

Εξαμηνιαία Εργασία

Μεταφραστής για Python-like γλώσσα

με χρήση των εργαλείων Flex/Bison

Tokens
Syntax Analysis
BNF Grammar
Lambda Calculus
Python
Lexical Analysis
Bison
Flex

ΜΕΛΗ ΟΜΑΔΑΣ :

ΑΛΕΞΙΟΥ ΣΤΑΥΡΟΣ AM 1059680

ΑΦΕΝΤΑΚΗ ΦΛΩΡΕΝΤΙΑ AM 1059576

ΝΙΚΟΛΑΔΑΚΗΣ ΣΤΕΛΙΟΣ AM 1041833 (236149)

ΣΙΓΟΥΡΟΥ ΑΛΚΗΣΤΙΣ - ΑΙΚΑΤΕΡΙΝΗ AM 1059661

ΠΕΡΙΕΧΟΜΕΝΑ

1.BNF Γραμματική	σελ.2
2.Flex	σελ.8
3.Bison	σελ.10
4.Calculator	σελ.19
5.Lambda Calculus	σελ.19
6.Dictionaries	σελ.20
7.TestCases	σελ.21

1. BNF Γραμματική (Python-like)

Παραδοχές

- ♦ < > - μη τερματικά σύμβολα/κανόνες
- ♦ CAPITALS – tokens
- ♦ %empty – μπορεί ο κανόνας να είναι κενός

program ::=

%empty | <line> <program> | <summerBodies> <program>

summerBodies ::=

<if_stmt> '\n' | <for_stmt> '\n' | <func_def> '\n' | <class_def> '\n'
| <expr_import> '\n'

summerBodies – οι κανόνες οι οποίοι αποτελούνται από Block κώδικα. Ουσιαστικά όσοι χρησιμοποιούν τις παραδοχές indent και dedent .

line ::=

'\n' | <assign> '\n' | <print> '\n' | <returns> '\n'
| <dictionary> '\n' | <func_items> '\n' | <func_setdefault> '\n'
| <lc_expression> '\n' | error '\n'

expr_import::=

IMPORT FILEVAR | IMPORT VAR FROM FILEVAR | IMPORT VAR AS VAR
| FROM VAR IMPORT VAR
| FROM VAR IMPORT AS VAR
| FROM VAR IMPORT '*' | IMPORT VAR FROM VAR

IMPORT statement

FILEVAR – token που αντιστοιχεί σε οποιαδήποτε λέξη που συνοδεύεται από την κατάληξη .py

IMPORT -token για την λέξη key της python “import”

FROM-token για την λέξη key της python “from”

Ως VAR ορίζουμε οποιαδήποτε ακολουθία χαρακτήρων που συμπίπτει με τους κανόνες της Python.

returns::=

VAR '=' <call> | <call> | VAR ':' <call>

Κλήση συνάρτησης

call::=

VAR '(' <param> ')'

class_def::=

<class_head> <clbody>

Ορισμός της κλάσης
CLASS – token για την λέξη
της python “class”

class_head::=

CLASS <class_name> ':' '\n'

class_name::=

VAR

BLOCK – token στο οποίο
συνδέεται το tab. Κάθε
BLOCK αντιστοιχεί σε ένα \t ή
\s.

clbody::=

BLOCK <assign> | BLOCK <method_def> | <clbody> <clbody>
| BLOCK | '\n'

method_def::=

<method_head> <method_body>

Ορισμός Μεθόδου

method_body::=

BLOCK <body> '\n' | <method_body> <method_body>

method_head::=

DEF <method_name> '(' <class_param> ')' ':' '\n'

method_name::=

VAR | INIT

class_param::=

| SELF | SELF ':' <args> | <args>

INIT- token για
την λέξη key
της Python
“init”

SELF-token για
την λέξη key
της Python
“self”

func_def::=

<func_head> <body>

func_head::=

DEF <func_name> '(' <param> ')' ':' '\n'

func_name::=

VAR

param::=

%empty | <args>

args::=

VAR | VAR ',' <args>

for_stmt ::=

FOR VAR IN VAR ':' body <optional_else>

if_stmt ::=

IF <bool_expr> ':' '\n' <body> <else_if> <optional_else>

| IF <bool_expr> ':' '\n' <body> <optional_else>

else_if ::=

%empty | ELIF <bool_expr> ':' '\n' <body>

| <else_if> ELIF <bool_expr> ':' '\n' <body>

optional_else ::=

%empty | ELSE ':' '\n' <body>

body::=

BLOCK <stmt> | BLOCK <body> | <body> < body> | '\n'

stmt::=

<assign> | <print> | <for_stmt> | <if_stmt>

Ορισμός Συνάρτησης
DEF - token για την
λέξη key της python
"def"

Ορισμός For Loop
FOR – token για την
λέξη key της python
"for"

IN – token για την
λέξη key της python
"in"

Ορισμός της IF-
statement

IF-token για την
λέξη key της
python "if"

ELIF - token για
την λέξη key της
python "elif"

ELSE – token για
την λέξη key της
python "else"

lc_expression ::=

```
(' LAMBDA <lc_ID> ':' <lc_expression> '(' <lc_args> ')' ')  
| (' LAMBDA <lc_ID> ':' <lc_expr> '(' <lc_args> ')' ')  
| LAMBDA <lc_ID> ':' <lc_expression> '(' <lc_args> ')  
| LAMBDA <lc_ID> ':' <lc_expr> '(' <lc_args> ')
```

