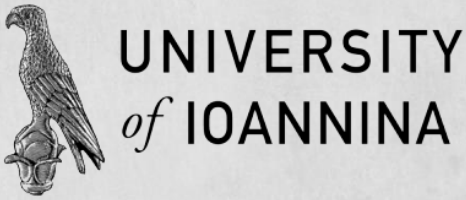
Τμήμα Μηχ. Η/Υ και Πληροφορικής



ΜΕΤΑΦΡΑΣΤΕΣ

CutePy

ΔΙΔΑΣΚΩΝ: Γ. Μανής

ΣΤΟΙΧΕΙΑ ΟΜΑΔΑΣ:

Πρίσκας Σπυρίδων ΑΜ: 4482

Παΐλα Αγνή ΑΜ: 4753

Ημερομηνία : 24/05/2023

Πίνακας περιεχομένων

Αρχικά, για να δώσουμε το αρχείο που θέλουμε να μεταγλωττίσουμε ως είσοδο από τη γραμμή εντολών, χρησιμοποιήσαμε τις ακόλουθες γραμμές κώδικα.

```
import sys
filename = sys.argv[1]

file = open(filename, 'r')
```

Έτσι μπορούμε να τρέχουμε τον κώδικά μας με τον εξής τρόπο:

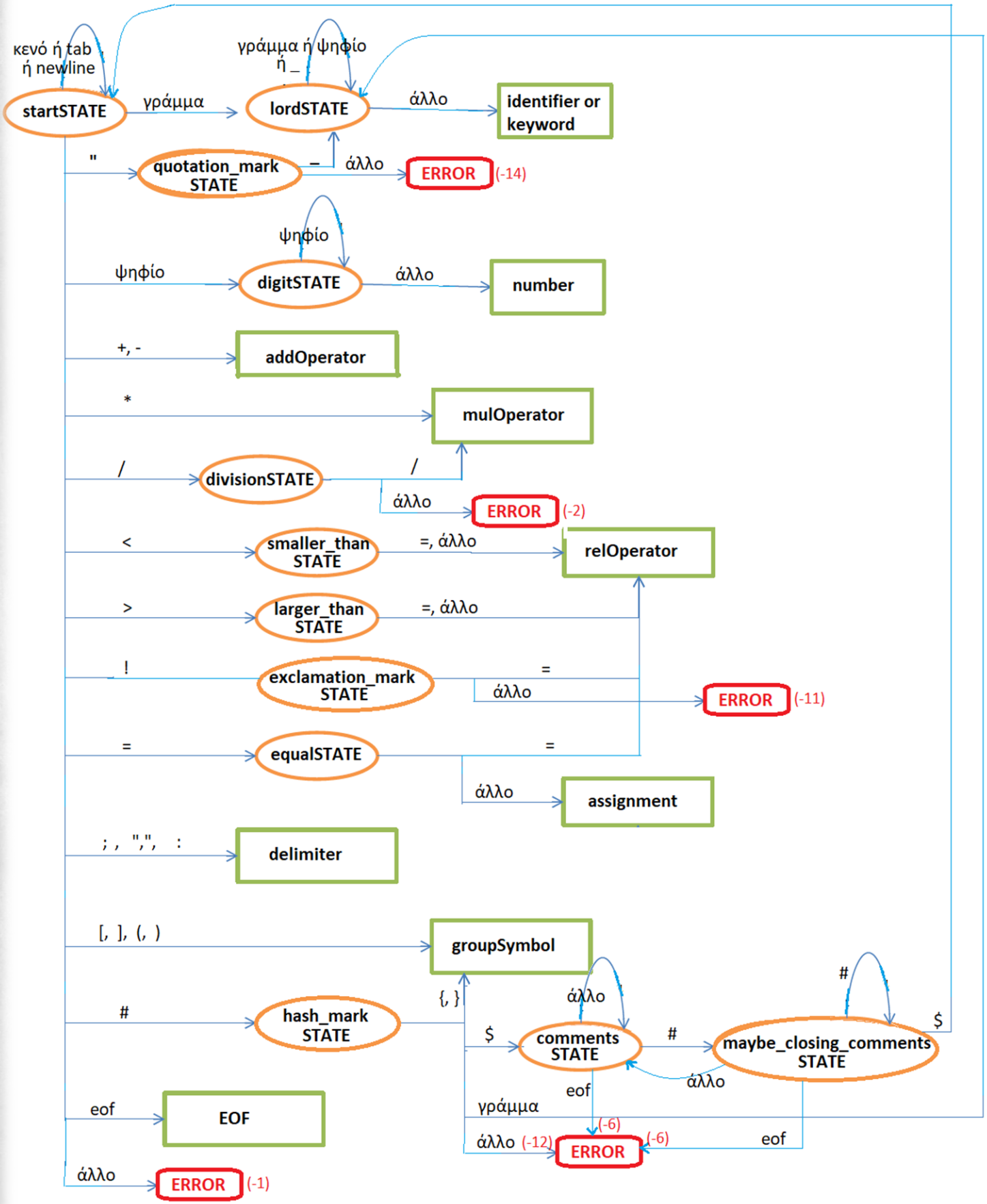
>python cutePy_4482_4753.py test.cpy

Όπου *test.py* είναι το *sys.argv[1]* που περνάμε στο *filename* μέσα στο πρόγραμμά μας.

Έπειτα με την 3η γραμμή ανοίγουμε το αρχείο για διάβασμα.

ΑΥΤΟΜΑΤΟ ΚΑΤΑΣΤΑΣΕΩΝ

Δημιουργήσαμε το ακόλουθο αυτόματο καταστάσεων το οποίο ξεκινά από την αρχική κατάσταση (*startSTATE*) και με κάθε διαφορετική είσοδο αλλάζει κατάσταση ώσπου να συναντήσει μια τελική κατάσταση.


κενό ή tab
ή newline
γράμμα ή ψηφίο
ή _
startSTATE
γράμμα
lordSTATE
άλλο
identifier or keyword
"
quotation_mark STATE
_
άλλο
ERROR
(-14)
ψηφίο
ψηφίο
digitSTATE
άλλο
number
+, -
addOperator
*
mulOperator
/
divisionSTATE
/
άλλο
ERROR
(-2)
<
smaller_than STATE
=, άλλο
relOperator
>
larger_than STATE
=, άλλο
!
exclamation_mark STATE
=
άλλο
ERROR
(-11)
=
equalSTATE
=
άλλο
assignment
; , ","," :
delimiter
[,], (,)
groupSymbol
#
hash_mark STATE
{, }
άλλο
$
comments STATE
#
maybe_closing_comments STATE
#
$
άλλο
eof
γράμμα
(-6)
άλλο
(-12)
ERROR
(-6)
eof
eof
EOF
άλλο
ERROR
(-1)

Το αυτόματο αναγνωρίζει το *αλφάβητο της cutePy* που αποτελείται από τους παρακάτω χαρακτήρες:

- τα μικρά και κεφαλαία γράμματα της λατινικής αλφαβήτου (A,…,Z και a,…,z),
- τα αριθμητικά ψηφία (0,…,9),
- την κάτω παύλα (_),
- τα σύμβολα των αριθμητικών πράξεων (+, -, *, //),
- τους τελεστές συσχέτισης (<, >, !=, <=, >=, ==)
- το σύμβολο ανάθεσης (=)
- τους διαχωριστές (;, “,”, :)
- τα σύμβολα ομαδοποίησης ([,], (,) , #{, #})
- και διαχωρισμού σχολίων (#$)

Οι 12 καταστάσεις του αυτομάτου (αριθμημένες από 0 έως 11) είναι οι ακόλουθες:

```
startSTATE = 0
lordSTATE = 1
digitSTATE = 2
divisionSTATE = 3
smaller_thanSTATE = 4
larger_thanSTATE = 5
exclamatiom_markSTATE = 6
equalSTATE = 7
hash_markSTATE = 8
commentsSTATE = 9
maybe_closing_commentsSTATE = 10
quotation_markSTATE = 11
```

Οι χαρακτήρες της γλώσσας cutePy είναι οι ακόλουθοι:

```
chars = ['blank', 'letters', 'digits', 'underscore', 'add', 'sub', 'mul', 'div', 'smaller_than', 'larger_than',
         'exclamatiom_mark', 'equal', 'question_mark', 'comma', 'colon', 'left_bracket', 'right_bracket',
         'left_parenthesis', 'right_parenthesis', 'hash_mark', 'start_of_block', 'end_of_block', 'dollar_sign',
         'quotation_mark', 'next_line', 'not_accepted_symbol', 'EOF']
```

Οι τελικές καταστάσεις (αριθμημένες από 100 έως 122) στις οποίες μπορούμε να οδηγηθούμε ακολουθώντας τις διάφορες καταστάσεις του αυτόματου, είναι οι ακόλουθες και σε αυτές ισχύει ότι έχουμε αναγνωρίσει μια λεκτική μονάδα.

```
id_kwrdTOKEN = 100
digitTOKEN = 101
addTOKEN = 102
subTOKEN = 103
```

```
mulTOKEN = 104
divTOKEN = 105
smaller_equalTOKEN = 106
smallerTOKEN = 107
larger_equalTOKEN = 108
largerTOKEN = 109
differentTOKEN = 110
equalTOKEN = 111
assignmentTOKEN = 112
question_markTOKEN = 113
commaTOKEN = 114
colonTOKEN = 115
left_bracketTOKEN = 116
right_bracketTOKEN = 117
left_parenthesisTOKEN = 118
right_parenthesisTOKEN = 119
start_of_blockTOKEN = 120
end_of_blockTOKEN = 121
EOFTOKEN = 122
```

Από τα αναγνωριστικά (*id_keywordTOKEN*) ξεχωρίζουμε τις δεσμευμένες λέξεις της γλώσσας cutePy, καθώς και τις αντίστοιχες τελικές καταστάσεις εάν έχει αναγνωριστεί λεκτική μονάδα η οποία αποτελεί δεσμευμένη λέξη.

```
#DESMEUMENES LEKSEIS PROGRAMMATOS
desmeumenes_lexeis = ['#declare','if','else','while','return','print',
                      'int','input','def','or','and','not','__name__','"__main__"']

#DESMEUMENA TOKEN (telikh katastash) PROGRAMMATOS
desmeumenes_lexeisTOKEN = ['declareTOKEN', 'ifTOKEN', 'elseTOKEN', 'whileTOKEN', 'returnTOKEN', 'printTOKEN', 'intTOKEN',
                           'inputTOKEN', 'defTOKEN', 'orTOKEN', 'andTOKEN', 'notTOKEN', 'nameTOKEN', 'mainTOKEN'  ]
```

Έπειτα ορίζουμε τα πιθανά *errors* (αριθμημένα από -1 έως -15), καθώς κατά τη διάρκεια εκτέλεσης του αυτόματου, μπορεί να έχουμε οδηγηθεί σε τελική κατάσταση η οποία είναι λανθασμένη. Με αυτόν τον τρόπο αναγνωρίζουμε την ύπαρξη λάθους.

```
ERROR_notAcceptedSymbol = -1
ERROR_singleSlash = -2
ERROR_letterAfterDigit = -3
ERROR_numberOutOfRange = -4
ERROR_identifierOver30Characters = -5
ERROR_EOFBeforeClosingComments = -6
ERROR_singleDollarSign = -7
ERROR_singleLeftCurlyBrackets = -8
ERROR_singleRightCurlyBrackets = -9
```

```
ERROR_identifierStartsWithUnderscore = -10
ERROR_singleExclamationMark = -11
ERROR_singleHashMark = -12
ERROR_identifierStartsWithHashMarkWithoutBeingTheKeywordDeclare = -13
ERROR_singleQuotationMark = -14
ERROR_identifierContainsQuoteMarkWithoutBeingTheKeywordMain = -15
```

ΠΙΝΑΚΑΣ ΜΕΤΑΒΑΣΕΩΝ

Έπειτα υλοποιούμε τον πίνακα μεταβάσεων (*transition_matrix*) για όλες τις καταστάσεις και όλους τους πιθανούς χαρακτήρες που μπορεί να διαβάσουμε ενώ βρισκόμαστε σε κάποια από αυτές. Ξεκινώντας από μια κατάσταση και έστω ότι μας έρχεται ο πρώτος χαρακτήρας από τον πίνακα *chars* που περιέχει τους χαρακτήρες της γλώσσας cutePy σκεφτόμαστε σε ποια ***STATE*** θα βρεθούμε ή σε πιο ***TOKEN*** ή σε ποιο ***ERROR***.

Έτσι προκύπτει ο ακόλουθος πίνακας για τις 11 καταστάσεις του αυτόματου και τους 27 χαρακτήρες που μπορεί να διαβάσει το πρόγραμμά μας:

```
transition_matrix=[
    #gia thn katastash startSTATE
    [startSTATE, lordSTATE, digitSTATE, lordSTATE, addTOKEN, subTOKEN, mulTOKEN,
     divisionSTATE, smaller_thanSTATE, larger_thanSTATE, exclamatiom_markSTATE, equalSTATE, question_markTOKEN, commaTOKEN, colonTOKEN,
     left_bracketTOKEN, right_bracketTOKEN, left_parenthesisTOKEN, right_parenthesisTOKEN, hash_markSTATE,
     ERROR_singleLeftCurlyBrackets, ERROR_singleRightCurlyBrackets, ERROR_singleDollarSign, quotation_markSTATE,
     startSTATE, ERROR_notAcceptedSymbol, EOFTOKEN
    ],

    #gia thn katastash lordSTATE
    [id_kwrdTOKEN, lordSTATE, lordSTATE, lordSTATE, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN,
     id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN,
     id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, id_kwrdTOKEN, lordSTATE, id_kwrdTOKEN,
     ERROR_notAcceptedSymbol, id_kwrdTOKEN
    ],

    #gia thn katastash digitSTATE
    [digitTOKEN, ERROR_letterAfterDigit, digitSTATE, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN,
     digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN,
     digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, digitTOKEN, ERROR_notAcceptedSymbol, digitTOKEN
    ],

    #gia thn katastash divisionSTATE
    [ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash,
     ERROR_singleSlash, divTOKEN, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash,
     ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash,
     ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_singleSlash, ERROR_notAcceptedSymbol,
     ERROR_singleSlash
    ],
```

```
#gia thn katastash smaller_thanSTATE
[smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN,
 smallerTOKEN, smallerTOKEN, smallerTOKEN, smaller_equalTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN,
 smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN, smallerTOKEN,
 ERROR_notAcceptedSymbol, smallerTOKEN
],

#gia thn katastash larger_thanSTATE
[largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN,
 largerTOKEN, largerTOKEN, larger_equalTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN,
 largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, largerTOKEN, ERROR_notAcceptedSymbol,
 largerTOKEN
],

#gia thn katastash exclamatiom_markSTATE
[ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark,
 ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark,
 ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, differentTOKEN,
 ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark,
 ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark,
 ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark, ERROR_singleExclamationMark,
 ERROR_singleExclamationMark, ERROR_notAcceptedSymbol, ERROR_singleExclamationMark
],

#gia thn katastash equalSTATE
[assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN,
 assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, equalTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN,
 assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, assignmentTOKEN,
 assignmentTOKEN, assignmentTOKEN, assignmentTOKEN, ERROR_notAcceptedSymbol, assignmentTOKEN
],
```

```
#gia thn katastash hash_markSTATE
[ERROR_singleHashMark, lordSTATE, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark,
 ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark,
 ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark, ERROR_singleHashMark,
 ERROR_singleHashMark, ERROR_singleHashMark, start_of_blockTOKEN, end_of_blockTOKEN, commentsSTATE, ERROR_singleHashMark,
 ERROR_singleHashMark, ERROR_notAcceptedSymbol, ERROR_singleHashMark
],

#gia thn katastash commentsSTATE
[commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, maybe_closing_commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, ERROR_EOFBeforeClosingComments
],

#gia thn katastash maybe_closing_commentsSTATE
[commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, maybe_closing_commentsSTATE, commentsSTATE, commentsSTATE, startSTATE, commentsSTATE,
 commentsSTATE, commentsSTATE, ERROR_EOFBeforeClosingComments
],

#gia thn katastash quotation_markSTATE
[ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, lordSTATE, ERROR_singleQuotationMark,
 ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark,
 ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark,
 ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark,
 ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark, ERROR_singleQuotationMark,
 ERROR_notAcceptedSymbol, ERROR_singleQuotationMark
]
```

ΑΡΧΙΚΟΠΟΙΗΣΗ ΤΩΝ FAMILIES

Δημιουργούμε τη μέθοδο *initfamily(current_state)* που παίρνει ως παράμετρο την τρέχουσα κατάσταση στην οποία βρισκόμαστε και ανάλογα με αυτή χωρίζονται τα *TOKEN* που αναγνωρίσαμε σε ομάδες, σύμφωνα με την εκφώνηση. Εάν κάποιο *TOKEN* μας είναι άγνωστο επιστρέφει *FAMILY: unknown*.

```
family = ''
###DHMIOYRGEI TA DIAFORETIKA FAMILIES GIA KATHE TOKEN
def initfamily(current_state):
        global family

        if(current_state == id_kwrdTOKEN):
                family = 'FAMILY: identifier'
        elif(current_state == digitTOKEN):
                family = 'FAMILY: number'
        elif(current_state == addTOKEN or current_state == subTOKEN):
                family = 'FAMILY: addOperator'
        elif(current_state == mulTOKEN or current_state == divTOKEN):
                family = 'FAMILY: mulOperator'
        elif(current_state == smallerTOKEN or current_state == smaller_equalTOKEN or current_state == largerTOKEN or current_state == larger_equalTOKEN or
                family = 'FAMILY: relOperator'
        elif(current_state == assignmentTOKEN):
                family = 'FAMILY: assignment'
        elif(current_state == commaTOKEN or current_state == question_markTOKEN or current_state == colonTOKEN):
                family = 'FAMILY: delimiter'
        elif(current_state == left_bracketTOKEN or current_state == right_bracketTOKEN or current_state == left_parenthesisTOKEN or current_state == right_
                family = 'FAMILY: groupSymbol'
        elif(current_state == EOFTOKEN):
                family = 'FAMILY: EOF'
        else:
                family = 'FAMILY unknown'

        return family
```

ΔΙΑΧΕΙΡΙΣΗ ΤΩΝ ERROR

Δημιουργούμε τη μέθοδο *errorMessages(current_line, word_unit)* που παίρνει ως παράμετρο την τρέχουσα γραμμή του αρχείου που βρισκόμαστε και την λεκτική μονάδα που έχει σχηματιστεί για να φτάσουμε σε λανθασμένη κατάσταση.

