PROJECT Αρχές γλωσσών και μεταφραστών

1) Tα παρακάτω είναι ο bnf συντακτικός ορισμός της γραμματικής της γλώσσας.

atom ::=

identifier | literal | enclosure

enclosure ::=

parenth\_form | list\_display

| generator\_expression | dict\_display

| string\_conversion | yield\_atom

literal ::=

stringliteral | integer | longinteger

| floatnumber | imagnumber

stringliteral ::=

stringliteralpiece

| stringliteral stringliteralpiece

parenth\_form ::=

"(" [expression\_list] ")"

list\_display ::=

"[" [expression\_list | list\_comprehension] "]"

list\_comprehension ::=

expression list\_for

list\_for ::=

"for" target\_list "in" old\_expression\_list

[list\_iter]

old\_expression\_list ::=

old\_expression

[("," old\_expression)+ [","]]

list\_iter ::=

list\_for | list\_if

list\_if ::=

"if" old\_expression [list\_iter]

generator\_expression ::=

"(" expression genexpr\_for ")"

genexpr\_for ::=

"for" target\_list "in" or\_test

[genexpr\_iter]

genexpr\_iter ::=

genexpr\_for | genexpr\_if

genexpr\_if ::=

"if" old\_expression [genexpr\_iter]

dict\_display ::=

"{" [key\_datum\_list] "}"

key\_datum\_list ::=

key\_datum ("," key\_datum)\* [","]

key\_datum ::=

expression ":" expression

string\_conversion ::=

"`" expression\_list "`"

yield\_atom ::=

"(" yield\_expression ")"

yield\_expression ::=

"yield" [expression\_list]

primary ::=

atom | attributeref

| subscription | slicing | call

attributeref ::=

primary "." identifier

subscription ::=

primary "[" expression\_list "]"

slicing ::=

simple\_slicing | extended\_slicing

simple\_slicing ::=

primary "[" short\_slice "]"

extended\_slicing ::=

primary "[" slice\_list "]"

slice\_list ::=

slice\_item ("," slice\_item)\* [","]

slice\_item ::=

expression | proper\_slice | ellipsis

proper\_slice ::=

short\_slice | long\_slice

short\_slice ::=

[lower\_bound] ":" [upper\_bound]

long\_slice ::=

short\_slice ":" [stride]

lower\_bound ::=

expression

upper\_bound ::=

expression

stride ::=

expression

ellipsis ::=

"..."

call ::=

primary "(" [argument\_list [","]

| expression genexpr\_for] ")"

argument\_list ::=

positional\_arguments ["," keyword\_arguments]

["," "\*" expression]

["," "\*\*" expression]

| keyword\_arguments ["," "\*" expression]

["," "\*\*" expression]

| "\*" expression ["," "\*\*" expression]

| "\*\*" expression

positional\_arguments ::=

expression ("," expression)\*

keyword\_arguments ::=

keyword\_item ("," keyword\_item)\*

keyword\_item ::=

identifier "=" expression

power ::=

primary ["\*\*" u\_expr]

u\_expr ::=

power | "-" u\_expr

| "+" u\_expr | "~" u\_expr

m\_expr ::=

u\_expr | m\_expr "\*" u\_expr

| m\_expr "//" u\_expr

| m\_expr "/" u\_expr

| m\_expr "%" u\_expr

a\_expr ::=

m\_expr | a\_expr "+" m\_expr

| a\_expr "-" m\_expr

shift\_expr ::=

a\_expr

| shift\_expr ( "<<" | ">>" ) a\_expr

and\_expr ::=

shift\_expr | and\_expr "&" shift\_expr

xor\_expr ::=

and\_expr | xor\_expr "^" and\_expr

or\_expr ::=

xor\_expr | or\_expr "|" xor\_expr

comparison ::=

or\_expr ( comp\_operator or\_expr )\*

comp\_operator ::=

"<" | ">" | "==" | ">=" | "<=" | "<>" | "!="

| "is" ["not"] | ["not"] "in"

expression ::=

conditional\_expression | lambda\_form

old\_expression ::=

or\_test | old\_lambda\_form

conditional\_expression ::=

or\_test ["if" or\_test "else" expression]

or\_test ::=

and\_test | or\_test "or" and\_test

and\_test ::=

not\_test | and\_test "and" not\_test

not\_test ::=

comparison | "not" not\_test

lambda\_form ::=

"lambda" [parameter\_list] ":" expression

old\_lambda\_form ::=

"lambda" [parameter\_list] ":" old\_expression

expression\_list ::=

expression ( "," expression )\* [","]