Ορισμός των
συναρτήσεων
Lambda calculus
πάνω στα operators
(+, -, *, /)
LAMBDA – token για
την λέξη key της
python “lambda”

lc_ID ::=

```
VAR | VAR ':' <lc_ID>
```

lc_expr ::=

```
<lc_exp_parts> | <lc_exp_parts> <praxi> <lc_expr>
```

Ικανοποιεί τις αναδρομικές
συναρτήσεις

praxi ::=

```
'+' | '*' | '-' | '/'
```

lc_args ::=

```
<lc_exp_parts> | <lc_exp_parts> ':' <lc_args>
```

lc_exp_parts ::=

```
<expr_i> | <expr_f> | VAR ;
```

dictionary ::=

```
VAR '=' '{' '\n' <obs> '\n' '}' '\n'  
| VAR '=' '{' <obs> '}' '\n'
```

Ορισμός των **Λεξικών**
της Python

obs ::=

```
STRINGB ':' <dvar> | STRINGB ':' <dvar> ':' '\n' <obs>  
| STRINGB ':' <dvar> ':' <obs>
```

Ορισμός των
συναρτήσεων των
λεξικών items() και
setdefault()

func_items ::=

```
VAR ':' ITEMS '(' ')' '\n'
```

ITEMS -token για την
λέξη key της python
“items”

func_setdefault ::=

```
VAR ':' SETDEFAULT '(' <dvar> ':' <dvar> ')' | VAR ':' SETDEFAULT '(' <dvar> ':' NONE ')
```

SETDEFAULT – token
για την λέξη key της
python “setdefault”

dvar ::=

```
STRINGB | INT | FLOAT
```

bool_expr ::=

<expr> <bools> <expr>

bools ::=

EQ | NOTEQ | LESS | GREATER | LESSEQ
| GREATEREQ;

expr ::=

STRINGA | STRINGB | VAR | <expr_i> | <expr_f>

print ::=

PRINT '(' STRINGA ')' | PRINT '(' STRINGB ')' | PRINT '(' <expr_i> ')'

| PRINT '(' <expr_f> ')' | PRINT '(' VAR ')' | PRINT '(' error ')'

assign ::=

VAR '=' VAR | VAR '=' <expr_f> | VAR '=' <expr_i> | VAR '=' STRINGA

| VAR '=' STRINGB | VAR ',' VAR '=' <expr_i> ',' <expr_i>

| VAR ',' VAR '=' <expr_f> ',' <expr_f>

| VAR ',' VAR '=' <expr_i> ',' <expr_f> | VAR ',' VAR '=' <expr_f> ',' <expr_i>

expr_f ::=

FLOAT

| <expr_f> '+' <expr_f> | <expr_f> '*' <expr_f> | <expr_f> '-' <expr_f>

| <expr_f> '/' <expr_f> | <expr_f> '+' <expr_i> | <expr_f> '*' <expr_i>

| <expr_f> '-' <expr_i> | <expr_f> '/' <expr_i> | <expr_f> '***' <expr_i>

| <expr_i> '+' <expr_f> | <expr_i> '*' <expr_f> | <expr_i> '-' <expr_f>

| <expr_i> '/' <expr_f>

expr_i ::=

INT

| <expr_i> '+' <expr_i> | <expr_i> '*' <expr_i> | <expr_i> '-' <expr_i>

| <expr_i> '/' <expr_i> | <expr_i> '***' <expr_i>

Ορισμός Boolean εκφράσεων

Bools – τα αντίστοιχα tokens για τους συμβολισμούς (==, !=, <, >, <=, >=)

STRINGB – token τύπου string, που αναγνωρίζει λέξεις που είναι ανάμεσα σε διπλά εισαγωγικά ("...") αντί για μονά ('...') που είναι το **STRINGA**

Ορισμός της print

PRINT- token για την λέξη key της python "print"