Για κάθε λάθος του λεκτικού αναλυτή τυπώνουμε αντίστοιχο μήνυμα λάθους αναφέροντας τη γραμμή στην οποία συνέβη αυτό και την λανθασμένη λεκτική μονάδα που σχηματίζεται κάθε φορά.

```
###ELEGXEI KAI TYPWNEI OLA TA LATHH
def errorMessages(current_line, word_unit):
        if(current_state == ERROR_notAcceptedSymbol):
                print("***ERROR(lex)***: Found a not accepted symbol ( " + word_unit + " ) in line: " + str(current_line))
        elif(current_state == ERROR_singleSlash):
                print("***ERROR(lex)***: Found a single / in line: " + str(current_line))
        elif(current_state == ERROR_letterAfterDigit):
                print("***ERROR(lex)***: Found a letter character after a digit character ( " + word_unit + " ) in line: " + str(current_line))
        elif(current_state == ERROR_numberOutOfRange):
                print("***ERROR(lex)***: The number ' " + word_unit + " ' is not inside the accepted range of [-((2^32)-1), (2^32)-1)] in line: " + str(current_line))
        elif(current_state == ERROR_identifierOver30Characters):
                print("***ERROR(lex)***: Found an identifier with over 30 characters ( " + word_unit + " ) in line: " + str(current_line))
        elif(current_state == ERROR_EOFBeforeClosingComments):
                print("***ERROR(lex)***: Comments begin correctly with #$ but we get EOF before they end in line: " + str(current_line))
        elif(current_state == ERROR_singleDollarSign):
                print("***ERROR(lex)***: Found a single $ in line: " + str(current_line))
        elif(current_state == ERROR_singleLeftCurlyBrackets):
                print("***ERROR(lex)***: Found a single { in line: " + str(current_line))
        elif(current_state == ERROR_singleRightCurlyBrackets):
                print("***ERROR(lex)***: Found a single } in line: " + str(current_line))
        elif(current_state == ERROR_identifierStartsWithUnderscore):
                print("***ERROR(lex)***: Found the word ' " + word_unit + " ' starting with _ in line: " + str(current_line))
        elif(current_state == ERROR_singleExclamationMark):
                print("***ERROR(lex)***: Found a single ! in line: " + str(current_line))
        elif(current_state == ERROR_singleHashMark):
                print("***ERROR(lex)***: Found a single # in line: " + str(current_line))
        elif(current_state == ERROR_identifierStartsWithHashMarkWithoutBeingTheKeywordDeclare):
                print("***ERROR(lex)***: Found the identifier ' " + word_unit + " ' starting with # (NOT #declare) in line: " + str(current_line))
        elif(current_state == ERROR_singleQuotationMark):
                print("***ERROR(lex)***: Found a single \" in line: " + str(current_line))
        elif(current_state == ERROR_identifierContainsQuoteMarkWithoutBeingTheKeywordMain):
                print("***ERROR(lex)***: Found the identifier ' " + word_unit + " ' containing \" (NOT \"__main__\") in line: " + str(current_line))
        return()
```

ΑΝΑΓΝΩΡΙΣΗ ΛΕΚΤΙΚΗΣ ΜΟΝΑΔΑΣ

Για να αναγνωρίσουμε μια λεκτική μονάδα δημιουργούμε την μέθοδο ***lex().*** Αρχικά η μέθοδος αυτή αρχικοποιεί τη λεκτική μονάδα στο κενό και φτιάχνει μια μεταβλητή *result* που είναι μια λίστα που θα κρατήσει το αποτέλεσμα της μεθόδου το οποίο θα έχει τη μορφή :

['FAMILY: …..', 'word_unit', line, code_number]

Η τρέχουσα κατάσταση είναι η αρχική (*startSTATE*) και η τρέχουσα γραμμή είναι η 1η. Έχουμε ορίσει τη μεταβλητή *line ίση με 1* έξω και πριν από τη μέθοδο ***lex()*** καθώς κάθε φορά που την καλούμε θέλουμε η τρέχουσα γραμμή να είναι αυτή στην οποία μείναμε την προηγούμενη φορά που κλήθηκε η συνάρτηση.

```
line = 1  ##arxika eimai sthn 1h grammh


def lex():
        global line
        global current_state
        word_unit = ''                #lektikh monada arxika einai kenh
        current_state = startSTATE    #trexousa katastash arxika sto start
        current_line = line           #trexoysa grammh kwdika toy arxeioy filename

### RESULT einai to apotelesma poy epistrefetai se lista kathws exoyme 4 pragmata na epistrecoyme se kathe klhsh ths lex
        result=[]
```

Έπειτα όσο βρισκόμαστε σε κάποια από τις 12 καταστάσεις του αυτόματου που έχουμε δημιουργήσει πρέπει να διαβάζουμε έναν-έναν τους χαρακτήρες από το αρχείο .cpy που έχουμε ανοίξει για διάβασμα. Αυτό γίνεται με την εντολή: *char_read = file.read(1)*

Η μεταβλητή *char_read* κρατάει κάθε φορά τον χαρακτήρα που διαβάζει και τον συγκρίνει με τους πιθανούς χαρακτήρες που έχει η γλώσσα cutePy, ώστε να αρχικοποιήσει μια μεταβλητή *char* στον αντίστοιχο χαρακτήρα που υπάρχει στη θέση του πίνακα chars με όλους τους χαρακτήρες που υπάρχουν. Αυτό μας βοηθάει, καθώς έπειτα από το διάβασμα κάθε χαρακτήρα θέλουμε να τον χρησιμοποιήσουμε για να περάσουμε από την τρέχουσα κατάσταση στην επόμενη κατάσταση, όπως ορίζεται από τον πίνακα μεταβάσεων.

Στην περίπτωση που ο χαρακτήρας που διαβάσαμε είναι η αλλαγή γραμμής (\n) αυξάνουμε τον μετρητή *current_line* κατά ένα.

```
while(0 <= current_state <= 11 ):
        char_read = file.read(1)  #diavazw ena xaraktira
        #print('char read: ', char_read)
        if (char_read == ' ' or char_read == '\t' ):
                char = chars[0]
        elif ('a'<= char_read <= 'z' or 'A' <= char_read <= 'Z'):
                char = chars[1]
        elif ('0'<= char_read <= '9'):
                char = chars[2]
        elif (char_read == '_'):
                char = chars[3]
        elif (char_read == '+'):
                char = chars[4]
        elif (char_read == '-'):
                char = chars[5]
        elif (char_read == '*'):
                char = chars[6]
        elif (char_read == '/'):
                char = chars[7]
        elif (char_read == '<'):
                char = chars[8]
        elif (char_read == '>'):
                char = chars[9]
        elif (char_read == '!'):
                char = chars[10]
        elif (char_read == '='):
                char = chars[11]
        elif (char_read == ';'):
                char = chars[12]
        elif (char_read == ','):
                char = chars[13]
        elif (char_read == ':'):
                char = chars[14]
        elif (char_read == '['):
                char = chars[15]
        elif (char_read == ']'):
                char = chars[16]
        elif (char_read == '('):
                char = chars[17]
        elif (char_read == ')'):
                char = chars[18]
        elif (char_read == '#'):
                char = chars[19]
        elif (char_read == '{'):
                char = chars[20]
        elif (char_read == '}'):
                char = chars[21]
        elif (char_read == '$'):
                char = chars[22]
        elif (char_read == '"'):
                char = chars[23]
        elif (char_read == '\n'):
                current_line += 1    #ean diavasw char allagh grammhs paw sthn epomenh seira
                char = chars[24]
        elif (char_read == ''):
                char = chars[26]
        else:
                char = chars[25]     #not_accepted_symbol

        positionofchar = chars.index(char)
        # nea current_state einai ean sthn trexousa katastash diabasw ton xarakthra char
        current_state = transition_matrix[current_state][positionofchar]
```

Αφού έχουμε διαβάσει έναν χαρακτήρα πλέον και έχουμε πάει στην επόμενη κατάσταση, θέλουμε αυτός ο χαρακτήρας να προστεθεί στη λεκτική μονάδα που θα σχηματίσουμε. Η πρόσθεση του χαρακτήρα μας στη λεκτική μονάδα γίνεται πάντα, εκτός από την περίπτωση πού βρισκόμαστε μέσα σε σχόλια όπου δεν μας ενδιαφέρει το τι βρίσκεται γραμμένο εκεί. Ακόμη, εάν η τωρινή μας κατάσταση παραμένει να είναι η αρχική κατάσταση, σημαίνει ότι διαβάσαμε έναν λευκό χαρακτήρα (*space*, *tab*, *newline*) που δεν θέλουμε να προστεθεί στην λεκτική μονάδα, καθώς πρέπει να τους αγνοούμε. Σε αυτές τις περιπτώσεις ορίζουμε η λεκτική μονάδα μας να είναι το κενό.

```
###SXHMATISE LEKTIKH MONADA PROSTHETONTAS TON XARAKTHRA POY DIABAZEIS KATHE FORA, EKTOS KAI AN EIMASTE SE SXOLIO 'H STHN ARXH GIA NA MHN PROSTHETEI TA KENA
            if(current_state != startSTATE and current_state != commentsSTATE and current_state != maybe_closing_commentsSTATE):
                word_unit += char_read
            else:
                word_unit = ''
```

Πρέπει επίσης καθώς προσθέτουμε χαρακτήρες να ελέγξουμε εάν η λεκτική μονάδα είναι αλφαριθμητικό το οποίο αποτελείται από περισσότερους από 30 χαρακτήρες, καθώς αυτό δεν είναι επιτρεπτό σύμφωνα με την εκφώνηση. Εάν ισχύει ωστόσο αυτή η περίπτωση **κάνουμε οπισθοδρόμηση**, καθώς θέλουμε να μας επιστραφεί ο τελευταίος χαρακτήρας, που διαβάστηκε χωρίς να είναι γράμμα, για να χρησιμοποιηθεί στην επόμενη επανάληψη και περνάμε στην κατάσταση λάθους.

```
###ELEGXW AN EINAI PANW APO 30 XARAKTHRES ENA ANAGNWRISTIKO
while(len(word_unit)>30 and current_state == id_kwrdTOKEN):
        char = file.seek(file.tell()-1,0)
        word_unit = word_unit[:-1]
        current_state = ERROR_identifierOver30Characters
```

ΟΠΙΣΘΟΔΡΟΜΗΣΗ

Όταν έχουν προστεθεί όλοι οι χαρακτήρες, έχει σχηματιστεί η λεκτική μονάδα. Πολλές φορές όμως μπορεί να χρειαστεί να καταναλώσουμε τον επόμενο χαρακτήρα για να τη σχηματίσουμε. Ο χαρακτήρας αυτός λοιπόν είναι αναγκαίο να ενσωματωθεί στην επόμενη λεκτική μονάδα. Σε όσες περιπτώσεις συμβαίνει αυτό ανανεώνουμε τη μεταβλητή *char* να είναι πλέον ο τελευταίος χαρακτήρας του αρχείου μας και ως λεκτική μονάδα (*word_unit*) παίρνουμε όλους τους χαρακτήρες εκτός από τον τελευταίο.

Πιο συγκεκριμένα οι περιπτώσεις που χρειάζεται να γίνει οπισθοδρόμηση είναι όταν έχουμε μια λέξη ή έναν αριθμό ή κάποια από τα σύμβολα <, >, =, !, “, /, # διότι σε αυτές τις περιπτώσεις πρέπει να καταναλώσουμε τον επόμενο χαρακτήρα για να καταλάβουμε ποια είναι η λεκτική μονάδα που σχηματίζεται τελικά.

```
###OPISTHODROMISH    %%% GINETAI EAN EIMAI SE KATASTASEIS POU KATANALWNW TON EPOMENO XARAKTHRA GIA NA BRW TO TOKEN
        if(current_state == id_kwrdTOKEN or current_state == digitTOKEN or current_state == smallerTOKEN or current_state == largerTOKEN
           or current_state == assignmentTOKEN or current_state == ERROR_singleSlash or current_state == ERROR_singleExclamationMark
           or current_state == ERROR_singleHashMark or current_state == ERROR_singleQuotationMark):
```

Αφού πλέον έχει γίνει η οπισθοδρόμηση, εάν βρισκόμαστε σε τελική κατάσταση *id_kwrdTOKEN* σημαίνει ότι μπορεί να έχουμε αναγνωρίσει είτε κάποιο αλφαριθμητικό (*identifier*) είτε κάποια δεσμευμένη λέξη (*keyword*). Διατρέχοντας τον πίνακα που περιέχει όλες τις δεσμευμένες λέξεις ελέγχουμε εάν η λεκτική μονάδα μας είναι κάποια από αυτές. Εάν ναι, προσθέτουμε στο αποτέλεσμα την οικογένεια τον δεσμευμένων λέξεων (*‘FAMILY: keyword’*) και ορίζουμε ως τρέχουσα κατάσταση την τελική κατάσταση της δεσμευμένης λέξης που αναγνωρίσαμε.

```
###ELEGXW AN EINAI DESMEYMENH LEKSI
        if(current_state == id_kwrdTOKEN):
                for i in range(len(desmeumenes_lexeis)):
                        if(word_unit == desmeumenes_lexeis[i]):
                                result.append('FAMILY: keyword')
                                current_state = desmeumenes_lexeisTOKEN[i]
```

Όσο βρισκόμαστε στην κατάσταση *id_kwrdTOKEN* πρέπει να ελέγξουμε εάν υπάρχει λέξη η οποία να ξεκινά από # και να μην είναι η δεσμευμένη λέξη #declare, καθώς δεν επιτρέπεται αναγνωριστικό να ξεκινάει από #. Σε αυτή την περίπτωση ορίζουμε τρέχουσα κατάσταση να είναι το ERROR που τυπώνει το αντίστοιχο μήνυμα λάθους.

```
###ELEGXW AN KSEKINAEI APO # ENW DEN EINAI DESMEYMENH LEKSI
if(word_unit[0] == '#' and word_unit != desmeumenes_lexeis[0] ):       #desmeumenes_lexeis[0] einai to '#declare'
        current_state = ERROR_identifierStartsWithHashMarkWithoutBeingTheKeywordDeclare
```

Παρόμοια ελέγχουμε εάν υπάρχει λέξη η οποία να ξεκινά από __ και να μην είναι η δεσμευμένη λέξη __name__ , καθώς δεν επιτρέπεται αναγνωριστικό να ξεκινάει από _ . Σε αυτή την περίπτωση ορίζουμε τρέχουσα κατάσταση να είναι το ERROR που τυπώνει το αντίστοιχο μήνυμα λάθους.

```
###ELEGXW AN KSEKINAEI APO _ ENW DEN EINAI DESMEYMENH LEKSI
if(word_unit[0] == '_' and (word_unit not in desmeumenes_lexeis) ):
        current_state = ERROR_identifierStartsWithUnderscore
```

Παρόμοια ελέγχουμε εάν υπάρχει λέξη η οποία να περιέχει ” χωρίς να είναι η δεσμευμένη λέξη “__main__” . Σε αυτή την περίπτωση ορίζουμε τρέχουσα κατάσταση να είναι το ERROR που τυπώνει το αντίστοιχο μήνυμα λάθους.

```
###ELEGXW AN EINAI H DESMEYMENH LEKSI "__main__" POY PERIEXEI " ALLIWS ERROR
if(word_unit != '"__main__"' and ( '"' in word_unit)):
        current_state = ERROR_identifierContainsQuoteMarkWithoutBeingTheKeywordMain
```

Εάν βρισκόμαστε στη τελική κατάσταση *digitTOKEN* και ο αριθμός που αναγνωρίσαμε είναι μεγαλύτερος από το 2^32, ορίζουμε τρέχουσα κατάσταση να είναι το ERROR που τυπώνει το αντίστοιχο μήνυμα λάθους.

```
###ELEGXW AN EINAI MESA STA EPITREPTA ORIA ENA NOYMERO (2^32)
        if(current_state == digitTOKEN and int(word_unit) >= pow(2,32)):
                current_state = ERROR_numberOutOfRange
```

Αφού έχουμε τελειώσει με τους περιορισμούς που μας θέτει η εκφώνηση καλούμε την μέθοδο *errorMessages(current_line, word_unit)* με την τρέχουσα γραμμή και την συγκεκριμένη λεκτική μονάδα που έχει σχηματιστεί κάθε φορά με την κλήση της ***lex()***, ώστε να τυπωθούν πιθανά μηνύματα λάθους που ίσως υπάρχουν και αφορούν τη συγκεκριμένη αυτή λεκτική μονάδα κάθε φορά.

```
###TYPWSE PITHANA ERRORS
        errorMessages(current_line, word_unit)
```

ΤΙ ΕΠΙΣΤΡΕΦΕΙ Η ΣΥΝΑΡΤΗΣΗ LEX()

Στην **1η θέση** του αποτελέσματος *result* τυπώνουμε την οικογένεια στην οποία ανήκει η λεκτική μονάδα που σχηματίστηκε.

```
## TI EPISTREFEI H SYNARTHSH LEX
## result = ['FAMILY: ______', 'token pou sxhmatisthke', 'LINE: ___', 'kwdikos kathe token' ]
        # result[0]
        if(current_state != id_kwrdTOKEN ):
                if(word_unit not in desmeumenes_lexeis):
                        result.append(initfamily(current_state))
                        exit
        else:
                result.append(initfamily(current_state))
                exit
```

Στη **2η θέση** του αποτελέσματος *result* τυπώνουμε την λεκτική μονάδα που σχηματίστηκε.

Στην **3η θέση** του αποτελέσματος *result* τυπώνουμε την γραμμή στην οποία εμφανίζεται η λεκτική μονάδα.