simple\_stmt ::= expression\_stmt

| assert\_stmt

| assignment\_stmt

| augmented\_assignment\_stmt

| pass\_stmt

| del\_stmt

| print\_stmt

| return\_stmt

| yield\_stmt

| raise\_stmt

| break\_stmt

| continue\_stmt

| import\_stmt

| global\_stmt

| exec\_stmt

expression\_stmt ::=

expression\_list

assert\_stmt ::=

"assert" expression ["," expression]

assignment\_stmt ::=

(target\_list "=")+

(expression\_list | yield\_expression)

target\_list ::=

target ("," target)\* [","]

target ::=

identifier

| "(" target\_list ")"

| "[" target\_list "]"

| attributeref

| subscription

| slicing

augmented\_assignment\_stmt ::=

target augop

(expression\_list | yield\_expression)

augop ::=

"+=" | "-=" | "\*=" | "/=" | "//=" | "%=" | "\*\*="

| ">>=" | "<<=" | "&=" | "^=" | "|="

pass\_stmt ::=

"pass"

del\_stmt ::=

"del" target\_list

print\_stmt ::=

"print" ( [expression ("," expression)\* [","]]

| ">>" expression [("," expression)+ [","]] )

return\_stmt ::=

"return" [expression\_list]

yield\_stmt ::=

yield\_expression

raise\_stmt ::=

"raise" [expression ["," expression

["," expression]]]

break\_stmt ::=

"break"

continue\_stmt ::=

"continue"

import\_stmt ::=

"import" module ["as" name]

( "," module ["as" name] )\*

| "from" relative\_module "import" identifier

["as" name]

( "," identifier ["as" name] )\*

| "from" relative\_module "import" "("

identifier ["as" name]

( "," identifier ["as" name] )\* [","] ")"

| "from" module "import" "\*"

module ::=

(identifier ".")\* identifier

relative\_module ::=

"."\* module | "."+

name ::=

identifier

global\_stmt ::=

"global" identifier ("," identifier)\*

exec\_stmt ::=

"exec" or\_expr

["in" expression ["," expression]]

compound\_stmt ::=

if\_stmt

| while\_stmt

| for\_stmt

| try\_stmt

| with\_stmt

| funcdef

| classdef

suite ::=

stmt\_list NEWLINE

| NEWLINE INDENT statement+ DEDENT

statement ::=

stmt\_list NEWLINE | compound\_stmt

stmt\_list ::=

simple\_stmt (";" simple\_stmt)\* [";"]

if\_stmt ::=

"if" expression ":" suite

( "elif" expression ":" suite )\*

["else" ":" suite]

while\_stmt ::=

"while" expression ":" suite

["else" ":" suite]

for\_stmt ::=

"for" target\_list "in" expression\_list

":" suite

["else" ":" suite]

try\_stmt ::= try1\_stmt | try2\_stmt

try1\_stmt ::=

"try" ":" suite

("except" [expression

["," target]] ":" suite)+

["else" ":" suite]

["finally" ":" suite]

try2\_stmt ::=

"try" ":" suite

"finally" ":" suite

with\_stmt ::=

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

funcdef ::=

[decorators] "def" funcname "(" [parameter\_list] ")"

":" suite

decorators ::=

decorator+

decorator ::=

"@" dotted\_name ["(" [argument\_list [","]] ")"] NEWLINE

dotted\_name ::=

identifier ("." identifier)\*

parameter\_list ::=

(defparameter ",")\*

("\*" identifier [, "\*\*" identifier]

| "\*\*" identifier

| defparameter [","] )

defparameter ::=

parameter ["=" expression]

sublist ::=

parameter ("," parameter)\* [","]

parameter ::=

identifier | "(" sublist ")"

funcname ::=

identifier

classdef ::=

"class" classname [inheritance] ":"

suite

inheritance ::=

"(" [expression\_list] ")"

classname ::=

identifier

file\_input ::=

(NEWLINE | statement)\*

interactive\_input ::=

[stmt\_list] NEWLINE | compound\_stmt NEWLINE

eval\_input ::=

expression\_list NEWLINE\*

input\_input ::=

expression\_list NEWLINE

2) Eίναι η περιγραφή του αντίστοιχου υποσυνόλου της Python σε BNF

list:

/\* Nothing \*/

| list program

;

program: ekf

| assignment

| parameters

| function

| ifstatement

| forstatement

| callfunction

| createclass

| comment

| print

| lambda

| metablhtes

;

