

[**ΑΡΧΕΣ**](https://eclass.upatras.gr/courses/CEID1091/) **ΓΛΩΣΣΩΝ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ**

**ΚΑΙ ΜΕΤΑΦΡΑΣΤΩΝ**

Εργασια Flex & Bison

**ΟΝΟΜΑΤΕΠΩΝΥΜΑ / ΑΜ:**

Χατζηγαβριήλ Ανδρέας , ΑΜ:1056645

Παλαμάς Ανδρέας , ΑΜ: 1049789

Γεωργίου Γεώργιος , ΑΜ:1049797

Κυριάκου Χρήστος , ΑΜ: 1050002

**Περιεχόμενα**

**Ι. ΕΙΣΑΓΩΓΗ**: ……..…………..………………………………………….. Σελ.3

**II. ΣΥΝΤΑΚΤΙΚΟΣ ΟΡΙΣΜΟΣ ΓΡΑΜΜΑΤΙΚΗΣ** ………..………….. Σελ.4

Α. Περιγραφή γραμματικής της γλώσσας σε BNF .……………… Σελ.4

**III. ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ** …….…….......………..……….. Σελ.12

**IV. ΟΛΟΚΛΗΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ** ……………………….………... Σελ.20

Α. Αρχείο LEXER ………….....................……....………………. Σελ.20

Β. Αρχείο BISON ……….………..………………………...……. Σελ.23

Γ. Αρχείο εκτύπωσης στην κονσόλα …..……………..…….….… Σελ.28

Δ. Αρχείο εισαγωγής Python …………………..…...……………. Σελ.32

**Ι . ΕΙΣΑΓΩΓΗ:**

Σε αυτό το Project ασχοληθήκαμε με την λεκτική αλλά και συντακτική ανάλυση ενός υποσυνόλου της γλώσσας προγραμματισμού Python.

Σύμφωνα με το documentation της Python και με την βοήθεια των εργαλείων Flex & Bison υλοποιήσαμε τα παρακάτω:

* Εισαγωγή από modules
* Αρχικοποίηση μεταβλητών
* Ορισμός κλάσης με τον επικείμενο constructor και δημιουργία αντικειμένου
* Ορισμός συνάρτησης και κλήση της
* Υποστήριξη σχολίων
* Εντολές βρόγχου και συνθήκης ( If , For loop )
* Υποστήριξη εμφάνισης μηνυμάτων
* Υλοποίησης των operators +, -, \*, /
* Υλοποίηση Lambda Calculus συναρτήσεων
* Αρχικοποίηση dictionaries

Δημιουργήσαμε συνολικά 4 αρχεία τα οποία είναι ο λεξικός αναλυτής (lexer), ο συντακτικός αναλυτής (parser), ένα αρχείο c και ένα αρχείο Python .

Ο λεξικός αναλυτής περιέχει το αλφάβητο της γλώσσας μας, δηλαδή καθορίζουμε στο πρόγραμμα μας τι να αναγνωρίζει η γλώσσα μας και τί όχι. Διαχωρίζει τους χαρακτήρες του προγράμματος σε ομάδες. Τις λέξεις που αναγνωρίζει ο λεξικός μας αναλυτής λέγονται και tokens.

Ο συντακτικός αναλυτής καθορίζει την σωστή δομή που θα έχει η γλώσσα που ορίσαμε στο λεξικό αναλυτή για να υπακούει στην γλώσσα που θέλουμε να φτιάξουμε.

Το αρχείο c περιέχει κώδικα ο οποίος μας εκτυπώνει τα αποτελέσματα στην κονσόλα.

Το αρχείο python (.py) περιέχει κώδικα γραμμένο σε γλώσσα python το οποίο είναι και το αρχείο εισόδου στον συντακτικό αναλυτή. Ο κώδικας ο οποίος καλύπτει όλες τις παραπάνω λειτουργίες που προαναφέραμε. Είναι το αρχείο το οποίο μας επιβεβαιώνει την ορθότητα της ανάλυσης μας.

**II. ΣΥΝΤΑΚΤΙΚΟΣ ΟΡΙΣΜΟΣ ΓΡΑΜΜΑΤΙΚΗΣ**

Α. Περιγραφή γραμματικής της γλώσσας σε BNF

<**program**> ::= <**line**> | <**program**> <**line**>

<**line**> ::= <**if\_stmt**> | <**for\_statement**> | <**def**> NEWLINE INDENT

| <**def**> NEWLINE <**indent2**> | <**def**> NEWLINE | <**def\_call**>

| <**comments**> NEWLINE | <**action**> | <**print**> NEWLINE

| <**import\_statement**> NEWLINE | <**class**> NEWLINE

| <**create\_object**> NEWLINE | <**dictionaries**> NEWLINE

| <**lambda\_expr**> NEWLINE | <**calc\_assigmnet**> NEWLINE

| NEWLINE

<**action**> ::= INDENT <**line**> | <**indent2**> <**line**> | <**indent3**> <**line**>

| <**indent4**> <**line**> | <**indent5**> <**line**>

<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

<**if\_stmt**>::= IF IDENTIFIER <**operators**> <**data\_type**> COLON NEWLINE

<**action**>

<**for\_statement**> ::= FOR IDENTIFIER IN <**def\_call**> COLON NEWLINE <**action**>

<**operators**> ::= AND\_OP | OR\_OP | EQ\_OP | GE\_OP | LE\_OP | NE\_OP

<**optional**\_**parameters**> ::= IDENTIFIER

| <**optional\_parameters**> COMMA IDENTIFIER

| COMMA

<**def**> ::= DEF IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR COLON

<**def\_call**> ::= IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <**data\_type**> COMMA

<**data\_type**> R\_PAR

<**comments**> ::= COMMENT

<**print**> ::= PRINT L\_PAR <**data\_type**> R\_PAR

<**import\_statement**> ::=

FROM IDENTIFIER IMPORT IDENTIFIER AS IDENTIFIER

| FROM IDENTIFIER IMPORT IDENTIFIER

| FROM IDENTIFIER IMPORT MUL

| IMPORT IDENTIFIER AS IDENTIFIER

| IMPORT IDENTIFIER

| FROM IDENTIFIER IMPORT IDENTIFIER COMMA IDENTIFIER

<**class**> ::= CLASS IDENTIFIER COLON NEWLINE INDENT <**class\_constructor**>

| CLASS IDENTIFIER L\_PAR R\_PAR COLON NEWLINE INDENT <**class\_constructor**>

<**class\_constructor**> ::= DEF IDENTIFIER L\_PAR <**constructor\_parameters**>

R\_PAR COLON NEWLINE <**constructor\_body**>

<**constructor\_parameters**> ::= SELF | IDENTIFIER

| <**constructor\_parameters**> COMMA IDENTIFIER

<**constructor\_body**> ::= <**object\_creation**>

| <**constructor\_body**><**object\_creation**>

<**object\_creation**> ::= INDENT SELF DOT IDENTIFIER ASSIGN

<**data\_type**> NEWLINE

<**create\_object**> ::= IDENTIFIER ASSIGN <**class\_call**>

<**class\_call**> ::= IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <**data\_type**> COMMA