Στην **4η και τελευταία θέση** του αποτελέσματος *result* τυπώνουμε έναν μοναδικό κωδικό αριθμό τον οποίο ορίσαμε για κάθε κατάσταση, είτε αυτή είναι κάποια από τις καταστάσεις του αυτόματου, είτε είναι κάποια τελική κατάσταση, είτε είναι κάποια κατάσταση ERROR. Αυτό μας βοηθάει για το επόμενο βήμα της άσκησης, τον συντακτικό αναλυτή.

```
        # result[1]
        result.append(word_unit)

        # result[2]
        #ln = 'LINE: ' + str(current_line)
        result.append(current_line)

        # result[3]
        if(word_unit in desmeumenes_lexeis):
                keyword_state_num = 200 + desmeumenes_lexeis.index(word_unit)
                result.append(keyword_state_num)
        else:
                result.append(current_state)
```

Έπειτα ενημερώνουμε τη μεταβλητή *line* ώστε να κρατάει την τρέχουσα γραμμή και να μην ξεκινάει πάλι από το 1. *Τυπώνουμε το αποτέλεσμα* ώστε να βλέπουμε στο τερματικό όλα τα TOKEN που δημιουργήθηκαν και *το επιστρέφουμε ώστε να μπορούμε να το περάσουμε ως είσοδο στην συνάρτηση syn() του συντακτικού αναλυτή.*

Εάν φτάσουμε στο τέλος αρχείου το οποίο έχει τη δική του οικογένεια '*FAMILY: EOF*' κάνουμε break καθώς έχουμε βρει πλέον όλα τα TOKEN του αρχείου .cpy

```
        line=current_line    #gia na synexizei h arithmish grammwn apo ekei poy emeine

        print(result)

        return result


while(True):
        l = lex()
        if (l[0] == 'FAMILY: EOF'):
                break
```

ΑΠΟΤΕΛΕΣΜΑΤΑ ΛΕΚΤΙΚΟΥ ΑΝΑΛΥΤΗ

Ενδεικτικά παραθέτουμε το αποτέλεσμα του λεκτικού αναλυτή για όλες τις λεκτικές μονάδες των πρώτων και τελευταίων 10 γραμμών του αρχείου *test.cpy* που περιέχει τα 4 προγράμματα που μας δόθηκαν στην εκφώνηση.

```
['FAMILY: keyword', 'def', 1, 208]
['FAMILY: identifier', 'main_factorial', 1, 100]
['FAMILY: groupSymbol', '(', 1, 118]
['FAMILY: groupSymbol', ')', 1, 119]
['FAMILY: delimiter', ':', 1, 115]
['FAMILY: groupSymbol', '#{', 2, 120]
['FAMILY: keyword', '#declare', 4, 200]
['FAMILY: identifier', 'x', 4, 100]
['FAMILY: keyword', '#declare', 5, 200]
['FAMILY: identifier', 'i', 5, 100]
['FAMILY: delimiter', ',', 5, 114]
['FAMILY: identifier', 'fact', 5, 100]
['FAMILY: identifier', 'x', 8, 100]
['FAMILY: assignment', '=', 8, 112]
['FAMILY: keyword', 'int', 8, 206]
['FAMILY: groupSymbol', '(', 8, 118]
['FAMILY: keyword', 'input', 8, 207]
['FAMILY: groupSymbol', '(', 8, 118]
['FAMILY: groupSymbol', ')', 8, 119]
['FAMILY: groupSymbol', ')', 8, 119]
['FAMILY: delimiter', ';', 8, 113]
['FAMILY: identifier', 'fact', 9, 100]
['FAMILY: assignment', '=', 9, 112]
['FAMILY: number', '1', 9, 101]
['FAMILY: delimiter', ';', 9, 113]
['FAMILY: identifier', 'i', 10, 100]
['FAMILY: assignment', '=', 10, 112]
['FAMILY: number', '1', 10, 101]
['FAMILY: delimiter', ';', 10, 113]
```

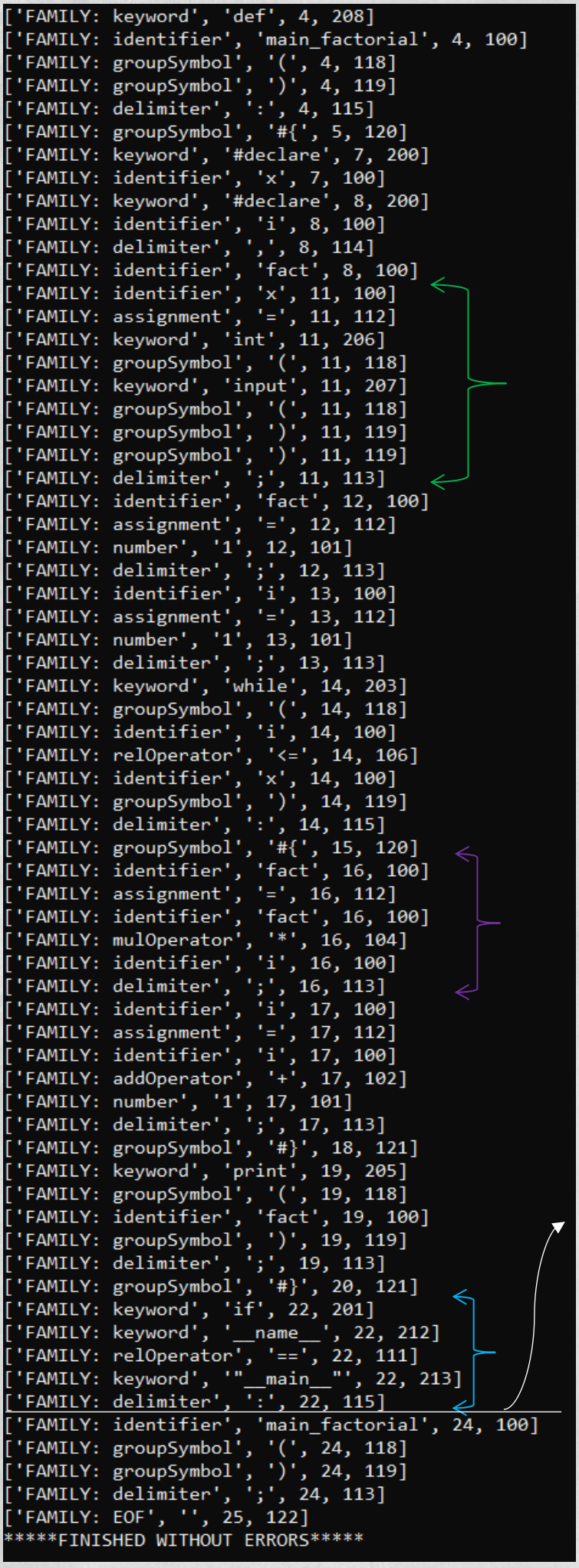
```
['FAMILY: identifier', 'i', 75, 100]
['FAMILY: addOperator', '+', 75, 102]
['FAMILY: number', '1', 75, 101]
['FAMILY: delimiter', ';', 75, 113]
['FAMILY: groupSymbol', '#}', 76, 121]
['FAMILY: keyword', 'if', 78, 201]
['FAMILY: keyword', '__name__', 78, 212]
['FAMILY: relOperator', '==', 78, 111]
['FAMILY: keyword', '"__main__"', 78, 213]
['FAMILY: delimiter', ':', 78, 115]
['FAMILY: identifier', 'main_factorial', 80, 100]
['FAMILY: groupSymbol', '(', 80, 118]
['FAMILY: groupSymbol', ')', 80, 119]
['FAMILY: delimiter', ';', 80, 113]
['FAMILY: identifier', 'main_fibonacci', 81, 100]
['FAMILY: groupSymbol', '(', 81, 118]
['FAMILY: groupSymbol', ')', 81, 119]
['FAMILY: delimiter', ';', 81, 113]
['FAMILY: identifier', 'main_countdigits', 82, 100]
['FAMILY: groupSymbol', '(', 82, 118]
['FAMILY: groupSymbol', ')', 82, 119]
['FAMILY: delimiter', ';', 82, 113]
['FAMILY: identifier', 'main_primes', 83, 100]
['FAMILY: groupSymbol', '(', 83, 118]
['FAMILY: groupSymbol', ')', 83, 119]
['FAMILY: delimiter', ';', 83, 113]
['FAMILY: EOF', '', 84, 122]
*****FINISHED WITHOUT ERRORS*****
```

Για να ελέγξουμε ότι ο λεκτικός αναλυτής λειτουργεί σωστά και εμφανίζει τα μηνύματα λάθους όπου αυτά συμβαίνουν, δημιουργούμε ένα μικρότερο αρχείο που περιέχει μόνο ένα από αυτά τα προγράμματα, ώστε να είναι και λιγότερες οι γραμμές στο αποτέλεσμα του τερματικού.

Καθώς στείλαμε όλα τα παραδείγματα που μας είχαν δοθεί, δεν θα τα επεξεργαστούμε όλα αναλυτικά, παρά μόνο το πρώτο πρόγραμμα το οποίο είναι το main_factorial για το οποίο είχαμε και το παράδειγμα από τις περσινές σημειώσεις ως αναφορά.

❖ Αρχείο main_factorial()

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_factorial():
#{
    #$ declarations #$
    #declare x
    #declare i,fact

    #$ body of main_factorial #$
    x = int(input());
    fact = 1;
    i = 1;
    while (i<=x):
    #{
        fact = fact * i;
        i = i + 1;
    #}
    print(fact);
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_factorial();

```



Εδώ βλέπουμε ότι ο λεκτικός αναλυτής που δημιουργήσαμε, τυπώνει σωστά όλες τις λεκτικές μονάδες που βρίσκει μέσα στο πρόγραμμα main_factorial.

Για κάθε λεκτική μονάδα αναγνωρίζει σωστά το αντίστοιχο family, όπως έχει οριστεί και από την εκφώνηση της άσκησης, καθώς σωστή είναι επίσης και η αρίθμηση των γραμμών στις οποίες ανιχνεύει την αντίστοιχη λεκτική μονάδα.

Στο τέλος εφόσον δεν έχει βρει κάποιο λάθος ο λεκτικός αναλυτής, τυπώνει και το αντίστοιχο μήνυμα

*****FINISHED WITHOUT ERRORS*****

ώστε να καταλάβουμε ότι όλα πήγαν καλά και ότι δεν υπήρχαν λάθη.

Για παράδειγμα στη γραμμή 11 έχουν αναγνωριστεί σωστά όλες οι λεκτικές μονάδες χωρίς καμία να χαθεί.

x = int(input());

Επίσης στη γραμμή 16 το ίδιο.

fact = fact * i;

Επίσης στη γραμμή 22 το ίδιο.

if __name__ == "__main__":

Επίσης για το σχόλιο που υπάρχει στη γραμμή 23 παρατηρούμε ότι σωστά δεν έχει δημιουργηθεί κάποια λεκτική μονάδα, συνεπώς δεν υπάρχει η γραμμή 23 στο αποτέλεσμα μας και περνάμε κατευθείαν στη γραμμή 24.

```
program     family:"keyword",  line: 1
factorial   family:"id",  line: 1
{           family:"groupSymbol",  line: 2
declare     family:"keyword",  line: 4
x           family:"id",  line: 4
;           family:"delimiter",  line: 4
declare     family:"keyword",  line: 5
i           family:"id",  line: 5
,           family:"delimiter",  line: 5
fact        family:"id",  line: 5
;           family:"delimiter",  line: 5
input       family:"keyword",  line: 8
(           family:"groupSymbol",  line: 8
x           family:"id",  line: 8
)           family:"groupSymbol",  line: 8
;           family:"delimiter",  line: 8
fact        family:"id",  line: 9
:=          family:"assignment",  line: 9
1           family:"number",  line: 9
;           family:"delimiter",  line: 9
i           family:"id",  line: 10
:=          family:"assignment",  line: 10
1           family:"number",  line: 10
;           family:"delimiter",  line: 10
while       family:"keyword",  line: 11
(           family:"groupSymbol",  line: 11
i           family:"id",  line: 11
<=          family:"relOperator",  line: 11
x           family:"id",  line: 11
```

```
)           family:"groupSymbol",  line: 11
{           family:"groupSymbol",  line: 12
fact        family:"id",  line: 13
:=          family:"assignment",  line: 13
fact        family:"id",  line: 13
*           family:"mulOperator",  line: 13
i           family:"id",  line: 13
;           family:"delimiter",  line: 13
i           family:"id",  line: 14
:=          family:"assignment",  line: 14
i           family:"id",  line: 14
+           family:"addOperator",  line: 14
1           family:"number",  line: 14
;           family:"delimiter",  line: 14
}           family:"groupSymbol",  line: 15
;           family:"delimiter",  line: 15
print       family:"keyword",  line: 16
(           family:"groupSymbol",  line: 16
fact        family:"id",  line: 16
)           family:"groupSymbol",  line: 16
;           family:"delimiter",  line: 16
}           family:"groupSymbol",  line: 17
.           family:"delimiter",  line: 17
```

Επιπλέον λαμβάνοντας υπόψη μας το παράδειγμα για το πρόγραμμα factorial που μας δόθηκε στις σημειώσεις για την λεκτική ανάλυση, παρατηρούμε πως κάνοντας τις απαραίτητες αλλαγές (π.χ. **:= → =** , **program → def** , **line:1 → line:4** [έχουμε προσθέσει στην αρχή σε σχόλια τα ονοματεπώνυμα μας]) έχει αναγνωρίσει πανομοιότυπα τις λεκτικές μονάδες αλλά και τις οικογένειες στις οποίες αυτές εντάσσονται. Ωστόσο καθώς το συγκεκριμένο παράδειγμα αναφέρεται στην περσινή γλώσσα και όχι στην CutePy την οποία εμείς φέτος εξετάζουμε, δεν περιέχει τις λεκτικές μονάδες που σχηματίστηκαν στην δική μας main, ενώ το δικό μας αποτέλεσμα τις έχει τυπώσει και εκείνες με σωστό τρόπο.

❖ Αρχείο main_fibonacci()

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_fibonacci():
#{
    #declare x
    def fibonacci(x):
    #{
        if (x<=1):
            return(x);
        else:
            return (fibonacci(x-1)+fibonacci(x-2));
    #}
    x = int(input());
    print(fibonacci(x));
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_fibonacci();

```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_fibonacci', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
['FAMILY: keyword', '#declare', 6, 200]
['FAMILY: identifier', 'x', 6, 100]
['FAMILY: keyword', 'def', 7, 208]
['FAMILY: identifier', 'fibonacci', 7, 100]
['FAMILY: groupSymbol', '(', 7, 118]
['FAMILY: identifier', 'x', 7, 100]
['FAMILY: groupSymbol', ')', 7, 119]
['FAMILY: delimiter', ':', 7, 115]
['FAMILY: groupSymbol', '#{', 8, 120]
['FAMILY: keyword', 'if', 9, 201]
['FAMILY: groupSymbol', '(', 9, 118]
['FAMILY: identifier', 'x', 9, 100]
['FAMILY: relOperator', '<=', 9, 106]
['FAMILY: number', '1', 9, 101]
['FAMILY: groupSymbol', ')', 9, 119]
['FAMILY: delimiter', ':', 9, 115]
['FAMILY: keyword', 'return', 10, 204]
['FAMILY: groupSymbol', '(', 10, 118]
['FAMILY: identifier', 'x', 10, 100]
['FAMILY: groupSymbol', ')', 10, 119]
['FAMILY: delimiter', ';', 10, 113]
['FAMILY: keyword', 'else', 11, 202]
['FAMILY: delimiter', ':', 11, 115]
['FAMILY: keyword', 'return', 12, 204]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'fibonacci', 12, 100]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: addOperator', '-', 12, 103]
['FAMILY: number', '1', 12, 101]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: addOperator', '+', 12, 102]
['FAMILY: identifier', 'fibonacci', 12, 100]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: addOperator', '-', 12, 103]
['FAMILY: number', '2', 12, 101]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: delimiter', ';', 12, 113]
['FAMILY: groupSymbol', '#}', 13, 121]
['FAMILY: identifier', 'x', 14, 100]
['FAMILY: assignment', '=', 14, 112]
['FAMILY: keyword', 'int', 14, 206]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: keyword', 'input', 14, 207]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: groupSymbol', ')', 14, 119]
['FAMILY: groupSymbol', ')', 14, 119]
['FAMILY: delimiter', ';', 14, 113]
['FAMILY: keyword', 'print', 15, 205]
['FAMILY: groupSymbol', '(', 15, 118]
['FAMILY: identifier', 'fibonacci', 15, 100]
['FAMILY: groupSymbol', '(', 15, 118]
['FAMILY: identifier', 'x', 15, 100]
['FAMILY: groupSymbol', ')', 15, 119]
['FAMILY: groupSymbol', ')', 15, 119]
['FAMILY: delimiter', ';', 15, 113]
['FAMILY: groupSymbol', '#}', 16, 121]
['FAMILY: keyword', 'if', 18, 201]
['FAMILY: keyword', '__name__', 18, 212]
['FAMILY: relOperator', '==', 18, 111]
['FAMILY: keyword', '"__main__"', 18, 213]
['FAMILY: delimiter', ':', 18, 115]
['FAMILY: identifier', 'main_fibonacci', 20, 100]
['FAMILY: groupSymbol', '(', 20, 118]
['FAMILY: groupSymbol', ')', 20, 119]
['FAMILY: delimiter', ';', 20, 113]
['FAMILY: EOF', '', 21, 122]
*****FINISHED WITHOUT ERRORS*****
```

Παραθέτονται επίσης τα αποτελέσματα της λεκτικής ανάλυσης του προγράμματος main_fibonacci. Βλέπουμε πως επίσης όλες οι λεκτικές μονάδες έχουν αναγνωριστεί σωστά χωρίς να υπάρχει κανένα πρόβλημα.

❖ Αρχείο main_countdigits()

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_countdigits():
#{
    #declare x, count
    x = int(input());
    count = 0;
    while (x>0):
    #{
        x = x // 10;
        count = count + 1;
    #}
    print(count);
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_countdigits();

```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_countdigits', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
['FAMILY: keyword', '#declare', 6, 200]
['FAMILY: identifier', 'x', 6, 100]
['FAMILY: delimiter', ',', 6, 114]
['FAMILY: identifier', 'count', 6, 100]
['FAMILY: identifier', 'x', 7, 100]
['FAMILY: assignment', '=', 7, 112]
['FAMILY: keyword', 'int', 7, 206]
['FAMILY: groupSymbol', '(', 7, 118]
['FAMILY: keyword', 'input', 7, 207]
['FAMILY: groupSymbol', '(', 7, 118]
['FAMILY: groupSymbol', ')', 7, 119]
['FAMILY: groupSymbol', ')', 7, 119]
['FAMILY: delimiter', ';', 7, 113]
['FAMILY: identifier', 'count', 8, 100]
['FAMILY: assignment', '=', 8, 112]
['FAMILY: number', '0', 8, 101]
['FAMILY: delimiter', ';', 8, 113]
['FAMILY: keyword', 'while', 9, 203]
['FAMILY: groupSymbol', '(', 9, 118]
['FAMILY: identifier', 'x', 9, 100]
['FAMILY: relOperator', '>', 9, 109]
['FAMILY: number', '0', 9, 101]
['FAMILY: groupSymbol', ')', 9, 119]
['FAMILY: delimiter', ':', 9, 115]
['FAMILY: groupSymbol', '#{', 10, 120]
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: mulOperator', '//', 11, 105]
['FAMILY: number', '10', 11, 101]
['FAMILY: delimiter', ';', 11, 113]
['FAMILY: identifier', 'count', 12, 100]
['FAMILY: assignment', '=', 12, 112]
['FAMILY: identifier', 'count', 12, 100]
['FAMILY: addOperator', '+', 12, 102]
['FAMILY: number', '1', 12, 101]
['FAMILY: delimiter', ';', 12, 113]
['FAMILY: groupSymbol', '#}', 13, 121]
['FAMILY: keyword', 'print', 14, 205]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: identifier', 'count', 14, 100]
['FAMILY: groupSymbol', ')', 14, 119]
['FAMILY: delimiter', ';', 14, 113]
['FAMILY: groupSymbol', '#}', 15, 121]
['FAMILY: keyword', 'if', 17, 201]
['FAMILY: keyword', '__name__', 17, 212]
['FAMILY: relOperator', '==', 17, 111]
['FAMILY: keyword', '"__main__"', 17, 213]
['FAMILY: delimiter', ':', 17, 115]
['FAMILY: identifier', 'main_countdigits', 19, 100]
['FAMILY: groupSymbol', '(', 19, 118]
['FAMILY: groupSymbol', ')', 19, 119]
['FAMILY: delimiter', ';', 19, 113]
['FAMILY: EOF', '', 20, 122]
*****FINISHED WITHOUT ERRORS*****
```

Παραθέτονται επίσης τα αποτελέσματα της λεκτικής ανάλυσης του προγράμματος main_countdigits. Βλέπουμε πως επίσης όλες οι λεκτικές μονάδες έχουν αναγνωριστεί σωστά χωρίς να υπάρχει κανένα πρόβλημα.

❖ Αρχείο main_primes()

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_primes():
#{
    #declare i
    def isPrime(x):
    #{
        #declare i
        def divides(x,y):
        #{
            if (y == (y//x) * x):
                return (1);
            else:
            #{
                return (0);
            #}
        #}
        i = 2;
        while (i<x):
        #{
            if (divides(i,x)==1):
                return (0);
            i = i + 1;
        #}
        return (1);
    #}


    #$ body of main_primes #$
    i = 2;
    while (i<=30):
    #{
        if (isPrime(i)==1):
            print(i);
        i = i + 1;
    #}
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_primes();

```

Παραθέτονται επίσης τα αποτελέσματα της λεκτικής ανάλυσης του προγράμματος main_primes. Βλέπουμε πως επίσης όλες οι λεκτικές μονάδες έχουν αναγνωριστεί σωστά χωρίς να υπάρχει κανένα πρόβλημα.