Ορισμός της ανάθεσης μεταβλητών

Ορισμός του Calculator

2. Flex & Bison

Αρχείο Flex

- Ο Flex είναι το εργαλείο που χρησιμοποιεί ο Parser μας , ώστε να «σπάσει» το αρχείο εισόδου σε επιμέρους τμήματα, τα λεγόμενα tokens .
- Ως tokens ορίζουμε λέξεις κλειδιά της Python, πχ. import , class ,if , επίσης ορίζουμε όλους τους ειδικούς χαρακτήρες όπως αριθμητικά σύμβολα ,σημεία στίξης, παρενθέσεις κ.ο.κ
- Στον Flex επίσης μπορούμε να ορίσουμε τον χαρακτήρα εξόδου, τα κενά , αρχικές συνθήκες για την υπαρξη errors, όπως και καταστάσεις (πχ. <MARIO>) διαφορετικές από τις αρχικές του Flex.

```
/* flex will read only one input file*/
%option noyywrap

/*%option case-insensitive      /*flex doesn't distinguish between uppercase and lowercase */

%{
#include "y.tab.h"      /* bison file */
extern YYSTYPE yyval;
/*#include "syntab.h"*/

#include <stdio.h>
#include <stdlib.h>
#include <string.h>

extern int yyval;

void yyerror(char *s);
int yylex();

int block=0;
int control=0;
int mylineno=1;
char c[1];

#define UNDEF 0

/*#define YY_DECL int yylex()*/
struct list_t* lookup(char *name);
void insert(char *name, int len,int lineno);
void incr_scope();
void init_scope();
%}

/*start case of regular expression*/
%x Start_Comment      MARIO

digit      [0-9]
num        {digit}+
real       {num}\.{num}
lambda     "lambda"
name       [a-zA-Z_]([a-zA-Z0-9_]*)?

/*litirely use of special character (""") */
comm       \"\"\"

/*          All PRINTABLE ASCII Character          */

printable  [#-&0-9A-Za-z~ ]*

stringa    \"' {printable} \"'
stringb    \" {printable} \"
%%
```



```
%[\\t]    { /* printf("\\n\\tFirst TAB\\n\\n");          */incr_scope(); BEGIN(MARIO);      return BLOCK;   }
%^       { /* printf("\\n\\tFirst SPACE\\n\\n");        */incr_scope(); BEGIN(MARIO);      return BLOCK;   }

<MARIO>[\\t]    { ++block;      incr_scope();          return BLOCK;   }
<MARIO>[ ]     { ++block;      incr_scope();          |      return BLOCK;   }
<MARIO>[^ ]    { BEGIN(INITIAL); /*printf("%c\\n",yytext[0]);*/ unput(yytext[0]); }

[\\t][ ]      { }

"\\n"         { mylineno++; block=0; init_scope(); return '\\n'; }

"#"[^\\n]*     { printf("\\n\\t\\t Comment at line %d :\\n\\n %s \\n\\n",mylineno,yytext);}

{comm}        { printf("\\n\\n\\t\\t\\t\\t\\tEat up comment from line %d\\n",mylineno); BEGIN(Start_Comment); }
<Start_Comment>:{comm}"\\n"      { mylineno++; printf("\\t\\t\\t\\t\\tuntil line %d\\n\\n\\n",mylineno); BEGIN(INITIAL); }

<Start_Comment>:{comm}[^\\n]+     { printf("\\t\\t\\t\\t\\tError! Expexted new line after comment at line %d\\n",mylineno); }
<Start_Comments>:[^comm]\\n"+   { /*printf("%d\\n",mylineno);*/ }
<Start_Comments>:"\\n"           { ++mylineno; /*printf("%d\\n",mylineno);*/ }

import        { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return IMPORT; }

from          { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return FROM; }
as            { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return AS; }

class         { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return CLASS; }
def           { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return DEF; }

__init__      { printf("Constructor %s was recogized in line %d\\n",yytext,mylineno); return INIT; }
self          { printf("Class member %s was recogized in line %d\\n",yytext,mylineno); return SELF; }

True          { printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); }
False         { printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); }

if            { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return IF; }
elif          { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return ELIF; }
else          { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return ELSE; }

for           { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return FOR; }
in            { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return IN; }
range         { printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); }

:             { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return ':'; }
\\.           { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno);*/ return '.'; }

print { /*printf("Keyword %s was recogized in line %d\\n",yytext,mylineno); */ return PRINT;}

"items"       { /*printf("items");*/ return ITEMS; }
"setdefault"  { /*printf("set");*/ return SETDEFAULT; }
"None"        { /*printf("none");*/ return NONE; }

{lambda}      { /*printf("%s\\n",yytext);*/ return LAMBDA;}

{name} { //    printf( "An identifier: %s in line %d\\n", yytext,mylineno );
              insert(yytext,strlen(yytext),mylineno);

              yylval.symtab_item=lookup(yytext);
              //printf("VariableName: %s \\n",lookup(yytext)->st_name);
              return VAR;
            }

{num} { /*printf("An Integer %s was recogized in line %d\\n",yytext,mylineno);*/ yylval.ival = atoi(yytext); return INT; }
{real} { /*printf("A Float %s was recogized in line %d\\n",yytext,mylineno);*/ yylval.fval = atof(yytext); return FLOAT; }
{stringa} {
    printf("A StringA %s was recogized in line %d\\n",yytext,mylineno);

    yylval.sval = malloc(yyleng * sizeof(char));
    strcpy(yylval.sval, yytext);
    return STRINGA;
}
{stringb} {
    printf("A StringB %s was recogized in line %d\\n",yytext,mylineno);

    yylval.sval = malloc(yyleng * sizeof(char));
    strcpy(yylval.sval, yytext);
    return STRINGB;
}
```

```

"+" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '+'; }
"*" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '*'; }
"_" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '-'; }
"/" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '/'; }
"," { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return ','; }
"'" { printf("Symbol %s was recognized in line %d\n",yytext,mylineno); }
\" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return Q; }

"(" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '('; }
")" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return ')'; }
"{" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '{'; }
"}" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '}'; }
"[" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '['; }
"]" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return ']'; }

"==" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return EQ ; }
"!==" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return NOTEQ ; }
"<" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return LESS ; }
">" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return GREATER ; }
"<=" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return LESSEQ ; }
">=" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return GREATEREQ ; }

"=" { /*printf("Symbol %s was recognized in line %d\n",yytext,mylineno);*/ return '='; }

\.\. {exit(0);}

{digit}[a-zA-Z_]+ {printf("wrong identifier at line no %d\n",mylineno);}

. { printf("\n\nUnrecognized character %s at line %d\n\n",yytext,mylineno); }

%%

