=

        <parameter>

        | <parameter> EQUAL expression

<defparameters> ::=
        <defparameter> COMMA
        | <defparameters> <defparameter> COMMA

<sublist> ::=

        <parameter>

        | <parameter> COMMA
        | <parameter> <parameters>

        | <parameter> <parameters> COMMA

<parameters> ::=

        COMMA <parameter>

        | <parameters> COMMA <parameter

<parameter> ::=

        <identifier>

        | LPAR <sublist> RPAR


<funcname> ::=

        <identifier>

<classdef> ::=

        CLASS <classname> COLON <statement_list>

        | CLASS classname> <inheritance COLON> <statement_list>

<inheritance> ::=

        LPAR RPAR

        | LPAR <expression_list> RPAR

<classname> ::=

        <identifier>

//.................................etc .....................................................................

<dict_items> ::=

        <identifier> DOT ITEMS LPAR RPAR
```

```
<dict_setdefault> ::=

        <identifier> DOT SETDEFAULT LPAR <expression> COMMA <expression> RPAR

<dict_display> ::=

        LBRA RBRA

        | LBRA <key_datum_list> RBRA

<key_datum_list> ::=

        <key_datum>

        | <key_datum > COMMA

        | <key_datum > <key_datums>

        | <key_datum > <key_datums> COMMA

<key_datums> ::=

        COMMA <key_datum>

        | <key_datums> COMMA <key_datum >

<key_datum> ::=

                <expression> COLON <expression>


<expression_list> ::=

        <expression_list> COMMA expression

        |LPAR <expression_list> COMMA <expression> RPAR

        |<expression>


<expression > ::=
        <atom>
        | LPAR <expression> RPAR
        | <expression> PLUS <expression>
        | <expression> MINUS <expression>
        | <expression> SLASH <expression>
        | <expression> STAR <expression>
        | <expression> <assignment_op> <expression>
        | <expression> <arithmetic_op> <expression>
        | <expression> <comparison_op> <expression>
        | <expression> logical_op> <expression>
        | <expression> <bitwise_op> <expression>
<atom> ::=
        <literal>
        | <identifier>
        | <integer>
        | <attr_identifier>
        | <dict_display>
        | <dict_setdefault>


<literal> ::=
         <string>
        | <longinteger>
        | <imagnumber>

<attr_identifier> ::=
        <identifier>
```

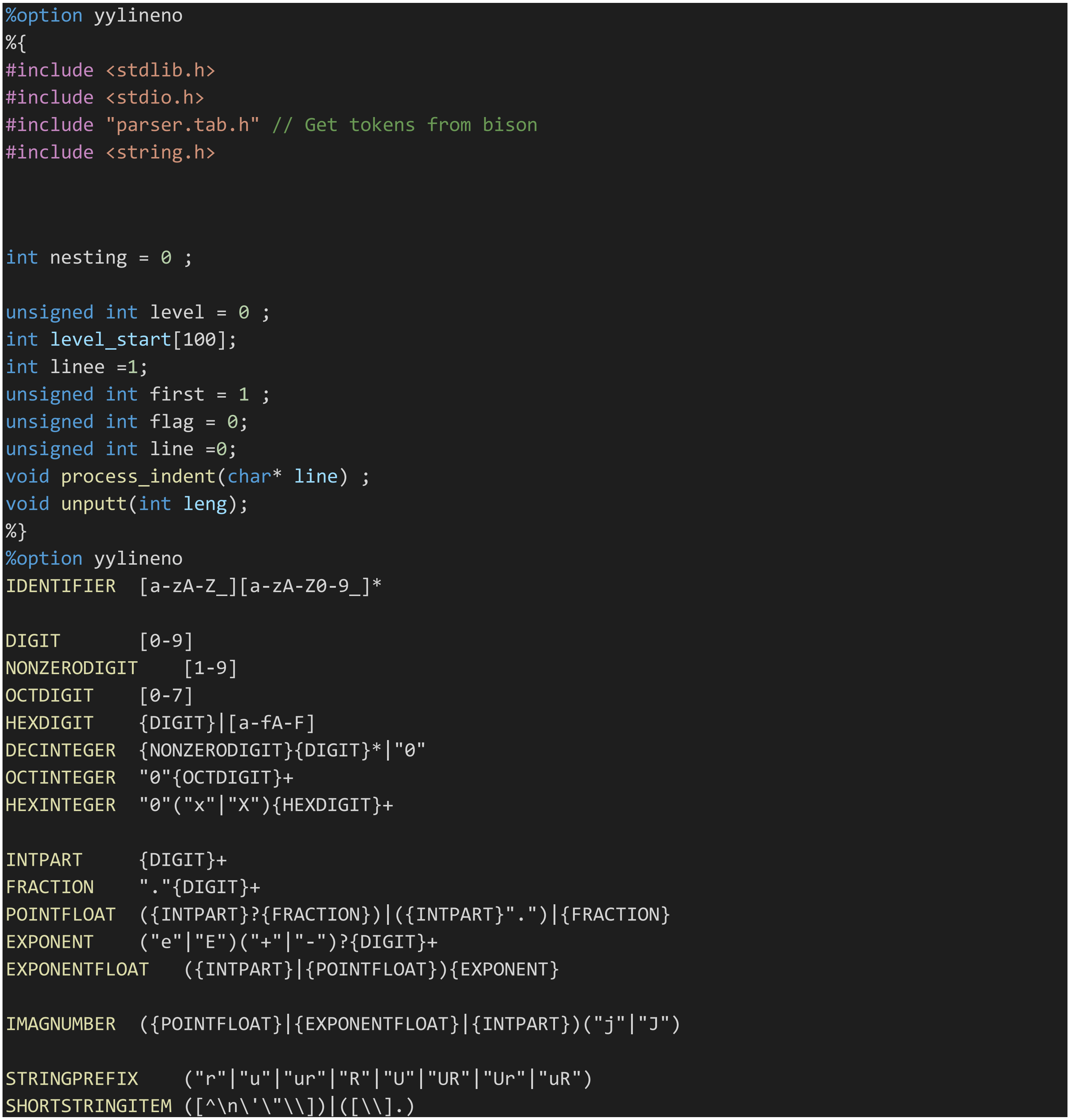
```
            | attr_<identifier> DOT <identifier>

            | <identifier> DOT <identifier>



<stringliteral> ::=
            <shortstring>
            |<longstring>

<shortstring>::=
             <any source character except "" or newline>

longstringitem ::=
             <any source character except '\' '>

imagnumber ::= (floatnumber | intpart) ("j" | "J")

longinteger ::=
             integer ("l" | "L")

integer ::=
             decimalinteger | octinteger | hexinteger

decimalinteger ::=
             nonzerodigit digit* | "0"

octinteger ::=
             "0" octdigit+

hexinteger ::=
             "0" ("x" | "X") hexdigit+

nonzerodigit ::=
              "1"..."9"

octdigit ::=
             "0"..."7"

hexdigit ::=
             digit | "a"..."f" | "A"..."F"


floatnumber ::=
             pointfloat | exponentfloat

pointfloat ::=
             [intpart] fraction | intpart "."
```

```
exponentfloat ::=
            (intpart | pointfloat)
             exponent

intpart ::=
            digit+

fraction ::=
            "." digit+

exponent ::=
            ("e" | "E") ["+" | "-"] digit+

identifier ::=
            (letter|"_") (letter | digit | "_")*

    (where the matched string is not a keyword)

letter ::=
            lowercase | uppercase

lowercase ::=
            "a"|"b"|...|"z"

uppercase ::=
            "A"|"B"|...|"Z"

digit ::=
            "0"|"1"|...|"9"

<assignment_op> ::=

            ADD_ASSIGN
           | SUB_ASSIGN
           | MUL_ASSIGN
           | POW_ASSIGN
           | DIV_ASSIGN
           | MOD_ASSIGN
           | AND_ASSIGN
           | XOR_ASSIGN
           | OR_ASSIGN
           | RIGHT_ASSIGN
           | LEFT_ASSIGN

<arithmetic_op> ::=
           PERCENT
           | DOUBLESTAR
           | DOUBLESLASH

<comparison_op> ::=
           EQ_OP
           | NE_OP
           | GREATER_THAN_OP
           | LESS_THAN_OP
           | LE_OP
           | GE_OP

<logical_op> ::=
           AND
           | NOT
           | OR
           | IS
           | IN
           | IS NOT
           | NOT IN

<bitwise_op> ::=
           AND_EXP
           | OR_SIGN
           | XOR
           | NOT_SIGN
           | LEFT_OP
           | RIGHT_OP
```

3.

Τελικά αρχεία Flex και Bison

Scan.l :

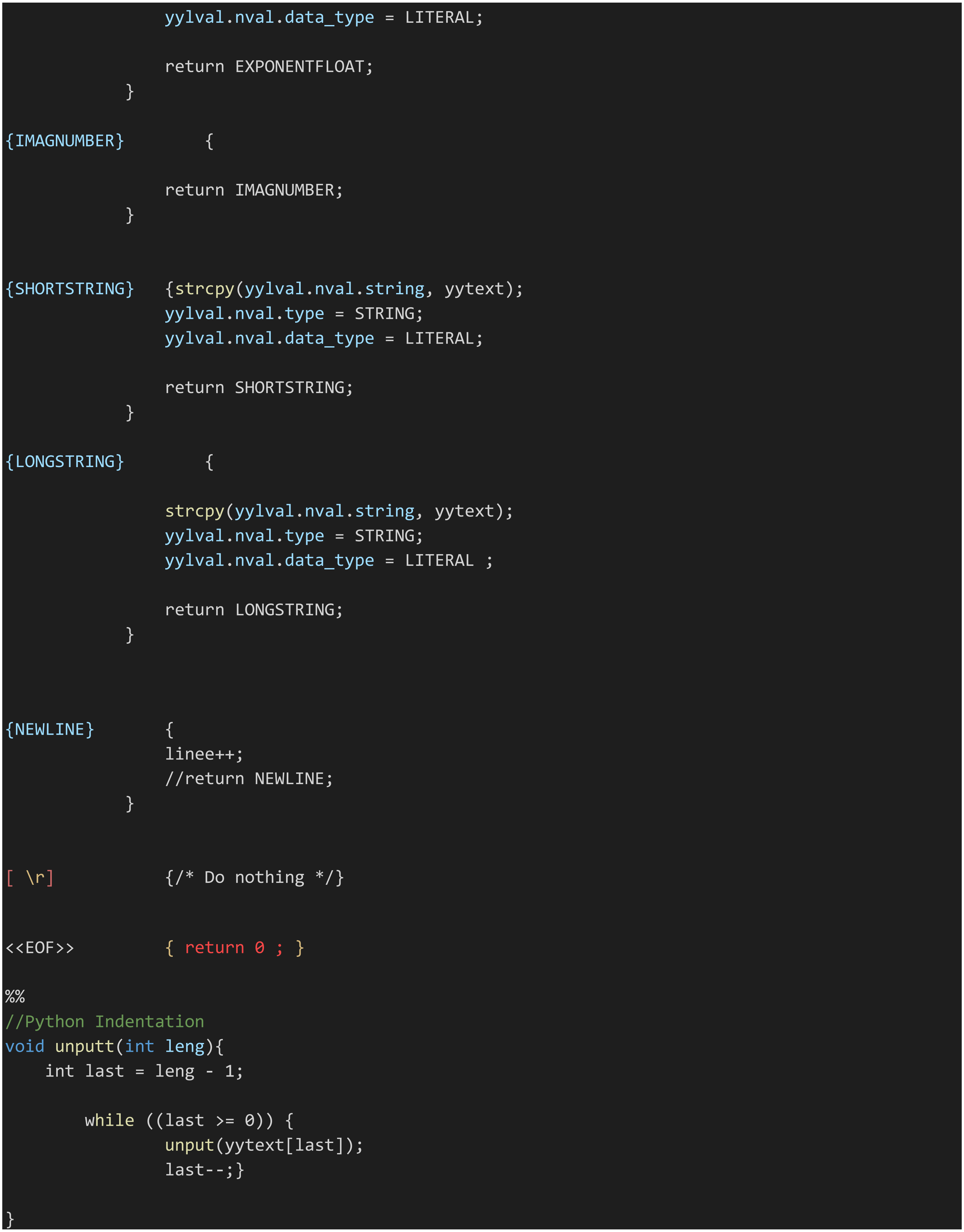


```
%option yylineno
%{
#include <stdlib.h>
#include <stdio.h>
#include "parser.tab.h" // Get tokens from bison
#include <string.h>



int nesting = 0 ;

unsigned int level = 0 ;
int level_start[100];
int linee =1;
unsigned int first = 1 ;
unsigned int flag = 0;
unsigned int line =0;
void process_indent(char* line) ;
void unputt(int leng);
%}
%option yylineno
IDENTIFIER  [a-zA-Z_][a-zA-Z0-9_]*

DIGIT       [0-9]
NONZERODIGIT    [1-9]
OCTDIGIT    [0-7]
HEXDIGIT    {DIGIT}|[a-fA-F]
DECINTEGER  {NONZERODIGIT}{DIGIT}*|"0"
OCTINTEGER  "0"{OCTDIGIT}+
HEXINTEGER  "0"("x"|"X"){HEXDIGIT}+

INTPART     {DIGIT}+
FRACTION    "."{DIGIT}+
POINTFLOAT  ({INTPART}?{FRACTION})|({INTPART}".")|{FRACTION}
EXPONENT    ("e"|"E")("+"|"-")?{DIGIT}+
EXPONENTFLOAT   ({INTPART}|{POINTFLOAT}){EXPONENT}

IMAGNUMBER  ({POINTFLOAT}|{EXPONENTFLOAT}|{INTPART})("j"|"J")

STRINGPREFIX    ("r"|"u"|"ur"|"R"|"U"|"UR"|"Ur"|"uR")
SHORTSTRINGITEM ([^\n\'\"\\])|([\\].)
```

```
LONGSTRINGITEM  ([^\\])|([\\].)
SHORTSTRING {STRINGPREFIX}?([']{SHORTSTRINGITEM}*['])|(["]{SHORTSTRINGITEM}*["])
LONGSTRING  {STRINGPREFIX}?(([']{3}{LONGSTRINGITEM}*[']{3})|(["]{3}{LONGSTRINGITEM}*["]{3}))

NEWLINE     \n
WHITESPACE  [ \t\v\n\f]


%option noyywrap

%%


^[ ]*\n             {/* Ignore blank lines. */ linee++ ;}
^[\t]*\n            {/* Ignore blank lines. */ ;}
^.+                 {process_indent(yytext); unputt(yyleng);/*Reads every line*/}
[#].*           { /*Ignore comments*/}


"if"            {level++ ; flag = 1;  return IF;}
"for"           {level++ ; flag = 1;  return FOR;}
"def"           {level++ ; flag = 1;  return DEF;}
"class"         {level++ ; flag = 1;  return CLASS;}
"elif"          {level++ ; flag = 1;  return ELIF;}
"else"          {level++ ; flag = 1;  return ELSE;}
"setdefault"    {return SETDEFAULT;}
"False"         {return FALSE;}
"None"          {return NONE;}
"True"          {return TRUE;}
"and"           {return AND;}
"as"            {return AS;}
"assert"        {return ASSERT;}
"break"         {return BREAK;}
"continue"      {return CONTINUE;}
"del"           {return DEL;}
"except"        {return EXCEPT;}
"finally"       {return FINALLY; }
"from"          {return FROM; }
"global"        {return GLOBAL;}
"import"        {return IMPORT;}
"in"            {return IN; }
"is"            {return IS; }
"lambda"        {return LAMBDA;}
"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;}
"range"         {return RANGE;}
"print"         {return PRINT;}
"exec"          {return EXEC;}
"items"         {return ITEMS;}
"L"         {return 'L';}
"l"         {return 'l';}

"++"                {return INC;}
"--"                {return DEC;}
"\'"            {return APOSTROPHE;}
"\""            {return QUOTATION;}
"..."           {return ELLIPSIS; }
">>="           {return RIGHT_ASSIGN; }
"<<="           {return LEFT_ASSIGN; }
"+="            {return ADD_ASSIGN; }
"-="            {return SUB_ASSIGN; }
"*="            {return MUL_ASSIGN; }
"**="           {return POW_ASSIGN; }
"/="            {return DIV_ASSIGN; }
"%="            {return MOD_ASSIGN; }
"&="            {return AND_ASSIGN; }
"^="            {return XOR_ASSIGN; }
"|="            {return OR_ASSIGN; }
">>"            {return RIGHT_OP; }
"<<"            {return LEFT_OP; }
"<="            {return LE_OP; }
">="            {return GE_OP; }
"=="            {return EQ_OP; }
"!="            {return NE_OP; }
"<>"            {return LR_OP;}
"!"         {return EXA;}
";"         {return COL;}
"_"         {return '_';}
","         {return COMMA;}
":"         {return COLON;}
"="         {return EQUAL;}
"("         {nesting++; return LPAR;}
")"         {nesting--; return RPAR;}
"["         {return '[';}
"]"         {return ']';}
"{"         {return LBRA;}
"}"         {return RBRA;}
"."         {return DOT;}
"&"         {return AND_EXP;}
"@"         {return PAPAKI;}
"~"         {return '~';}
"-"         {return MINUS;}
```