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_primes', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
['FAMILY: keyword', '#declare', 6, 200]
['FAMILY: identifier', 'i', 6, 100]
['FAMILY: keyword', 'def', 7, 208]
['FAMILY: identifier', 'isPrime', 7, 100]
['FAMILY: groupSymbol', '(', 7, 118]
['FAMILY: identifier', 'x', 7, 100]
['FAMILY: groupSymbol', ')', 7, 119]
['FAMILY: delimiter', ':', 7, 115]
['FAMILY: groupSymbol', '#{', 8, 120]
['FAMILY: keyword', '#declare', 9, 200]
['FAMILY: identifier', 'i', 9, 100]
['FAMILY: keyword', 'def', 10, 208]
['FAMILY: identifier', 'divides', 10, 100]
['FAMILY: groupSymbol', '(', 10, 118]
['FAMILY: identifier', 'x', 10, 100]
['FAMILY: delimiter', ',', 10, 114]
['FAMILY: identifier', 'y', 10, 100]
['FAMILY: groupSymbol', ')', 10, 119]
['FAMILY: delimiter', ':', 10, 115]
['FAMILY: groupSymbol', '#{', 11, 120]
['FAMILY: keyword', 'if', 12, 201]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'y', 12, 100]
['FAMILY: relOperator', '==', 12, 111]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'y', 12, 100]
['FAMILY: mulOperator', '//', 12, 105]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: mulOperator', '*', 12, 104]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: delimiter', ':', 12, 115]
['FAMILY: keyword', 'return', 13, 204]
['FAMILY: groupSymbol', '(', 13, 118]
['FAMILY: number', '1', 13, 101]
['FAMILY: groupSymbol', ')', 13, 119]
['FAMILY: delimiter', ';', 13, 113]
['FAMILY: keyword', 'else', 14, 202]
['FAMILY: delimiter', ':', 14, 115]
['FAMILY: groupSymbol', '#{', 15, 120]
['FAMILY: keyword', 'return', 16, 204]
['FAMILY: groupSymbol', '(', 16, 118]
['FAMILY: number', '0', 16, 101]
['FAMILY: groupSymbol', ')', 16, 119]
['FAMILY: delimiter', ';', 16, 113]
['FAMILY: groupSymbol', '#}', 17, 121]
['FAMILY: groupSymbol', '#}', 18, 121]
['FAMILY: identifier', 'i', 19, 100]
['FAMILY: assignment', '=', 19, 112]
['FAMILY: number', '2', 19, 101]
['FAMILY: delimiter', ';', 19, 113]
['FAMILY: keyword', 'while', 20, 203]
['FAMILY: groupSymbol', '(', 20, 118]
['FAMILY: identifier', 'i', 20, 100]
['FAMILY: relOperator', '<', 20, 107]
['FAMILY: identifier', 'x', 20, 100]
['FAMILY: groupSymbol', ')', 20, 119]
['FAMILY: delimiter', ':', 20, 115]
['FAMILY: groupSymbol', '#{', 21, 120]
['FAMILY: keyword', 'if', 22, 201]
['FAMILY: groupSymbol', '(', 22, 118]
['FAMILY: identifier', 'divides', 22, 100]
['FAMILY: groupSymbol', '(', 22, 118]
['FAMILY: identifier', 'divides', 22, 100]
['FAMILY: groupSymbol', '(', 22, 118]
['FAMILY: identifier', 'i', 22, 100]
['FAMILY: delimiter', ',', 22, 114]
['FAMILY: identifier', 'x', 22, 100]
['FAMILY: groupSymbol', ')', 22, 119]
['FAMILY: relOperator', '==', 22, 111]
['FAMILY: number', '1', 22, 101]
['FAMILY: groupSymbol', ')', 22, 119]
['FAMILY: delimiter', ':', 22, 115]
['FAMILY: keyword', 'return', 23, 204]
['FAMILY: groupSymbol', '(', 23, 118]
['FAMILY: number', '0', 23, 101]
['FAMILY: groupSymbol', ')', 23, 119]
['FAMILY: delimiter', ';', 23, 113]
['FAMILY: identifier', 'i', 24, 100]
['FAMILY: assignment', '=', 24, 112]
['FAMILY: identifier', 'i', 24, 100]
['FAMILY: addOperator', '+', 24, 102]
['FAMILY: number', '1', 24, 101]
['FAMILY: delimiter', ';', 24, 113]
['FAMILY: groupSymbol', '#}', 25, 121]
['FAMILY: keyword', 'return', 26, 204]
['FAMILY: groupSymbol', '(', 26, 118]
['FAMILY: number', '1', 26, 101]
['FAMILY: groupSymbol', ')', 26, 119]
['FAMILY: delimiter', ';', 26, 113]
['FAMILY: groupSymbol', '#}', 27, 121]
['FAMILY: identifier', 'i', 31, 100]
['FAMILY: assignment', '=', 31, 112]
['FAMILY: number', '2', 31, 101]
['FAMILY: delimiter', ';', 31, 113]
['FAMILY: keyword', 'while', 32, 203]
['FAMILY: groupSymbol', '(', 32, 118]
['FAMILY: identifier', 'i', 32, 100]
['FAMILY: relOperator', '<=', 32, 106]
['FAMILY: number', '30', 32, 101]
['FAMILY: groupSymbol', ')', 32, 119]
['FAMILY: delimiter', ':', 32, 115]
['FAMILY: groupSymbol', '#{', 33, 120]
['FAMILY: keyword', 'if', 34, 201]
['FAMILY: groupSymbol', '(', 34, 118]
['FAMILY: identifier', 'isPrime', 34, 100]
['FAMILY: groupSymbol', '(', 34, 118]
['FAMILY: identifier', 'i', 34, 100]
['FAMILY: groupSymbol', ')', 34, 119]
['FAMILY: relOperator', '==', 34, 111]
['FAMILY: number', '1', 34, 101]
['FAMILY: groupSymbol', ')', 34, 119]
['FAMILY: delimiter', ':', 34, 115]
['FAMILY: keyword', 'print', 35, 205]
['FAMILY: groupSymbol', '(', 35, 118]
['FAMILY: identifier', 'i', 35, 100]
['FAMILY: groupSymbol', ')', 35, 119]
['FAMILY: delimiter', ';', 35, 113]
['FAMILY: identifier', 'i', 36, 100]
['FAMILY: assignment', '=', 36, 112]
['FAMILY: identifier', 'i', 36, 100]
['FAMILY: addOperator', '+', 36, 102]
['FAMILY: number', '1', 36, 101]
['FAMILY: delimiter', ';', 36, 113]
['FAMILY: groupSymbol', '#}', 37, 121]
['FAMILY: groupSymbol', '#}', 38, 121]
['FAMILY: keyword', 'if', 40, 201]
['FAMILY: keyword', '__name__', 40, 212]
['FAMILY: relOperator', '==', 40, 111]
['FAMILY: keyword', '"__main__"', 40, 213]
['FAMILY: delimiter', ':', 40, 115]
['FAMILY: identifier', 'main_primes', 42, 100]
['FAMILY: groupSymbol', '(', 42, 118]
['FAMILY: groupSymbol', ')', 42, 119]
['FAMILY: delimiter', ';', 42, 113]
['FAMILY: EOF', '', 43, 122]
*****FINISHED WITHOUT ERRORS*****
```

Ανίχνευση λανθασμένων λεκτικών μονάδων από τον λεκτικό αναλυτή.

Παρακάτω παραθέτουμε όλα τα πιθανά error που μπορούσαμε να σκεφτούμε για την γλώσσα CutePy καθώς φτιάχναμε τον λεκτικό αναλυτή μας. Εφόσον ο αναλυτής εντοπίζει αυτά τα λάθη και τυπώνει αντίστοιχο μήνυμα λάθους, τερματίζοντας το πρόγραμμα όταν αυτό βρεθεί, συμπεραίνουμε ότι η λεκτική μας ανάλυση έχει γίνει με επιτυχία, τουλάχιστον για όλες αυτές τις περιπτώσεις που καταφέραμε να σκεφτούμε.

- ERROR για εύρεση μη αποδεκτού συμβόλου της CutePy.

```
4   def main_factorial():
5   #{
6       #$ declarations #$
7       #declare x
8       #declare i,fact
9       @
10      #$ body of main_factorial #$
```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_factorial', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
['FAMILY: keyword', '#declare', 7, 200]
['FAMILY: identifier', 'x', 7, 100]
['FAMILY: keyword', '#declare', 8, 200]
['FAMILY: identifier', 'i', 8, 100]
['FAMILY: delimiter', ',', 8, 114]
['FAMILY: identifier', 'fact', 8, 100]
***ERROR(lex)***: Found a not accepted symbol ( @ ) in line: 9
```

- ERROR για εύρεση μονού / χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας για την γλώσσα CutePy.

```
15      #{
16          fact = fact / i;
17          i = i + 1;
```

```
['FAMILY: identifier', 'fact', 16, 100]
['FAMILY: assignment', '=', 16, 112]
['FAMILY: identifier', 'fact', 16, 100]
***ERROR(lex)***: Found a single / in line: 16
```

- ERROR για εύρεση αλφαριθμητικού το οποίο να ξεκινά με ψηφίο, κάτι το οποίο δεν είναι αποδεκτό για την γλώσσα CutePy.

```
10      #$ body of main_factorial #$
11      x = int(input());
12      fact = 14a;
13      i = 1;
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: keyword', 'input', 11, 207]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: groupSymbol', ')', 11, 119]
['FAMILY: groupSymbol', ')', 11, 119]
['FAMILY: delimiter', ';', 11, 113]
['FAMILY: identifier', 'fact', 12, 100]
['FAMILY: assignment', '=', 12, 112]
***ERROR(lex)***: Found a letter character after a digit character ( 14a ) in line: 12
```

- ERROR για εύρεση ακέραιας σταθεράς η οποία να αποτελείται από ακολουθία αριθμητικών ψηφίων της οποίας η τιμή ξεπερνά το 2^{32} - 1, κάτι το οποίο δεν πρέπει να συμβαίνει στην γλώσσα CutePy.

```
11      x = int(input());
12      fact = 999999999999999999999;
13      i = 1;
```

```
['FAMILY: identifier', 'fact', 12, 100]
['FAMILY: assignment', '=', 12, 112]
***ERROR(lex)***: The number ' 99999999999999999999 ' is not inside the accepted range of [-((2^32)-1), (2^32)-1)] in line: 12
```

- ERROR για εύρεση αναγνωριστικού του οποίου η συμβολοσειρά ξεπερνάει τους 30 χαρακτήρες, κάτι το οποίο δεν είναι αποδεκτό στην γλώσσα CutePy.

```
10      #$ body of main_factorial #$
11      xdbfhbrefeyrfyhrfrfrhhfwuheubfiw = int(input());
12      fact = 1;
```

```
['FAMILY: keyword', '#declare', 8, 200]
['FAMILY: identifier', 'i', 8, 100]
['FAMILY: delimiter', ',', 8, 114]
['FAMILY: identifier', 'fact', 8, 100]
***ERROR(lex)***: Found an identifier with over 30 characters ( xdbfhbrefeyrfyhrfrfrhhfwuheubfiw ) in line: 11
```

- ERROR για εύρεση EOF πριν προλάβουν να κλείσουν τα σχόλια που έχουμε ανοίξει προς το τέλος του αρχείου.

```
22  if __name__ == "__main__":
23      #$ call of main functions #$
24      main_factorial();
25      #$ eof error
26
```

```
['FAMILY: identifier', 'main_factorial', 24, 100]
['FAMILY: groupSymbol', '(', 24, 118]
['FAMILY: groupSymbol', ')', 24, 119]
['FAMILY: delimiter', ';', 24, 113]
***ERROR(lex)***: Comments begin correctly with #$ but we get EOF before they end in line: 26
```

- ERROR για εύρεση μονού $ χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας από μόνος του για την γλώσσα CutePy, καθώς θα έπρεπε να προηγείται χαρακτήρας # για να θεωρηθεί όλο μαζί (#$) σύμβολο για άνοιγμα ή κλείσιμο σχολίων.

```
12      fact = 1;
13      i = 1; $
14      while (i<=x):
```

```
['FAMILY: identifier', 'i', 13, 100]
['FAMILY: assignment', '=', 13, 112]
['FAMILY: number', '1', 13, 101]
['FAMILY: delimiter', ';', 13, 113]
***ERROR(lex)***: Found a single $ in line: 13
```

- ERROR για εύρεση μονού { χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας από μόνος του για την γλώσσα CutePy, καθώς θα έπρεπε να προηγείται χαρακτήρας # για να θεωρηθεί όλο μαζί (#{) σύμβολο για το άνοιγμα ενός block.

```
4   def main_factorial():
5   {
6       #$ declarations #$
```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_factorial', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
***ERROR(lex)***: Found a single { in line: 5
```

- ERROR για εύρεση μονού } χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας από μόνος του για την γλώσσα CutePy, καθώς θα έπρεπε να προηγείται χαρακτήρας # για να θεωρηθεί όλο μαζί (#}) σύμβολο για το κλείσιμο ενός block.

```
16          fact = fact * i;
17          i = i + 1; }
18      #}
```

```
['FAMILY: identifier', 'i', 17, 100]
['FAMILY: assignment', '=', 17, 112]
['FAMILY: identifier', 'i', 17, 100]
['FAMILY: addOperator', '+', 17, 102]
['FAMILY: number', '1', 17, 101]
['FAMILY: delimiter', ';', 17, 113]
***ERROR(lex)***: Found a single } in line: 17
```

- ERROR για εύρεση αναγνωριστικού το οποίο να ξεκινάει από τον χαρακτήρα _, κάτι το οποίο δεν είναι αποδεκτό για τη γλώσσα CutePy.

```
11      _x = int(input());
12      fact = 1;
```

```
['FAMILY: keyword', '#declare', 8, 200]
['FAMILY: identifier', 'i', 8, 100]
['FAMILY: delimiter', ',', 8, 114]
['FAMILY: identifier', 'fact', 8, 100]
***ERROR(lex)***: Found the word ' _x ' starting with _ in line: 11
```

- ERROR για εύρεση μονού ! χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας από μόνος του για την γλώσσα CutePy, καθώς θα έπρεπε να ακολουθεί χαρακτήρας = για να θεωρηθεί όλο μαζί (!=) ο τελεστής συσχέτισης που δηλώνει το διαφορετικό.

```
12      fact = ! 1;
13      i = 1;
```

```
['FAMILY: delimiter', ';', 11, 113]
['FAMILY: identifier', 'fact', 12, 100]
['FAMILY: assignment', '=', 12, 112]
***ERROR(lex)***: Found a single ! in line: 12
```

- ERROR για εύρεση μονού # χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας από μόνος του για την γλώσσα CutePy, καθώς θα έπρεπε να ακολουθεί είτε χαρακτήρας { για να θεωρηθεί όλο μαζί (#{) σύμβολο για το άνοιγμα ενός block, είτε χαρακτήρας } για να θεωρηθεί όλο μαζί (#}) σύμβολο για το κλείσιμο ενός block, είτε χαρακτήρας $ για να θεωρηθεί όλο μαζί (#$) σύμβολο για άνοιγμα ή κλείσιμο σχολίων.

```
14      while (i<=x):
15      #
16          fact = fact * i;
```

```
['FAMILY: groupSymbol', ')', 14, 119]
['FAMILY: delimiter', ':', 14, 115]
***ERROR(lex)***: Found a single # in line: 15
```

- ERROR για εύρεση αναγνωριστικού το οποίο να ξεκινάει με το σύμβολο # και ωστόσο να μην είναι το αναγνωριστικό #declare, το οποίο είναι το μόνο δέχεται η γλώσσα CutePy να ξεκινάει με #, καθώς αποτελεί δεσμευμένη λέξη.

```
6       #$ declarations #$
7       #dec x
8       #declare i,fact
```

```
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
***ERROR(lex)***: Found the identifier ' #dec ' starting with # (NOT #declare) in line: 7
```

- ERROR για εύρεση “ χαρακτήρα, ο οποίος δεν είναι αποδεκτός χαρακτήρας για την γλώσσα CutePy και μπορούμε να τον συναντήσουμε μόνο ως μέρος της δεσμευμένης λέξης “__main__”.

```
14      while ( " i<=x):
15      #{
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: groupSymbol', '(', 14, 118]
***ERROR(lex)***: Found a single " in line: 14
```

- ERROR για εύρεση αναγνωριστικού το οποίο περιέχει το σύμβολο “ και ωστόσο δεν είναι η δεσμευμένη λέξη “__main__”, η οποία είναι η μόνη λέξη που περιέχει το “ και την δέχεται η γλώσσα CutePy, καθώς αποτελεί δεσμευμένη λέξη.

```
22  if __name__ == "__world__" :
```

```
['FAMILY: keyword', 'if', 22, 201]
['FAMILY: keyword', '__name__', 22, 212]
['FAMILY: relOperator', '==', 22, 111]
***ERROR(lex)***: Found the identifier ' "__world__" ' containing " (NOT "__main__") in line: 22
```

Επίσης παραθέτουμε κάποιες περιπτώσεις οι οποίες σύμφωνα με την εκφώνηση που μας έχει δοθεί για τη γλώσσα πρέπει να περνάνε την λεκτική ανάλυση, κάτι το οποίο συμβαίνει και στο δικό μας πρόγραμμα, και μας επιτρέπει να συμπεράνουμε ότι **το αυτόματο των καταστάσεων που δημιουργήσαμε, τις έχει συμπεριλάβει ορθά.**

- Σωστή αναγνώριση αναγνωριστικού το οποίο περιέχει γράμματα και ψηφία, εφόσον ξεκινάει από γράμμα.

```
7     #declare xx2
8     #declare i,fact
9
10    #$ body of main_factorial #$
11    xx2 = int(input());
12    fact = 1;
13    i = 1;
14    while (i<=xx2):
```

```
7     #declare xx2x
8     #declare i,fact
9
10    #$ body of main_factorial #$
11    xx2x = int(input());
12    fact = 1;
13    i = 1;
14    while (i<=xx2x):
```

- Σωστή αναγνώριση αναγνωριστικού το οποίο περιέχει γράμματα και κάτω παύλα, εφόσον ξεκινάει από γράμμα.

```
7     #declare xx_
8     #declare i,fact
9
10    #$ body of main_factorial #$
11    xx_ = int(input());
12    fact = 1;
13    i = 1;
14    while (i<=xx_):
```

```
7     #declare xx_x
8     #declare i,fact
9
10    #$ body of main_factorial #$
11    xx_x = int(input());
12    fact = 1;
13    i = 1;
14    while (i<=xx_x):
```

- Σωστή αναγνώριση αναγνωριστικού το οποίο περιέχει γράμματα, ψηφία και κάτω παύλα, εφόσον ξεκινάει από γράμμα.

```
7     #declare xx_23x
8     #declare i,fact
9
10    #$ body of main_factorial #$
11    xx_23x = int(input());
12    fact = 1;
13    i = 1;
14    while (i<=xx_23x):
```

Όλες οι παραπάνω περιπτώσεις λειτουργούν σωστά και στο τέλος τυπώνεται το μήνυμα: `*****FINISHED WITHOUT ERRORS*****`

ΑΠΟΤΕΛΕΣΜΑΤΑ ΣΥΝΤΑΚΤΙΚΟΥ ΑΝΑΛΥΤΗ

Για να ελέγξουμε ότι ο συντακτικός αναλυτής λειτουργεί σωστά και εμφανίζει τα σωστά μηνύματα λάθους όπου αυτά συμβαίνουν, φτιάχνουμε μόνοι μας διάφορα λανθασμένα συντακτικά παραδείγματα και βλέπουμε ότι το πρόγραμμά μας τα εντοπίζει και εφόσον τυπώσει το λάθος που βρήκε, τερματίζει.

❖ Αρχείο main_factorial

- ERROR καθώς η κύρια συνάρτηση πρέπει να ξεκινάει με def.