```

Αρχείο Bison

- Το εργαλείο Bison χρησιμοποιείται για να ορίσουμε τους διάφορους κανόνες από τους οποίους θα αποτελείται η γλώσσα που περιγράφουμε.
- Οι κανόνες, βασίζονται στην BNF γραμματική που ορίσαμε στην αρχή της αναφοράς μας.
- Αρχικά ορίζουμε ένα Union , το union αντιστοιχεί στις μεταβλητές που θα χρησιμοποιήσουμε στους κανόνες μας. Κάθε μεταβλητή αντιστοιχεί σε έναν τύπο (πχ ival,sval).Ο καθορισμός τύπου ,είναι απαραίτητος στον Bison και συνδέεται με τα actions κάθε κανόνα.
- Στην συνέχεια παραθέτουμε τα tokens, τα οποία χωρίζει ο Lexer. Με την εντολή **%token** .
- Επίσης tokens ή κανόνες που υπάγονται σε συγκεκριμένο τύπο (πχ ival,sval), τα ορίζουμε ομοίως με τις εντολές **%token<τύπος>** ή **%type<τύπος>**.
- Πριν ακριβώς ξεκινήσουν οι κανόνες , οφείλουμε να ορίσουμε τον αρχικό κανόνα τον οποίο θα ψάξει πρώτα να αντιστοιχίσει ο parser μας όταν λάβει αρχείο εισόδου. Η αρχικοποίηση γίνεται με την εντολή **%start** .
- Στο τέλος του αρχείου έχουμε την main μας (κώδικας σε C)

```

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

/*#include "lex.yy.c"*/

#include "syntab.h"

void yyerror(char *);

extern FILE *yyin;
extern FILE *yyout;
extern int yylex();
extern int mylineno;

int counter=1;
int cv=1;

Param* cool;

%}

%union {
    int ival;
    float fval;
    char* sval;
    struct list_t* syntab_item;
    struct Param* parameter;

}

%token BLOCK DEF IF ELIF ELSE PRINT FOR IN CLASS INIT Q ITEMS SETDEFAULT NONE

%token LAMBDA
%token FROM AS

%token<ival> INT
%token<fval> FLOAT
%token<sval> STRINGA STRINGB
%token<syntab_item> VAR
%token<syntab_item> SELF
%token<sval> FILEVAR

%token<sval> IMPORT

%token '\n' ':' '(' ')' ' ' '.' '{' '}' '[' ']'

%left ' ','

%right '='

```

%left EQ NOTEQ LESS GREATER LESSEQ GREATEREQ

%left '+' '-'

%left '*' '/'

%type<ival> expr_i
%type<fval> expr_f
//%type<syntab_item> variable
%type<syntab_item> assign
%type<syntab_item> func_name
%type<syntab_item> class_name
%type<syntab_item> method_name
%type<syntab_item> call
%type<syntab_item> dictionary

%type<syntab_item> lc_ID

%type<parameter> obs

%start program

```
program:
| line program
| summerBodies program
;

summerBodies: if_stmt { printf("If diagnosed\n"); }
| for_stmt { printf("For diagnosed\n"); }
| func_def { printf("Function Definition ENDED at line :%d\n\n",mylineno-1); }
| class_def { printf("Class Definition ENDED at line :%d\n\n",mylineno-1); }
| expr_import '\n'
;

expr_import: IMPORT FILEVAR { printf("Import Python File: %s \n",$2); }
| IMPORT VAR FROM FILEVAR { printf("Import Python File: %s from : %s\n",$2->st_name,$4); }
| IMPORT VAR { printf("Import OK \n"); }
| IMPORT VAR AS VAR { printf("Import OK \n"); }
| FROM VAR IMPORT VAR { printf("Import OK \n"); }
| FROM VAR IMPORT AS VAR { printf("Import OK \n"); }
| FROM VAR IMPORT '*' { printf("Import OK \n"); }
| IMPORT VAR FROM VAR { printf("Import OK \n"); }
;

returns: VAR '=' call
{
if($3->st_type=="Function")
{
printf("%s's Function Call at line:%d\t",$3->st_name,mylineno);
printf("(Function Value is stored to %s)\n\n",$1->st_name);
}
else if($3->st_type=="Class")
{
printf("%s is %s Object\n",$1->st_name,$3->st_name);
$1->st_type=$3->st_name;
}
}

| call
{
if(strcmp(lookup($1->st_name)->st_type,"Function")==0)
// if(st_return=="")
printf("%s's Function Call at line:%d\t(VOID)\n\n",$1->st_name,mylineno);
}

| VAR '.' call
{
if($3->st_type=="Method")
{
printf("%s's Method Call at line:%d\t",$3->st_name,mylineno);
printf("(called by Object %s)\n\n",$1->st_name);
}
}

;
```

```

call: VAR '(' param ')'
>st_type=="Method")
{
    if(lookup($1->st_name)->st_type=="Function"||lookup($1->st_name)->st_type=="Class"||lookup($1->st_name)-
        $$=$1;
    else
        printf("\t\tNOT a Class or a Function member\n\n\n");
}

;

class_def: class_head clbody
;

class_head: CLASS class_name ':' '\n' {printf("%s's Class Definition STARTED at line :%d\n",$2->st_name,mylineno); }

;

class_name:VAR
{ /*printf("Function's Name: %s\n",$1->st_name); */
    $1->st_type="Class";
}

;

clbody: BLOCK assign
| BLOCK method_def
{
    printf("Method Definition ENDED at line :%d\n",mylineno-1);
}
// 6 shift/reduce conflict
// \n => \t 2shift reduce
| clbody clbody
| BLOCK
| '\n'
;

method_def: method_head method_body
;

method_body: BLOCK body '\n' //tzoufio
|BLOCK BLOCK member '\n'
|method_body method_body
|BLOCK method_def {printf("Method Definition ENDED at line %d",mylineno); }
| BLOCK method_body // \n => \t 2shift reduce
| '\n'
;

member:SELF '.' VAR '=' expr { printf("Object's member\n"); }
|SELF '.' VAR '=' VAR { printf("Object's member\n"); }
;

method_head: DEF method_name '(' class_param ')' ':' '\n' { }
;

method_name: VAR
{
    printf("%s's Method Definition STARTED at line :%d\n",$1->st_name,mylineno);
    /*printf("Function's Name: %s\n",$1->st_name); */
    $1->st_type="Method";
}
| INIT
{
    printf("Constructor Definition STARTED at line :%d\n",mylineno);
}
;

class_param:
| SELF
| SELF ',' args
| args
;

func_def: func_head body
;

func_head: DEF func_name '(' param ')' ':' '\n' {printf("%s's Function Definition STARTED at line :%d\n",$2->st_name,mylineno); }
;

func_name: VAR
{ /*printf("Function's Name: %s\n",$1->st_name); */
    $1->st_type="Function"; }
;

param:
| args
;

args: VAR
|VAR ',' args
;

for_stmt : FOR VAR IN VAR ':' body optional_else { printf("%s Control Type \t",$4->st_type);
    printf("Can be executed properly");
    printf("\tFOR\n");
}