```
"+"         {return PLUS;}
"%"         {return PERCENT;}
"<"         {return LESS_THAN_OP;}
">"         {return GREATER_THAN_OP;}
"^"         {return XOR;}
"|"         {return OR_SIGN;}
"`"         {return '`'; }
"*"         {return STAR;}
"**"            {return DOUBLESTAR;}
"/"         {return SLASH;}
"//"            {return DOUBLESLASH;}

{IDENTIFIER}        {
                strcpy(yylval.nval.name, yytext);
                yylval.nval.type = IDENT;
                yylval.nval.data_type = LITERAL;
                return IDENTIFIER;
            }

{DECINTEGER}        {
                yylval.nval.ival = atoi(yytext);
                yylval.nval.type = INTEGER;
                yylval.nval.data_type = LITERAL;
                return DECINTEGER;
            }

{OCTINTEGER}        {
                yylval.nval.ival = atoi(yytext);
                yylval.nval.type = INTEGER;
                yylval.nval.data_type = LITERAL;
                return OCTINTEGER;
            }

{HEXINTEGER}        {
                yylval.nval.ival = atoi(yytext);
                yylval.nval.type = INTEGER;
                yylval.nval.data_type = LITERAL;
                return HEXINTEGER;
            }

{POINTFLOAT}        {   yylval.nval.fval = atof(yytext);
                yylval.nval.type = FLOAT;
                yylval.nval.data_type = LITERAL;

                return POINTFLOAT;
            }

{EXPONENTFLOAT}     {   yylval.nval.fval = atof(yytext);
                yylval.nval.type = FLOAT;
```

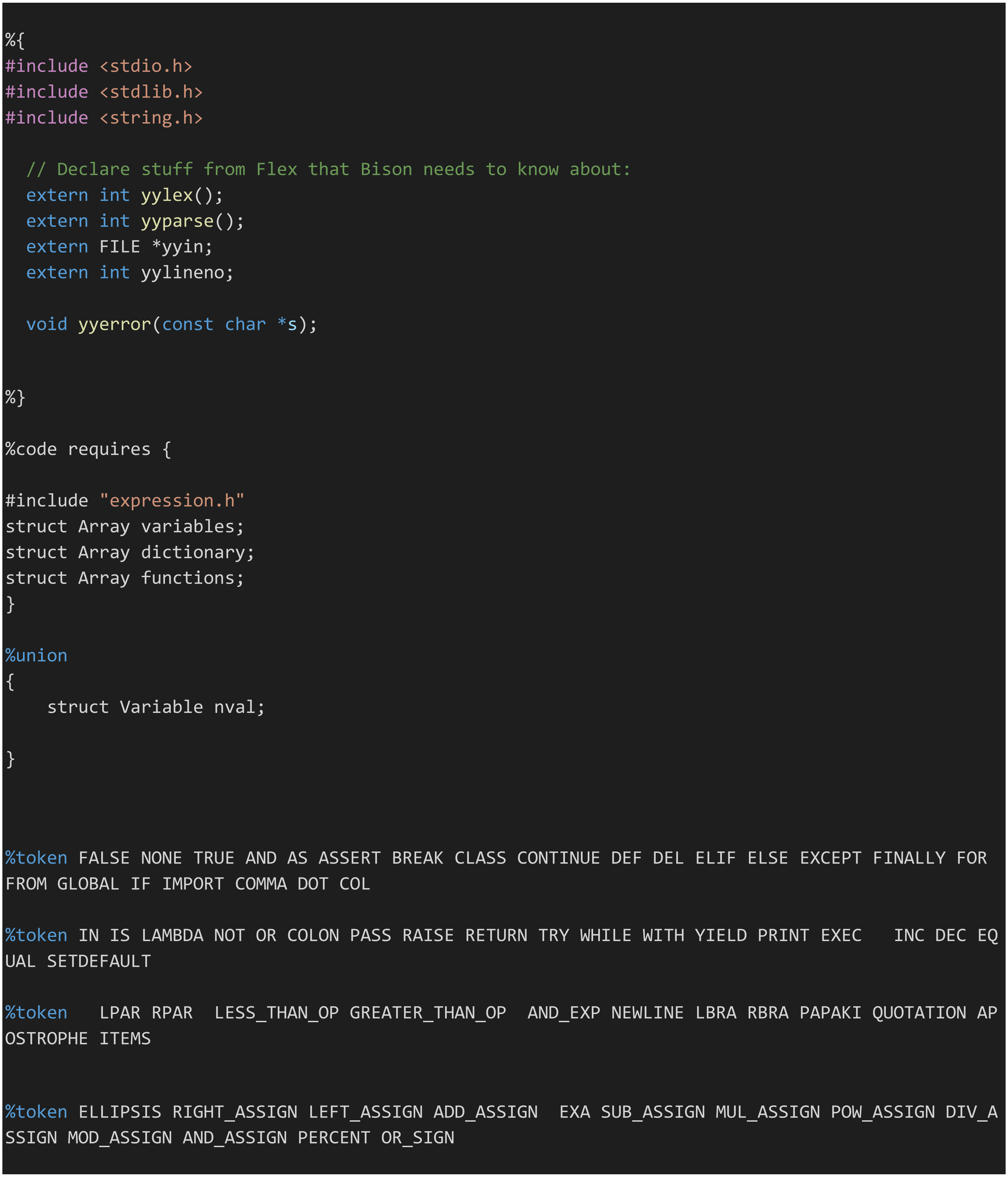


```
                yylval.nval.data_type = LITERAL;

                return EXPONENTFLOAT;
            }

{IMAGNUMBER}        {

                return IMAGNUMBER;
            }


{SHORTSTRING}   {strcpy(yylval.nval.string, yytext);
                yylval.nval.type = STRING;
                yylval.nval.data_type = LITERAL;

                return SHORTSTRING;
            }

{LONGSTRING}        {

                strcpy(yylval.nval.string, yytext);
                yylval.nval.type = STRING;
                yylval.nval.data_type = LITERAL ;

                return LONGSTRING;
            }



{NEWLINE}       {
                linee++;
                //return NEWLINE;
            }


[ \r]           {/* Do nothing */}


<<EOF>>         { return 0 ; }

%%
//Python Indentation
void unputt(int leng){
    int last = leng - 1;

        while ((last >= 0)) {
                unput(yytext[last]);
                last--;}

}
```

```
unsigned int white_count(char* line) {
  unsigned int count = 0 ;

  while (*line == ' ' || *line == '\t'){
    if(*line == ' ')
        count++;
    else
        count = count + 8;

    line++;

 }

   return count;
}

void process_indent(char* line) {


  if (nesting)
    /* Ignore indents while nested. */
    return ;

  unsigned int indent = white_count(line) ;


  if ((indent ==level_start[level] && !flag) ||(flag && indent >level_start[level-1])) {

    level_start[level] = indent;
    flag=0;
    return ;
  }
  else if(flag){
    printf("Line:%d --> Indentation error\n",yylineno);
    exit(1);
    return;}

   if (indent > level_start[level] && !flag) {

    printf("Line:%d --> Indentation error\n",yylineno);
    exit(1);
    return ;
  }

  int temp = level;
  while(indent != level_start[temp] && level >=0) {
    flag=0;
    temp--;
  }

  if(temp>=0)
    level = temp ;
  else{
    printf("Line:%d --> Indentation error\n",yylineno);
    exit(1);}
return;




}
```

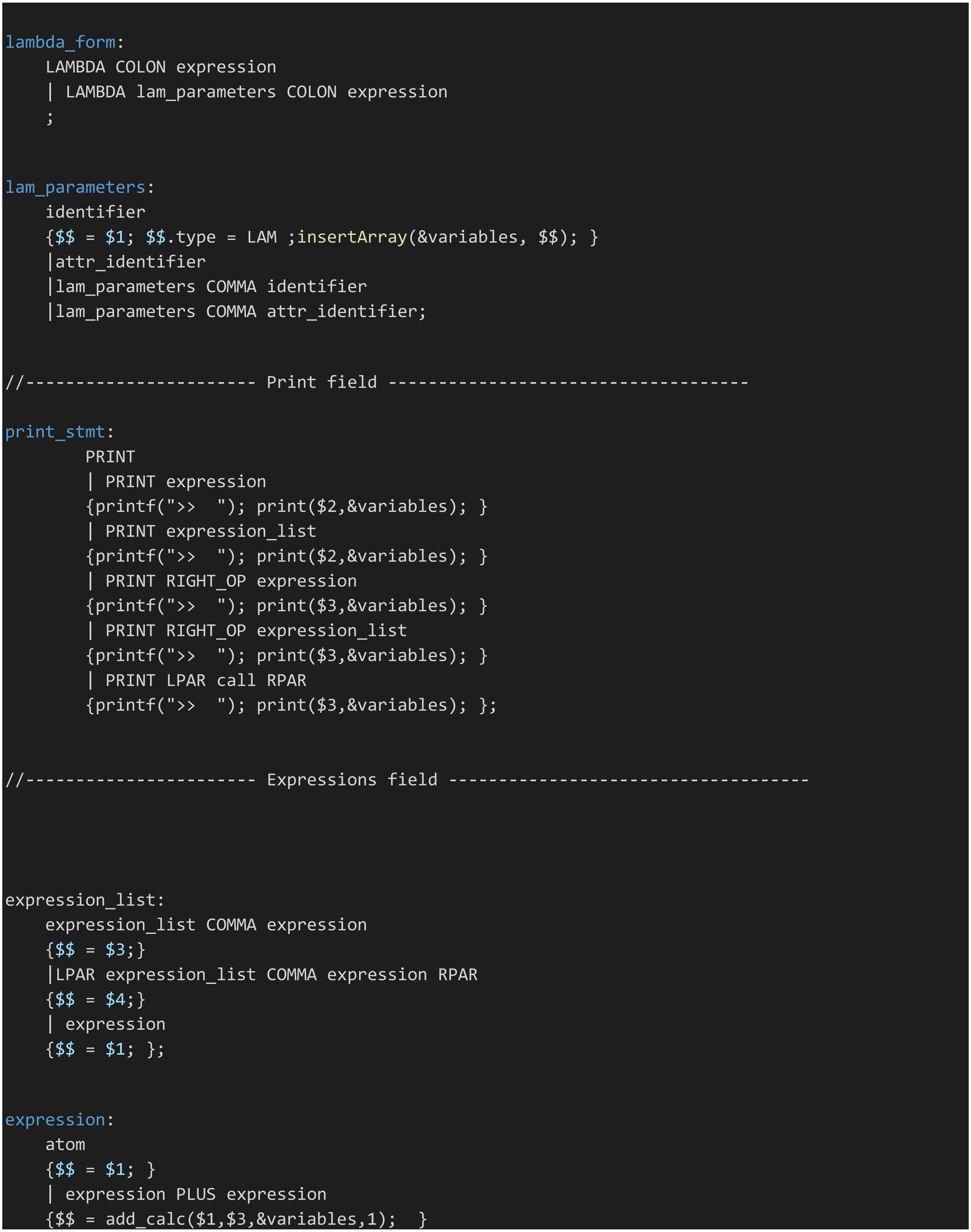
End of file scan.l

Parser.y:

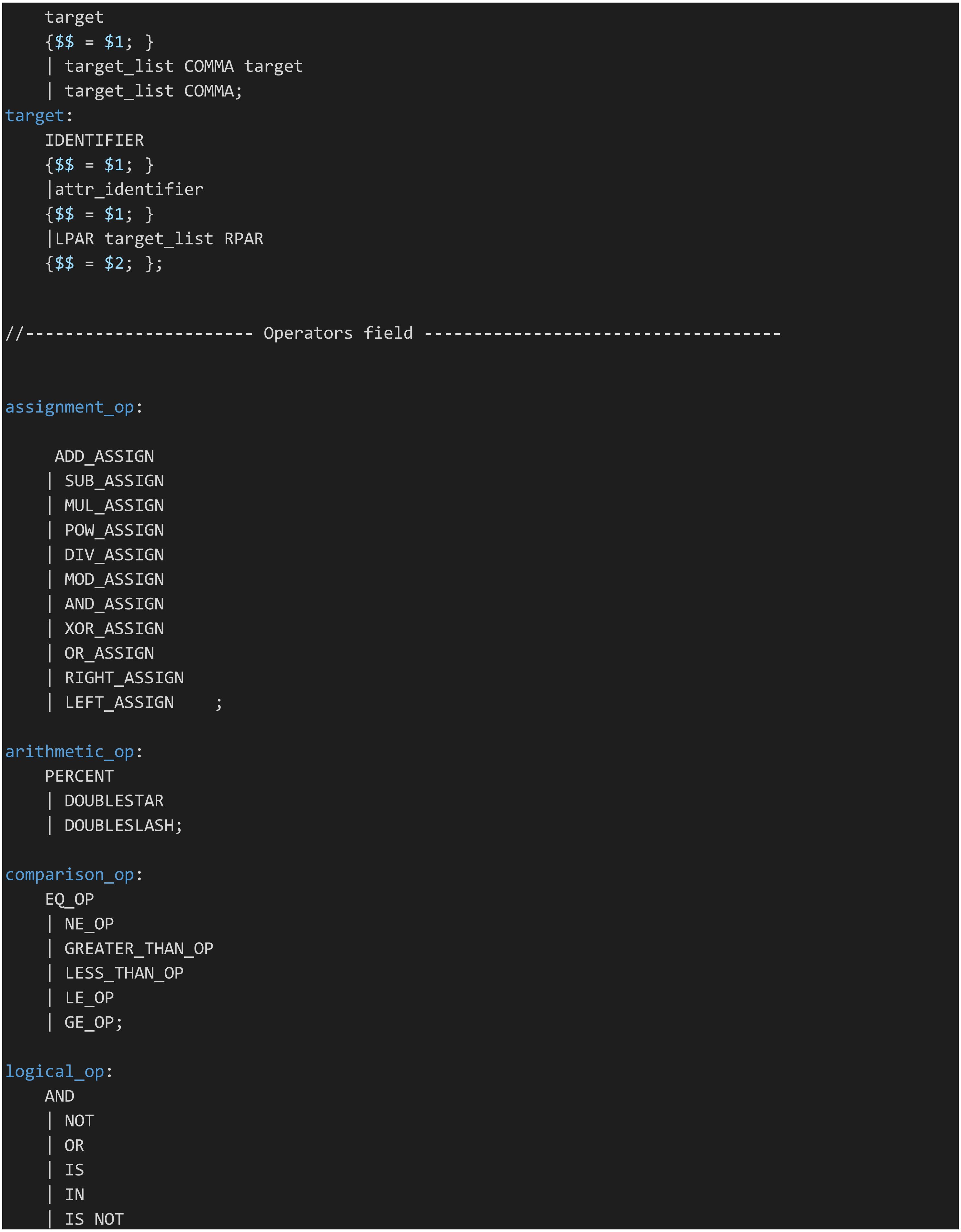


```
%{
#include <stdio.h>
#include <stdlib.h>
#include <string.h>

  // Declare stuff from Flex that Bison needs to know about:
  extern int yylex();
  extern int yyparse();
  extern FILE *yyin;
  extern int yylineno;

  void yyerror(const char *s);


%}

%code requires {

#include "expression.h"
struct Array variables;
struct Array dictionary;
struct Array functions;
}

%union
{
    struct Variable nval;

}



%token FALSE NONE TRUE AND AS ASSERT BREAK CLASS CONTINUE DEF DEL ELIF ELSE EXCEPT FINALLY FOR FROM GLOBAL IF IMPORT COMMA DOT COL

%token IN IS LAMBDA NOT OR COLON PASS RAISE RETURN TRY WHILE WITH YIELD PRINT EXEC   INC DEC EQUAL SETDEFAULT

%token   LPAR RPAR  LESS_THAN_OP GREATER_THAN_OP  AND_EXP NEWLINE LBRA RBRA PAPAKI QUOTATION APOSTROPHE ITEMS


%token ELLIPSIS RIGHT_ASSIGN LEFT_ASSIGN ADD_ASSIGN  EXA SUB_ASSIGN MUL_ASSIGN POW_ASSIGN DIV_ASSIGN MOD_ASSIGN AND_ASSIGN PERCENT OR_SIGN
```

```
%token XOR_ASSIGN OR_ASSIGN RIGHT_OP LEFT_OP PTR_OP LE_OP GE_OP EQ_OP NE_OP  DOUBLESTAR  DOUBLE
SLASH RANGE  LR_OP  XOR NOT_SIGN


%left  PLUS MINUS
%left  STAR SLASH


%token<nval>    DECINTEGER
%token<nval>    OCTINTEGER
%token<nval>    HEXINTEGER
%token<nval>    POINTFLOAT
%token<nval>    EXPONENTFLOAT
%token<nval>    IDENTIFIER
%token<nval>    SHORTSTRING
%token<nval>    LONGSTRING
%token<nval>    IMAGNUMBER