<**data\_type**> R\_PAR

<**dictionaries**> ::= IDENTIFIER ASSIGN L\_BRACE <**dictionary\_data**> R\_BRACE

| IDENTIFIER ASSIGN IDENTIFIER L\_PAR L\_BRACK L\_PAR

<**dictionary\_data**> R\_PAR R\_BRACK R\_PAR

| IDENTIFIER ASSIGN IDENTIFIER L\_PAR <**dictionary\_data**>

<**optional\_parameters**> <**dictionary\_data**> R\_PAR

<**dictionary\_data**> ::= <**data\_type**> COLON <**data\_type**>

| <**data\_type**> COLON <**data\_type**> COMMA <**dictionary\_data**>

| <**data\_type**> COMMA <**data\_type**><**optional\_parameters**>

| IDENTIFIER ASSIGN <**data\_type**>

<**lambda\_expr**> ::= IDENTIFIER ASSIGN LAMBDA IDENTIFIER COLON

<**expressions**>

| LAMBDA IDENTIFIER COLON <**expressions**>

<**expressions**> ::= <**lambda\_expr**> | <**term**> | <**expressions**> MINUS <**term**>

| <**expressions**> PLUS <**term**> | <**expressions**> MUL <**term**>

| <**expressions**> DIV <**term**> | <**def\_call**>

<**term**> ::= <**data**\_**type**>

<**calc\_assignment**> ::= IDENTIFIER ASSIGN <**int\_op**>

<**int\_op**> ::= <**int\_data**>

| <**int\_op**> PLUS <**int\_data**> | <**int\_op**> MINUS <**int\_data**>

| <**int\_op**> MUL <**int\_data**> | <**int\_op**> DIV <**int\_data**>

<**int\_data**> ::= INTEGER | IDENTIFIER

**Επεξήγηση:**

Κατασκευάσαμε την πιο πάνω BNF η οποία υλοποιεί το υποσύνολο της γλώσσας της Python όπως προαναφέραμε πιο πάνω.

Αρχικά ξεκινάμε πάντα με το starting symbol το οποίο στην περίπτωση μας είναι ο κανόνας **program**.

Το program το οποίο περιέχει το line και την αναδρομή του program με το line για την εκτέλεση περισσότερες από μία φορές του line.

Στην συνέχεια υπάρχει ο κανόνας line ο οποίος περιέχει ουσιαστικά όλο μας το πρόγραμμα. Δηλαδή την εισαγωγή των modules, την αρχικοποίηση μεταβλητών, ορισμό κλάσης, τον επικείμενο constructor, δημιουργία αντικειμένου, ορισμό συνάρτηση και κλήση της , σχόλια, εντολές βρόγχου και συνθήκης, lambda calculus και αρχικοποίηση dictionaries.

Κάτω από το line υπάρχουν όλοι οι βοηθητικοί κανόνες για την εκτέλεση των πιο πάνω λειτουργιών.

Για τον ορισμό της εντολής IMPORT MODULE χρησιμοποιούνται οι πιο κάτω κανόνες:

<**import\_statement**> ::=

FROM IDENTIFIER IMPORT IDENTIFIER AS IDENTIFIER

| FROM IDENTIFIER IMPORT IDENTIFIER

| FROM IDENTIFIER IMPORT MUL

| IMPORT IDENTIFIER AS IDENTIFIER

| IMPORT IDENTIFIER

| FROM IDENTIFIER IMPORT IDENTIFIER COMMA IDENTIFIER

Για τον ορισμό της αρχικοποίησης μεταβλητών χρησιμοποιούνται οι πιο κάτω κανόνες:

<**calc\_assignment**> ::= IDENTIFIER ASSIGN <**int\_op**>

<**int\_op**> ::= <**int\_data**>

| <**int\_op**> PLUS <**int\_data**> | <**int\_op**> MINUS <**int\_data**>

| <**int\_op**> MUL <**int\_data**> | <**int\_op**> DIV <**int\_data**>

<**int\_data**> ::= INTEGER | IDENTIFIER

Για τον ορισμό του ορισμού κλάσης, δημιουργία του επικείμενου constructor , δημιουργία αντικειμένου και κλήση της χρησιμοποιούνται οι πιο κάτω κανόνες:

<**class**> ::= CLASS IDENTIFIER COLON NEWLINE INDENT <**class\_constructor**>

| CLASS IDENTIFIER L\_PAR R\_PAR COLON NEWLINE INDENT <**class\_constructor**>

<**class\_constructor**> ::= DEF IDENTIFIER L\_PAR <**constructor\_parameters**>

R\_PAR COLON NEWLINE <**constructor\_body**>

<**constructor\_parameters**> ::= SELF | IDENTIFIER

| <**constructor\_parameters**> COMMA IDENTIFIER

<**constructor\_body**>::= <**object\_creation**>

|<**constructor\_body**><**object\_creation**>

<**object\_creation**> ::= INDENT SELF DOT IDENTIFIER ASSIGN

<**data\_type**> NEWLINE

<**create\_object**> ::= IDENTIFIER ASSIGN <**class\_call**>

<**class\_call**> ::= IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <**data\_type**> COMMA <**data\_type**> R\_PAR

<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

Για τον ορισμό συνάρτησης και κλήσης της χρησιμοποιούνται οι πιο κάτω κανόνες:

<**def**> ::= DEF IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR COLON

<**def\_call**> ::= IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> COMMA

<**data\_type**> R\_PAR

<**optional**\_**parameters**> ::= IDENTIFIER

| <**optional\_parameters**> COMMA IDENTIFIER

| COMMA

<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

Για τον ορισμό σχολίων χρησιμοποιείτε ο πιο κάτω κανόνας:

<**comments**> ::= COMMENT

Για τον ορισμό της εντολής βρόγχου IF χρησιμοποιούνται οι πιο κάτω κανόνες:

<**if\_stmt**> ::= IF IDENTIFIER<operators><data\_type>COLON NEWLINE <**action**>

<**operators**> ::= AND\_OP | OR\_OP | EQ\_OP | GE\_OP | LE\_OP | NE\_OP

<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

<**action**> ::= INDENT <**line**> | <**indent2**> <**line**> | <**indent3**> <**line**>

| <**indent4**> <**line**> | <**indent5**> <**line**>

Για τον ορισμό της εντολής συνθήκης FOR χρησιμοποιούνται οι πιο κάτω κανόνες:

<**for\_statement**> ::= FOR IDENTIFIER IN <def\_call> COLON NEWLINE <**action**>

<**def\_call**> ::= IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> COMMA <**data\_type**> R\_PAR

<**action**> ::= INDENT <**line**> | <**indent2**> <**line**> | <**indent3**> <**line**>

| <**indent4**> <**line**> | <**indent5**> <**line**>

Για τον ορισμό της συνάρτησης lambda calculus χρησιμοποιούνται οι πιο κάτω κανόνες:

<**lambda\_expr**> ::= IDENTIFIER ASSIGN LAMBDA IDENTIFIER COLON

<**expressions**>

| LAMBDA IDENTIFIER COLON <**expressions**>

<**expressions**> ::= <**lambda\_expr**> | <**term**> | <**expressions**> MINUS <**term**>

| <**expressions**> PLUS <**term**> | <**expressions**> MUL <**term**>

| <**expressions**> DIV <**term**> | <**def\_call**>

<**term**> ::= <**data**\_**type**>

<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

<**def\_call**> ::= IDENTIFIER L\_PAR <**optional\_parameters**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> R\_PAR

| IDENTIFIER L\_PAR <**data\_type**> COMMA <data\_type> COMMA

<**data\_type**> R\_PAR

<**optional**\_**parameters**> ::= IDENTIFIER

| <**optional\_parameters**> COMMA IDENTIFIER

| COMMA

Για τον ορισμό της αρχικοποίησης dictionaries χρησιμοποιούνται οι πιο κάτω κανόνες:

<**dictionaries**> ::= IDENTIFIER ASSIGN L\_BRACE <**dictionary\_data**> R\_BRACE

| IDENTIFIER ASSIGN IDENTIFIER L\_PAR L\_BRACK L\_PAR

<**dictionary\_data**> R\_PAR R\_BRACK R\_PAR

| IDENTIFIER ASSIGN IDENTIFIER L\_PAR <**dictionary\_data**>

<**optional\_parameters**><**dictionary\_data**> R\_PAR

<**dictionary\_data**> ::= <**data\_type**> COLON <**data\_type**>

| <**data\_type**> COLON <**data\_type**> COMMA <**dictionary\_data**>

| <**data\_type**> COMMA <**data\_type**><**optional\_parameters**>

| IDENTIFIER ASSIGN <**data\_type**>

<**optional**\_**parameters**> ::= IDENTIFIER

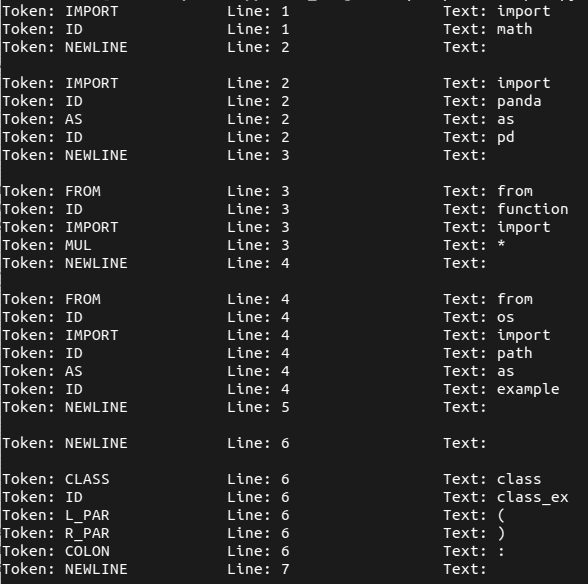
| <**optional\_parameters**> COMMA IDENTIFIER

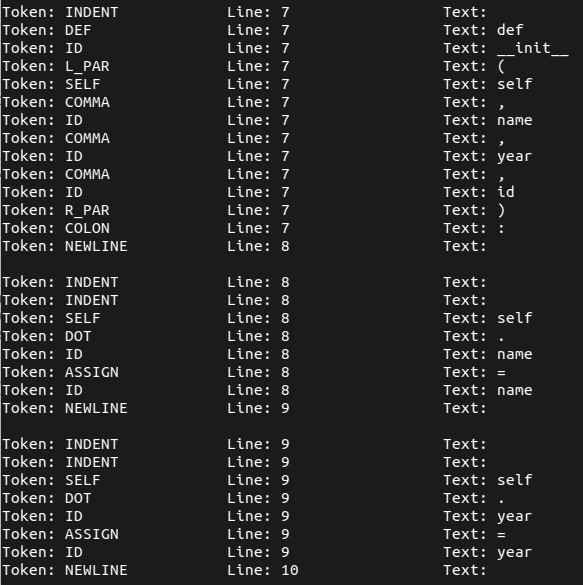
| COMMA

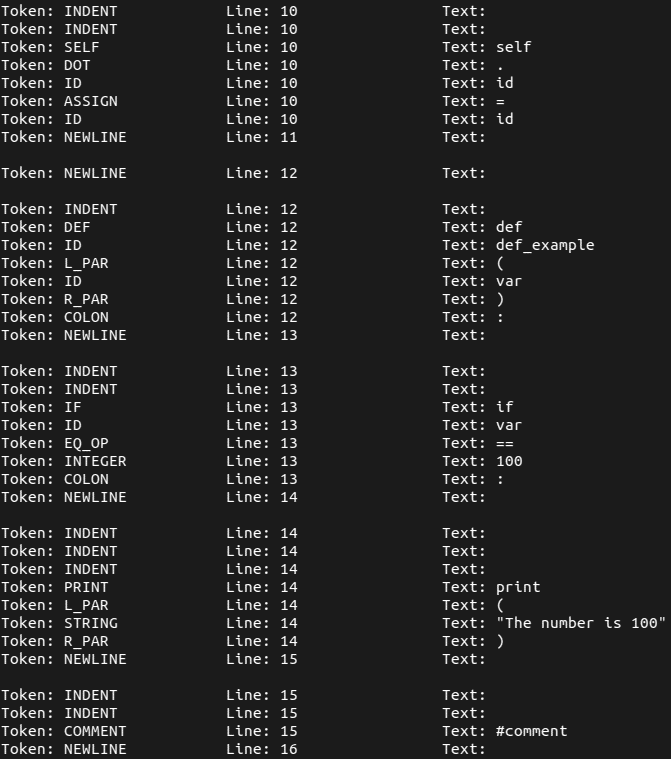
<**data**\_**type**> ::= FLOAT | CHAR | STRING | INTEGER | IDENTIFIER