```

```

if_stmt: IF bool_expr ':' body else_if optional_else //second \n
;

else_if: /* optional */
|ELIF bool_expr ':' body { printf("\tELSE IF\n");}
|else_if ELIF bool_expr ':' body { printf("\tELSE IF\n");}
;

optional_else: /* optional */
|ELSE ':' body { printf("\tELSE\n");}
;

body: BLOCK stmt // { printf("\tBody\n");}
|BLOCK body // 6 shift/reduce conflict
|body body // 1 shift/reduce conflictss
|'\n'
;

stmt: assign
| print
| for_stmt
| if_stmt
;

line: '\n'
| assign '\n'
| print '\n'
| returns '\n'
| dictionary '\n'
| func_items '\n'
| func_setdefault '\n'
| lc_expression '\n'
| error '\n'
// expr '\n' { printf("\n\tCalculatorMode\t\t\tTesting Area\n"); }
// summerBodies // emfolvrmenes if = 8 shift/reduce & 7 reduce/reduce
// me | xwris \n afxanoun ta reduce conflicts
;

lc_expression: '(' LAMBDA lc_ID ':' lc_expression '('lc_args')' ')' { }
| '(' LAMBDA lc_ID ':' lc_expr '('lc_args')' ')' { { printf("%s at line:%d\n",$3->st_type,mylineno); } }
| LAMBDA lc_ID ':' lc_expression '('lc_args')' { }
| LAMBDA lc_ID ':' lc_expr '('lc_args')' { { printf("%s at line:%d\n",$2->st_type,mylineno); } }
|lc_practical
;

lc_practical: LAMBDA VAR ':' VAR '+' expr_i '('expr_i')' { { $2->st_val=$8;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$2->st_name,$2->st_val);
printf("%d\n",$4->st_val+$6); } }
| LAMBDA VAR ':' VAR '-' expr_i '('expr_i')' { { $2->st_val=$8;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$2->st_name,$2->st_val);
printf("%d\n",$4->st_val-$6); } }
| LAMBDA VAR ':' VAR '*' expr_i '('expr_i')' { { $2->st_val=$8;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$2->st_name,$2->st_val);
printf("%d\n",$4->st_val*$6); } }
| LAMBDA VAR ':' VAR '/' expr_i '('expr_i')' { { $2->st_val=$8;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$2->st_name,$2->st_val);
printf("%d\n",$4->st_val/$6); } }
| '(' LAMBDA VAR ':' VAR '+' expr_i '('expr_i')' ')' { { $3->st_val=$9;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$3->st_name,$3->st_val);
printf("%d\n",$5->st_val+$7); } }
| '(' LAMBDA VAR ':' VAR '-' expr_i '('expr_i')' ')' { { $3->st_val=$9;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$3->st_name,$3->st_val);
printf("%d\n",$5->st_val-$7); } }
| '(' LAMBDA VAR ':' VAR '*' expr_i '('expr_i')' ')' { { $3->st_val=$9;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$3->st_name,$3->st_val);
printf("%d\n",$5->st_val*$7); } }
| '(' LAMBDA VAR ':' VAR '/' expr_i '('expr_i')' ')' { { $3->st_val=$9;
printf("LAMBDA at line:%d\t%s\t%d:\t",mylineno,$3->st_name,$3->st_val);
printf("%d\n",$5->st_val/$7); } }
;

lc_ID : VAR { $1->st_type="Lambda"; $$=$1; }
|VAR ',' lc_ID { $1->st_type="Lambda"; $$=$1; }
;

lc_expr: lc_exp_parts
|lc_exp_parts praxi lc_expr
;

praxi: '+' | '*' | '-' | '/';

lc_args: lc_exp_parts { /*$$=$1;*/ }
|lc_exp_parts ',' lc_args { /*$$=$1;*/ }
;

```

```

lc_exp_parts: expr_i
| expr_f
| VAR
;

dictionary : VAR '=' '{' '\n' obs '\n' '}' '\n' { $1->st_type="Dictionary\n";
// printf("OBS=%s\n",$5);
// strcpy(lookup_P($1)->param_name,$5);
// printf("Parameter : %s\n", print_P($1->parameters)->param_name);
//print_P($1);
// $$=$1; printf("Dictionary : %s\n",$$->st_name);
}
|VAR '=' '{' obs '}' '\n' { $1->st_type="Dictionary";
// printf("OBS=%s\n",$4);
// strcpy(lookup_P($1)->param_name,$4);
// printf("Parameter : %s\n",lookup_P($1)->param_name);
// print_P($1);
//
// $$=$1;
// printf("Dictionary : %s\n",$$->st_name);
}
;
obs : STRINGB ':' dvar { // printf("OBS=%s\n",$$->param_name);
//$$=insert_P($1);
//$$->val=$3;
}
| STRINGB ':' dvar ',' '\n' obs { //printf("OBS=%s\n",$$->param_name);
//$$=insert_P($1);
//$$->val=$3;
//$$->next=$6;
}
| STRINGB ':' dvar ',' obs { //printf("OBS=%s\n",$$->param_name);
//$$=insert_P($1);
//$$->val=$3;
//$$->next=$5;
}
;

func_items : VAR '.' ITEMS '(' ')' {printf("Func Item\n");}
;

func_setdefault : VAR '.' SETDEFAULT '(' dvar ',' dvar ')' {printf("Func Set Default !!\n");}
|VAR '.' SETDEFAULT '(' dvar ',' NONE ')' {printf("Func Set Default\tYou got it bitch !!\n");}
;