%type<nval>     imagnumber
%type<nval>     identifier
%type<nval>     integer
%type<nval>     stringliteral
%type<nval>     floatnumber
%type<nval>     literal
%type<nval>     atom
%type<nval>     expression
%type<nval>     print_stmt
%type<nval>     target
%type<nval>     target_list
%type<nval>     assignment_stmt
%type<nval>     assignment_stmt_targer_list
%type<nval>     expression_list
%type<nval>     attr_identifier
%type<nval>     longinteger
%type<nval>     call
%type<nval>     primary
%type<nval>     dict_display
%type<nval>     dict_setdefault

%type<nval>     lam_parameters
%type<nval>     lambda_form
%type<nval>     funcname


%%


program:
    //empty
    {printf("Success! You are awesome. \n");}
    | statement_list
    {printf("Success! You are awesome. \n");}
    ;

statement_list :
    statement_list statement
    | statement;

statement:
    import_stmt
    | assignment_stmt
    | if_stmt
    | for_stmt
    | print_stmt
    | funcdef
    | classdef
    | call
    | return_stmt
    | break_stmt
    | lambda_form
    | dict_setdefault
    | dict_items
    ;

break_stmt:
    BREAK;

return_stmt:
        RETURN
        | RETURN expression_list;
call:
    primary LPAR RPAR

    | primary LPAR expression_list RPAR

    | identifier EQUAL primary LPAR  RPAR

    | identifier EQUAL primary LPAR expression_list RPAR



primary:
    identifier
    {$$ = $1;}
    |attr_identifier
    {$$ = $1;}
    ;
```

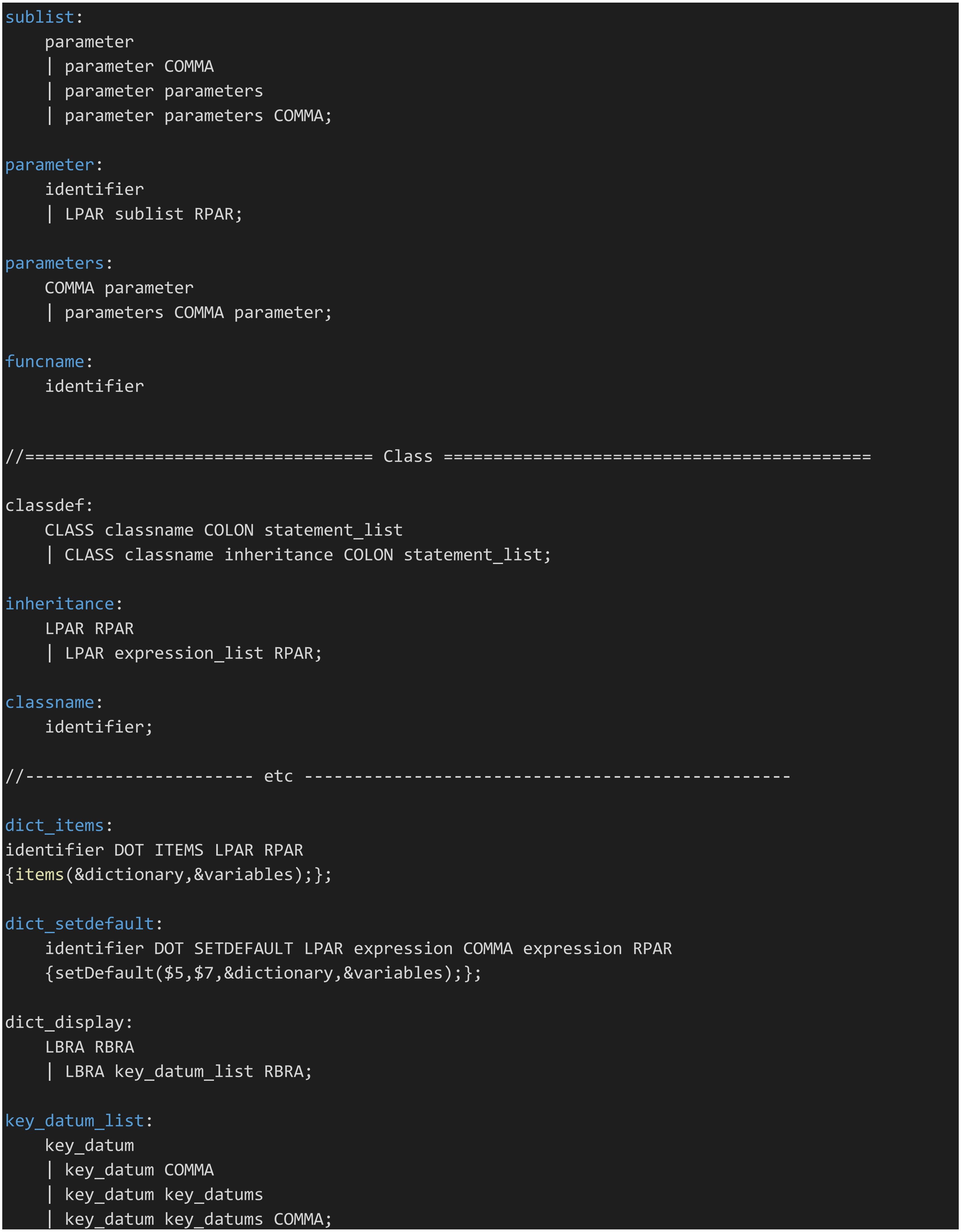


```
lambda_form:
    LAMBDA COLON expression
    | LAMBDA lam_parameters COLON expression
    ;


lam_parameters:
    identifier
    {$$ = $1; $$.type = LAM ;insertArray(&variables, $$); }
    |attr_identifier
    |lam_parameters COMMA identifier
    |lam_parameters COMMA attr_identifier;


//----------------------- Print field -----------------------------------

print_stmt:
        PRINT
        | PRINT expression
        {printf(">>  "); print($2,&variables); }
        | PRINT expression_list
        {printf(">>  "); print($2,&variables); }
        | PRINT RIGHT_OP expression
        {printf(">>  "); print($3,&variables); }
        | PRINT RIGHT_OP expression_list
        {printf(">>  "); print($3,&variables); }
        | PRINT LPAR call RPAR
        {printf(">>  "); print($3,&variables); };


//----------------------- Expressions field -----------------------------------




expression_list:
    expression_list COMMA expression
    {$$ = $3;}
    |LPAR expression_list COMMA expression RPAR
    {$$ = $4;}
    | expression
    {$$ = $1; };


expression:
    atom
    {$$ = $1; }
    | expression PLUS expression
    {$$ = add_calc($1,$3,&variables,1);  }
    | expression MINUS expression
    {$$ = minus_calc($1,$3,&variables,1);   }
    | expression STAR expression
    {$$ = mul_calc($1,$3,&variables,1); }
    | expression SLASH expression
    {$$ = div_calc($1,$3,&variables,1);   }


    | expression assignment_op expression
    | expression arithmetic_op expression
    | expression comparison_op expression
    | expression logical_op expression
    | expression bitwise_op expression;
    | LPAR expression RPAR
    {$$ = $2;}

atom:
    literal
    {$$ = $1;}
    | identifier
    {$$ = $1; }
    |integer
    {$$ = $1; }
    | attr_identifier
    {$$ = $1; }
    | dict_display
    {$$ = $1; }
    |dict_setdefault
    {$$ = $1; };



//----------------------- Assignment field ------------------------------------
assignment_stmt:
    assignment_stmt_targer_list expression_list
    {insertArray(&variables,value_assign($1,$2,&variables));   }

    |assignment_stmt_targer_list call

    |assignment_stmt_targer_list lambda_form
    {insertArray(&variables,value_assign($1,$2,&variables));}
    ;

assignment_stmt_targer_list:
    target_list EQUAL
    {$$ = $1; }
    | assignment_stmt_targer_list target_list EQUAL;


target_list:
```



```
    target
    {$$ = $1; }
    | target_list COMMA target
    | target_list COMMA;
target:
    IDENTIFIER
    {$$ = $1; }
    |attr_identifier
    {$$ = $1; }
    |LPAR target_list RPAR
    {$$ = $2; };


//----------------------- Operators field ------------------------------------


assignment_op:

     ADD_ASSIGN
    | SUB_ASSIGN
    | MUL_ASSIGN
    | POW_ASSIGN
    | DIV_ASSIGN
    | MOD_ASSIGN
    | AND_ASSIGN
    | XOR_ASSIGN
    | OR_ASSIGN
    | RIGHT_ASSIGN
    | LEFT_ASSIGN    ;

arithmetic_op:
    PERCENT
    | DOUBLESTAR
    | DOUBLESLASH;

comparison_op:
    EQ_OP
    | NE_OP
    | GREATER_THAN_OP
    | LESS_THAN_OP
    | LE_OP
    | GE_OP;

logical_op:
    AND
    | NOT
    | OR
    | IS
    | IN
    | IS NOT
    | NOT IN;

bitwise_op:
    AND_EXP
    | OR_SIGN
    | XOR
    | NOT_SIGN
    | LEFT_OP
    | RIGHT_OP;


literal:
    //integer
    //{$$ = $1; }
     floatnumber
    {$$ = $1; }
    | stringliteral
    {$$ = $1; }
    | longinteger
    {$$ = $1; }
    | imagnumber
    {$$ = $1; };




//----------------------- Import filed --------------------------------------
import_stmt:

    IMPORT module
    | IMPORT module AS name
    | IMPORT modules modules
    | IMPORT modules AS name modules
    | FROM relative_module IMPORT identifier
    | FROM relative_module IMPORT identifier AS name
    | FROM relative_module IMPORT identifier import_stmt_identifiers
    | FROM relative_module IMPORT identifier AS name import_stmt_identifiers
    | FROM relative_module IMPORT LPAR identifier RPAR
    | FROM relative_module IMPORT LPAR identifier AS name RPAR
    | FROM relative_module IMPORT LPAR identifier import_stmt_identifiers RPAR
    | FROM relative_module IMPORT LPAR identifier AS name import_stmt_identifiers RPAR
    | FROM relative_module IMPORT LPAR identifier COMMA RPAR
    | FROM relative_module IMPORT LPAR identifier AS name COMMA RPAR
    | FROM relative_module IMPORT LPAR identifier import_stmt_identifiers COMMA RPAR
    | FROM relative_module IMPORT LPAR identifier AS name import_stmt_identifiers COMMA RPAR
    | FROM relative_module IMPORT STAR;


module:
    module DOT identifier
    | identifier;

relative_module:
     module
    | dots module
    | dots;


dots: DOT
    | dots DOT;

modules: modules COMMA  module
    | modules COMMA  module AS name
    | COMMA module
    | COMMA module AS name;


import_stmt_identifiers:
    COMMA identifier
    | COMMA identifier AS name
    | import_stmt_identifiers COMMA identifier
    | import_stmt_identifiers COMMA identifier AS name;

name: IDENTIFIER ;


//----------------------- Compound_stmt field -----------------------------------------------

//=================================== If =============================================
if_stmt:
    IF  expression  COLON statement_list
    | IF  expression COLON statement_list ELSE COLON statement_list
    | IF expression  COLON statement_list elif_stmt
    | IF  expression  COLON statement_list elif_stmt ELSE COLON statement_list;

elif_stmt:
    ELIF  expression  COLON  statement_list
    | elif_stmt ELIF  expression  COLON statement_list;

//=================================== For ===========================================

for_stmt:
    FOR for_target_list IN expression_list COLON statement_list
    |FOR for_target_list IN RANGE LPAR expression_list RPAR  COLON statement_list
    | FOR for_target_list IN expression_list COLON statement_list ELSE COLON statement_list;

for_target_list:
    for_target
    | for_target_list COMMA target
    | for_target_list COMMA;
```

```
for_target:
    identifier
    |LPAR for_target_list RPAR;

//================================== Function  ================================

funcdef:
    DEF funcname LPAR RPAR COLON statement_list
    | decorators DEF funcname LPAR RPAR COLON statement_list
    | DEF funcname LPAR parameter_list RPAR COLON statement_list
    | decorators DEF funcname LPAR parameter_list RPAR COLON statement_list;

decorators:
    decorator
    | decorators decorator;

decorator:
    PAPAKI dotted_name NEWLINE
    | PAPAKI dotted_name LPAR RPAR NEWLINE;

dotted_name:
    identifier
    | identifier dot_identifiers;

dot_identifiers:
    DOT identifier
    | dot_identifiers DOT identifier;

parameter_list:
    STAR identifier
    | STAR identifier COMMA DOUBLESTAR identifier
    | DOUBLESTAR identifier
    | defparameter
    | defparameter COMMA
    | defparameters STAR identifier
    | defparameters STAR identifier COMMA DOUBLESTAR identifier
    | defparameters DOUBLESTAR identifier
    | defparameters defparameter
    | defparameters defparameter COMMA;

defparameter:
    parameter
    | parameter EQUAL expression;

defparameters:
    defparameter COMMA
    | defparameters defparameter COMMA;
```



```
sublist:
    parameter
    | parameter COMMA
    | parameter parameters
    | parameter parameters COMMA;

parameter:
    identifier
    | LPAR sublist RPAR;

parameters:
    COMMA parameter
    | parameters COMMA parameter;

funcname:
    identifier


//=================================== Class ===========================================

classdef:
    CLASS classname COLON statement_list
    | CLASS classname inheritance COLON statement_list;

inheritance:
    LPAR RPAR
    | LPAR expression_list RPAR;

classname:
    identifier;

//----------------------- etc -------------------------------------------------

dict_items:
identifier DOT ITEMS LPAR RPAR
{items(&dictionary,&variables);};

dict_setdefault:
    identifier DOT SETDEFAULT LPAR expression COMMA expression RPAR
    {setDefault($5,$7,&dictionary,&variables);};

dict_display:
    LBRA RBRA
    | LBRA key_datum_list RBRA;

key_datum_list:
    key_datum
    | key_datum COMMA
    | key_datum key_datums
    | key_datum key_datums COMMA;
```

```
key_datums:
    COMMA key_datum
    | key_datums COMMA key_datum;

key_datum:
        expression COLON expression
        { insertArray(&dictionary,$1); insertArray(&dictionary,$3);};

//======================================================================================
attr_identifier:
    identifier
    {$$ = $1; }
    | attr_identifier DOT identifier
    {$$ = $1;  }
    |identifier DOT identifier
;

identifier:
        IDENTIFIER
        {$$ = $1;  } ;



stringliteral:
    SHORTSTRING
    {$$ = $1;}
    | LONGSTRING
     {$$ = $1;}
;

longinteger:
    integer 'l'
    {$$ = $1;}
    | integer 'L'
    {$$ = $1;}
    ;


integer:
    DECINTEGER
    {$$ = $1;}
    | OCTINTEGER
    {$$ = $1;}
    | HEXINTEGER
    {$$ = $1;}
;

floatnumber:
    POINTFLOAT
    {$$ = $1;}
    | EXPONENTFLOAT
    {$$ = $1;}
;
imagnumber:
    IMAGNUMBER
    {$$ = $1;}
;

%%


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


  initArray(&variables, 5);  // initially 5 elements
  initArray(&dictionary,5);


   extern int yydebug;
   //yydebug = 1;

  // Open a file
  FILE *myfile = fopen(argv[1], "r");
  //  is valid?
  if (!myfile) {

    return -1;
  }
  // read the file
  yyin = myfile;

  // Parse through the input:
  yyparse();