```
3
4   main_factorial():
5   #{
6       #$ declarations #$
```

```
['FAMILY: identifier', 'main_factorial', 4, 100]
***ERROR(syn)***:{def_main_function()}: Program doesn't start with def statement in line:  4
```

- ERROR καθώς δεν έχουμε δώσει νόμιμα ονόματα στην κύρια συνάρτηση μας. Αυτό γιατί πρέπει οι κύριες συναρτήσεις μας να ξεκινούν με το διακριτικό main_.

```
4   def mai_factorial():
5   #{
6       #$ declarations #$
```

```
4   def factorial():
5   #{
6       #$ declarations #$
```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'mai_factorial', 4, 100]
***ERROR(syn)***:{def_main_function()}: Function's name does not start with 'main_' in line:  4
```

- ERROR καθώς και στις 2 παρακάτω περιπτώσεις συναντάμε παρένθεση που κλείνει χωρίς πιο πριν να έχουμε συναντήσει παρένθεση που να ανοίγει.

```
4    def main_factorial):
```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_factorial', 4, 100]
['FAMILY: groupSymbol', ')', 4, 119]
***ERROR(syn)***:{def_main_function()}: Missing left parenthesis "(" in line:  4
```

```
14        while i<=x):
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: identifier', 'i', 14, 100]
***ERROR(syn)***:{while_stat()}: Missing left parenthesis "(" in line:  14
```

- ERROR καθώς έχουμε συναντήσει σωστή σύνταξη σε ολόκληρη τη γραμμή, ωστόσο στο τέλος λείπει η : η οποία είναι απαραίτητη για τον σωστό ορισμό της κύριας συνάρτησης main_factorial.

```
4    def main_factorial()
```

```
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: groupSymbol', '#{', 5, 120]
***ERROR(syn)***:{def_main_function()}: Missing colon ":" in line:  4
```

- ERROR καθώς λείπει το απαραίτητο σύμβολο #{ από την γραμμή 5, για να υποδηλώσει την αρχή του block της κυρίας συνάρτησης. Το error βρίσκει σωστά ότι λείπει το συγκεκριμένο σύμβολο, ωστόσο το ζητάει στη γραμμή 7 καθώς εκεί βρίσκει την επόμενη λεκτική μονάδα (#declare), εφόσον τα σχόλια στη γραμμή 6 απλά τα προσπερνάει.

```
4    def main_factorial():
5
6        #$ declarations #$
7        #declare x
8        #declare i,fact
```

```
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: keyword', '#declare', 7, 200]
***ERROR(syn)***:{def_main_function()}: Missing start of block statement "#{" in line:  7
```

- ERROR καθώς λείπει το ; που είναι απαραίτητο στο τέλος της ανάθεσης.

```
11      x = int(input())
12      fact = 1;
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: keyword', 'input', 11, 207]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: groupSymbol', ')', 11, 119]
['FAMILY: groupSymbol', ')', 11, 119]
['FAMILY: identifier', 'fact', 12, 100]
***ERROR(syn)***:{assignment_stat()}: Missing ";" in line:  11
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει άλλο ένα σύμβολο) για να κλείσει και η δεύτερη παρένθεση που ανοίγει, αλλά δεν το διάβασε.

```
11      x = int(input();
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: keyword', 'input', 11, 207]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: groupSymbol', ')', 11, 119]
['FAMILY: delimiter', ';', 11, 113]
***ERROR(syn)***:{assignment_stat()}: Missing right parenthesis ")" in line:  11
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε μετά το int και την αριστερή παρένθεση (να διαβάσει input για να συνεχίσει, αλλά δεν έγινε αυτό και για αυτό βγάζει error.

```
11      x = int(inpt());
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: identifier', 'inpt', 11, 100]
***ERROR(syn)***:{assignment_stat()}: Missing "input" in line:  11
```

- ERROR καθώς λείπει το σύμβολο της ανάθεσης μεταξύ του αναγνωριστικού fact και του αριθμού 1.

```
12      fact  1;
```

```
['FAMILY: identifier', 'fact', 12, 100]
['FAMILY: number', '1', 12, 101]
***ERROR(syn)***:{assignment_stat()}: Missing assignment "=" in line:  12
```

- ERROR καθώς λείπει η : που είναι απαραίτητη στο τέλος της σύνταξης του while statement.

```
14      while (i<=x)
15      #{
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: identifier', 'i', 14, 100]
['FAMILY: relOperator', '<=', 14, 106]
['FAMILY: identifier', 'x', 14, 100]
['FAMILY: groupSymbol', ')', 14, 119]
['FAMILY: groupSymbol', '#{', 15, 120]
***ERROR(syn)***:{while_stat()}: Missing ":" in line:  14
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει το σύμβολο) για να κλείσει και η παρένθεση που ανοίγει στο while statement, αλλά δεν το διάβασε.

```
14      while (i<=x:
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: identifier', 'i', 14, 100]
['FAMILY: relOperator', '<=', 14, 106]
['FAMILY: identifier', 'x', 14, 100]
['FAMILY: delimiter', ':', 14, 115]
***ERROR(syn)***:{while_stat()}: Missing right parenthesis ")" in line:  14
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει ένα condition ανάμεσα στις παρενθέσεις για το οποίο να εκτελεί τον κώδικα που υπάρχει μέσα στο while statement, αλλά δεν το διάβασε.

```
14      while ():
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: groupSymbol', ')', 14, 119]
***ERROR(syn)***:{factor()}: Missing INTEGER or expression or identifier in line:  14
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει έναν αριθμό ή μια έκφραση ώστε να μπορεί να κάνει τη σύγκριση <= με το x, αλλά δεν το διάβασε.

```
14      while (<=x):
```

```
['FAMILY: keyword', 'while', 14, 203]
['FAMILY: groupSymbol', '(', 14, 118]
['FAMILY: relOperator', '<=', 14, 106]
***ERROR(syn)***:{factor()}: Missing INTEGER or expression or identifier in line:  14
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει το σύμβολο) για να κλείσει και η παρένθεση που ανοίγει στο print statement, αλλά δεν το διάβασε.

```
19      print(fact;
```

```
['FAMILY: keyword', 'print', 19, 205]
['FAMILY: groupSymbol', '(', 19, 118]
['FAMILY: identifier', 'fact', 19, 100]
['FAMILY: delimiter', ';', 19, 113]
***ERROR(syn)***:{print_stat()}: Missing right parenthesis ")" in line:  19
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει τη δεσμευμένη λέξη __name__ μετά το if της main συνάρτησής μας, αλλά δεν την διάβασε.

```
22    if   == "__main__":
```

```
['FAMILY: keyword', 'if', 22, 201]
['FAMILY: relOperator', '==', 22, 111]
***ERROR(syn)***:{call_main_part()}: Missing "__name__" in line:  22
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει το σύμβολο == στη σύνταξη της main συνάρτησής μας, αλλά δεν το διάβασε.

```
22    if __name__ = "__main__":
```

```
['FAMILY: keyword', 'if', 22, 201]
['FAMILY: keyword', '__name__', 22, 212]
['FAMILY: assignment', '=', 22, 112]
***ERROR(syn)***:{call_main_part()}: Missing "==" in line:  22
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει τη δεσμευμένη λέξη if στην αρχή της main συνάρτησής μας, αλλά δεν το διάβασε.

```
22      __name__ == "__main__":
23          #$ call of main functions #$
24          main_factorial();
25
```

```
['FAMILY: delimiter', ';', 19, 113]
['FAMILY: groupSymbol', '#}', 20, 121]
['FAMILY: keyword', '__name__', 22, 212]
***ERROR(syn)***:{call_main_part()}: Missing "if" in line:  22
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει τη δεσμευμένη λέξη __name__ μετά το if της main συνάρτησής μας, αλλά δεν την διάβασε.

```
24          main_factorial;
```

```
['FAMILY: identifier', 'main_factorial', 24, 100]
['FAMILY: delimiter', ';', 24, 113]
***ERROR(syn)***:{main_function_call()}: Missing left parenthesis "(" in line:  24
```

❖ Αρχείο main_primes

- ERROR καθώς λείπει το απαραίτητο σύμβολο #{ , για να υποδηλώσει την αρχή του block της συνάρτησης divides.

```
 9          #declare i
10          def divides(x,y):
11
12              if (y == (y//x) * x):
13                  return (1);
```

```
['FAMILY: keyword', 'def', 10, 208]
['FAMILY: identifier', 'divides', 10, 100]
['FAMILY: groupSymbol', '(', 10, 118]
['FAMILY: identifier', 'x', 10, 100]
['FAMILY: delimiter', ',', 10, 114]
['FAMILY: identifier', 'y', 10, 100]
['FAMILY: groupSymbol', ')', 10, 119]
['FAMILY: delimiter', ':', 10, 115]
['FAMILY: keyword', 'if', 12, 201]
***ERROR(syn)***:{def_function()}: Missing start of block statement "#{" in line:  12
```

- ERROR καθώς λείπει κάποιο statement μέσα στο block του if statement όταν η συνθήκη αληθεύει και περνάει απευθείας στο else.

```
12              if (y == (y//x) * x):
13
14              else:
```

```
['FAMILY: keyword', 'if', 12, 201]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'y', 12, 100]
['FAMILY: relOperator', '==', 12, 111]
['FAMILY: groupSymbol', '(', 12, 118]
['FAMILY: identifier', 'y', 12, 100]
['FAMILY: mulOperator', '//', 12, 105]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: mulOperator', '*', 12, 104]
['FAMILY: identifier', 'x', 12, 100]
['FAMILY: groupSymbol', ')', 12, 119]
['FAMILY: delimiter', ':', 12, 115]
['FAMILY: keyword', 'else', 14, 202]
***ERROR(syn)***:{statement()}: Incorrect statement in line:  14
```

- ERROR καθώς λείπει κάποιο expression ή η έκφραση int(input()) μέσα στο return statement.

```
12          if (y == (y//x) * x):
13              return ();
14          else:
```

```
['FAMILY: keyword', 'return', 13, 204]
['FAMILY: groupSymbol', '(', 13, 118]
['FAMILY: groupSymbol', ')', 13, 119]
***ERROR(syn)***:{factor()}: Missing INTEGER or expression or identifier in line:  13
```

- ERROR καθώς ο συντακτικός αναλυτής περίμενε να διαβάσει το σύμβολο (αμέσως μετά το print statement, αλλά δεν το διάβασε.

```
12          if (y == (y//x) * x):
13              return ;
14          else:
```

```
['FAMILY: keyword', 'return', 13, 204]
['FAMILY: delimiter', ';', 13, 113]
***ERROR(syn)***:{return_stat()}: Missing left parenthesis "(" in line:  13
```

- ERROR καθώς λείπει η : που είναι απαραίτητη στο τέλος του else για την σύνταξη του if statement.

```
14          else
```

```
['FAMILY: keyword', 'else', 14, 202]
['FAMILY: groupSymbol', '#{', 15, 120]
***ERROR(syn)***:{if_stat()}: Missing ":" in line:  14
```

❖ Αρχείο main_fibonacci

- ERROR καθώς λείπει η (που είναι απαραίτητη στην έκφραση *int(input());*

```
11      x = int input());
12      print(fibonacci(x));
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: keyword', 'input', 11, 207]
***ERROR(syn)***:{assignment_stat()}: Missing left parenthesis "(" in line:  11
```

- ERROR καθώς λείπει το identifier input που είναι απαραίτητο στην έκφραση *int(input());*

```
11      x = int();
12      print(fibonacci(x));
```

```
['FAMILY: identifier', 'x', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: keyword', 'int', 11, 206]
['FAMILY: groupSymbol', '(', 11, 118]
['FAMILY: groupSymbol', ')', 11, 119]
***ERROR(syn)***:{assignment_stat()}: Missing "input" in line:  11
```

ΑΠΟΤΕΛΕΣΜΑΤΑ ΕΝΔΙΑΜΕΣΟΥ ΚΩΔΙΚΑ

Για να ελέγξουμε ότι ο ενδιάμεσος κώδικας λειτουργεί σωστά και σχηματίζει τις σωστές τετράδες, συμβουλευτήκαμε το πρόγραμμα ifWhile που υπάρχει στις σημειώσεις για τον ενδιάμεσο κώδικα, κάνοντας τις απαραίτητες αλλαγές (π.χ. **:=** → **=** , **program** → **def** , **ifWhile** → **main_ifWhile**, **line:1** → **line:4** [έχουμε προσθέσει στην αρχή σε σχόλια τα ονοματεπώνυμα μας]) όπως φαίνεται παρακάτω και στο screenshot το πως υλοποιήσαμε το ίδιο πρόγραμμα σε CutePy για να ανταποκρίνεται στις απαιτήσεις της φετινής άσκησης.

Θα επεξεργαστούμε αναλυτικά το παράδειγμα αυτό καθώς υπήρχε και το ανάλογο υλικό για να το μελετήσουμε εκτενώς και να υλοποιήσουμε τον ενδιάμεσο κώδικα στην άσκησή μας.

❖ Αρχείο main_ifWhile

2.4 Ένα πιο σύνθετο παράδειγμα, με φωλιασμένες δομές

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_ifWhile():
#{
    #declare c,a,b,t
    a=1;
    while (a+b<1 and b<5):
    #{
        if (t==1):
            c=2;
        else:
            if (t==2):
                c=4;
            else:
                c=0;
        while (a<1):
            if (a==2):
                while(b==1):
                    c=2;
    #}
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_ifWhile();
```

```
program ifWhile
{
    declare c,a,b,t;

    a:=1;
    while (a+b<1 and b<5)
    {
        if (t=1)
            c:=2;
        else
            if (t=2)
                c:=4;
            else

                c:=0;
        while (a<1)
            if (a=2)
                while(b=1)
                    c:=2;
    }
}
```

```
['FAMILY: keyword', 'def', 4, 208]
['FAMILY: identifier', 'main_ifWhile', 4, 100]
['FAMILY: groupSymbol', '(', 4, 118]
['FAMILY: groupSymbol', ')', 4, 119]
['FAMILY: delimiter', ':', 4, 115]
['FAMILY: groupSymbol', '#{', 5, 120]
['FAMILY: keyword', '#declare', 6, 200]
['FAMILY: identifier', 'c', 6, 100]
['FAMILY: delimiter', ',', 6, 114]
['FAMILY: identifier', 'a', 6, 100]
['FAMILY: delimiter', ',', 6, 114]
['FAMILY: identifier', 'b', 6, 100]
['FAMILY: delimiter', ',', 6, 114]
['FAMILY: identifier', 't', 6, 100]
['FAMILY: identifier', 'a', 7, 100]
['FAMILY: assignment', '=', 7, 112]
['FAMILY: number', '1', 7, 101]
['FAMILY: delimiter', ';', 7, 113]
['FAMILY: keyword', 'while', 8, 203]
['FAMILY: groupSymbol', '(', 8, 118]
['FAMILY: identifier', 'a', 8, 100]
['FAMILY: addOperator', '+', 8, 102]
['FAMILY: identifier', 'b', 8, 100]
['FAMILY: relOperator', '<', 8, 107]
['FAMILY: number', '1', 8, 101]
['FAMILY: keyword', 'and', 8, 210]
['FAMILY: identifier', 'b', 8, 100]
['FAMILY: relOperator', '<', 8, 107]
['FAMILY: number', '5', 8, 101]
['FAMILY: groupSymbol', ')', 8, 119]
['FAMILY: delimiter', ':', 8, 115]
['FAMILY: groupSymbol', '#{', 9, 120]
['FAMILY: keyword', 'if', 10, 201]
['FAMILY: groupSymbol', '(', 10, 118]
['FAMILY: identifier', 't', 10, 100]
['FAMILY: relOperator', '==', 10, 111]
['FAMILY: number', '1', 10, 101]
['FAMILY: groupSymbol', ')', 10, 119]
['FAMILY: delimiter', ':', 10, 115]
['FAMILY: identifier', 'c', 11, 100]
['FAMILY: assignment', '=', 11, 112]
['FAMILY: number', '2', 11, 101]
['FAMILY: delimiter', ';', 11, 113]
['FAMILY: keyword', 'else', 12, 202]
['FAMILY: delimiter', ':', 12, 115]
['FAMILY: keyword', 'if', 13, 201]
['FAMILY: groupSymbol', '(', 13, 118]
['FAMILY: identifier', 't', 13, 100]
['FAMILY: relOperator', '==', 13, 111]
['FAMILY: number', '2', 13, 101]
['FAMILY: groupSymbol', ')', 13, 119]
['FAMILY: delimiter', ':', 13, 115]
['FAMILY: identifier', 'c', 14, 100]
['FAMILY: assignment', '=', 14, 112]
['FAMILY: number', '4', 14, 101]
['FAMILY: delimiter', ';', 14, 113]
['FAMILY: keyword', 'else', 15, 202]
['FAMILY: delimiter', ':', 15, 115]
['FAMILY: identifier', 'c', 16, 100]
['FAMILY: assignment', '=', 16, 112]
['FAMILY: number', '0', 16, 101]
['FAMILY: delimiter', ';', 16, 113]
['FAMILY: keyword', 'while', 17, 203]
['FAMILY: groupSymbol', '(', 17, 118]
['FAMILY: identifier', 'a', 17, 100]
['FAMILY: relOperator', '<', 17, 107]
['FAMILY: number', '1', 17, 101]
['FAMILY: groupSymbol', ')', 17, 119]
['FAMILY: delimiter', ':', 17, 115]
['FAMILY: keyword', 'if', 18, 201]
['FAMILY: groupSymbol', '(', 18, 118]
['FAMILY: identifier', 'a', 18, 100]
['FAMILY: relOperator', '==', 18, 111]
['FAMILY: number', '2', 18, 101]
['FAMILY: groupSymbol', ')', 18, 119]
['FAMILY: delimiter', ':', 18, 115]
['FAMILY: keyword', 'while', 19, 203]
['FAMILY: groupSymbol', '(', 19, 118]
['FAMILY: identifier', 'b', 19, 100]
['FAMILY: relOperator', '==', 19, 111]
['FAMILY: number', '1', 19, 101]
['FAMILY: groupSymbol', ')', 19, 119]
['FAMILY: delimiter', ':', 19, 115]
['FAMILY: identifier', 'c', 20, 100]
['FAMILY: assignment', '=', 20, 112]
['FAMILY: number', '2', 20, 101]
['FAMILY: delimiter', ';', 20, 113]
['FAMILY: groupSymbol', '#}', 21, 121]
['FAMILY: groupSymbol', '#}', 22, 121]
['FAMILY: keyword', 'if', 24, 201]
['FAMILY: keyword', '__name__', 24, 212]
['FAMILY: relOperator', '==', 24, 111]
['FAMILY: keyword', '"__main__"', 24, 213]
['FAMILY: delimiter', ':', 24, 115]
['FAMILY: identifier', 'main_ifWhile', 26, 100]
['FAMILY: groupSymbol', '(', 26, 118]
['FAMILY: groupSymbol', ')', 26, 119]
['FAMILY: delimiter', ';', 26, 113]
['FAMILY: EOF', '', 27, 122]
*****FINISHED WITHOUT ERRORS*****
```

Παραθέτουμε και το αποτέλεσμα της λεκτικής και συντακτικής ανάλυσης, καθώς το συγκεκριμένο πρόγραμμα δεν το είχαμε χρησιμοποιήσει προηγουμένως.