```



```
dvar : STRINGB{${<sval>$=$1;}| INT {${<ival>$=$1;}| FLOAT {${<fval>$=$1;};
```

```
bool_expr: expr bools expr
;
```

```
bools:EQ|NOTEQ
|LESS // { printf("LESS\n "); } //if ($1.val<$3.val) return TRUE; else FALSE; }
|GREATER|LESSEQ|GREATEREQ;
```

```
print: PRINT '(' STRINGA ')' { printf("PRINT:\t%s\n",$3); }
| PRINT '(' STRINGB ')' { printf("PRINT:\t%s\n",$3); }
| PRINT '(' expr_i ')' { printf("PRINT:\t%d\n",$3); }
| PRINT '(' expr_f ')' { printf("PRINT:\t%f\n",$3); }
| PRINT '(' VAR ')' {
{
if($3->st_type=="Float")
printf("PRINT:\t%f (%s)\n",$3->st_fval,$3->st_name);
if($3->st_type=="Integer")
printf("PRINT:\t%d (%s)\n",$3->st_ival,$3->st_name);
if($3->st_type=="String")
printf("PRINT:\t%s (%s)\n",$3->st_sval,$3->st_name);
}
| PRINT '(' error ')' { yyerrok;printf("You can't print that %d\n ",yyerrok);}
;
```

```
assign: VAR '=' VAR
{
$1->st_type=$3->st_type;
if($1->st_type=="Float")
$1->st_fval=$3->st_fval;
if($1->st_type=="Integer")
$1->st_ival=$3->st_ival;
if($1->st_type=="String")
$1->st_sval=$3->st_sval;
}
```

```
|VAR '=' expr_f { $1->st_type="Float"; $1->st_fval=$3;
//printf("\tVariable: %s \tValue:%f \tType:%s\n",$1->st_name,$1->st_fval,$1->st_type);
}
| VAR '=' expr_i { $1->st_type="Integer"; $1->st_ival=$3;
//printf("\tVariable: %s \tValue:%d \tType:%s\n",$1->st_name,$1->st_ival,$1->st_type);
}
| VAR '=' STRINGA // { $1->st_type="String"; $1->st_sval=$3;
printf("\tVariable: %s \tValue:%s \tType:%s\n",$1->st_name,$1->st_sval,$1->st_type);
}
| VAR '=' STRINGB // { $1->st_type="String"; $1->st_sval=$3;
printf("\tVariable: %s \tValue:%s \tType:%s\n",$1->st_name,$1->st_sval,$1->st_type);
}
| VAR','VAR '=' expr_i ',' expr_i {
{
$1->st_type="Integer"; $1->st_ival=$5;
$3->st_type="Integer"; $3->st_ival=$7;
// printf("\tVariable: %s \tValue:%d \tType:%s\n",$1->st_name,$1->st_ival,$1->st_type);
// printf("\tVariable: %s \tValue:%d \tType:%s\n",$3->st_name,$3->st_ival,$3->st_type);
}
}
| VAR','VAR '=' expr_f ',' expr_f {
{
$1->st_type="Float"; $1->st_fval=$5;
$3->st_type="Float"; $3->st_fval=$7;
// printf("\tVariable: %s \tValue:%f \tType:%s\n",$1->st_name,$1->st_fval,$1->st_type);
// printf("\tVariable: %s \tValue:%f \tType:%s\n",$3->st_name,$3->st_fval,$3->st_type);
}
}
| VAR','VAR '=' expr_i ',' expr_f {
{
$1->st_type="Integer"; $1->st_ival=$5;
$3->st_type="Float"; $3->st_fval=$7;
printf("\tVariable: %s \tValue:%d \tType:%s\n",$1->st_name,$1->st_ival,$1->st_type);
printf("\tVariable: %s \tValue:%f \tType:%s\n",$3->st_name,$3->st_fval,$3->st_type);
}
}
| VAR','VAR '=' expr_f ',' expr_i {
{
$1->st_type="Float"; $1->st_fval=$5;
$3->st_type="Integer"; $3->st_ival=$7;
// printf("\tVariable: %s \tValue:%f \tType:%s\n",$1->st_name,$1->st_fval,$1->st_type);
// printf("\tVariable: %s \tValue:%d \tType:%s\n",$3->st_name,$3->st_ival,$3->st_type);
}
}
;
```



```

expr:  expr_i                                { /*fprintf(yyout, "%i\n", $1); printf("%i\n",$1); */}
      | expr_f                                { /*fprintf(yyout, "%f\n", $1); printf("%f\n",$1); */}
      | STRINGA                              { /*fprintf(yyout, "%s\n", $1); printf("%s\n",$1); */}
      | STRINGB                              { /*fprintf(yyout, "%s\n", $1); printf("%s\n",$1); */}
      | VAR                                  { }
      ;