}


void yyerror(const char* s) {
    fprintf(stderr, "Line: %d --> Parser error\n", yylineno);
    exit(1);
}
```

End of file parser.y

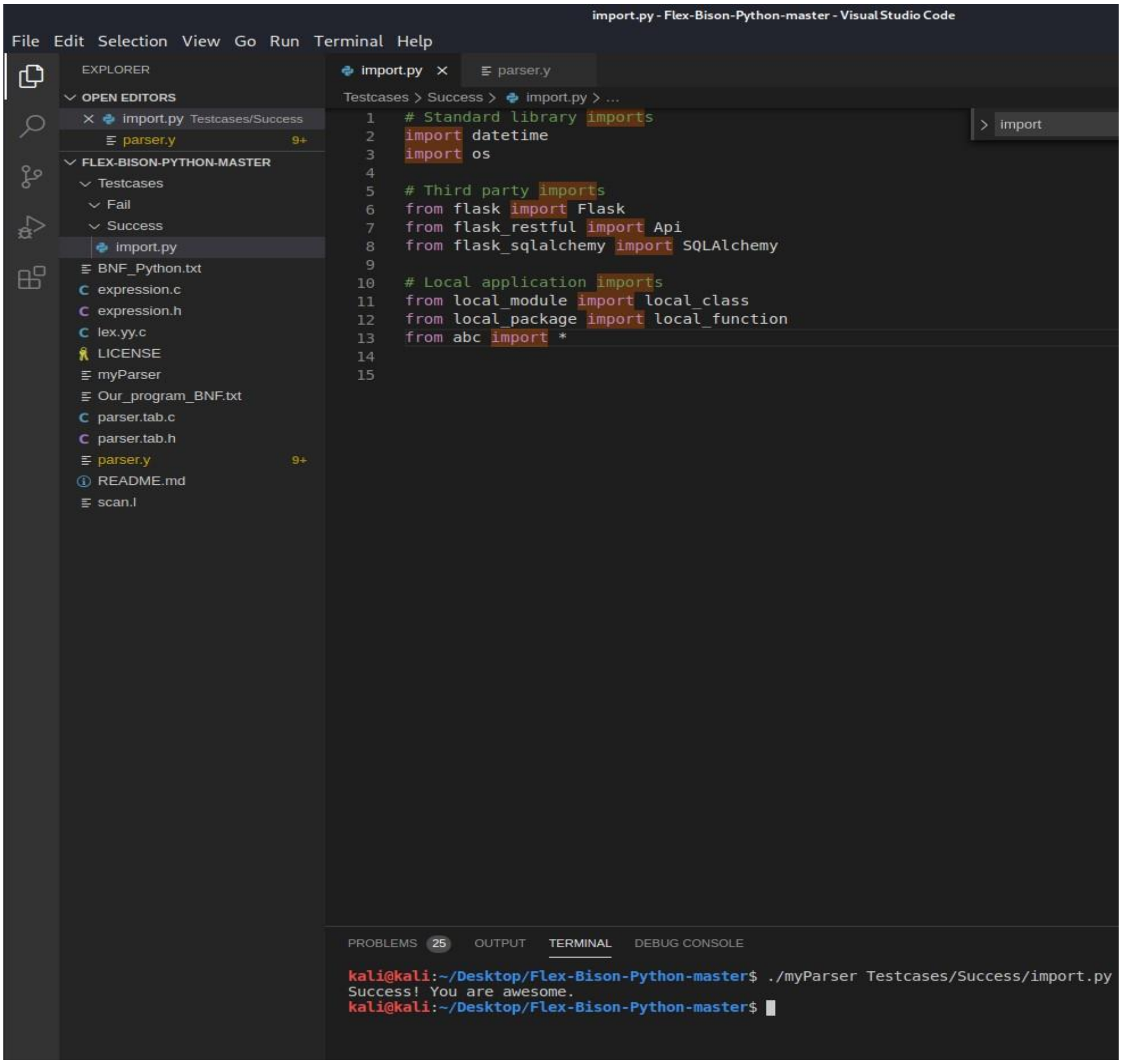
4.

Παραδείγματα εφαρμογής

A)

Εντολή import

Επιτυχημένη προσπάθεια:

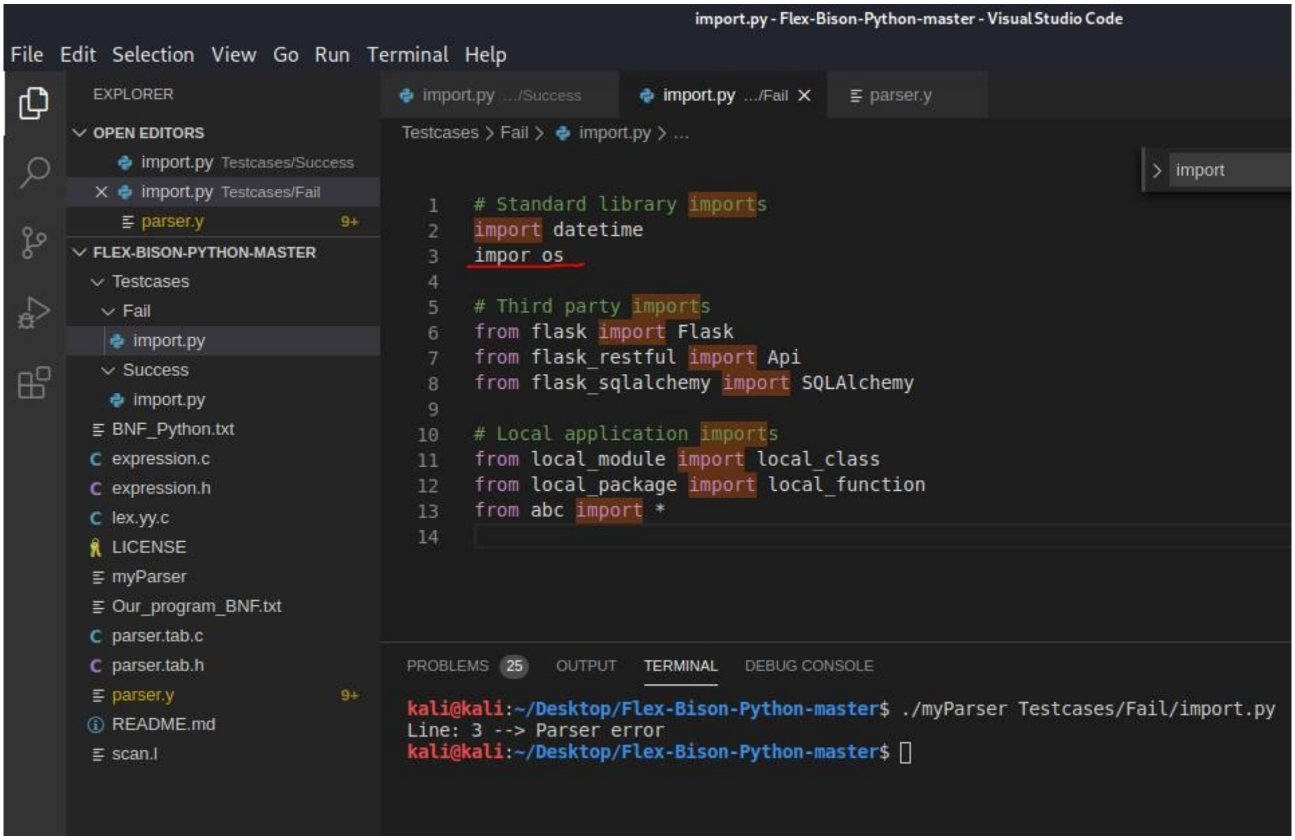


```python
# Standard library imports
import datetime
import os

# Third party imports
from flask import Flask
from flask_restful import Api
from flask_sqlalchemy import SQLAlchemy

# Local application imports
from local_module import local_class
from local_package import local_function
from abc import *
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/import.py
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

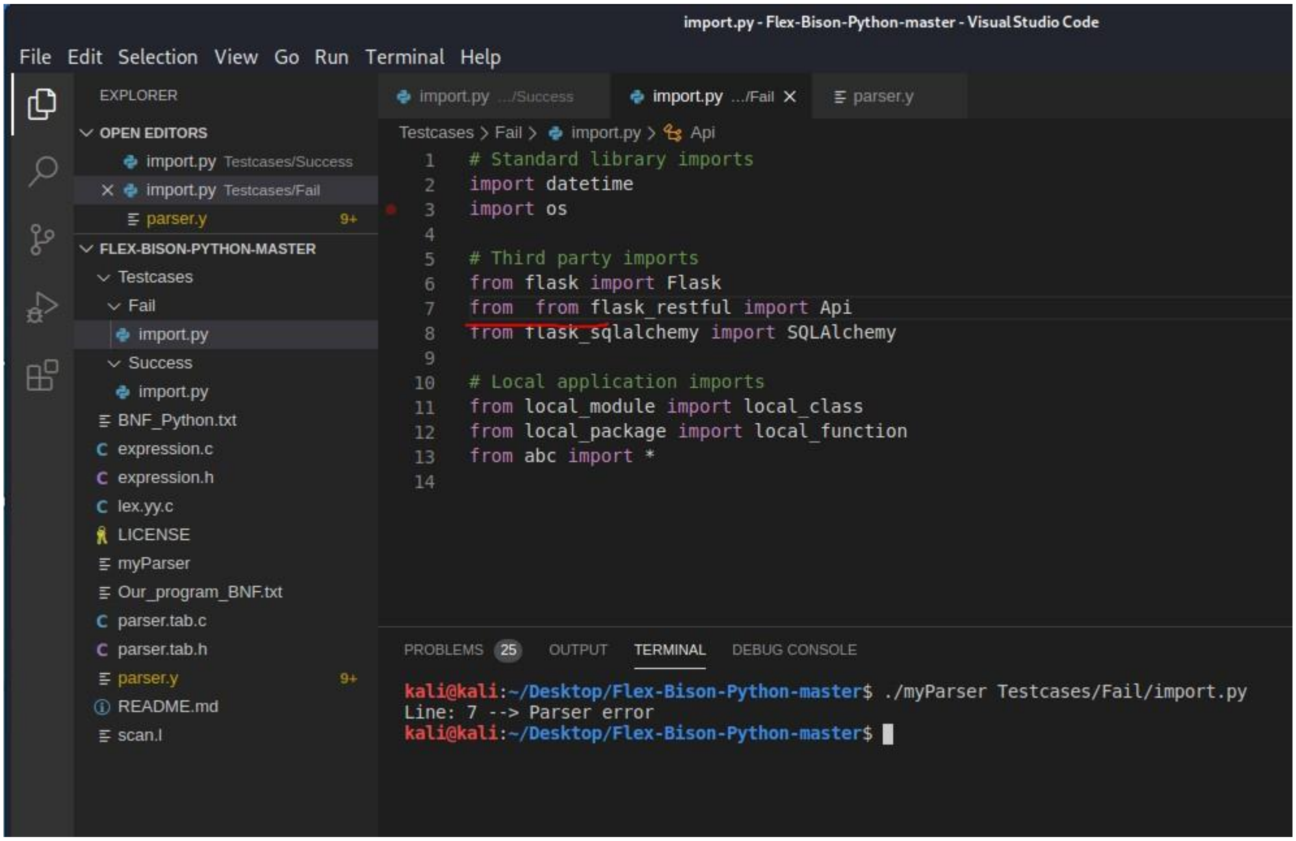
Αναγνώριση όλων των τύπων εντολών import και ενημέρωση του χρήστη για την επιτυχημένη προσπάθεια μεταγλώττισης.

Αποτυχημένες προσπάθειες:



```
# Standard library imports
import datetime
impor os

# Third party imports
from flask import Flask
from flask_restful import Api
from flask_sqlalchemy import SQLAlchemy

# Local application imports
from local_module import local_class
from local_package import local_function
from abc import *
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/import.py
Line: 3 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```



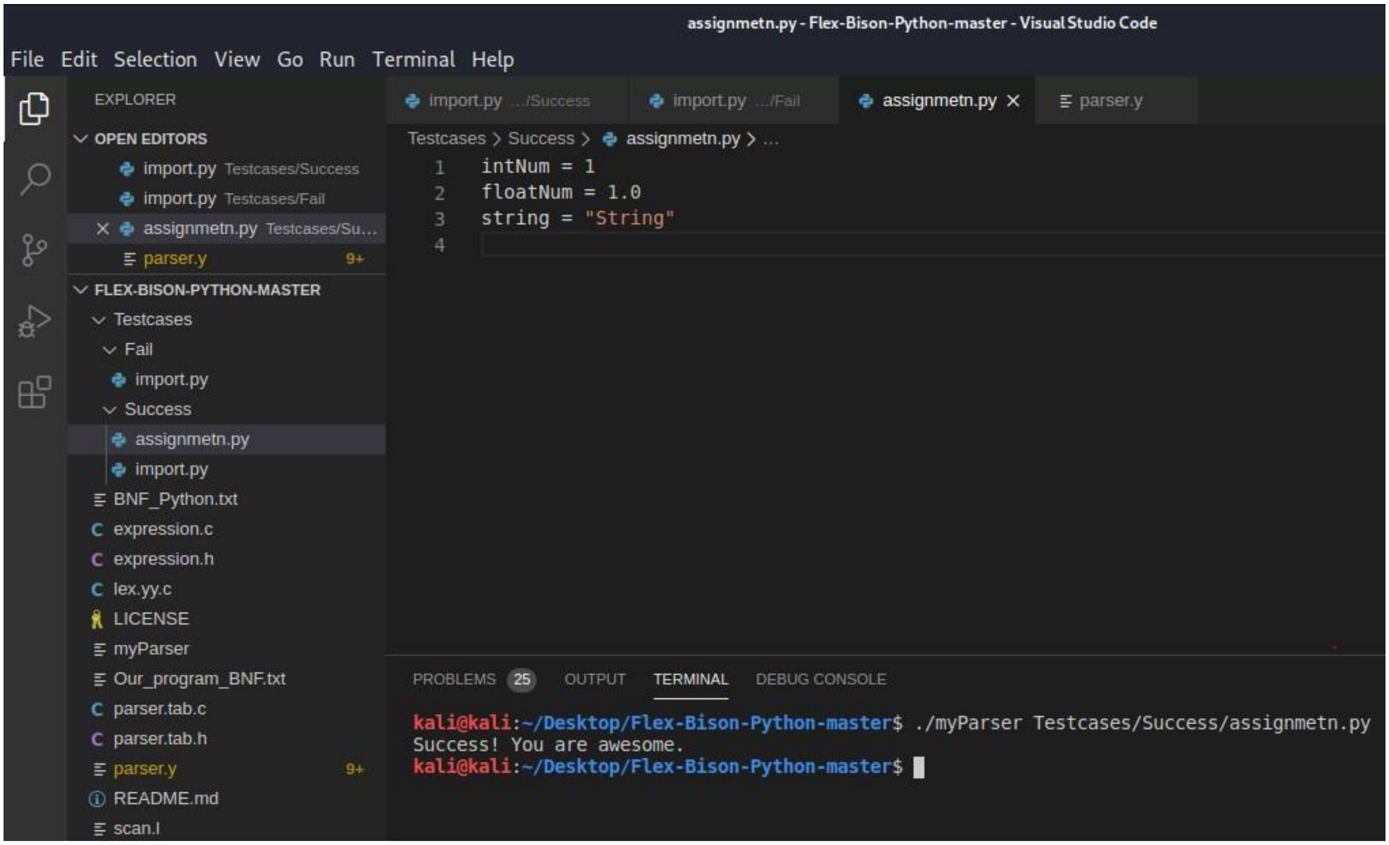
```
# Standard library imports
import datetime
import os

# Third party imports
from flask import Flask
from  from flask_restful import Api
from flask_sqlalchemy import SQLAlchemy

# Local application imports
from local_module import local_class
from local_package import local_function
from abc import *
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/import.py
Line: 7 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

B)

Αρχικοποίηση μεταβλητών

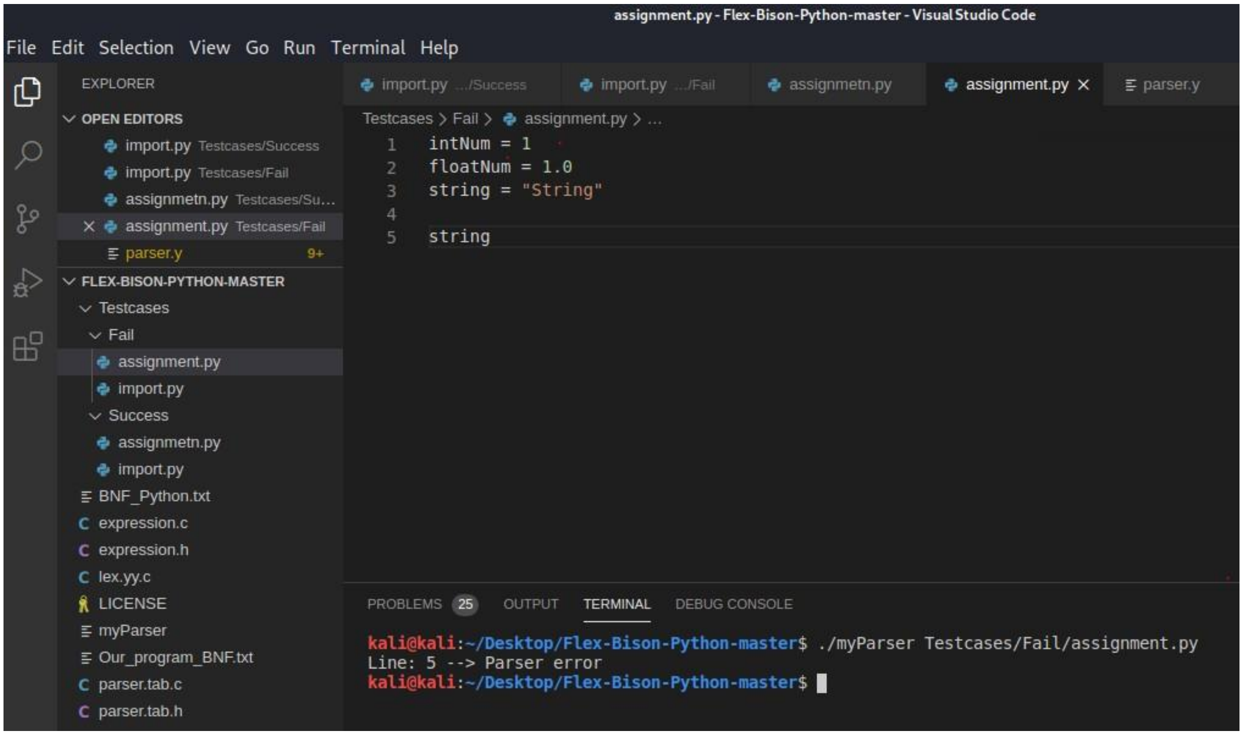
Επιτυχημένη προσπάθεια:



```
intNum = 1
floatNum = 1.0
string = "String"
```