Όπως φαίνεται οι αλλαγές που κάναμε ήταν σωστές σύμφωνα με την γλώσσα CutePy, καθώς δεν υπάρχουν ούτε λεκτικά ούτε συντακτικά λάθη.

```
1   1   begin_block   main_ifWhile   _   _
2   2   =   1   _   a
3   3   +   a   b   %1
4   4   <   %1   1   6
5   5   jump   _   _   28
6   6   <   b   5   8
7   7   jump   _   _   28
8   8   ==   t   1   10
9   9   jump   _   _   12
10  10   =   2   _   c
11  11   jump   _   _   17
12  12   ==   t   2   14
13  13   jump   _   _   16
14  14   =   4   _   c
15  15   jump   _   _   17
16  16   =   0   _   c
17  17   <   a   1   19
18  18   jump   _   _   27
19  19   ==   a   2   21
20  20   jump   _   _   26
21  21   ==   b   1   23
22  22   jump   _   _   25
23  23   =   2   _   c
24  24   jump   _   _   21
25  25   jump   _   _   26
26  26   jump   _   _   17
27  27   jump   _   _   3
28  28   end_block   main_ifWhile   _   _
29  29   begin_block   "__main__"   _   _
30  30   call   main_ifWhile   _   _
31  31   halt   _   _   _
32  32   end_block   "__main__"   _   _
33
```

```
1 :   begin_block, main_ifWhile, _, _
2 :   :=, 1, _, a
3 :   +, a, b, T_1
4 :   <, T_1, 1, 6
5 :   jump, _, _, 28
6 :   <, b, 5, 8
7 :   jump, _, _, 28
8 :   =, t, 1, 10
9 :   jump, _, _, 12
10 :   :=, 2, _, c
11 :   jump, _, _, 17
12 :   =, t, 2, 14
13 :   jump, _, _, 16
14 :   :=, 4, _, c
15 :   jump, _, _, 17
16 :   :=, 0, _, c
17 :   <, a, 1, 19
18 :   jump, _, _, 27
19 :   =, a, 2, 21
20 :   jump, _, _, 26
21 :   =, b, 1, 23
22 :   jump, _, _, 25
23 :   :=, 2, _, c
24 :   jump, _, _, 21
25 :   jump, _, _, 26
26 :   jump, _, _, 17
27 :   jump, _, _, 3
28 :   halt, _, _, _
29 :   end_block, main_ifWhile, _, _
```

Όπως φαίνεται από τις δύο εικόνες παραπάνω, οι τετράδες που σχηματίζονται για τον ενδιάμεσο κώδικα είναι ίδιες. Η μόνη διαφορά πλέον είναι στο ότι η γλώσσα CutePy δεν ξεκινάει αμέσως με την έναρξη του προγράμματος να μεταφράζει την *main* συνάρτησή του. Τώρα η *main* βρίσκεται στο τέλος και συνεπώς σχηματίζει διαφορετικό block από την κύρια συνάρτηση, δηλαδή την main_ifWhile. Οπότε, οι τρεις επιπλέον τετράδες που σχηματίζονται, αφορούν αυτόν τον σκοπό.

Η ορθότητα όσων εμφανίζει ο ενδιάμεσος κώδικας, περιγράφεται αναλυτικά παρακάτω με έναν συνδυασμό της εξήγησης που μας έχει δοθεί στις σημειώσεις και όσων προσθέσαμε ώστε να απευθύνεται στην γλώσσα CutePy.

Η συντακτική ανάλυση θα ξεκινήσει από το def και θα περάσει στο #declare, χωρίς να δημιουργηθεί ενδιάμεσος κώδικας. Η πρώτη τετράδα θα παραχθεί ότι συναντήσουμε το a=1; και θα είναι μία begin_block. Στη συνέχεια θα παραχθεί η τετράδα για το a=1;

1 : begin_block, main_ifWhile , _, _
2 : =, 1, _, a

Μετά ξεκινάει η μετάφραση του while. Πρώτα θα αποτιμηθεί η έκφραση a+b και θα τοποθετηθεί στην προσωρινή μεταβλητή %1. Αυτή η εντολή είναι και η πρώτη του while, οπότε πρέπει κάπου να φυλάξουμε τον αριθμό της τετράδας για να μπορέσουμε να κάνουμε το άλμα προς τα πίσω, όταν φτάσουμε στο τέλος του while. Όταν υπολογιστεί το %1, μετά θα γίνει η σύγκριση:

3 : +, a, b, %1
4 : <, %1, 1, _
5 : jump, _, _, _

Στο 6 θα μεταφραστεί το b<5 και θα γίνει backpatch() το true της a+b<1, δηλαδή η 4. Ο κώδικας ως τώρα είναι:

1 : begin_block, main_ifWhile , _, _
2 : =, 1, _, a
3 : +, a, b, %1
4 : <, %1, 1, 6
5 : jump, _, _, _
6 : <, b, 5, 8
7 : jump, _, _, _

Έχουν μείνει ασυμπλήρωτες οι τετράδες του false, δηλαδή οι 5 και 7.
Η τετράδα 8 είναι η πρώτη μέσα στο βρόχο της while και η πρώτη της if και πιο συγκεκριμένα της συνθήκης t=1. Άρα έχουμε τις μη συμπληρωμένες τετράδες:

8 : =, t, 1, _
9 : jump, _, _, _

Η 8 αντιστοιχεί στο true και θα κάνει εδώ backpatch() που θα τοποθετηθεί ο κώδικας για το c=2.

8 : =, t, 1, 10

Αμέσως μετά θα τοποθετηθεί το jump που θα μας βγάλει έξω από το if και θα είναι φυσικά ακόμα ασυμπλήρωτο. Αργότερα θα δείξει στο while(a<1), όταν γνωρίσουμε τον αριθμό της τετράδας που θα δημιουργηθεί για το a<1. Ο νέος κώδικας που παράγεται στο σημείο αυτό είναι οι τετράδες 10 και 11:

10 : =, 2, _, c
11 : jump, _, _, _

Συνεχίζουμε με το else το οποίο έχει μέσα του άλλο ένα if. Μέσα στο else κάνει backpatch() το false του t=1,

9 : jump, _, _, 12

ενώ δημιουργούνται οι τετράδες για το t=1.

12 : =, t, 2, _
13 : jump, _, _, _

και αμέσως συμπληρώνεται το 12, αφού ακολουθεί το κυρίως σώμα του if.

12 : =, t, 2, 14

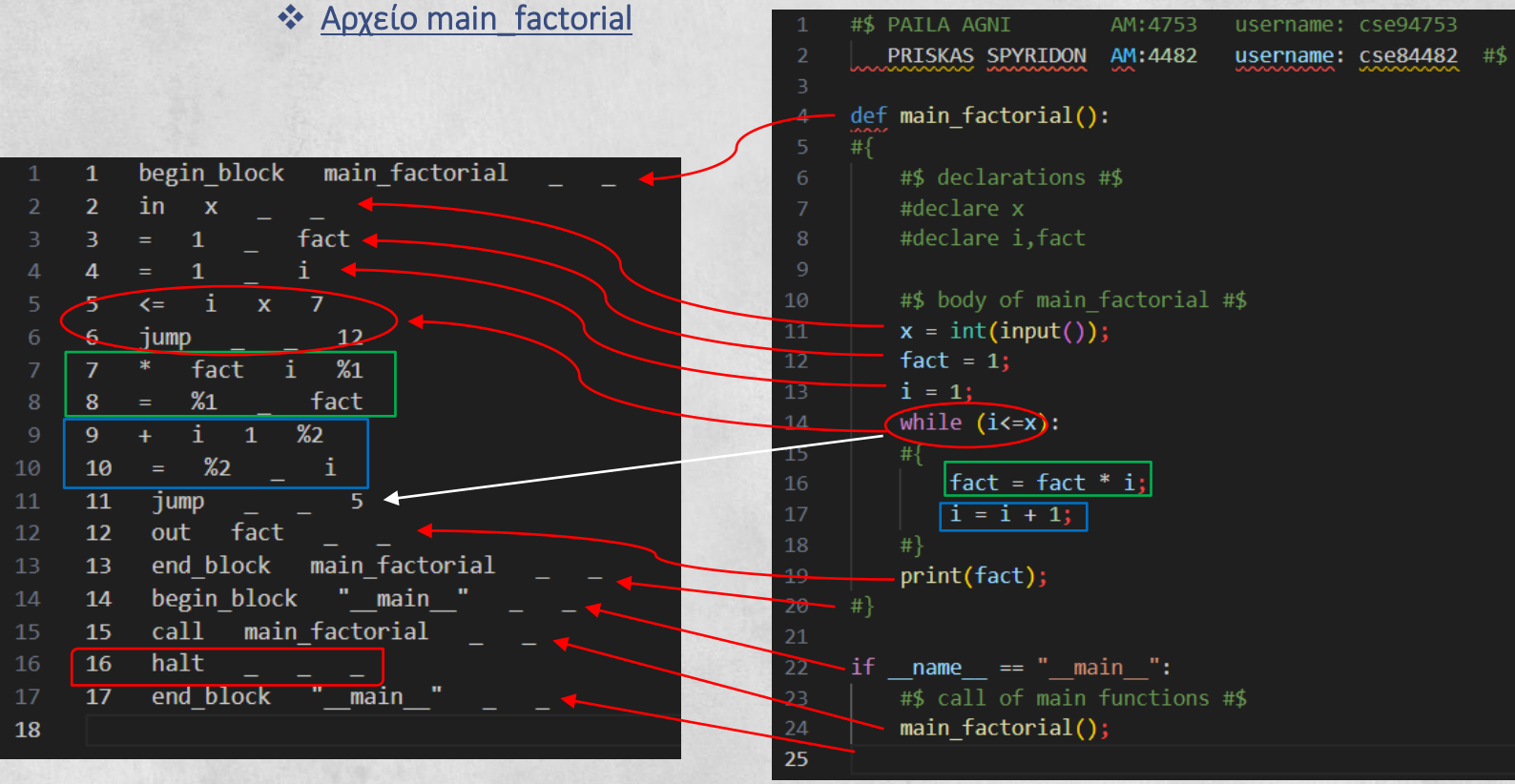
Το κυρίως σώμα του if και το ασυμπλήρωτο jump που θα παρακάμψει το else μας δίνουν τις εξής τετράδες:

14 : =, 4, _, c
15 : jump, _, _, _
Αμέσως μετά ακολουθεί ο κώδικας του else
16 : =, 0, _, c
Μην ξεχάσουμε ότι στον κώδικα του else, δηλαδή στην 16, πρέπει να κάνει backpatch() το false του if που βρίσκεται στην 13:
13 : jump, _, _, 16
Μετά ο έλεγχος κυλάει στο while(a<1). Εκεί είχαμε αφήσει κάποιες τετράδες που έπρεπε να γίνουν backpatch().
Πρόκειται για τις 11 και 15 που αφορούν τετράδες παρακάμπτουν else:
11 : jump, _, _, 17
15 : jump, _, _, 17
Σημειώνουμε την 17, διότι αφού έχουμε while θα γίνει αργότερα κάποιο άλμα προς τα πίσω στο σημείο αυτό, και θα παραχθούν οι τετράδες για τη συνθήκη:
17 : <, a, 1, _
18 : jump, _, _, _
Στη συνέχεια γίνεται ο έλεγχος αν a=2, που οφείλεται στο if, ενώ σε αυτό κάνει backpatch() το true στο 17, από το while. Έτσι η 17 γίνεται:
17 : <, a, 1, 19
και οι δύο νέες τετράδες που δημιουργούνται:
19 : =, a, 2, _
20 : jump, _, _, _
Το 19 ως true του if θα κάνει backpatch() στο nextQuad():
19 : =, a, 2, 21
όπου θα τοποθετηθούν οι μη συμπληρωμένες τετράδες που αντιστοιχούν στη λογική συνθήκη b=1 του while.
21 : =, b, 1, _
22 : jump, _, _, _
Το 21 θα σημειωθεί ως πρώτη εντολή του while για να μπορέσουμε να κάνουμε το άλμα της επανεξέτασης της λογικής συνθήκης.
Το 21 θα συμπληρωθεί με την ετικέτα 23, ως true της if:
21 : =, b, 1, 23
ενώ ο κώδικας για το c=2 θα τοποθετηθεί στην 23:
23 : :=, 2, _, c
Στη συνέχεια στον κώδικά μας θα εμφανιστούν μια σειρά από jump. Επίσης, θα πρέπει να εκτελεστεί και μία σειρά από backpatch(), τόσο για τα jump που έχουν ήδη παραχθεί και είναι ασυμπλήρωτα, όσο και για τα ασυμπλήρωτα jump θα παραχθούν παρακάτω.
Το πρώτο από αυτά τα jump είναι αυτό που θα επιτρέψει, μετά το σώμα του while (b=1), να επιστρέψουμε στη συνθήκη για νέο έλεγχο. Η συνθήκη ξεκινάει στο 21, άρα έχουμε:
24 : jump, _, _, 21
Επειδή ο κώδικας έχει μεγαλώσει αρκετά και έχει γίνει και αρκετά πιο πολύπλοκος, λόγω των πολλών φωλιασμένων βρόχων, ας θυμηθούμε τι έχουμε αφήσει ασυμπλήρωτο στον κώδικα που έχουμε ήδη παράγει.
Η τετράδα 5 έχει ένα μη συμπληρωμένο jump. Το jump αυτό οφείλεται στο false του while (a+b<1 and b<5). Όμοια και στην τετράδα 7, έχουμε ένα μη συμπληρωμένο jump λόγω του while (a+b<1 and b<5). Η τετράδα 18 έχει για τον ίδιο λόγο ένα ασυμπλήρωτο jump από την την while (a<1). Το ασυμπλήρωτο jump της 26 οφείλεται στο false του if (a=2), ενώ της 22, στο false της while(b=1).

Επιστρέφουμε στην παραγωγή νέων τετράδων. Στο σημείο της μετάφρασης που βρισκόμαστε, ολοκληρώθηκε η μετάφραση για το σώμα της if (a=2). Αναζητώντας στον υπάρχοντα ενδιάμεσο κώδικα, μπορούμε να βρούμε ποιος πρέπει να κάνει άλμα μετά το σώμα του if. Δεν είναι δύσκολο να δούμε ότι το άλμα αυτό έχει προκύψει από την ψευδή αποτίμηση της συνθήκης του if, δηλαδή το άλμα που βρίσκεται στην εντολή 22.
Έτσι, κάνουμε το κατάλληλο backpatch() σε αυτό:
22 : jump, _, _, 25
Θα χρειαστεί ένα jump για να μην εκτελεστεί πιθανό else. Εδώ μπορεί να μην υπάρχει κάποιο else, αφού αυτό είναι προαιρετικό, αλλά το jump σύμφωνα με το σχέδιο ενδιάμεσου κώδικα θα παραχθεί. Και αφού δεν υπάρχει else θα παραχθεί jump στην επόμενη εντολή. Άρα:
25 : jump, _, _, 26
Η επόμενη τετράδα που θα δημιουργηθεί είναι η 26. Βρισκόμαστε στο τέλος του βρόχου while (a<1). Χρειάζεται, δηλαδή ένα άλμα προς τα πίσω στη συνθήκη a<1, η οποία βρίσκεται στο 26, άρα:
26 : jump, _, _, 17
Αμέσως μετά τελειώνει ο βρόχος του while (a+b<1 and b<5). Άρα και εδώ, ακριβώς όπως και προηγουμένως, χρειάζεται άλμα προς τα πίσω στη συνθήκη a+b<1 and b<5 η οποία ξεκινάει στην εντολή 3:
27 : jump, _, _, 3
Εφόσον τελειώνει το block της main_ifWhile χρειαζόμαστε μία τετράδα με το end_block:
28 : end_block, main_ifWhile , _, _
Επίσης χρειαζόμαστε μία τετράδα για το begin_block της main συνάρτησης και την κλήση της συνάρτησης main_ifWhile:
29 : begin_block, "__main__", _, _
30 : call, main_ifWhile , _, _
Κάπου εδώ τελειώνουν οι εντολές του προγράμματος. Για τέλος, χρειαζόμαστε μία halt, αφού μιλάμε για το κυρίως πρόγραμμα και μία end_block, για το block της main συνάρτησης:
31 : halt, _, _, _
32 : end_block, "__main__", _, _

Για τα υπόλοιπα 4 παραδείγματα που έχουμε στείλει σημειώνουμε αναλυτικά για το καθένα από αυτά παρακάτω με βελάκια και διάφορα σχήματα, κάθε γραμμή του κώδικα του προγράμματος ποιες τετράδες δημιούργησε.

❖ Αρχείο main_factorial



Παρατηρούμε ότι οι τετράδες που δημιουργούνται αντιστοιχούν ορθά στον κώδικα που έχουμε δώσει. Τα **begin_block** και **end_block** γίνονται στα σωστά σημεία, στα οποία είτε ξεκινάει το block μια συνάρτησης είτε τελειώνει το block αυτής. Τα input δημιουργούν τετράδες οι οποίες ξεκινάνε με το **in** και τα print δημιουργούν τετράδες οι οποίες ξεκινάνε με το **out**. Οι αναθέσεις έχουν δημιουργήσει τις σωστές τετράδες στις σειρές 3,4 ενώ οι λίγο πιο περίπλοκες αναθέσεις, στις οποίες υπάρχει και κάποιος υπολογισμός, έχουν χρησιμοποιήσει σωστά τις προσωρινές μεταβλητές για να συμβεί σωστή ανάθεση (τετράδες 7,8 και 9,10). Στην συνάρτηση main έχει επίσης δημιουργηθεί σωστά η τετράδα που ξεκινάει με **call** για την κλήση της συνάρτησης main_factorial. Πριν γίνει το end_block της main συνάρτησης βλέπουμε και την τετράδα με το **halt**.

Όσες περιπτώσεις αναφέρθηκαν σε αυτό το παράδειγμα και υπάρχουν και στα επόμενα, δεν θα ξανά αναλυθούν, εφόσον βλέπουμε ότι λειτουργούν σωστά.