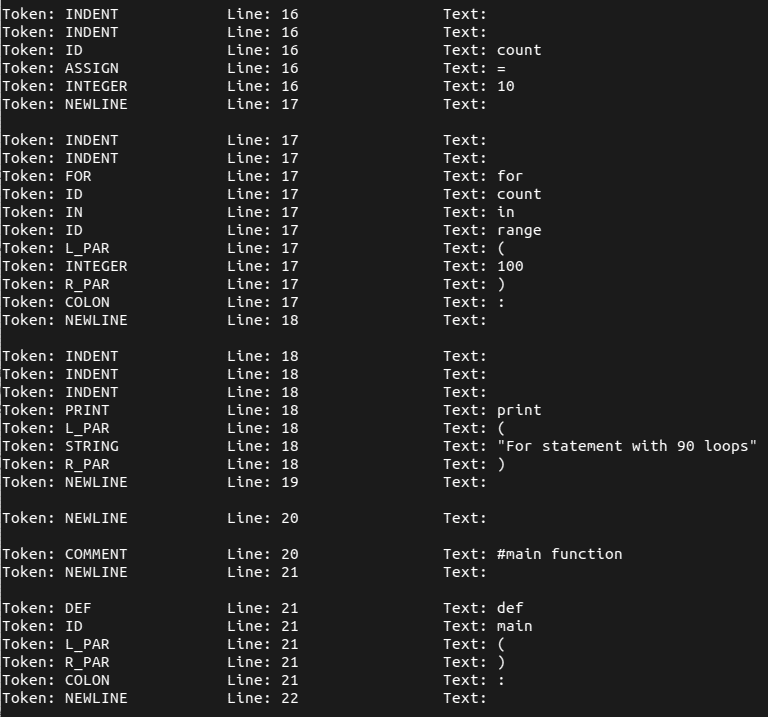
|  |
| --- |
| import math  import panda as pd  from function import \*  from os import path as example  class class\_ex():  def \_\_init\_\_(self,name,year,id):  self.name = name  self.year = year  self.id = id  def def\_example(var):  if var == 100:  print("The number is 100")  #comment  count=10  for count in range(100):  print("For statement with 90 loops")  #main function  def main():  print("Compiler Testing:\n")  #list dictionary  list1 = { Country1 : "GREECE" , Country2 : "CYPRUS"}  list2 = dict1( [ (Town1, "Athens") , (Town2, "Thessaloniki"),  (Town3, "Patra") , (Town4, "Chania") ] )  list3 = dict2( Athens= "Town1" , Thessaloniki= "Town2" ,  Patra= "Town3" , Chania= "Town4" )  #create an object  obj = class\_ex("CEID", 2020, 11111111)  #function call  def\_example(100)  #arithmetic expressions  num1 = 100  num2 = 10  plus = num1+num2  mul = num1\*num2  var1 = 10  var2 = 15  #lambda calculus  lambda\_exx = lambda vari : def\_example(var1)  lambda\_ex = lambda variable : var2+var1  lambda\_ex1 = lambda variable1 : lambda variable2 : var1\*var2 |