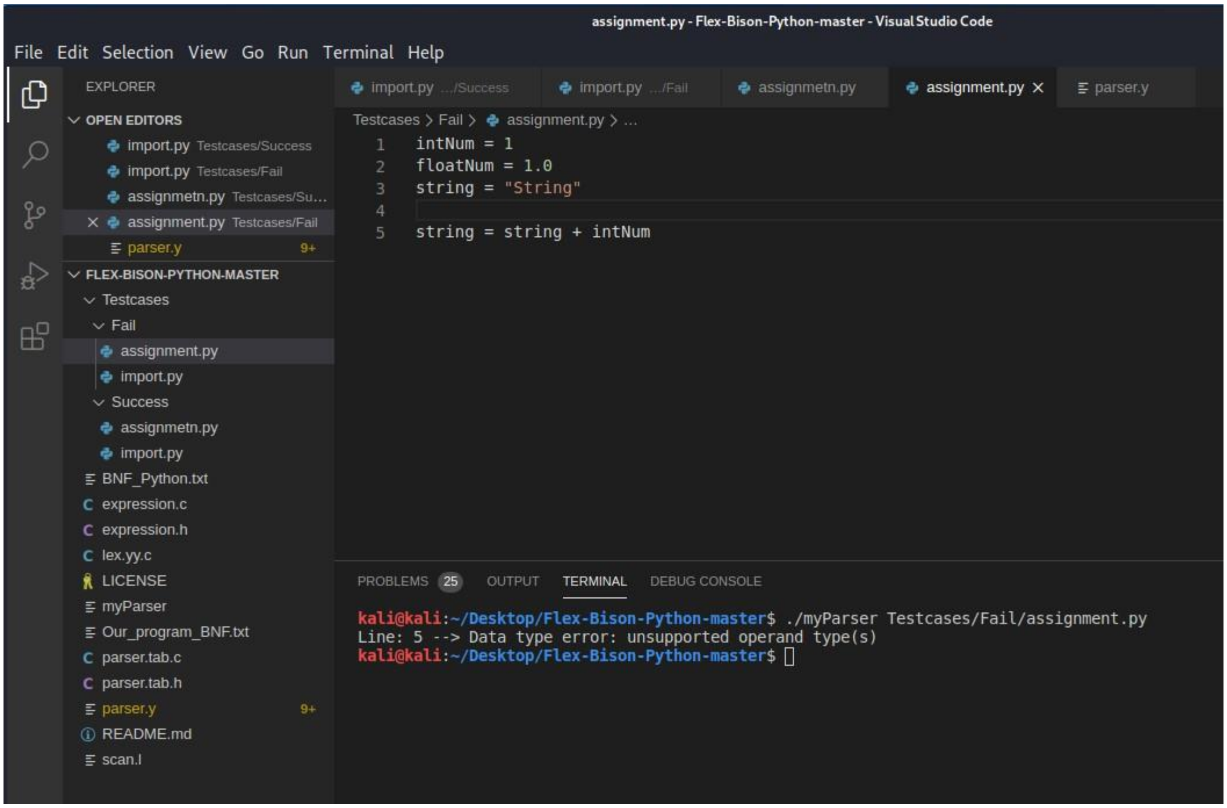
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/assignmetn.py
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Αποτυχημένες προσπάθειες:



```
1  intNum = 1
2  floatNum = 1.0
3  string = "String"
4
5  string
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/assignment.py
Line: 5 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```



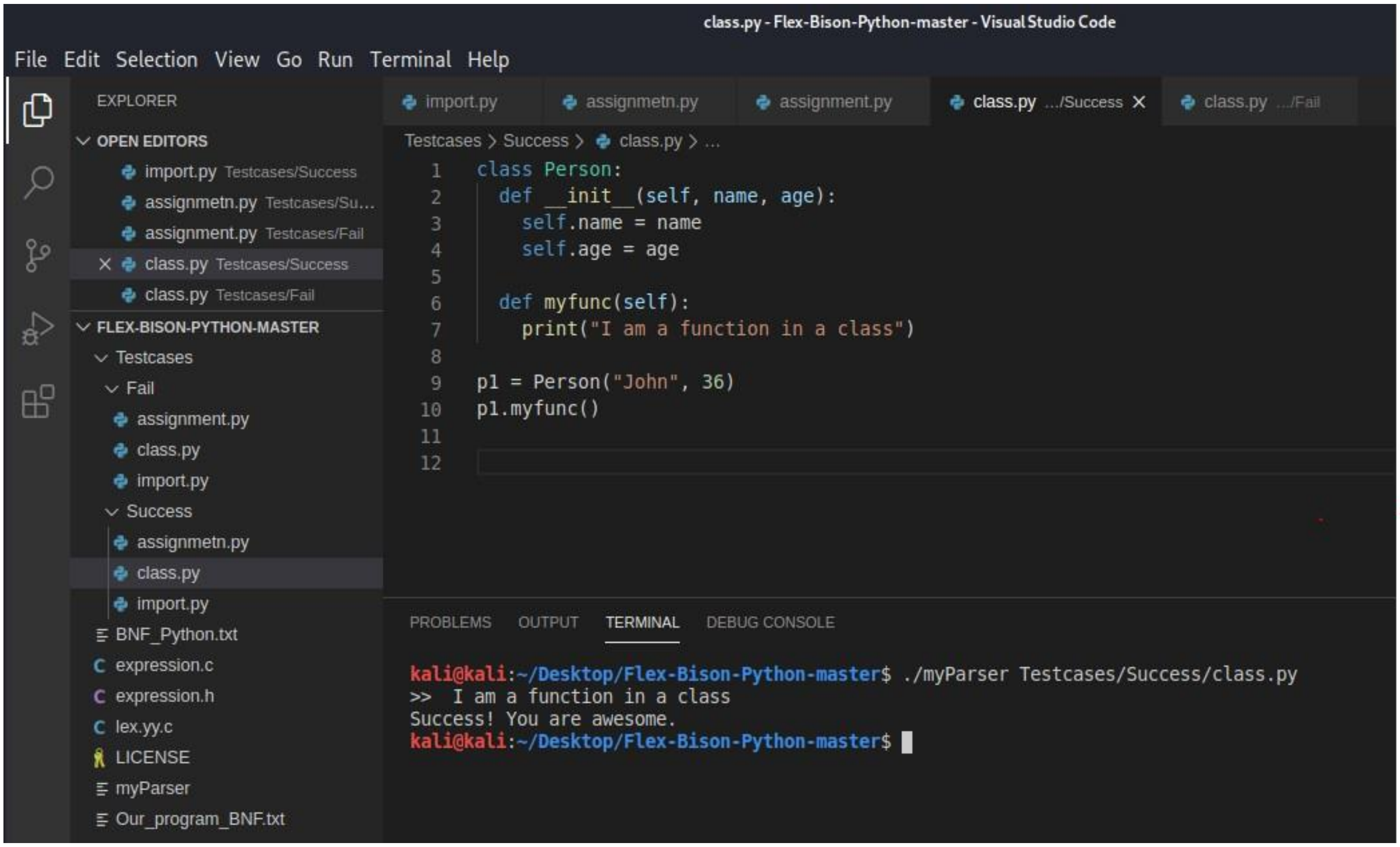
```
1  intNum = 1
2  floatNum = 1.0
3  string = "String"
4
5  string = string + intNum
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/assignment.py
Line: 5 --> Data type error: unsupported operand type(s)
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Γ)

Αρχικοποίηση κλάσης και αντικειμένου

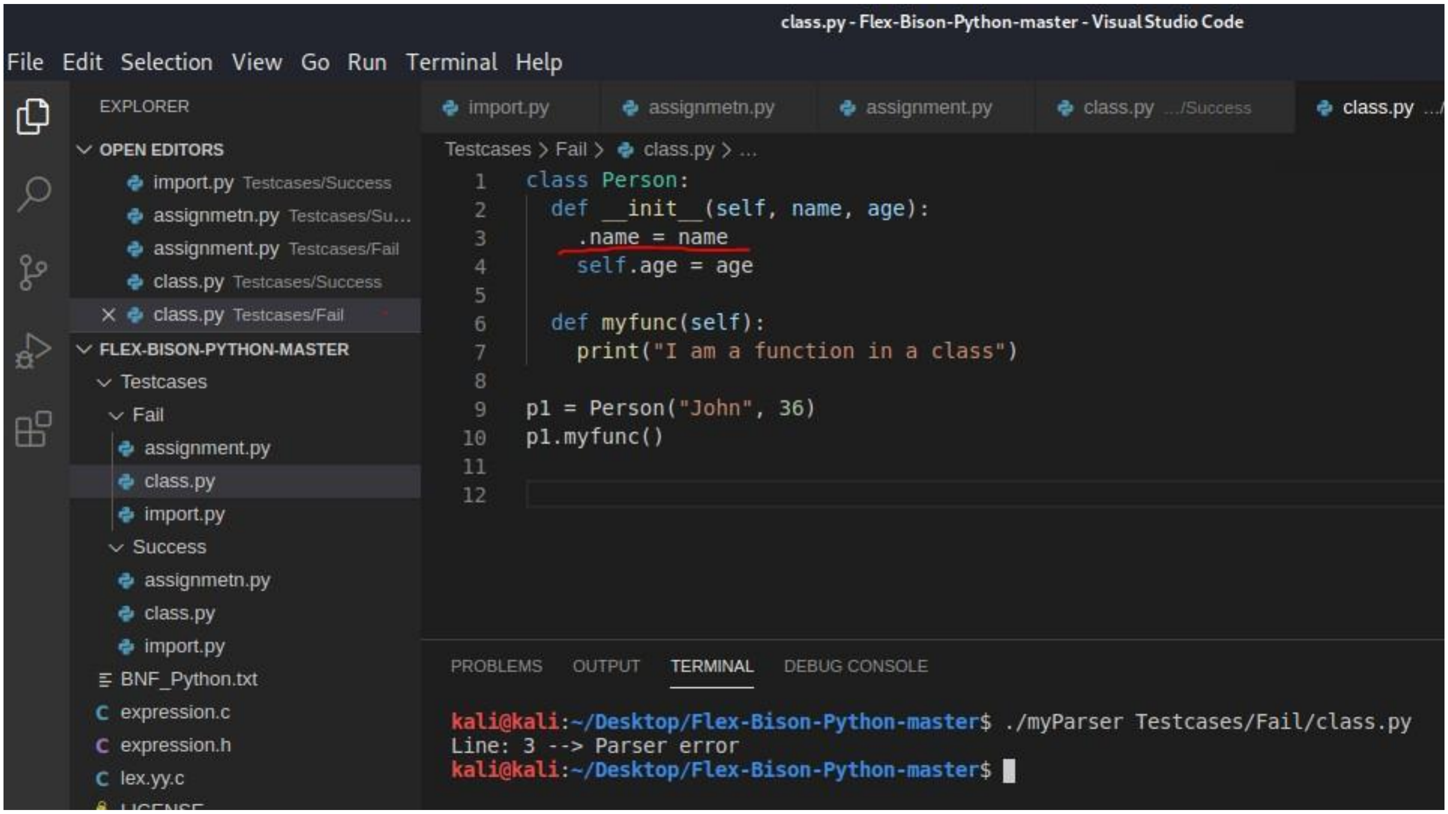
Επιτυχημένη προσπάθεια:



```python
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("I am a function in a class")

p1 = Person("John", 36)
p1.myfunc()
```

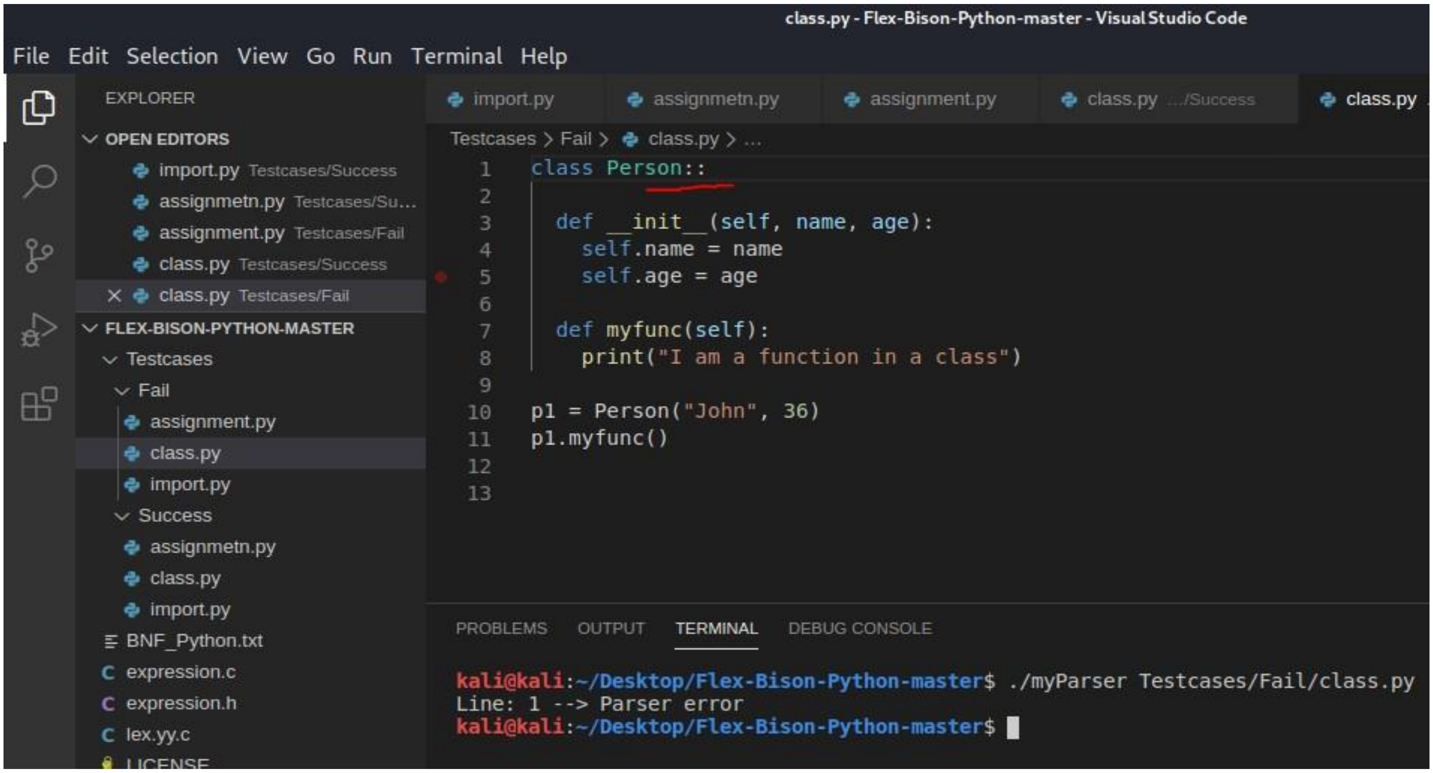
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/class.py
>>  I am a function in a class
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