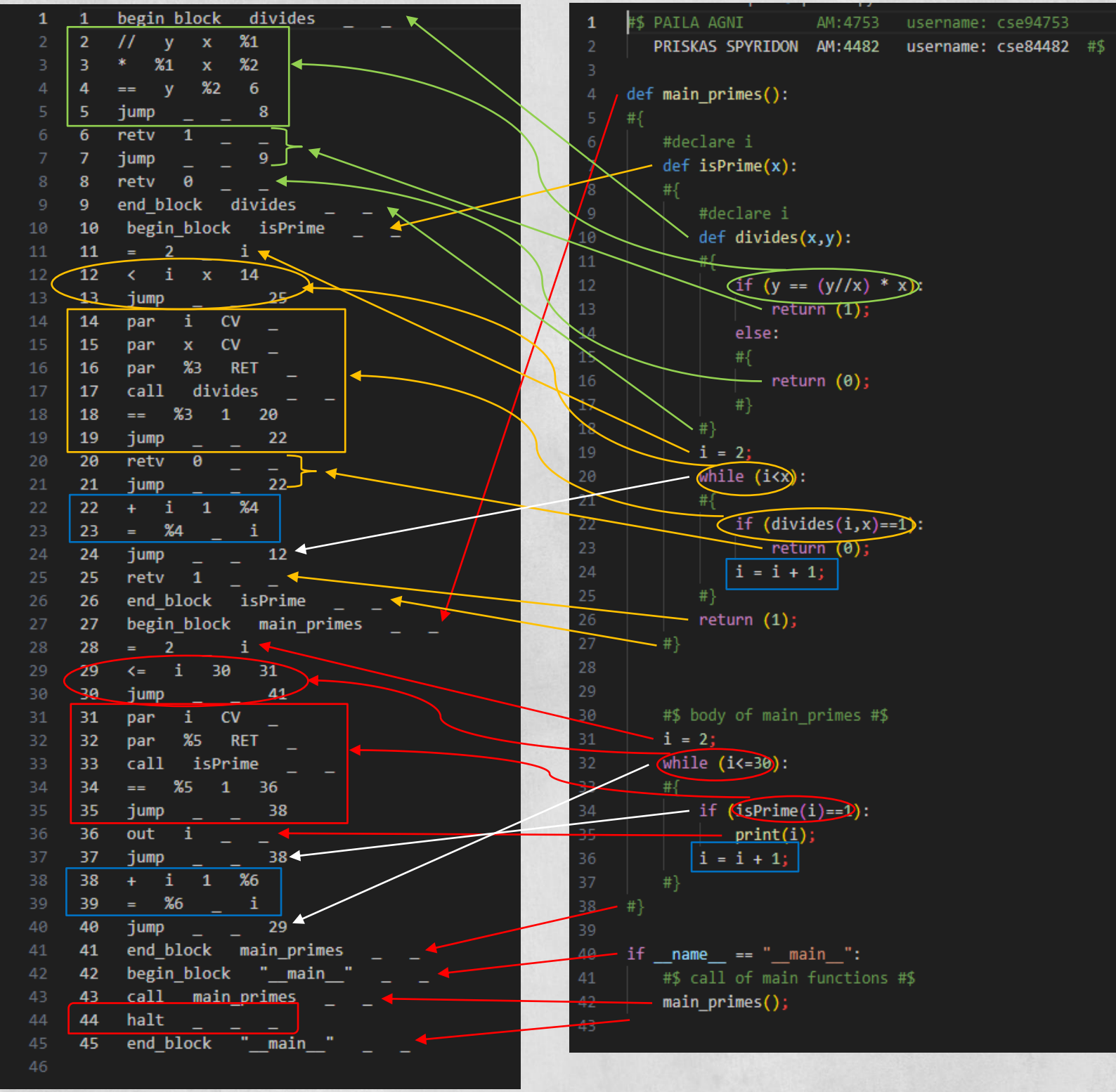
❖ Αρχείο main_fibonacci

```
1   begin_block   fibonacci   _   _
2   <=   x   1   4
3   jump   _   _   6
4   retv   x   _   _
5   jump   _   _   16
6   -   x   1   %1
7   par   %1   CV   _
8   par   %2   RET   _
9   call   fibonacci   _   _
10   -   x   2   %3
11   par   %3   CV   _
12   par   %4   RET   _
13   call   fibonacci   _   _
14   +   %2   %4   %5
15   retv   %5   _   _
16   end_block   fibonacci   _   _
17   begin_block   main_fibonacci   _   _
18   in   x   _   _
19   par   x   CV   _
20   par   %6   RET   _
21   call   fibonacci   _   _
22   out   %6   _   _
23   end_block   main_fibonacci   _   _
24   begin_block   "__main__"   _   _
25   call   main_fibonacci   _   _
26   halt   _   _   _
27   end_block   "__main__"   _   _
28
```

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_fibonacci():
#{
    #declare x
    def fibonacci(x):
    #{
        if (x<=1):
            return(x);
        else:
            return (fibonacci(x-1)+fibonacci(x-2));
    #}
    x = int(input());
    print(fibonacci(x));
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_fibonacci();
```

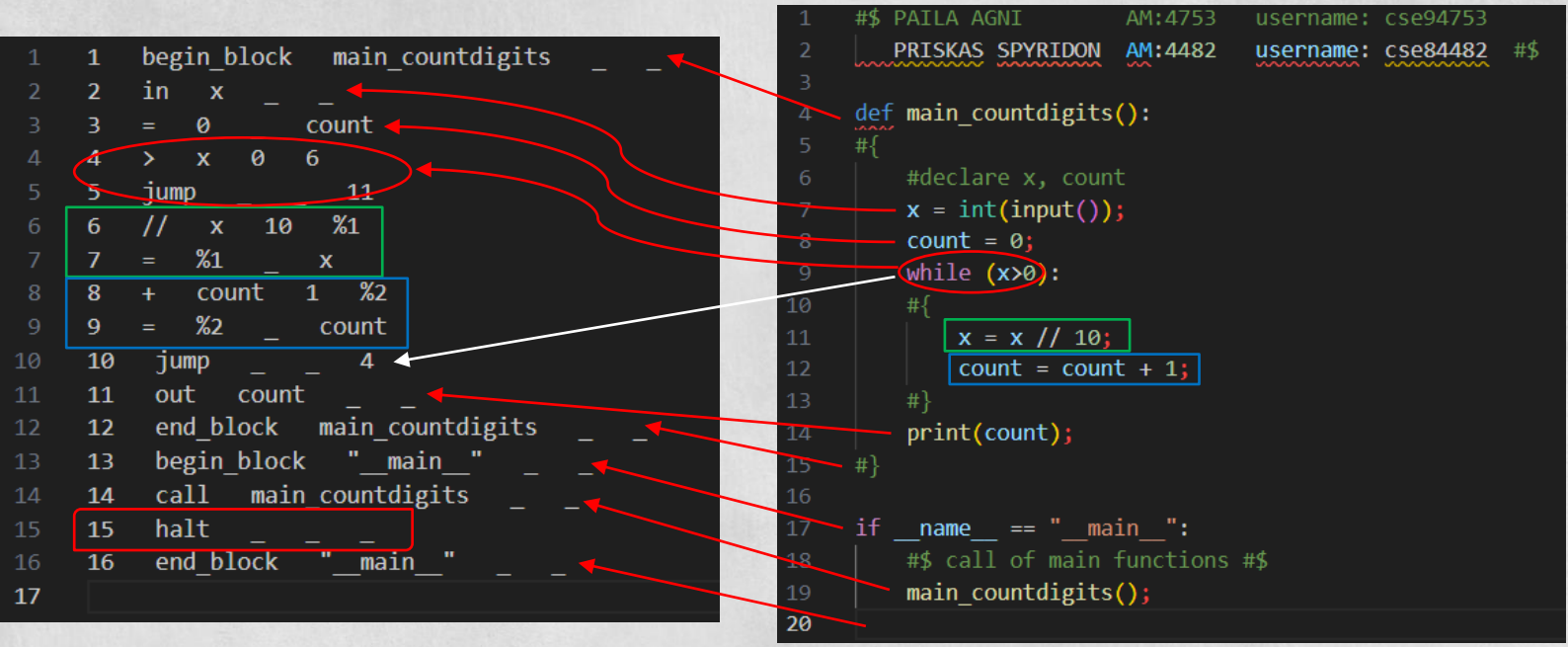
Παρατηρούμε ότι οι τετράδες που δημιουργούνται αντιστοιχούν ορθά στον κώδικα που έχουμε δώσει. Τα **begin_block** και **end_block** γίνονται στα σωστά σημεία, στα οποία είτε ξεκινάει το block μια συνάρτησης είτε τελειώνει το block αυτής. Τα return δημιουργούν τετράδες οι οποίες ξεκινάνε με το **retv** και σε περίπτωση που χρειάζεται να παραλείψουν επόμενες γραμμές κώδικα δημιουργούν και το αντίστοιχο **jump** όπως για παράδειγμα οι τετράδες 4,5 στις οποίες θέλουμε μετά το return να παραλείψουμε το else και να βγούμε από το if statement, πηγαίνοντας κατευθείαν στην τετράδα 16 η οποία μας βγάζει από το block της fibonacci. Το **if statement** έχει δημιουργήσει σωστά τετράδα για όταν ισχύει η έκφραση που περιέχει και αντίστοιχα το jump για όταν δεν ισχύει η συγκεκριμένη έκφραση. Επιπλέον όταν περνάμε μια παράμετρο με τιμή εμφανίζεται η τετράδα **par, …, CV, _** που υποδηλώνει ακριβώς αυτό. Επιπλέον με την τετράδα **par, …, RET, _** αποθηκεύουμε το αποτέλεσμα της κλήσης της συνάρτησης σε μια προσωρινή μεταβλητή, ώστε να μπορέσουμε να το τυπώσουμε.

❖ Αρχείο main_primes



Όσες περιπτώσεις υπάρχουν σε αυτό το παράδειγμα έχουν ξανά αναφερθεί, οπότε δεν θα αναλυθούν περαιτέρω, εφόσον βλέπουμε ότι λειτουργούν σωστά.

❖ Αρχείο main_countdigits



```
1   begin_block   main_countdigits   _   _
2   in   x   _   _
3   =   0   count
4   >   x   0   6
5   jump   _   _   11
6   //   x   10   %1
7   =   %1   _   x
8   +   count   1   %2
9   =   %2   _   count
10   jump   _   _   4
11   out   count   _   _
12   end_block   main_countdigits   _   _
13   begin_block   "__main__"   _   _
14   call   main_countdigits   _   _
15   halt   _   _   _
16   end_block   "__main__"   _   _
17
```

```
#$ PAILA AGNI        AM:4753   username: cse94753
   PRISKAS SPYRIDON  AM:4482   username: cse84482  #$

def main_countdigits():
#{
    #declare x, count
    x = int(input());
    count = 0;
    while (x>0):
    #{
        x = x // 10;
        count = count + 1;
    #}
    print(count);
#}

if __name__ == "__main__":
    #$ call of main functions #$
    main_countdigits();
```

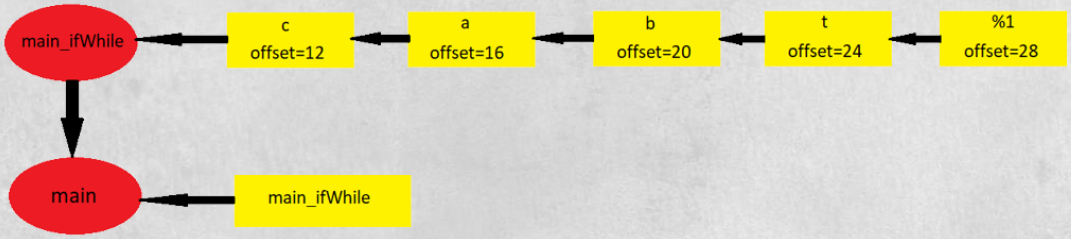
Παρατηρούμε ότι οι τετράδες που δημιουργούνται αντιστοιχούν ορθά στον κώδικα που έχουμε δώσει.

Όσες περιπτώσεις υπάρχουν σε αυτό το παράδειγμα έχουν ξανά αναφερθεί, οπότε δεν θα αναλυθούν περαιτέρω, εφόσον βλέπουμε ότι λειτουργούν σωστά.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΠΙΝΑΚΑ ΣΥΜΒΟΛΩΝ

❖ Αρχείο main_ifwhile

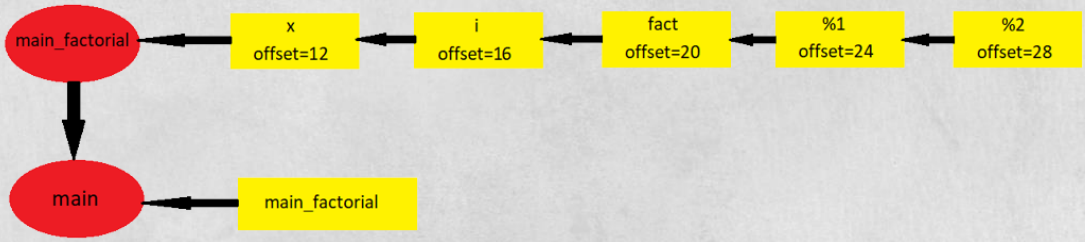
```
1  ********************************************************************************
2
3  ________________________________________________________________________________
4  *SCOPE*:  name: main_ifWhile         |nestingLevel: 1
5     *ENTITY*:  name: c       |type: variable         |offset: 12
6     *ENTITY*:  name: a       |type: variable         |offset: 16
7     *ENTITY*:  name: b       |type: variable         |offset: 20
8     *ENTITY*:  name: t       |type: variable         |offset: 24
9     *ENTITY*:  name: %1      |type: temporary variable   |offset: 28
10 ________________________________________________________________________________
11 *SCOPE*:  name: main          |nestingLevel: 0
12    *ENTITY*:  name: main_ifWhile    |type: function
13 ________________________________________________________________________________
14 ********************************************************************************
15
16
17 ********************************************************************************
18 ________________________________________________________________________________
19 *SCOPE*:  name: main          |nestingLevel: 0
20    *ENTITY*:  name: main_ifWhile    |type: function
21 ________________________________________________________________________________
22 ********************************************************************************
23
```



Αρχικά δημιουργείται το **Scope για την main** [***nestingLevel=0***] , το οποίο έχει ως Entity την function main_ifWhile. Αφού η main_ifWhile είναι συνάρτηση, δημιουργείται και το **Scope για την main_ifWhile** [***nestingLevel=1***] πάνω από αυτό της main. Σε αυτό το scope προστίθενται ως Entity οι τέσσερις μεταβλητές της συνάρτησης c,a,b,t και η μία προσωρινή μεταβλητή %1, οι οποίες έχουν **offset** που ξεκινάει από 12 και αυξάνεται κατά 4 κάθε φορά αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του.

❖ Αρχείο main_factorial

```
1  ********************************************************************************
2
3  *SCOPE*:  name: main_factorial       |nestingLevel: 1
4      *ENTITY*:  name: x       |type: variable         |offset: 12
5      *ENTITY*:  name: i       |type: variable         |offset: 16
6      *ENTITY*:  name: fact        |type: variable         |offset: 20
7      *ENTITY*:  name: %1      |type: temporary variable   |offset: 24
8      *ENTITY*:  name: %2      |type: temporary variable   |offset: 28
9
10 *SCOPE*:  name: main          |nestingLevel: 0
11     *ENTITY*:  name: main_factorial  |type: function
12
13 ********************************************************************************
14
15
16 ********************************************************************************
17
18 *SCOPE*:  name: main          |nestingLevel: 0
19     *ENTITY*:  name: main_factorial  |type: function
20
21 ********************************************************************************
22
```

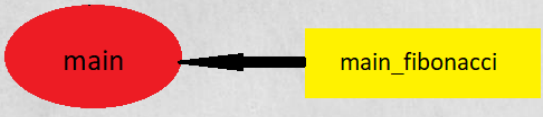


Αρχικά δημιουργείται το **Scope για την main** [***nestingLevel=0***] , το οποίο έχει ως Entity την function main_factorial. Αφού η main_factorial είναι συνάρτηση, δημιουργείται και το **Scope για την main_factorial** [***nestingLevel=1***] πάνω από αυτό της main. Σε αυτό το scope προστίθενται ως Entity οι τρεις μεταβλητές της συνάρτησης x,i,fact και οι δύο προσωρινές μεταβλητές %1, %2, οι οποίες έχουν **offset** που ξεκινάει από 12 και αυξάνεται κατά 4 κάθε φορά αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του.

❖ Αρχείο main_fibonacci

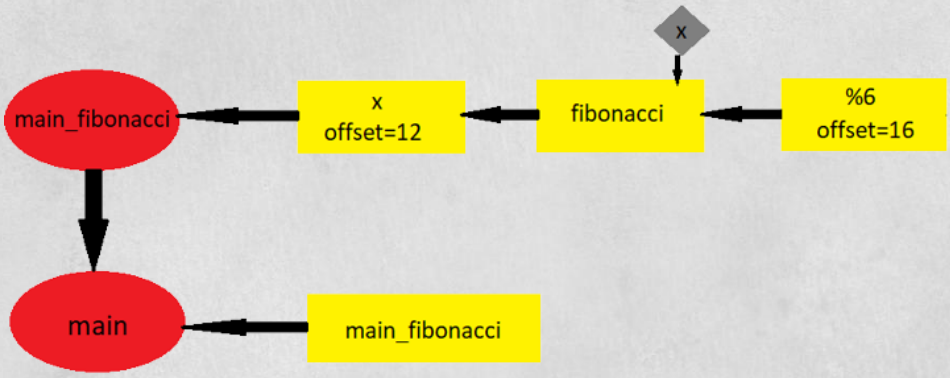
```
********************************************************************************

--------------------------------------------------------------------------------
*SCOPE*:  name: fibonacci        |nestingLevel: 2
    *ENTITY*:  name: x       |type: parameter        |offset: 12
    *ENTITY*:  name: %1      |type: temporary variable   |offset: 16
    *ENTITY*:  name: %2      |type: temporary variable   |offset: 20
    *ENTITY*:  name: %3      |type: temporary variable   |offset: 24
    *ENTITY*:  name: %4      |type: temporary variable   |offset: 28
    *ENTITY*:  name: %5      |type: temporary variable   |offset: 32
--------------------------------------------------------------------------------
*SCOPE*:  name: main_fibonacci       |nestingLevel: 1
    *ENTITY*:  name: x       |type: variable         |offset: 12
    *ENTITY*:  name: fibonacci   |type: function
        *ARGUMENT*:  name: x
--------------------------------------------------------------------------------
*SCOPE*:  name: main         |nestingLevel: 0
    *ENTITY*:  name: main_fibonacci  |type: function
--------------------------------------------------------------------------------
********************************************************************************



********************************************************************************
--------------------------------------------------------------------------------
*SCOPE*:  name: main_fibonacci       |nestingLevel: 1
    *ENTITY*:  name: x       |type: variable         |offset: 12
    *ENTITY*:  name: fibonacci   |type: function
        *ARGUMENT*:  name: x
    *ENTITY*:  name: %6      |type: temporary variable   |offset: 16
--------------------------------------------------------------------------------
*SCOPE*:  name: main         |nestingLevel: 0
    *ENTITY*:  name: main_fibonacci  |type: function
--------------------------------------------------------------------------------
********************************************************************************