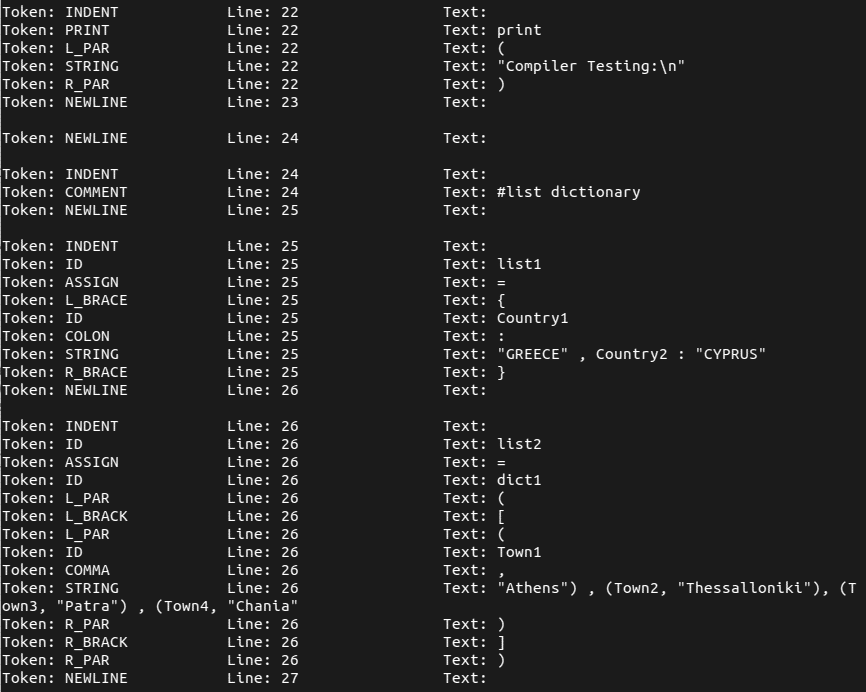
**III. ΠΑΡΑΔΕΙΓΜΑΤΑ ΕΚΤΕΛΕΣΗΣ**

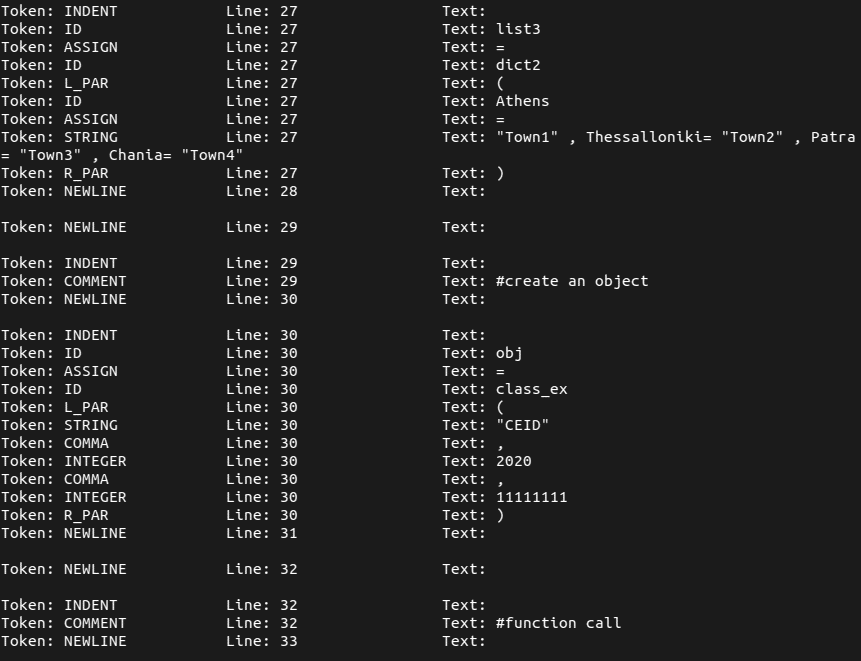
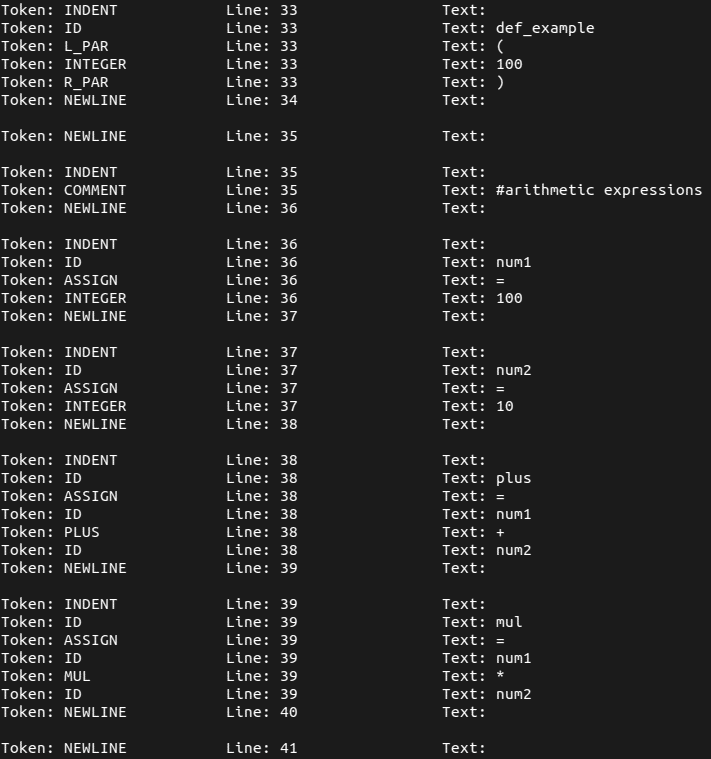
Τρέξαμε το παραπάνω πρόγραμμα (input.py), που έχουμε δημιουργήσει το οποίο καλύπτει όλες τις λειτουργίες του υποσυνόλου της γλώσσας Python. Κατά την εκτέλεση το πρόγραμμα αναλύεται με τον παρακάτω τρόπο:

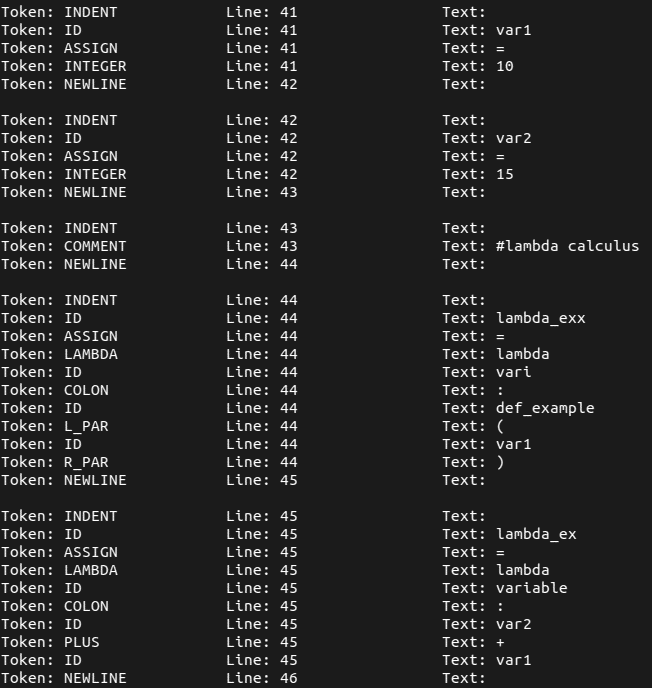
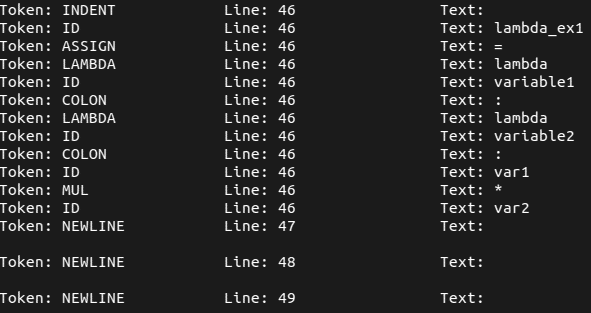


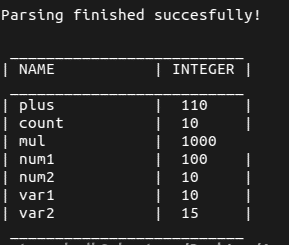












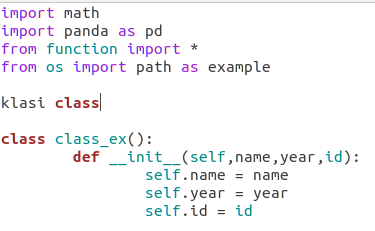
**Επεξήγηση:**

Πιο πάνω βλέπουμε ότι η ανάλυση γίνεται ανά γραμμή και ελέγχεται αν είναι όλα όπως τα έχουμε ορίσει στον λεξικό και συντακτικό αναλυτή.

Για την κάθε γραμμή εκτυπώνουμε τα tοkens την γραμμή και το text.