Αποτυχημένες προσπάθειες:



```
class Person:
  def __init__(self, name, age):
    .name = name
    self.age = age

  def myfunc(self):
    print("I am a function in a class")

p1 = Person("John", 36)
p1.myfunc()
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/class.py
Line: 3 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```



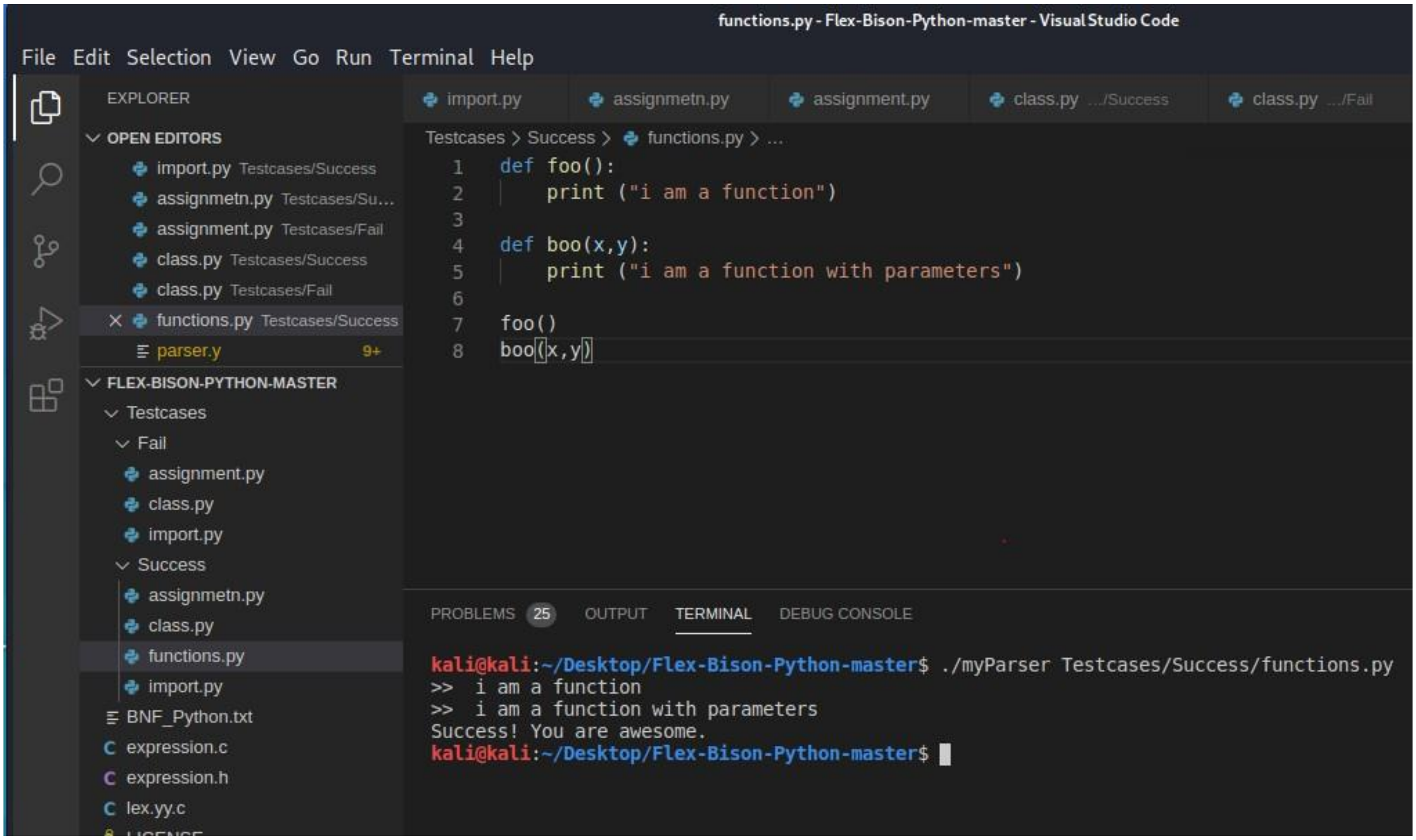
```
class Person::

  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("I am a function in a class")

p1 = Person("John", 36)
p1.myfunc()
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/class.py
Line: 1 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Δ)

Ορισμός συνάρτησης και κλήση της

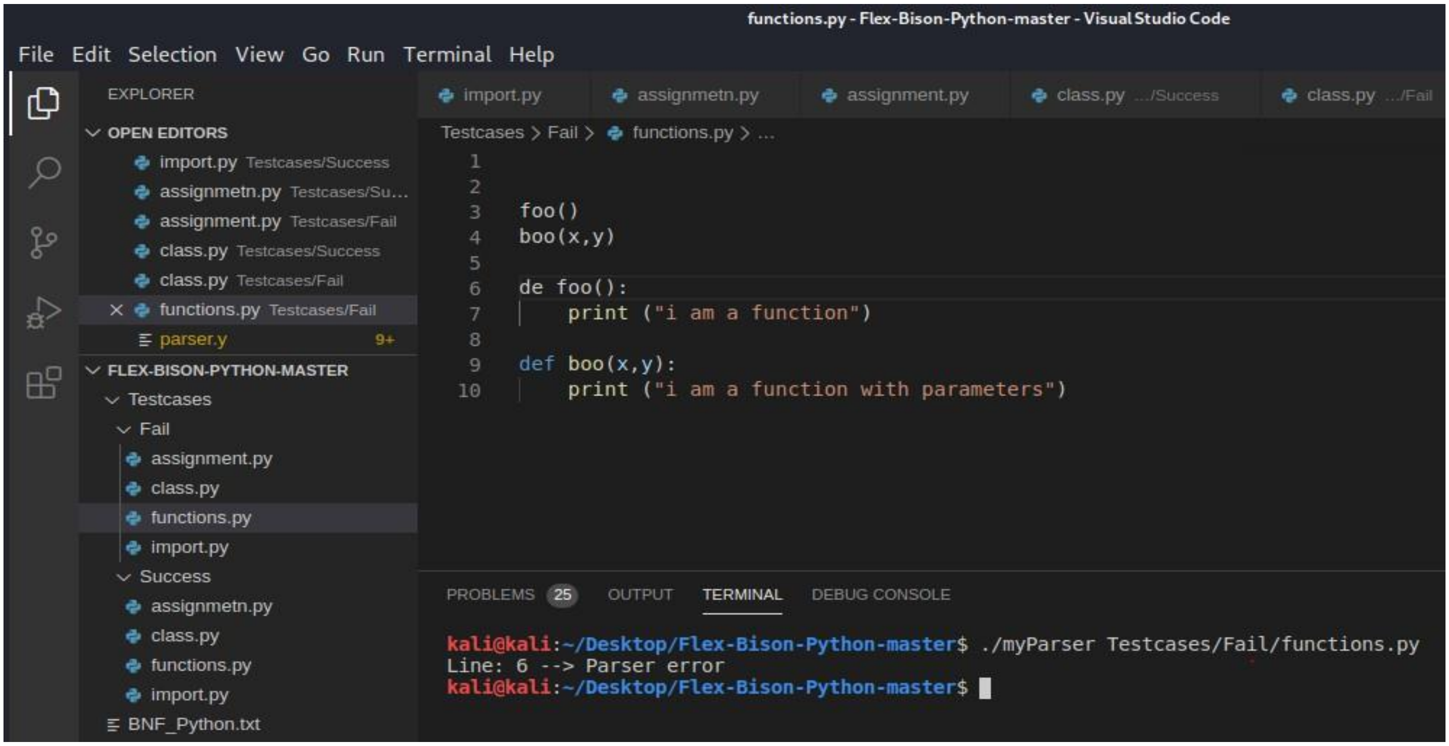
Επιτυχημένη προσπάθεια:



```
def foo():
    print ("i am a function")

def boo(x,y):
    print ("i am a function with parameters")

foo()
boo(x,y)
```

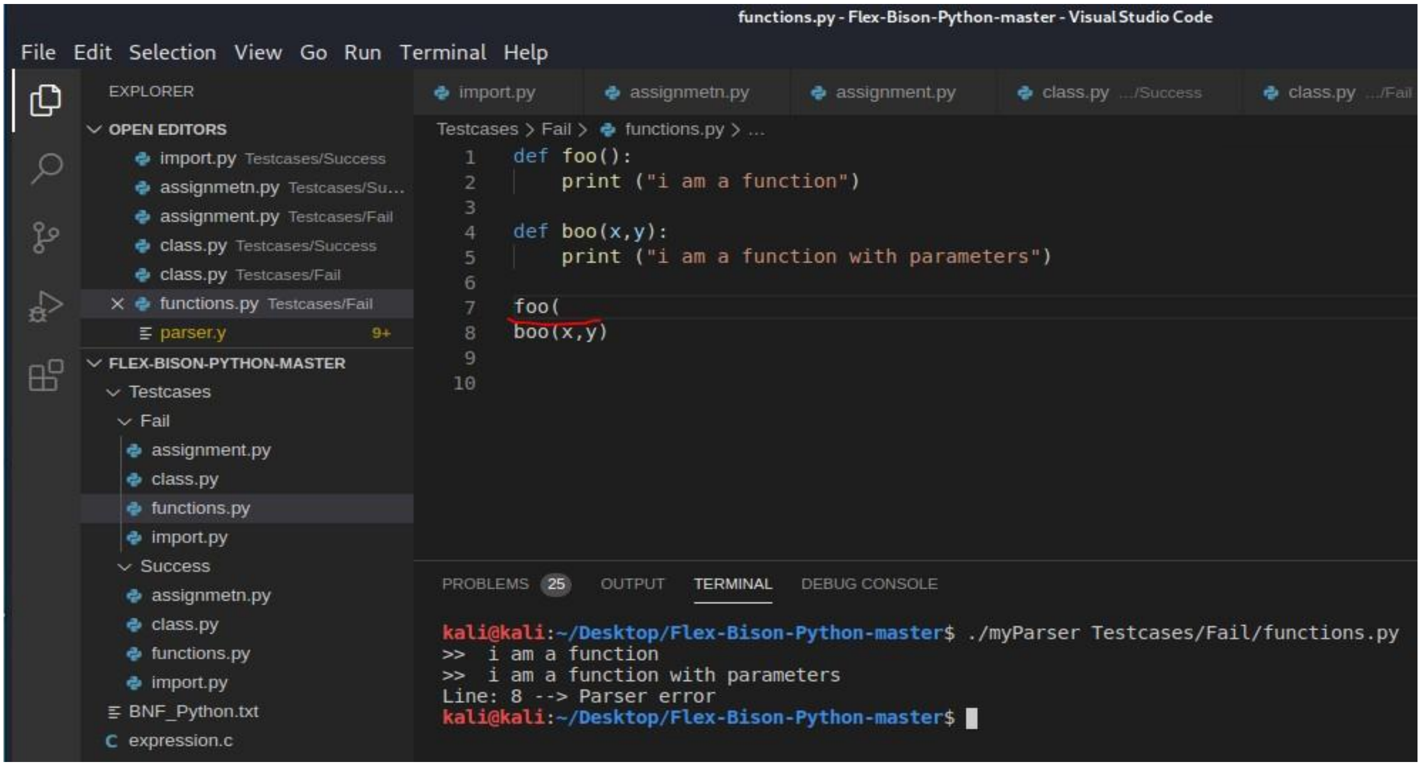
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/functions.py
>>  i am a function
>>  i am a function with parameters
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Αποτυχημένες προσπάθειες:



```


foo()
boo(x,y)

de foo():
    print ("i am a function")

def boo(x,y):
    print ("i am a function with parameters")
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/functions.py
Line: 6 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```



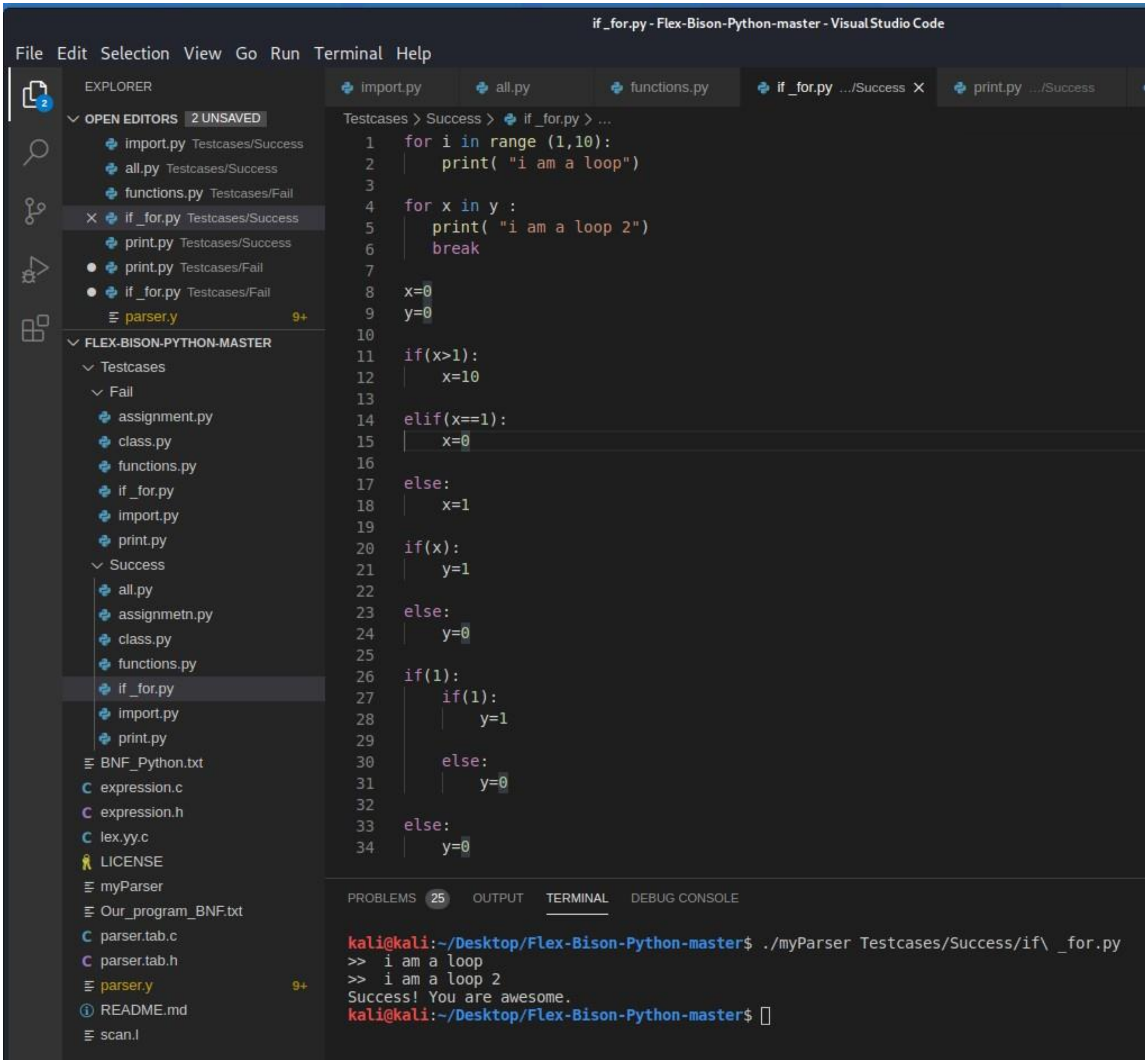
```
def foo():
    print ("i am a function")

def boo(x,y):
    print ("i am a function with parameters")

foo(
boo(x,y)


```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/functions.py
>>  i am a function
>>  i am a function with parameters
Line: 8 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

E)

Εντολές βρόγχου και συνθήκη

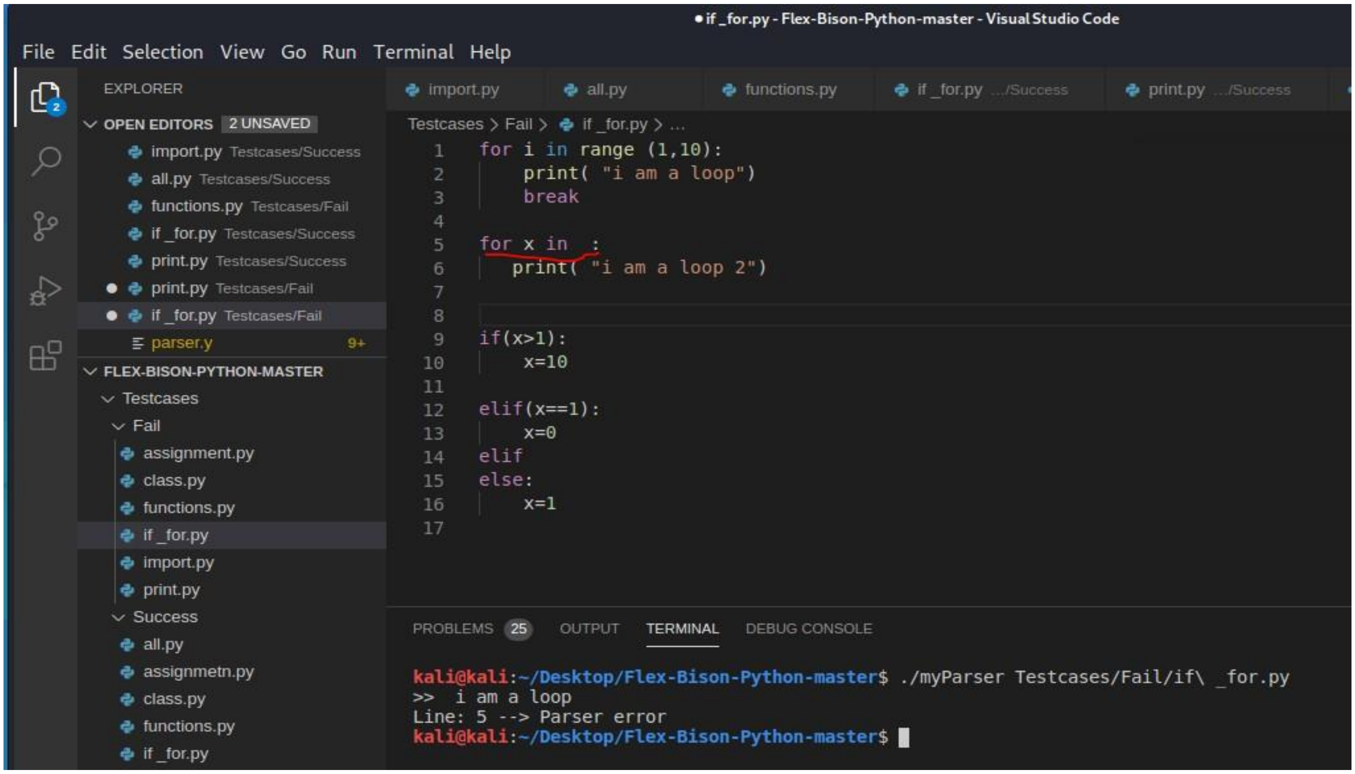
Επιτυχημένη προσπάθεια:



```
for i in range (1,10):
    print( "i am a loop")

for x in y :
   print( "i am a loop 2")
   break

x=0
y=0

if(x>1):
    x=10

elif(x==1):
    x=0

else:
    x=1

if(x):
    y=1

else:
    y=0

if(1):
    if(1):
        y=1

    else:
        y=0

else:
    y=0
```

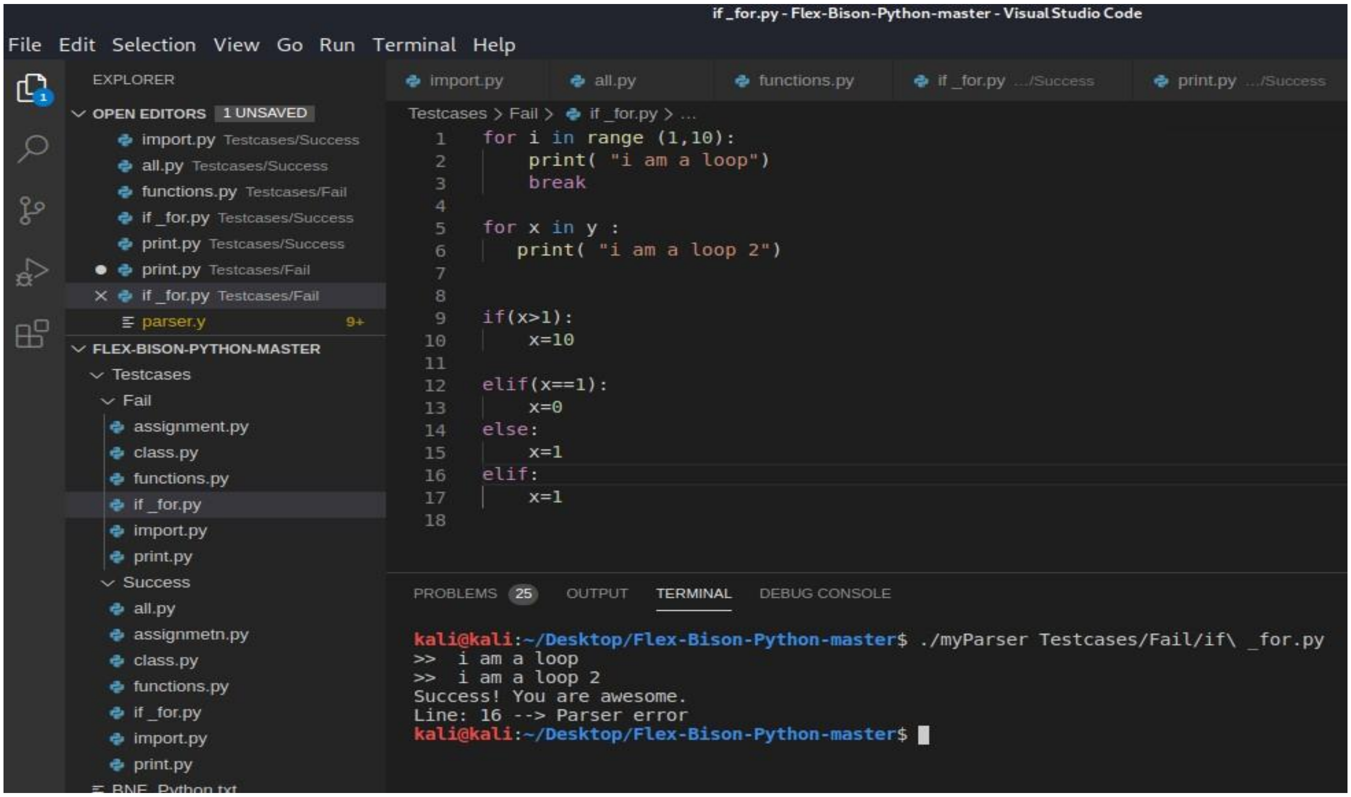
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/if\ _for.py
>>  i am a loop
>>  i am a loop 2
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Αποτυχημένες προσπάθειες:



```
for i in range (1,10):
    print( "i am a loop")
    break

for x in  :
  print( "i am a loop 2")


if(x>1):
    x=10

elif(x==1):
    x=0
elif
else:
    x=1
```

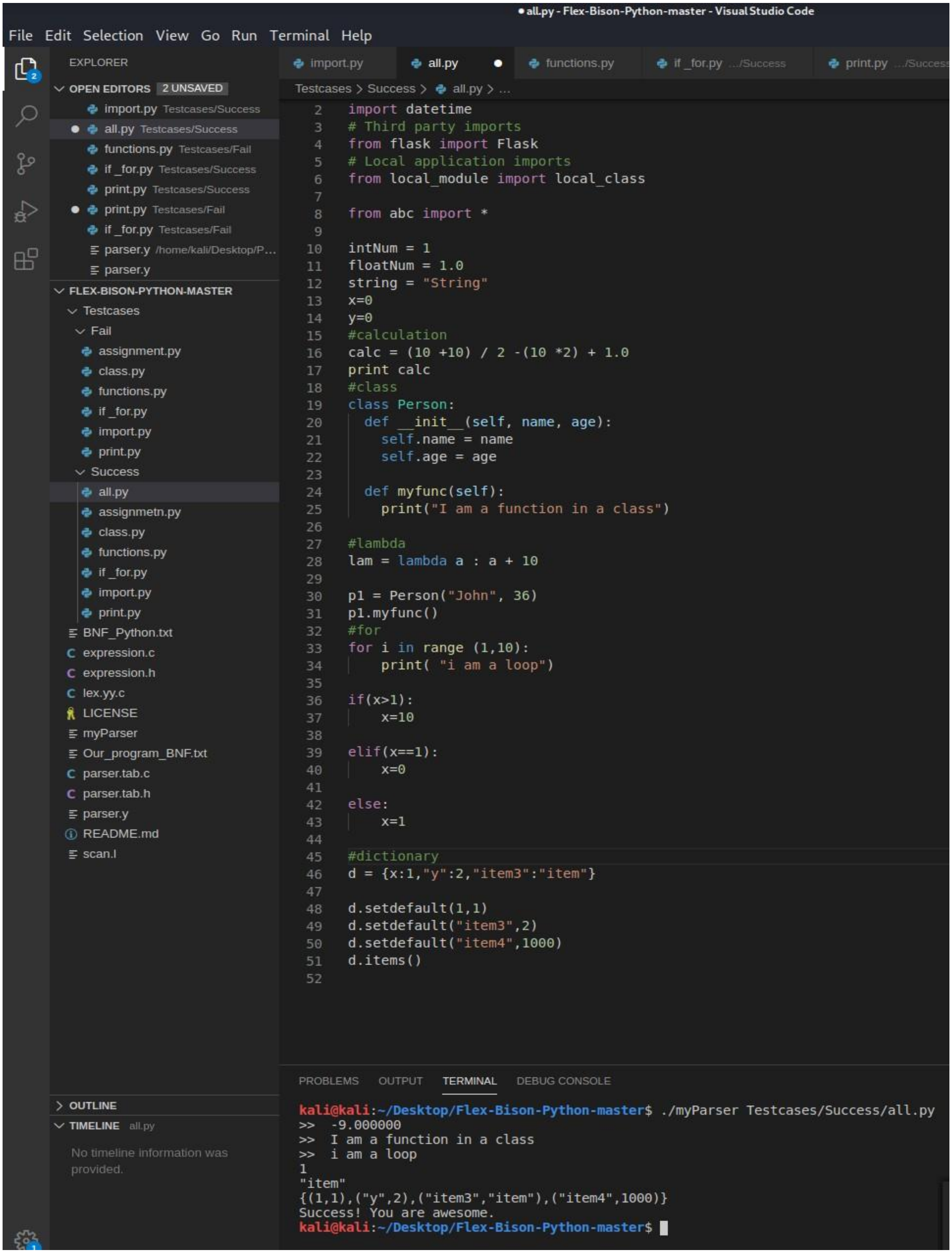
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/if\ _for.py
>>  i am a loop
Line: 5 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```



```
for i in range (1,10):
    print( "i am a loop")
    break

for x in y :
   print( "i am a loop 2")


if(x>1):
    x=10

elif(x==1):
    x=0
else:
    x=1
elif:
    x=1

```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/if\ _for.py
>>  i am a loop
>>  i am a loop 2
Success! You are awesome.
Line: 16 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

Αρχείο με πολλαπλές εντολές (Dictionaries και Lambda)

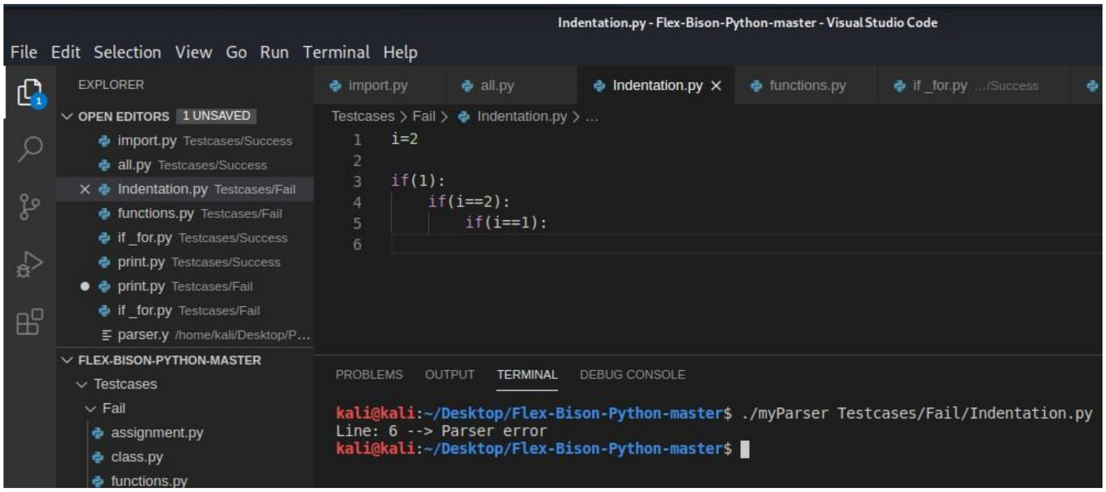


```python
import datetime
# Third party imports
from flask import Flask
# Local application imports
from local_module import local_class

from abc import *

intNum = 1
floatNum = 1.0
string = "String"
x=0
y=0
#calculation
calc = (10 +10) / 2 -(10 *2) + 1.0
print calc
#class
class Person:
  def __init__(self, name, age):
    self.name = name
    self.age = age

  def myfunc(self):
    print("I am a function in a class")

#lambda
lam = lambda a : a + 10

p1 = Person("John", 36)
p1.myfunc()
#for
for i in range (1,10):
    print( "i am a loop")

if(x>1):
    x=10

elif(x==1):
    x=0

else:
    x=1

#dictionary
d = {x:1,"y":2,"item3":"item"}

d.setdefault(1,1)
d.setdefault("item3",2)
d.setdefault("item4",1000)
d.items()

```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Success/all.py
>>  -9.000000
>>  I am a function in a class
>>  i am a loop
1
"item"
{(1,1),("y",2),("item3","item"),("item4",1000)}
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

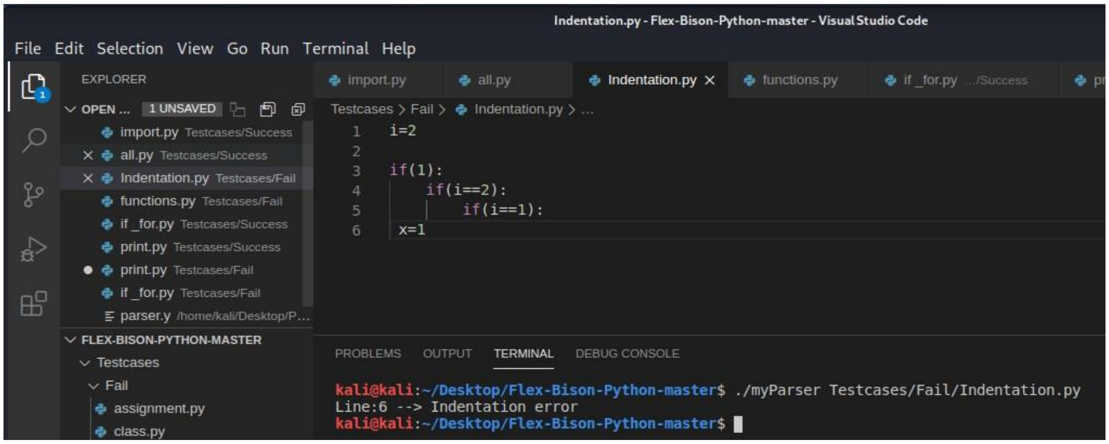
Python Indentation

Στα προηγούμενα παραδείγματα παρουσιάζεται η σωστή εκτέλεση του indentation στις εντολές if. Παρακάτω αναφέρονται περιπτώσεις που απαιτούν την τύπωση σφάλματος.


Indentation.py - Flex-Bison-Python-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
import.py Testcases/Success
all.py Testcases/Success
Indentation.py Testcases/Fail
functions.py Testcases/Fail
if _for.py Testcases/Success
print.py Testcases/Success
print.py Testcases/Fail
if _for.py Testcases/Fail
parser.y /home/kali/Desktop/P...
FLEX-BISON-PYTHON-MASTER
Testcases
Fail
assignment.py
class.py
functions.py
import.py
all.py
Indentation.py
functions.py
if _for.py .../Success
Testcases > Fail > Indentation.py > ...
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE

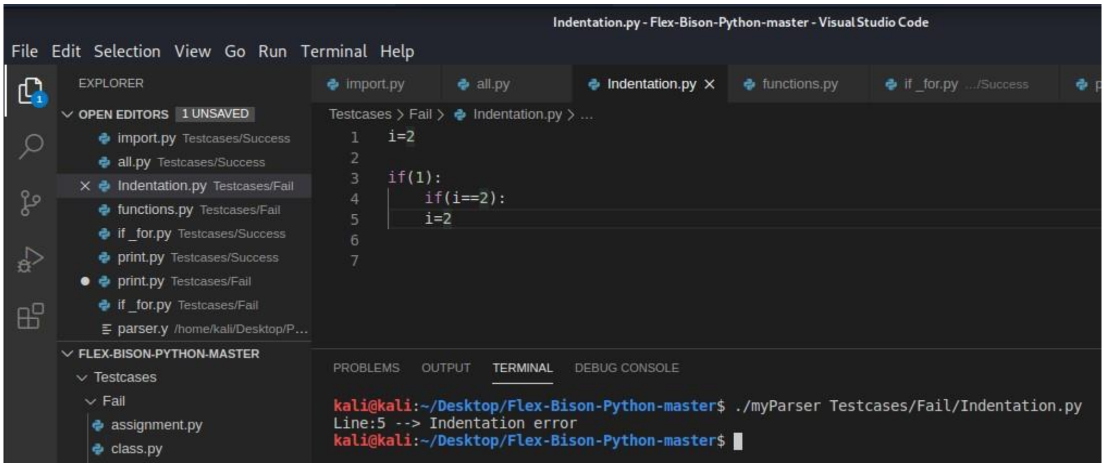
```
i=2

if(1):
    if(i==2):
        if(i==1):

```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/Indentation.py
Line: 6 --> Parser error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```


Indentation.py - Flex-Bison-Python-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN ... 1 UNSAVED
import.py Testcases/Success
all.py Testcases/Success
Indentation.py Testcases/Fail
functions.py Testcases/Fail
if _for.py Testcases/Success
print.py Testcases/Success
print.py Testcases/Fail
if _for.py Testcases/Fail
parser.y /home/kali/Desktop/P...
FLEX-BISON-PYTHON-MASTER
Testcases
Fail
assignment.py
class.py
import.py
all.py
Indentation.py
functions.py
if _for.py .../Success
Testcases > Fail > Indentation.py > ...
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE

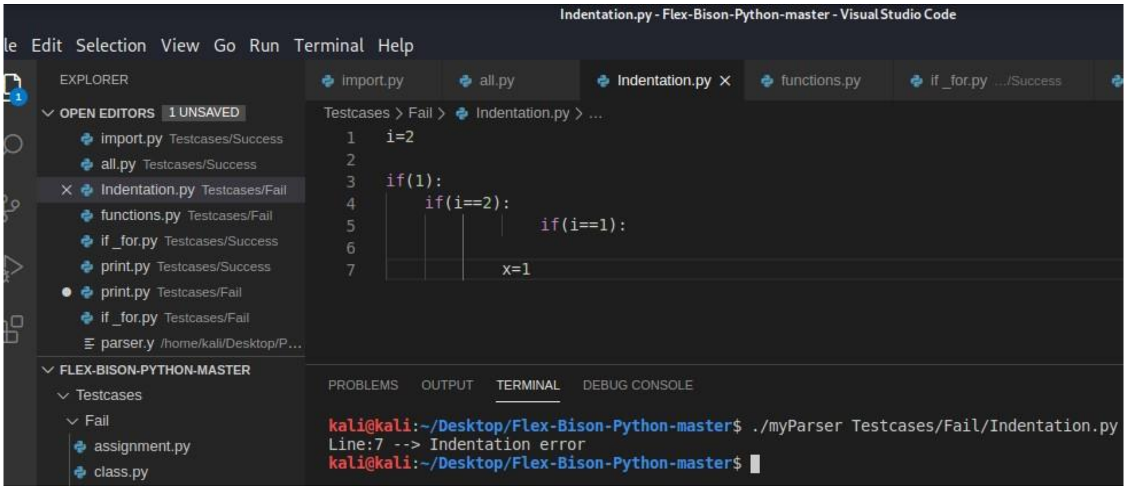
```
i=2

if(1):
    if(i==2):
        if(i==1):
 x=1
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/Indentation.py
Line:6 --> Indentation error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```