Το program ορίζει τις επόμενες εκφράσεις που θα χρησιμοποιηθούν

ekf:

INTNUM {;}

| FLOATNUM {;}

| ekf '+' ekf

| ekf '-' ekf

| ekf '\*' ekf

| ekf '/' ekf

| '(' ekf ')' {;}

|VAR

Assignment: VAR '=' ekf

|VAR

|VAR '<' ekf

|VAR '>' ekf

|VAR '!' '=' ekf

|VAR '>' '=' ekf

|VAR '<' '=' ekf

;

Το assignment εκτελεί συγκρίσεις μεταξύ αριθμών

parameters: var

| var ',' var

;

Το parameters περνάει τις παραμέτρους στην function

function: DEF VAR '(' ')' ':'

| DEF VAR'(' parameters ')' ':'

;

H function είναι η συνάρτηση

callfunction: VAR '(' ')'

| VAR '(' parameters ')'

;

Η κλήση της συνάρτησης

ifstatement: IF assignment ':'

| ELIF assignment ':'

| ELSE ':'

;

Η if

forstatement: FOR VAR IN RANGE '(' ekf ')' ':'

;

H for

createclass: CLASS VAR ':'

| VAR'=' VAR '(' parameters ')'

;

H δημιουργία κλάσης

comment: HASHTAG {YYACCEPT;}

;

Σχόλιο

print: PRINT '('''' VAR ''' ')'

| PRINT '(' ''' VAR ''' ',' VAR ')' parameters;

Εκτύπωση

lambda: LAMBDA [parameters] ":" program

λάμδα calculus

metablhtes: INT assignment

|FLOAT assignment

;

3) Aυτό είναι το τελικό αρχείο περιγραφής της γλώσσας, το οποίο δίνεται σαν είσοδος στον parser όπου γίνεται γενικός ελέγχος της λειτουργίας τόσο της γραμματικής όσο και του συντακτικού του compiler της python.

class Person:

age = 10

float child

def greet(self):

print(Hello)

if age<10:

print(underaged)

elif age>20:

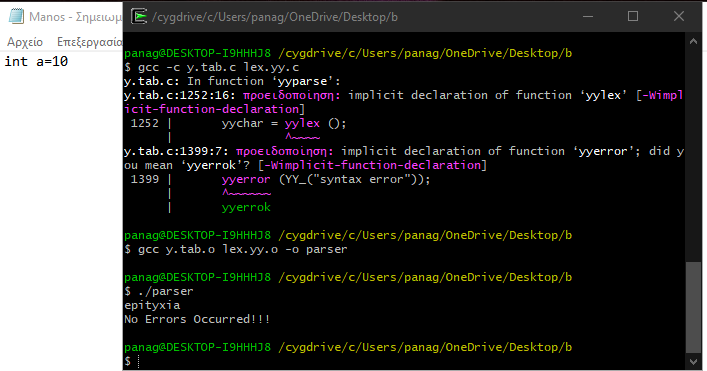
print(teenager)

else :

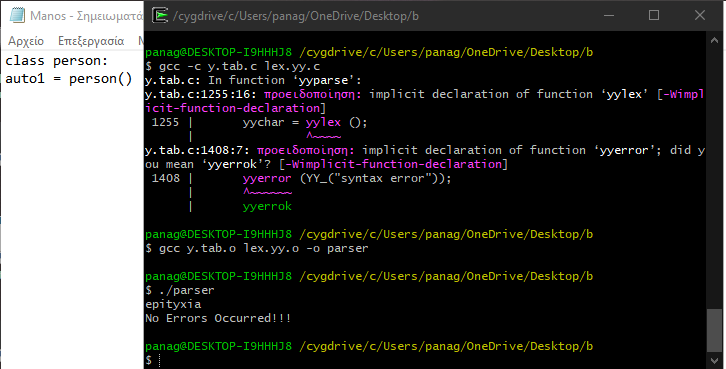
print(getout)