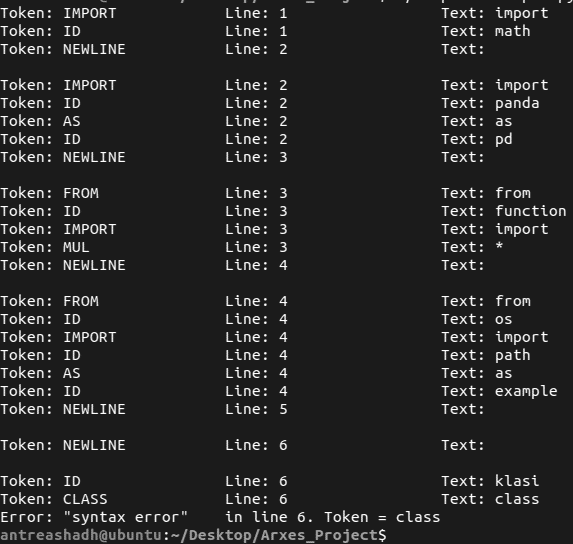
Εάν όλα είναι ορθά και συντακτικά σωστά γραμμένα, εκτυπώνει το μήνυμα “Parsing finished successfully !” και τέλος εκτυπώνεται ένας πίνακας με τα δεδομένα που έχει η κάθε μεταβλητή.

Εάν προκύψει κάποιο σφάλμα κατά την εκτέλεση καλείτε η συνάρτηση yyerror() και εμφανίζεται κατάλληλο μήνυμα ή “unknown character” για άγνωστο χαρακτήρα ή “syntax error” για συντακτικό λάθος. Για παράδειγμα:



Σκόπιμο λάθος!

Κατά την εκτέλεση του πιο πάνω εμφανίζεται μήνυμα συντακτικού λάθους. Αυτό οφείλετε στο ότι δεν υπάρχει κάποιος συντακτικός κανόνας που να το υλοποιεί. Επίσης κατά τον τερματισμό εμφανίζεται και η γραμμή που προκύπτει το σφάλμα:



**IV. ΟΛΟΚΛΗΡΩΜΕΝΟΣ ΚΩΔΙΚΑΣ**

Α. Αρχείο LEXER (lexer.l):

|  |
| --- |
| %option noyywrap  %{  #include <stdio.h>  #include <stdlib.h>  #include <string.h>  #include <regex.h>  #include "parser.tab.h"  extern FILE \*yyin;  extern FILE \*yyout;    int line\_no = 1;  //the function of lexer analysis. Return the token  int yylex();  //error function  void yyerror();  //print statement function  void print\_return(char \*token);    %}    %x ML\_COMMENT  alphabet [a-zA-Z]  digit [0-9]  alphanumeric {alphabet}|{digit}  print [ -~]  underscore \_  identifier ({alphabet}|{underscore})+({alphanumeric}|{underscore})\*  integer "0"|[0-9]{digit}\*  float\_number "0"|{digit}\*"."{digit}+  char \'{print}\'  string \".\*\"  %%  "#".\* { print\_return("COMMENT"); return COMMENT; }  "break" { print\_return("BREAK"); return BREAK; }  "continue" { print\_return("CONTINUE" ); return CONTINUE; }  "if" { print\_return("IF"); return IF; }  "elif" { print\_return("ELIF"); return ELIF; }  "else" { print\_return("ELSE"); return ELSE; }  "for" { print\_return("FOR"); return FOR; }  "in" { print\_return("IN"); return IN; }  "return" { print\_return("RETURN"); return; }  "boolean" { print\_return("BOOLEAN"); return BOOLEAN; }  "true" { print\_return("TRUE"); return TRUE; }  "false" { print\_return("FALSE"); return FALSE; }  "null" { print\_return("NIL"); return NIL; }  "lambda" { print\_return("LAMBDA"); return LAMBDA; }  "def" { print\_return("DEF"); return DEF; }  "class" { print\_return("CLASS"); return CLASS; }  "print" { print\_return("PRINT"); return PRINT; }  "from" { print\_return("FROM"); return FROM; }  "import" { print\_return("IMPORT"); return IMPORT; }  "init" { print\_return("INIT"); return INIT; }  "self" { print\_return("SELF"); return SELF; }  "as" { print\_return("AS"); return AS; }  "\n" { line\_no++; print\_return("NEWLINE"); return NEWLINE; }  "\t" { print\_return("INDENT"); return INDENT; }  ">>=" { print\_return("RIGHT\_ASSIGN"); return RIGHT\_ASSIGN; }  "<<=" { print\_return("LEFT\_ASSIGN"); return LEFT\_ASSIGN; }  "+=" { print\_return("ADD\_ASSIGN"); return ADD\_ASSIGN; }  "-=" { print\_return("SUB\_ASSIGN"); return SUB\_ASSIGN; }  "/=" { print\_return("DIV\_ASSIGN"); return DIV\_ASSIGN; }  "%=" { print\_return("MOD\_ASSIGN"); return MOD\_ASSIGN; }  "--" { print\_return("DEC\_OP"); return DEC\_OP; }  "++" { print\_return("INC\_OP"); return INC\_OP; }  "&&" { print\_return("AND\_OP"); return AND\_OP; }  "||" { print\_return("OR\_OP"); return OR\_OP; }  "==" { print\_return("EQ\_OP"); return EQ\_OP; }  ">=" { print\_return("GE\_OP"); return GE\_OP; }  "<=" { print\_return("LE\_OP"); return LE\_OP; }  "!=" { print\_return("NE\_OP"); return NE\_OP; }  "{" { print\_return("L\_BRACE"); return L\_BRACE; }  "}" { print\_return("R\_BRACE"); return R\_BRACE; }  "," { print\_return("COMMA"); return COMMA; }  ":" { print\_return("COLON"); return COLON; }  "=" { print\_return("ASSIGN"); return ASSIGN; }  "(" { print\_return("L\_PAR"); return L\_PAR; }  ")" { print\_return("R\_PAR"); return R\_PAR;}  "[" { print\_return("L\_BRACK"); return L\_BRACK; }  "]" { print\_return("R\_BRACK"); return R\_BRACK;}  "." { print\_return("DOT"); return DOT; }  "\_" { print\_return("UNDERSCORE"); return UNDERSCORE; }  "-" { print\_return("MINUS"); return MINUS; }  "+" { print\_return("PLUS"); return PLUS; }  "\*" { print\_return("MUL"); return MUL; }  "/" { print\_return("DIV"); return DIV; }  "<" { print\_return("LT"); return LT; }  ">" { print\_return("GT"); return GT; }  [ ] ;  . { yyerror("Unkown character"); }  {identifier} { print\_return("ID"); strcpy(yylval.name, yytext); return IDENTIFIER; }  {integer} { yylval.integer\_val = atoi(yytext); print\_return("INTEGER"); return INTEGER; }  {float\_number} { print\_return("FLOAT"); return FLOAT; }  {string} { print\_return("STRING"); return STRING; }  {char} { print\_return("CHAR"); return CHAR; }  %%  /\* -------------------------------------------- C FUNCTIONS ---------------------------------------------- \*/  void print\_return(char \*token)  {  printf("Token: %s\t\t Line: %d\t\t Text: %s\n", token, line\_no, yytext);  } |