Indentation.py - Flex-Bison-Python-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
import.py Testcases/Success
all.py Testcases/Success
Indentation.py Testcases/Fail
functions.py Testcases/Fail
if _for.py Testcases/Success
print.py Testcases/Success
print.py Testcases/Fail
if _for.py Testcases/Fail
parser.y /home/kali/Desktop/P...
FLEX-BISON-PYTHON-MASTER
Testcases
Fail
assignment.py
class.py
import.py
all.py
Indentation.py
functions.py
if _for.py .../Success
Testcases > Fail > Indentation.py > ...
PROBLEMS
OUTPUT
TERMINAL
DEBUG CONSOLE

```
i=2

if(1):
    if(i==2):
    i=2


```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/Indentation.py
Line:5 --> Indentation error
kali@kali:~/Desktop/Flex-Bison-Python-master$
```



```
i=2

if(1):
    if(i==2):
                if(i==1):

            x=1
```

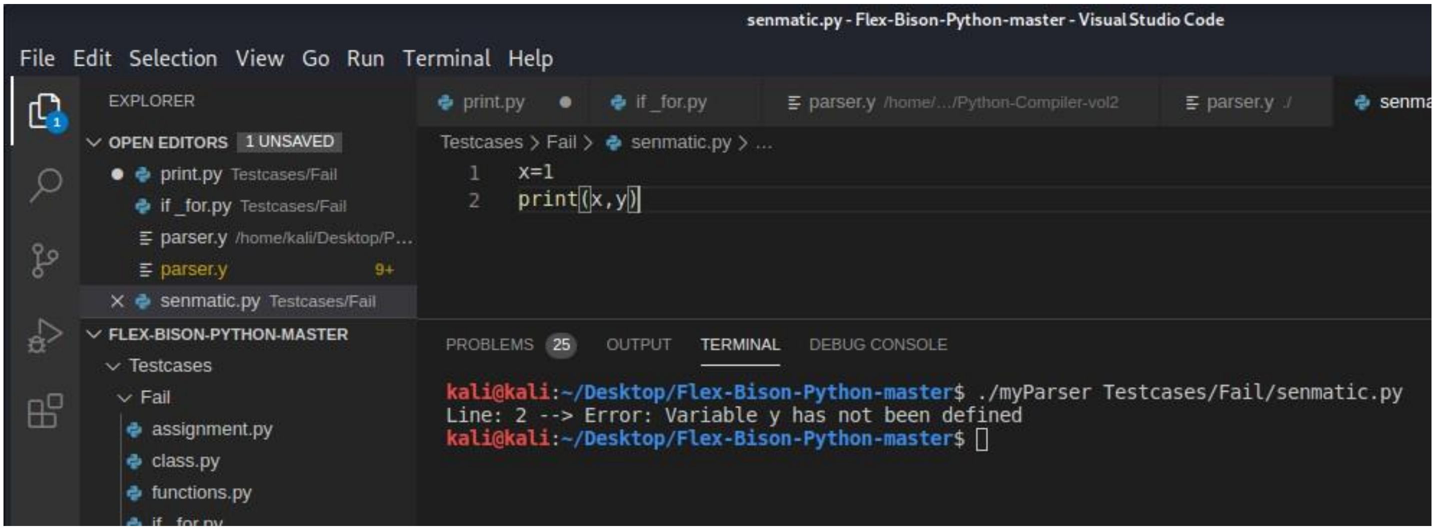
```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/Indentation.py
Line:7 --> Indentation error
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

Παρουσίαση Σημασιολογικής Λειτουργικότητας

Παρακάτω θα δείξουμε ορισμένες περιπτώσεις που ο compiler υλοποιεί και την σημασιολογική ανάλυση όπως:

A) Ανίχνευση μη ορισμένων μεταβλητών.
B) Ανίχνευση πράξεων που δεν επιτρέπονται (πχ. Integer με String).
Γ) Ανίχνευση πράξεων μεταξύ integer και float και μετατροπή του αποτελέσματος σε τιμή float.

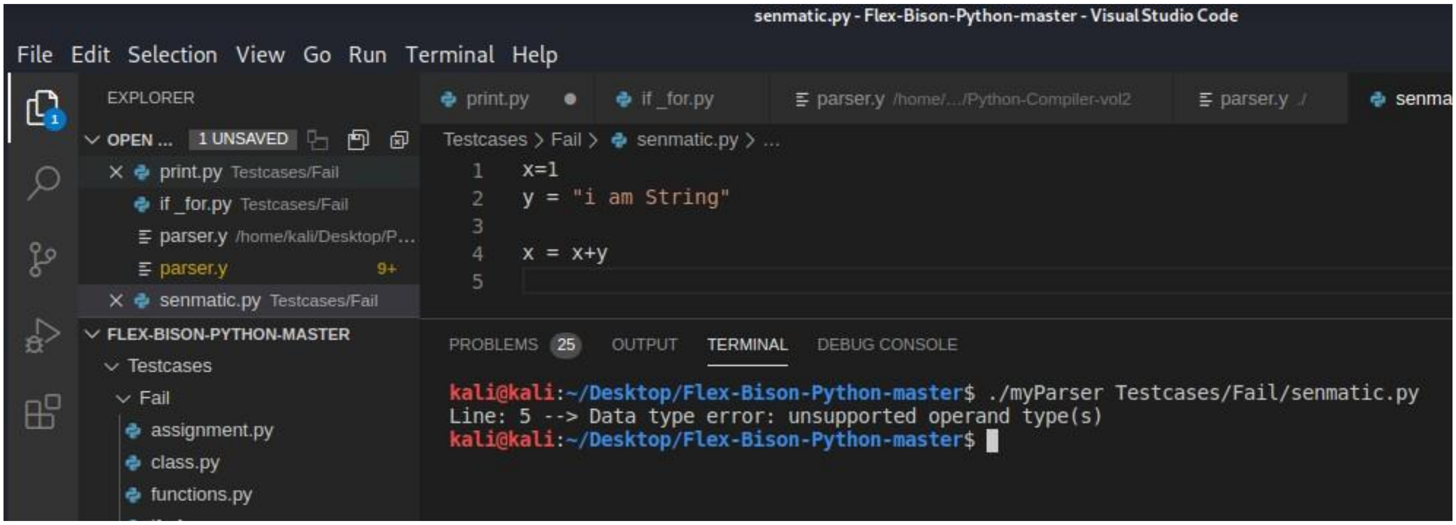
A)



```
x=1
print(x,y)
```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/senmatic.py
Line: 2 --> Error: Variable y has not been defined
kali@kali:~/Desktop/Flex-Bison-Python-master$ 
```

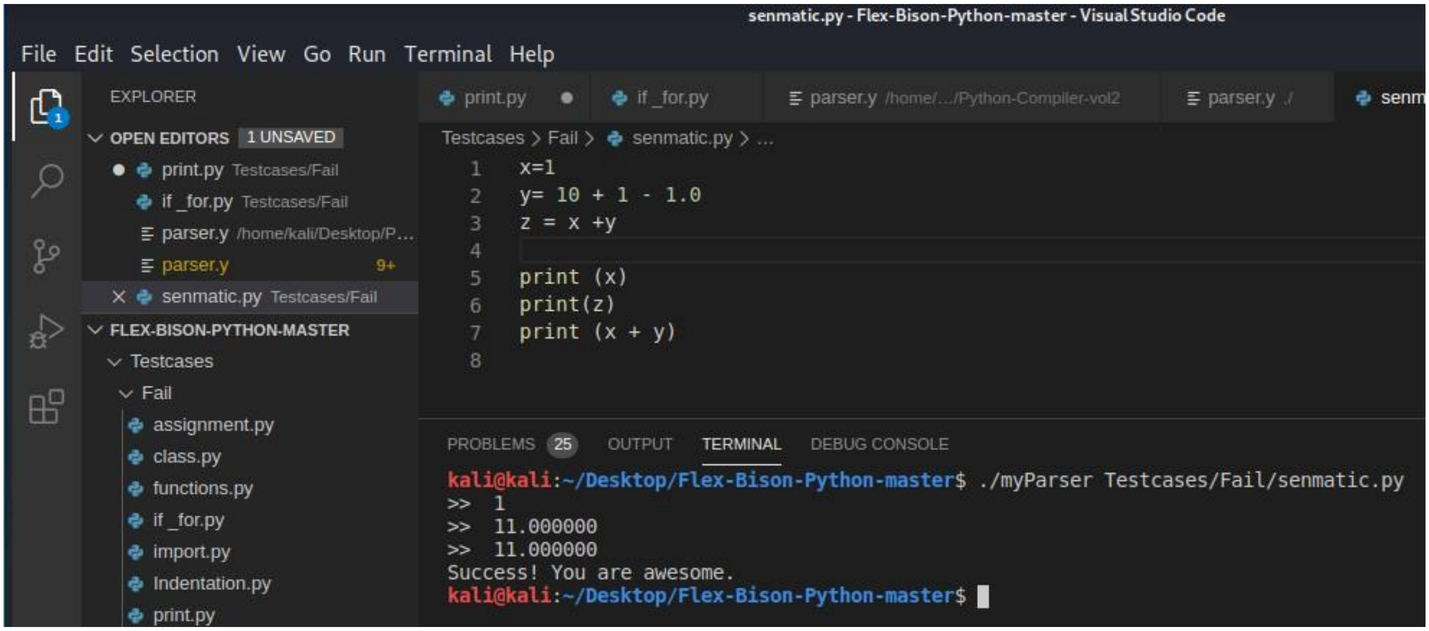
Β)


senmatic.py - Flex-Bison-Python-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN ... 1 UNSAVED
print.py Testcases/Fail
if _for.py Testcases/Fail
parser.y /home/kali/Desktop/P...
parser.y 9+
senmatic.py Testcases/Fail
FLEX-BISON-PYTHON-MASTER
Testcases
Fail
assignment.py
class.py
functions.py
print.py
if _for.py
parser.y /home/.../Python-Compiler-vol2
parser.y ./
senma
Testcases > Fail > senmatic.py > ...
PROBLEMS 25
OUTPUT
TERMINAL
DEBUG CONSOLE

```
x=1
y = "i am String"

x = x+y

```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/senmatic.py
Line: 5 --> Data type error: unsupported operand type(s)
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Γ)

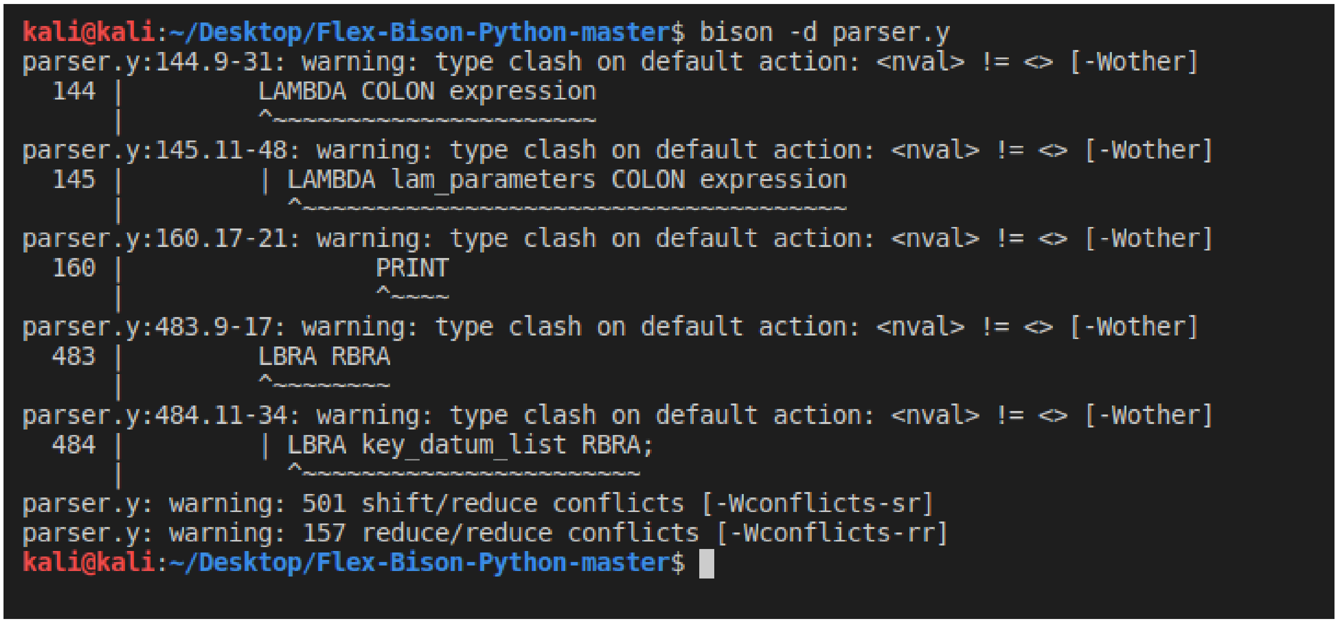

senmatic.py - Flex-Bison-Python-master - Visual Studio Code
File Edit Selection View Go Run Terminal Help
EXPLORER
OPEN EDITORS 1 UNSAVED
print.py Testcases/Fail
if _for.py Testcases/Fail
parser.y /home/kali/Desktop/P...
parser.y 9+
senmatic.py Testcases/Fail
FLEX-BISON-PYTHON-MASTER
Testcases
Fail
assignment.py
class.py
functions.py
if _for.py
import.py
Indentation.py
print.py
print.py
if _for.py
parser.y /home/.../Python-Compiler-vol2
parser.y ./
senm
Testcases > Fail > senmatic.py > ...
PROBLEMS 25
OUTPUT
TERMINAL
DEBUG CONSOLE

```
x=1
y= 10 + 1 - 1.0
z = x +y

print (x)
print(z)
print (x + y)

```

```
kali@kali:~/Desktop/Flex-Bison-Python-master$ ./myParser Testcases/Fail/senmatic.py
>>  1
>>  11.000000
>>  11.000000
Success! You are awesome.
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

5.

Διευκρινήσεις σχετικά με τα warnings



```
kali@kali:~/Desktop/Flex-Bison-Python-master$ bison -d parser.y
parser.y:144.9-31: warning: type clash on default action: <nval> != <> [-Wother]
  144 |         LAMBDA COLON expression
      |         ^~~~~~~~~~~~~~~~~~~~~~
parser.y:145.11-48: warning: type clash on default action: <nval> != <> [-Wother]
  145 |         | LAMBDA lam_parameters COLON expression
      |           ^~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
parser.y:160.17-21: warning: type clash on default action: <nval> != <> [-Wother]
  160 |                 PRINT
      |                 ^~~~~
parser.y:483.9-17: warning: type clash on default action: <nval> != <> [-Wother]
  483 |         LBRA RBRA
      |         ^~~~~~~~~
parser.y:484.11-34: warning: type clash on default action: <nval> != <> [-Wother]
  484 |         | LBRA key_datum_list RBRA;
      |           ^~~~~~~~~~~~~~~~~~~~~~~
parser.y: warning: 501 shift/reduce conflicts [-Wconflicts-sr]
parser.y: warning: 157 reduce/reduce conflicts [-Wconflicts-rr]
kali@kali:~/Desktop/Flex-Bison-Python-master$
```

Σχετικά με τα warnings που εμφανίζονται στην γραμμή 144, 145, 160, 483, 484 αφορούν τις περιπτώσεις που ορισμένα τερματικά και μη τερματικά σύμβολα δεν έχουν λάβει τον τύπο δεδομένων “nval” ενώ οι κανόνες που περιέχουν ανήκουν σε αυτό τον τύπο δεδομένων.

Τέλος, για τις προειδοποιήσεις της γραμμής 501 και 157 συμβαίνουν εάν υπάρχουν δύο ή περισσότεροι κανόνες που ισχύουν για την ίδια ακολουθία εισόδου.