```

```

expr_f:  FLOAT                                { $$ = $1; /*printf("\t\t\t\t\t\t\t%f \n",$1);*/ }
      | expr_f '+' expr_f                    { $$ = $1 + $3; printf("%f + %f\n",$1,$3); }
      | expr_f '*' expr_f                    { $$ = $1 * $3; printf("%f * %f\n",$1,$3); }
      | expr_f '-' expr_f                    { $$ = $1 - $3; printf("%f - %f\n",$1,$3); }
      | expr_f '/' expr_f                    { $$ = $1 / $3; printf("%f / %f\n",$1,$3); }

      | expr_f '+' expr_i                    { $$ = $1 + $3; printf("%f + %i\n",$1,$3); }
      | expr_f '*' expr_i                    { $$ = $1 * $3; printf("%f * %i\n",$1,$3); }
      | expr_f '-' expr_i                    { $$ = $1 - $3; printf("%f - %i\n",$1,$3); }
      | expr_f '/' expr_i                    { $$ = $1 / $3; printf("%f / %i\n",$1,$3); }
      {
        float temp=1;
        printf("%f ** %i\n",$1,$4) ;
        while($4!=0){
          temp=temp*$1;
          $4--;}
        $$=temp;
      }

      | expr_i '+' expr_f                    { $$ = $1 + $3; printf("%i + %f\n",$1,$3); }
      | expr_i '*' expr_f                    { $$ = $1 * $3; printf("%i * %f\n",$1,$3); }
      | expr_i '-' expr_f                    { $$ = $1 - $3; printf("%i - %f\n",$1,$3); }
      | expr_i '/' expr_f                    { $$ = $1 / $3; printf("%i / %f\n",$1,$3); }
      ;

```

```

expr_i:  INT                                { $$ = $1; /*printf("\t\t\t\t\t\t\t%i \n",$1);*/}
      | expr_i '+' expr_i                    { $$ = $1 + $3; printf("%i + %i\n",$1,$3); }
      | expr_i '*' expr_i                    { $$ = $1 * $3; printf("%i * %i\n",$1,$3); }
      | expr_i '-' expr_i                    { $$ = $1 - $3; printf("%i - %i\n",$1,$3); }
      | expr_i '/' expr_i                    { $$ = $1 / $3; printf("%i / %i\n",$1,$3); }

      | expr_i '*' '*' expr_i                {
        int temp=1;
        printf("%i ** %i\n",$1,$4) ;
        while($4!=0){
          temp=temp*$1;
          $4--;}
        $$=temp;
      }

```

```

%%

void yyerror(char *s) {
    fprintf(stderr, "%s\n", s);
    exit(1);
}

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

    ++argv; --argc;
    if ( argc > 0 )
        yyin = fopen( argv[0], "r" );
    else
        yyin = stdin;

    do {
        yyparse();
    } while(!feof(yyin));

    yyout = fopen ( "output", "wb" );
    symtab_dump(yyout);
    fclose(yyout);

    exit(0);
}

```


4. Υλοποίηση Dictionaries, Items() & Setdefault()

a. Dictionaries

```
dictionary : VAR '=' '{' '\n' obs '\n' '}' '\n' { $1->st_type="Dictionary"; $$=$1;
printf("Dictionary : %s\n",$$->st_name); printf("OBS=%s\n",$$);
strcpy($1->parameters->param_name,"hello");
printf("Parameter : %s\n",$1->parameters->param_name);}
|VAR '=' '{' obs '}' '\n' { $1->st_type="Dictionary"; $$=$1;
printf("Dictionary : %s\n",$$->st_name);
strcpy($1->parameters->param_name,$4);
printf("Parameter : %s\n",$1->parameters->param_name); }
;

obs : STRINGB ':' dvar { $$=$1; printf("OBS=%s\n",$$);}
| STRINGB ':' dvar ',' '\n' obs { $$=$1; printf("OBS=%s\n",$$);}
| STRINGB ':' dvar ',' obs { $$=$1; printf("OBS=%s\n",$$);}
;

dvar : STRINGB| INT | FLOAT |;
```

STRINGB – token τύπου string, που αναγνωρίζει λέξει που είναι ανάμεσα σε διπλά εισαγωγικά ("...") αντι για μονα ('...') που είναι το STRINGA

```
stringa \'{printable}\'  
stringb \'{printable}\'
```

b. Items()

Η items() είναι μια μέθοδος των λεξικών της python, η οποία επιστρέφει τα ορίσματα του λεξικού για το οποίο καλείται. Η σύνταξη της είναι της μορφής `dictionary.items()`, όπου `dictionary` το όνομα του λεξικού.

```
func_items : VAR '.' ITEMS '(' ')' {printf("Func Item\n");};
```

c. Setdefault()

Η setdefault() είναι μια μέθοδος των λεξικών της python, η οποία επιστρέφει την τιμή ενός συγκεκριμένου ορίσματος, του λεξικού, για το οποίο καλείται. Η σύνταξη της είναι της μορφής `dictionary.setdefault(keyname, value)`, όπου `dictionary` το όνομα του λεξικού, `keyname` το όνομα του ορίσματος και `value` η τιμή του. Η default τιμή για το `value` είναι `none`.

```
func_setdefault : VAR '.' SETDEFAULT '(' dvar ',' dvar ')' {printf("Func Set Default !!\n");}
|VAR '.' SETDEFAULT '(' dvar ',' NONE ')' {printf("Func Set Default\t");};
;

dvar : STRINGB| INT | FLOAT ;
```

NONE – token της λέξης key της python "None"
SETDEFAULT – token της λέξης key της python "setdefault"