Β. Αρχείο BISON (parser.y):

|  |
| --- |
| %{  #include <stdio.h>  #include <stdlib.h>  #include <math.h>  #include "print\_console.c"  //pointer to input file of lexer  extern FILE \*yyin;  //pointer to output file of lexer  extern FILE \*yyout;  //line counter  extern int line\_no;  //reads the input stream generates tokens  extern int yylex();  //temporary token save  extern char\* yytext;  //Function Initilize  int yylex();  void yyerror(char \*message);  %}  //struct for print\_console  %union  {  char name[500];  int integer\_val;  }  /\* ------------------------------------------------- TOKENS -------------------------------------------------- \*/  //starting symbol  %start programm  %token COMMENT  %token BREAK  %token CONTINUE  %token IF  %token ELIF  %token ELSE  %token FOR  %token RETURN  %token BOOLEAN  %token TRUE  %token FALSE  %token NIL  %token LAMBDA  %token DEF  %token CLASS  %token NEWLINE  %token INDENT  %token RIGHT\_ASSIGN  %token LEFT\_ASSIGN  %token ADD\_ASSIGN  %token SUB\_ASSIGN  %token DIV\_ASSIGN  %token MOD\_ASSIGN  %token DEC\_OP  %token INC\_OP  %token AND\_OP  %token OR\_OP  %token EQ\_OP  %token GE\_OP  %token LE\_OP  %token NE\_OP  %token L\_BRACE  %token R\_BRACE  %token COMMA  %token COLON  %token ASSIGN  %token L\_PAR  %token R\_PAR  %token L\_BRACK  %token R\_BRACK  %token DOT  %token MINUS  %token PLUS  %token MUL  %token DIV  %token LT  %token GT  %token FLOAT  %token CHAR  %token STRING  %token IN  %token PRINT  %token FROM  %token IMPORT  %token AS  %token UNDERSCORE  %token INIT  %token SELF  %token EXIT  %token <name> IDENTIFIER  %token <integer\_val> INTEGER  //type for access to $$  %type <integer\_val> line int\_op int\_data  %type <name> calc\_assignment  %%  /\* ------------------------------------------------ BNF GRAMMAR ----------------------------------------- \*/  /\* --- Starting symbol: programm --- \*/  programm: line | programm line | /\* empty \*/;  line: if\_stmt {;}| for\_statement {;} | def NEWLINE INDENT {;}| def NEWLINE indent2 {;}  | def NEWLINE {;} | def\_call {;} | comments NEWLINE {;} | action {;}  | print NEWLINE {;} | import\_statement NEWLINE {;}  | class NEWLINE {;} | create\_object NEWLINE {;} | dictionaries NEWLINE {;}  | lambda\_expr NEWLINE {;} | calc\_assignment NEWLINE {;}  | NEWLINE {;} ;  /\* --- Indents and Actions --- \*/  indent2:' '' ';  indent3:' '' '' ';  indent4:' '' '' '' ';  indent5:' '' '' '' '' ';  action: INDENT line | indent2 line | indent3 line | indent4 line | indent5 line | /\* empty \*/ ;  /\* --- Data Type --- \*/  data\_type: FLOAT | CHAR | STRING | INTEGER | IDENTIFIER ;  /\* --- Statements --- \*/  if\_stmt: IF IDENTIFIER operators data\_type COLON NEWLINE action;  for\_statement: FOR IDENTIFIER IN def\_call COLON NEWLINE action;  /\* --- Operators --- \*/  operators: AND\_OP | OR\_OP | EQ\_OP | GE\_OP | LE\_OP | NE\_OP;  optional\_parameters: IDENTIFIER | optional\_parameters COMMA IDENTIFIER | COMMA | /\* empty \*/;  /\* --- Functions --- \*/  def: DEF IDENTIFIER L\_PAR optional\_parameters R\_PAR COLON;  def\_call: IDENTIFIER L\_PAR optional\_parameters R\_PAR  | IDENTIFIER L\_PAR data\_type R\_PAR  | IDENTIFIER L\_PAR data\_type COMMA data\_type R\_PAR  | IDENTIFIER L\_PAR data\_type COMMA data\_type COMMA data\_type R\_PAR;  /\* --- Comments --- \*/  comments: COMMENT;  /\* --- Print --- \*/  print: PRINT L\_PAR data\_type R\_PAR;  /\* --- From-Import --- \*/  import\_statement: FROM IDENTIFIER IMPORT IDENTIFIER AS IDENTIFIER  | FROM IDENTIFIER IMPORT IDENTIFIER | FROM IDENTIFIER IMPORT MUL  | IMPORT IDENTIFIER AS IDENTIFIER | IMPORT IDENTIFIER  | FROM IDENTIFIER IMPORT IDENTIFIER COMMA IDENTIFIER;  /\* --- Class --- \*/  class: CLASS IDENTIFIER COLON NEWLINE INDENT class\_constructor  | CLASS IDENTIFIER L\_PAR R\_PAR COLON NEWLINE INDENT class\_constructor;  class\_constructor: DEF IDENTIFIER L\_PAR constructor\_parameters R\_PAR COLON NEWLINE constructor\_body;  constructor\_parameters: SELF | IDENTIFIER | constructor\_parameters COMMA IDENTIFIER;  constructor\_body: object\_creation | constructor\_body object\_creation;  object\_creation: INDENT SELF DOT IDENTIFIER ASSIGN data\_type NEWLINE  | /\* empty \*/;  /\* --- Create an object and call a Class --- \*/  create\_object: IDENTIFIER ASSIGN class\_call;  class\_call: IDENTIFIER L\_PAR data\_type R\_PAR  | IDENTIFIER L\_PAR data\_type COMMA data\_type R\_PAR  | IDENTIFIER L\_PAR data\_type COMMA data\_type COMMA data\_type R\_PAR;  /\* --- Dictionaries --- \*/  dictionaries: IDENTIFIER ASSIGN L\_BRACE dictionary\_data R\_BRACE  | IDENTIFIER ASSIGN IDENTIFIER L\_PAR L\_BRACK L\_PAR dictionary\_data R\_PAR R\_BRACK R\_PAR  | IDENTIFIER ASSIGN IDENTIFIER L\_PAR dictionary\_data optional\_parameters dictionary\_data R\_PAR ;  dictionary\_data: data\_type COLON data\_type  |data\_type COLON data\_type COMMA dictionary\_data  | data\_type COMMA data\_type optional\_parameters  | IDENTIFIER ASSIGN data\_type | /\* empty \*/ ;  /\* --- Lambda calculus --- \*/  lambda\_expr: IDENTIFIER ASSIGN LAMBDA IDENTIFIER COLON expressions  | LAMBDA IDENTIFIER COLON expressions;  expressions: lambda\_expr | term | expressions MINUS term | expressions PLUS term  | expressions MUL term | expressions DIV term | def\_call;  term: data\_type;    /\* --- calculation --- \*/  calc\_assignment: IDENTIFIER ASSIGN int\_op { Change($1, $3); };    int\_op: int\_data { $$ = $1; }  | int\_op PLUS int\_data { $$ = $1 + $3; }  | int\_op MINUS int\_data { $$ = $1 - $3; }  | int\_op MUL int\_data { $$ = $1 \* $3; }  | int\_op DIV int\_data { $$ = $1 / $3; } ;  int\_data: INTEGER { $$ = $1; }  | IDENTIFIER { $$ = Search($1) -> integer\_val; };  %%  /\* ------------------------------------------------ C FUNCTIONS -------------------------------------------- \*/  void yyerror(char \*message){  printf("Error: \"%s\"\t in line %d. Token = %s\n", message, line\_no, yytext);  exit(1);  }  /\* ------------------------------------------ MAIN FUNCTION --------------------------------------------- \*/  int main(int argc, char \*argv[]){  hashTable = (hash \*) calloc(SIZE, sizeof(hash));  int flag;  yyin = fopen(argv[1],"r");  //yyparse(): reads tokens, executes actions  flag = yyparse();  fclose(yyin);  printf("Parsing finished succesfully!\n\n");  printf(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  Print();  printf(" \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");  return flag;  } |