child = age/2

4)Για την αρχικοποίηση έχουμε



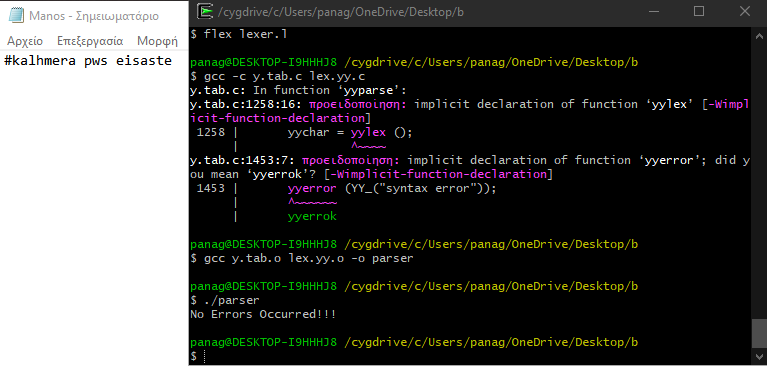
Για την κλάση έχουμε



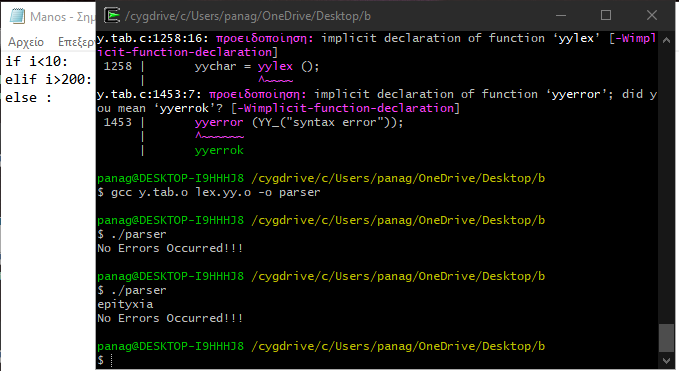
Για τον ορισμό συνάρτησης και κλήση της



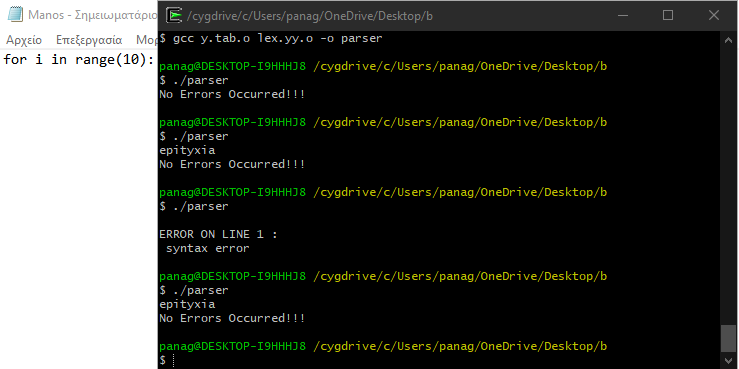
Για σχόλια



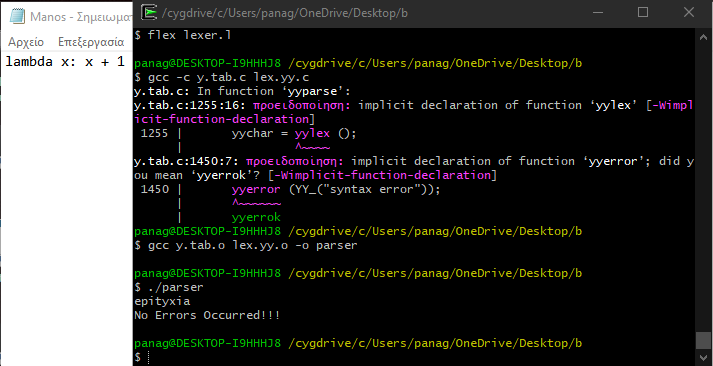
Για το if



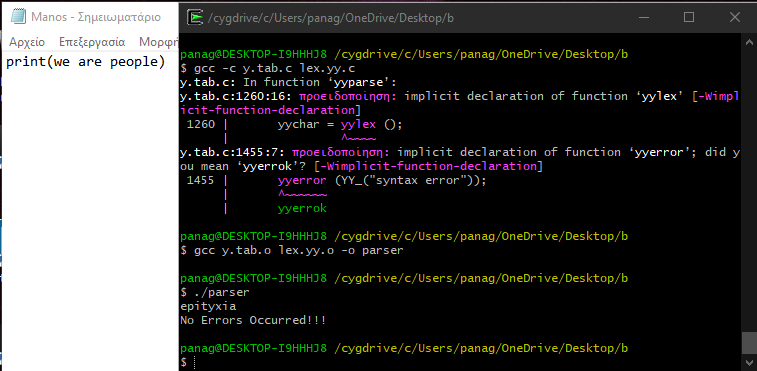
Για το for



Για το print



Για το lambda



Για πράξεις