5. Test Cases

- Προαιρετική εισαγωγή από modules

```
import extern_file.py
import test2 from test.py
```

```
import test3
import test4 as four
```

```
from urllib import download
from pandas import as pd
from sqlite3 import *
```

```
Python file: extern_file.py was recognized in line 1
Import Python File: extern_file.py
Python file: test.py was recognized in line 2
Import Python File: test2 from : test.py
Import OK
Import OK
Import OK
Import OK
Import OK
```

- Αρχικοποίηση μεταβλητών (εντολές ανάθεσης) σύμφωνα με τους κανόνες ονομάτων μεταβλητών της Python

```
year , exper= 2016, 2.5
```

- Ορισμός κλάσης με τον επικείμενο constructor και δημιουργία αντικειμένου

```
class worker :
    name=""
    salary=0
    months=0
    def __init__(self):
        self.name="Unknown"
        self.salary=520
        self.months=1
    def stoixia(name):
        name=giorgos
        print (name)
```

```
worker1=worker()
```

```
worker's Class Definition STARTED at line :18
Constructor __init__ was recognized in line 21
Constructor Definition STARTED at line :21
Class member self was recognized in line 21
Class member self was recognized in line 22
Object's member
Class member self was recognized in line 23
Object's member
Class member self was recognized in line 24
Object's member
stoixia's Method Definition STARTED at line :25
Method Definition ENDED at line 29Method Definition ENDED at line :28
Class Definition ENDED at line :28
```

```
worker1 is worker Object
stoixia's Method Call at line:30 (called by Object worker1)
```

- Ορισμός συνάρτησης και κλήση της

```
def sinartisi(year,money):  
    print(Apolieste!)
```

```
sinartisi(year,salary)
```

```
Variable: year  Value:2016      Type:Integer  
Variable: exper Value:2.500000  Type:Float  
sinartisi's Function Definition STARTED at line :73  
  
Unrecognized character ! at line 73  
  
Function Definition ENDED at line :76  
  
sinartisi's Function Call at line:77    (VOID)
```

- Υποστήριξη σχολίων

```
# lambda z:lambda y: y+3(z)(5)
```

```
# lambda z:lambda y: y+3(z)(5)
```

```
"""
```

```
Hello Over There My Friends
```

```
"""
```

```
Eat up comment from line 32  
until line 35
```

- Εντολές βρόγχου και συνθήκης If statement και For loop statement

```
temp=0
for i in record:
    salary=temp
    salary=680
    print(salary)

if salary<1000 :
    print("Raise")
    salary=800+800*0.2
    print(salary)
elif salary > 1500:
    print("Good Work")
else :
    salary=1000
    print(salary)
```

```
PRINT: 680 (salary)
Dictionary
Control Type Can be executed properly FOR
For diagnosed
PRINT: "Raise"
800 * 0.200000
800 + 160.000000
PRINT: 960.000000 (salary)
PRINT: "Good Work"
ELSE IF
PRINT: 1000 (salary)
ELSE
If diagnosed
```

Παρατηρούμε ότι δοκιμάζεται η δημιουργία αντικειμένου και η κλήση μεθόδου κλάσης η οποία δεν έχει ακόμα δημιουργηθεί. Ο Parser μας σε τέτοιες περιπτώσεις εκτυπώνει ανάλογο μήνυμα .

```
worker1=worker()
worker1.stoixia()
```

```
class worker :
    name=""
    salary=0
    months=0
    def __init__(self):
        self.name="Unknown"
        self.salary=520
        self.months=1
    def stoixia(name):
        name=giorgos
        print (name)
```

NOT a Class or a Function member

NOT a Class or a Function member

- Lambda Calculus

```
lambda x,y,z: x +y-z (2,3,4)
(lambda z: z + 1(5))
lambda x: x + 1(4)
# lambda z:lambda y: y+3(z)(5)
```

```
Lambda at line:64
LAMBDA at line:65      z      5:      6
LAMBDA at line:66      x      4:      5

      Comment at line 67 :

# lambda z:lambda y: y+3(z)(5)
```

- Dictionaries

```
record = {
    "First" : "Mike",
    "Last" : "FAsolakis",
    "Salary" : 1200
}

record.items()
record.setdefault("Salary",520)
```

```
Dictionary : record
Func Item
Func Set Default !!
```


- **Υποστήριξη εμφάνισης μηνυμάτων και error handling**

Στην έκδοση του parser pytho που παρουσιάζουμε διαχειριζόμαστε λάθη, όπως το όνομα μεταβλητής να ξεκινάει με αριθμό, να υπάρχουν σύμβολα μετά από τερματισμό πολλαπλών γραμμών σχολίων ("""), οι οποίοι μάλιστα, λαμβάνουν χώρα στον λεκτικό αναλυτή και μία πληθώρα από ελέγχους στους τύπους μεταβλητών ανάλογα την απαίτηση του εκάστοτε κανόνα, οι οποίοι γίνονται στον γραμματικό αναλυτή.

```
2metabliti
wrong identifier at line no 1
"""

                                Eat up comment from line 2
foasdp
odpof
o""lathos

                                Error! Expexted new line after comment at line
def sinartisi():
    blasis=5
    why=not
o""

                                until line 10

def sinartisi():
sinartisi's Function Definition STARTED at line :11
    vlasis="vlasteros"
    koula="soula"
print(vlasis)
Function Definition ENDED at line :12

PRINT: "vlasteros" (vlasis)
sinartisi()
sinartisi's Function Call at line:14    (VOID)

miOrismeniSinartisi()
    NOT a Class or a Function member

returning=miOrismeniSinartisi()
    NOT a Class or a Function member

rerurning=sinartisi()
sinartisi's Function Call at line:17    (Function Value is stored to rerurning)
```