Γ. Αρχείο εκτύπωσης στην κονσόλα (print\_console.c):

|  |
| --- |
| #define SIZE 10  #include <stdio.h>  #include <string.h>  #include <stdlib.h>  typedef struct node  {  char name[100];  int integer\_val;  struct node \*next;  }var;  typedef struct hash  {  var \*head;  int count;  }hash;  static hash \*hashTable = NULL;  //memory allocation  var \*newNode(char n[], int i)  {  var \*temp = (var \*)malloc(sizeof(var));    strcpy(temp->name, n);  temp->integer\_val = i;  temp->next = NULL;    return temp;  };  //insert data into hash table  void Insert(char n[], int i)  {  int hashIndex, h = 0;  var \*newnode = newNode(n, i);    // hash function  for (int c = 0; n[c] != '\0'; c++)  h = (h + (unsigned char)n[c]);  hashIndex = h % SIZE;    if (!hashTable[hashIndex].head)  {  hashTable[hashIndex].head = newnode;  hashTable[hashIndex].count = 1;  return;  }    //adding new node to the list  newnode->next = (hashTable[hashIndex].head);    //update the head of the list and no of nodes in the current bucket  hashTable[hashIndex].head = newnode;  hashTable[hashIndex].count++;    return;  };  //print the value of hashtable  void Print()  {  var \*myNode;  int i;    printf("| NAME\t\t | INTEGER |\n ");  printf("\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\n");    for (i = 0; i < SIZE; i++)  {  if (hashTable[i].count == 0)  continue;    myNode = hashTable[i].head;  if (!myNode)  continue;    while (myNode != NULL)  {  printf("| %s\t\t | ", myNode->name);  printf(" %d\t |\n", myNode->integer\_val);  myNode = myNode->next;  }  }  return;  };  //search value in hashtable and return the variable  var \*Search(char n[])  {  int hashIndex, h = 0, flag = 0;  var \*temp = NULL;    for (int i = 0; n[i] != '\0'; i++)  h = (h + (unsigned char)n[i]);    hashIndex = h % SIZE;    temp = hashTable[hashIndex].head;  if (!temp) {  printf("Search element not found in hash table\n");  return temp;  }  while (temp != NULL) {  if (strcmp(temp->name, n) == 0){  flag = 1;  break;  }    temp = temp->next;  }    if (!flag)  printf("Search element not found in hash table\n");    return temp;  }  //change the value  void Change(char n[], int i)  {  int hashIndex, h = 0, flag = 0;  var \*temp;    //hash function  for (int i = 0; n[i] != '\0'; i++)  h = (h + (unsigned char)n[i]);  hashIndex = h % SIZE;    temp = hashTable[hashIndex].head;  if (!temp)  {  Insert(n, i);  return;  }    while (temp != NULL)  {  if (strcmp(temp->name, n) == 0){  temp->integer\_val = i;  flag = 1;  break;  }  temp = temp->next;  }    if (!flag)  printf("Search element not found in hash table\n");    return;  }; |

Δ. Αρχείο εισαγωγής Python (input.py):

|  |
| --- |
| import math  import panda as pd  from function import \*  from os import path as example  class class\_ex():  def \_\_init\_\_(self,name,year,id):  self.name = name  self.year = year  self.id = id  def def\_example(var):  if var == 100:  print("The number is 100")  count=10  for count in range(100):  print("For statement with 90 loops")  #main function  def main():  print("Compiler Testing:\n")  #list dictionary  list1 = { Country1 : "GREECE" , Country2 : "CYPRUS"}  list2 = dict1( [ (Town1, "Athens") , (Town2, "Thessalloniki"), (Town3, "Patra") , (Town4, "Chania") ] )  list3 = dict2( Athens= "Town1" , Thessalloniki= "Town2" , Patra= "Town3" , Chania= "Town4" )  #create an object  obj = class\_ex("CEID", 2020, 11111111)  #function call  def\_example(100)  #arithmetic expressions  num1 = 100  num2 = 10  plus = num1+num2  mul = num1\*num2  var1 = 10  var2 = 15  #lambda calculus  lambda\_exx = lambda vari : def\_example(var1)  lambda\_ex = lambda variable : var2+var1  lambda\_ex1 = lambda variable1 : lambda variable2 : var1\*var2 |