********************************************************************************
--------------------------------------------------------------------------------
*SCOPE*:  name: main         |nestingLevel: 0
    *ENTITY*:  name: main_fibonacci  |type: function
--------------------------------------------------------------------------------
********************************************************************************

```

Βήμα 1



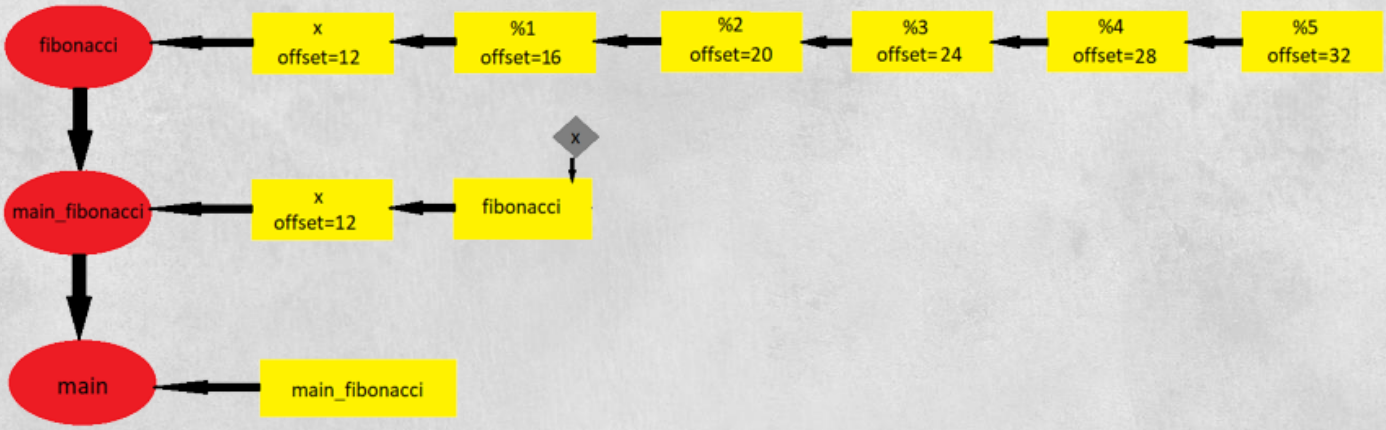
Αρχικά δημιουργείται το **Scope για την main** [***nestingLevel=0***] , το οποίο έχει ως Entity την function main_fibonacci.

Βήμα 2



Αφού η main_fibonacci είναι συνάρτηση, δημιουργείται και το **Scope για την main_fibonacci** [***nestingLevel=1***] πάνω από αυτό της main. Σε αυτό το scope προστίθεται ως Entity η μεταβλητή της συνάρτησης x με offset=12. Έπειτα ως Entity προστίθεται και η συνάρτηση fibonacci η οποία έχει ως παράμετρο Argument ένα x. Τέλος προστίθεται ως Entity και μία προσωρινή μεταβλητή %6 με offset=16. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του, καθώς και το **όνομα** του Argument.

Βήμα 3

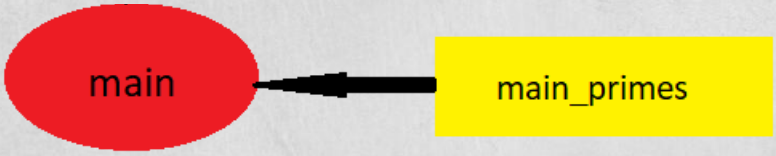


Αφού η fibonacci είναι συνάρτηση, δημιουργείται και το **Scope για την fibonacci** [***nestingLevel=2***] πάνω από αυτό της main_fibonacci. Σε αυτό το scope προστίθεται ως Entity η μια μεταβλητή της συνάρτησης x και οι πέντε προσωρινές μεταβλητές %1, %2, %3, %4, %5, οι οποίες έχουν **offset** που ξεκινάει από 12 και αυξάνεται κατά 4 κάθε φορά αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του, καθώς και το **όνομα** του Argument.

❖ Αρχείο main_primes

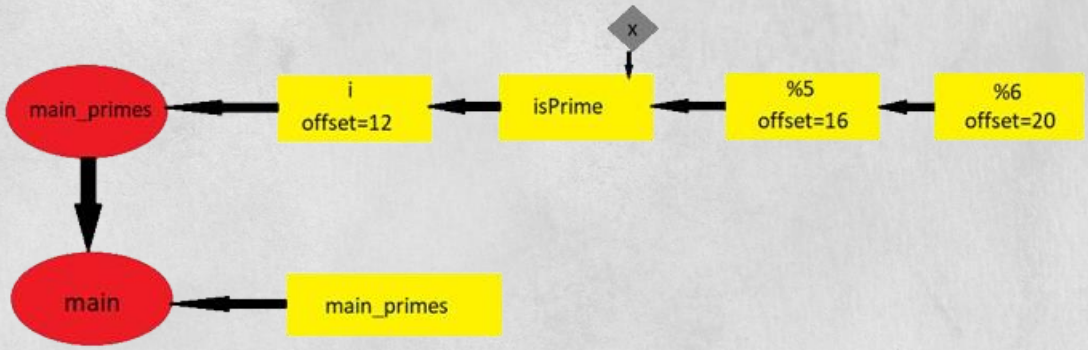
```
*****************************************************************************

-----------------------------------------------------------------------------
*SCOPE*:  name: divides       |nestingLevel: 3
    *ENTITY*:  name: x        |type: parameter          |offset: 12
    *ENTITY*:  name: y        |type: parameter          |offset: 16
    *ENTITY*:  name: %1       |type: temporary variable   |offset: 20
    *ENTITY*:  name: %2       |type: temporary variable   |offset: 24

-----------------------------------------------------------------------------
*SCOPE*:  name: isPrime       |nestingLevel: 2
    *ENTITY*:  name: x        |type: parameter          |offset: 12
    *ENTITY*:  name: i        |type: variable           |offset: 16
    *ENTITY*:  name: divides      |type: function
        *ARGUMENT*:  name: x
        *ARGUMENT*:  name: y

-----------------------------------------------------------------------------
*SCOPE*:  name: main_primes       |nestingLevel: 1
    *ENTITY*:  name: i        |type: variable           |offset: 12
    *ENTITY*:  name: isPrime      |type: function
        *ARGUMENT*:  name: x

-----------------------------------------------------------------------------
*SCOPE*:  name: main          |nestingLevel: 0
    *ENTITY*:  name: main_primes      |type: function

-----------------------------------------------------------------------------
*****************************************************************************



*****************************************************************************

-----------------------------------------------------------------------------
*SCOPE*:  name: isPrime       |nestingLevel: 2
    *ENTITY*:  name: x        |type: parameter          |offset: 12
    *ENTITY*:  name: i        |type: variable           |offset: 16
    *ENTITY*:  name: divides      |type: function
        *ARGUMENT*:  name: x
        *ARGUMENT*:  name: y
    *ENTITY*:  name: %3       |type: temporary variable   |offset: 20
    *ENTITY*:  name: %4       |type: temporary variable   |offset: 24

-----------------------------------------------------------------------------
*SCOPE*:  name: main_primes       |nestingLevel: 1
    *ENTITY*:  name: i        |type: variable           |offset: 12
    *ENTITY*:  name: isPrime      |type: function
        *ARGUMENT*:  name: x

-----------------------------------------------------------------------------
*SCOPE*:  name: main          |nestingLevel: 0
    *ENTITY*:  name: main_primes      |type: function

-----------------------------------------------------------------------------
*****************************************************************************



*****************************************************************************

-----------------------------------------------------------------------------
*SCOPE*:  name: main_primes       |nestingLevel: 1
    *ENTITY*:  name: i        |type: variable           |offset: 12
    *ENTITY*:  name: isPrime      |type: function
        *ARGUMENT*:  name: x
    *ENTITY*:  name: %5       |type: temporary variable   |offset: 16
    *ENTITY*:  name: %6       |type: temporary variable   |offset: 20

-----------------------------------------------------------------------------
*SCOPE*:  name: main          |nestingLevel: 0
    *ENTITY*:  name: main_primes      |type: function

-----------------------------------------------------------------------------
*****************************************************************************



*****************************************************************************

-----------------------------------------------------------------------------
*SCOPE*:  name: main          |nestingLevel: 0
    *ENTITY*:  name: main_primes      |type: function

-----------------------------------------------------------------------------
*****************************************************************************
```

Βήμα 1



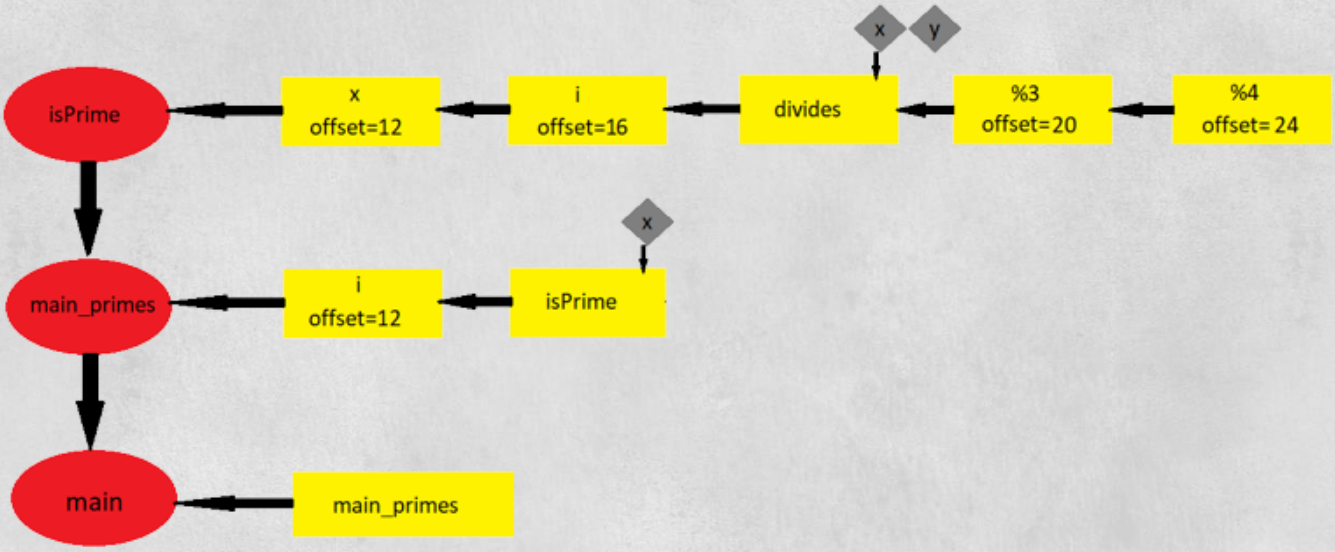
Αρχικά δημιουργείται το **Scope για την main** [***nestingLevel=0***] , το οποίο έχει ως Entity την function main_primes.

Βήμα 2



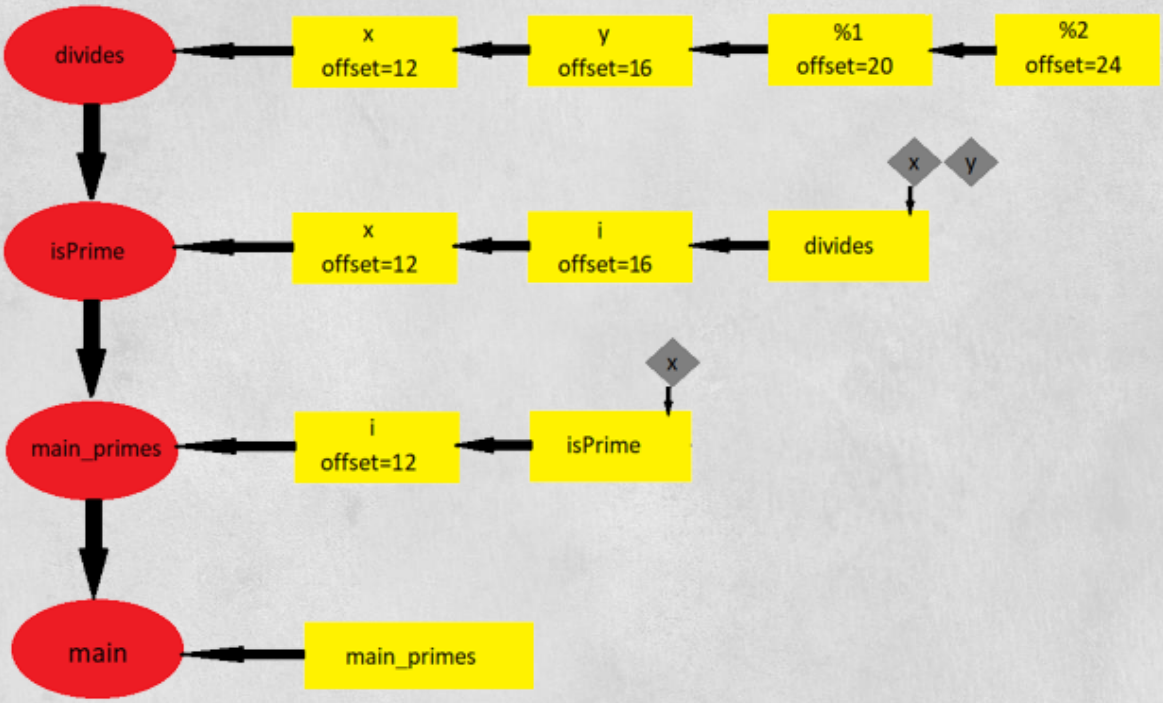
Αφού η main_primes είναι συνάρτηση, δημιουργείται και το **Scope για την main_primes** [***nestingLevel=1***] πάνω από αυτό της main. Σε αυτό το scope προστίθεται ως Entity η μεταβλητή της συνάρτησης i με offset=12. Έπειτα ως Entity προστίθεται και η συνάρτηση isPrime η οποία έχει ως παράμετρο Argument ένα x. Τέλος προστίθενται ως Entity οι προσωρινές μεταβλητές %5, %6 με offset 16 και 20 αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του, καθώς και το **όνομα** του Argument.

Βήμα 3



Αφού η isPrime είναι συνάρτηση, δημιουργείται και το **Scope για την isPrime** [***nestingLevel=2***] πάνω από αυτό της main_primes. Σε αυτό το scope προστίθενται ως Entity οι μεταβλητές της συνάρτησης x,i με offset 12 και 16 αντίστοιχα. Έπειτα ως Entity προστίθεται και η συνάρτηση divides η οποία έχει ως παραμέτρους Arguments τα x,y. Τέλος προστίθενται ως Entity οι προσωρινές μεταβλητές %3, %4 με offset 20 και 24 αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του, καθώς και τα **ονόματα** των Arguments.

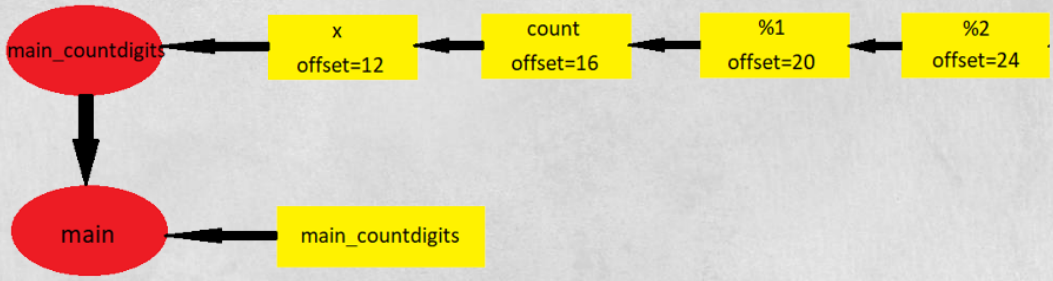
Βήμα 4



Αφού η divides είναι συνάρτηση, δημιουργείται και το **Scope για την divides** [***nestingLevel=3***] πάνω από αυτό της isPrime. Σε αυτό το scope προστίθενται ως Entity οι μεταβλητές της συνάρτησης x,y και οι δύο προσωρινές μεταβλητές %1, %2, οι οποίες έχουν **offset** που ξεκινάει από 12 και αυξάνεται κατά 4 κάθε φορά αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του, καθώς και τα **ονόματα** των Arguments.

❖ Αρχείο main_countdigits

```
********************************************************************************

_____________________________________________________________________________
*SCOPE*:  name: main_countdigits         |nestingLevel: 1
    *ENTITY*:  name: x       |type: variable         |offset: 12
    *ENTITY*:  name: count       |type: variable         |offset: 16
    *ENTITY*:  name: %1      |type: temporary variable   |offset: 20
    *ENTITY*:  name: %2      |type: temporary variable   |offset: 24
_____________________________________________________________________________
*SCOPE*:  name: main         |nestingLevel: 0
    *ENTITY*:  name: main_countdigits    |type: function
_____________________________________________________________________________
********************************************************************************



********************************************************************************

_____________________________________________________________________________
*SCOPE*:  name: main         |nestingLevel: 0
    *ENTITY*:  name: main_countdigits    |type: function
_____________________________________________________________________________
********************************************************************************

```



Αρχικά δημιουργείται το **Scope για την main** [***nestingLevel=0***] , το οποίο έχει ως Entity την function main_countdigits. Αφού η main_countdigits είναι συνάρτηση, δημιουργείται και το **Scope για την main_countdigits** [***nestingLevel=1***] πάνω από αυτό της main. Σε αυτό το scope προστίθενται ως Entity οι δύο μεταβλητές της συνάρτησης x,count και οι προσωρινές μεταβλητές %1, %2, οι οποίες έχουν **offset** που ξεκινάει από 12 και αυξάνεται κατά 4 κάθε φορά αντίστοιχα. Επιπλέον τυπώνουμε τον **τύπο** του κάθε Entity δίπλα από το **όνομά** του.

ΑΠΟΤΕΛΕΣΜΑΤΑ ΤΕΛΙΚΟΥ ΚΩΔΙΚΑ

Ο τελικός κώδικας που υλοποιήσαμε δεν δουλεύει για όλες τις περιπτώσεις σωστά.

Προσπαθήσαμε να φτιάξουμε όσες περισσότερες περιπτώσεις γινόταν με βάση και τις σημειώσεις του τελικού κώδικα που μας είχαν δοθεί. Όλες οι πιο «απλές» περιπτώσεις λειτουργούν σωστά και τις έχουμε σημειώσει στα αποτελέσματά μας με βελάκια και διάφορα σχήματα για να γίνουν πιο ευδιάκριτες.

Πιο συγκεκριμένα καταφέραμε να κάνουμε τον τελικό κώδικα που αφορά

- τα **begin_block** και **end_block** είτε αυτά αφορούν την main συνάρτηση είτε όχι
- το **halt** της προτελευταίας τετράδας
- το **retv**
- την ανάθεση (**=**)
- τα relational operations (**==**, **!=**, **>**, **<**, **>=**, **<=**)
- τις αριθμητικές πράξεις (**+**, **-**, *****, **//**)
- το **jump** *(ενώ έπρεπε να γράφεται ως b Lxx, όπου xx ο αριθμός του Label που θέλουμε να κάνει jump, δεν το κατανοήσαμε εγκαίρως παρά μόνο όταν φτάσαμε στην συγγραφή την αναφοράς και το είχαμε παραδώσει ως b (jump) xx για να το κατανοούμε εμείς καλύτερα)*

♦Επιπλέον καταφέραμε να υλοποιήσουμε **το αρχικό jump στο label της main** συνάρτησης, το οποίο πρέπει να υπάρχει στο L0: ώστε να εκτελείται πρώτο. Αυτό λειτουργεί σωστά, καθώς σε κάθε πρόγραμμα που στείλαμε και παραθέτουμε παρακάτω, κάνει jump στο σωστό label που έχει παραχθεί από την τετράδα *begin_block, “__main__”, _, _* .

❖ Αρχείο main_factorial

Edit | Execute

telikos.asm*

```
1   L0:
2   b(jump) 14
3   L1:
4   sw ra,(sp)
5   L2:
6   li a7,5
7   ecall
8   L3:
9   li t1,1
10  sw t1,-20(sp)
11  L4:
12  li t1,1
13  sw t1,-16(sp)
14  L5:
15  lw t1,-16(sp)
16  lw t2,-12(sp)
17  ble,t1,t2,7
18  L6:
19  b(jump) 12
20  L7:
21  lw t1,-20(sp)
22  lw t2,-16(sp)
23  mul t1,t1,t2
24  L8:
25  sw t1,-20(sp)
26  L9:
27  lw t1,-16(sp)
28  li t2,1
29  add t1,t1,t2
30  L10:
31  sw t1,-16(sp)
32  L11:
33  b(jump) 5
34  L12:
35  lw t1,-20(sp)
36  li a7,1
37  ecall
38  la a0,str_nl
39  li a7,4
40  ecall
41  L13:
42  lw ra,(sp)
43  jr ra
44  L14:
45  addi sp,sp,12
46  mv gp,sp
47  L15:
48  sw sp,-4(fp)
49  addi sp,sp,32
50  jal _
51  addi sp,sp,32
52  L16:
53  li a0,0
54  li a7,93
55  ecall
56  L17:
57
```

```
1   begin_block   main_factorial   _   _
2   in   x   _   _
3   =   1   _   fact
4   =   1   _   i
5   <=   i   x   7
6   jump   _   _   12
7   *   fact   i   %1
8   =   %1   _   fact
9   +   i   1   %2
10  =   %2   _   i
11  jump   _   _   5
12  out   fact   _   _
13  end_block   main_factorial   _   _
14  begin_block   "__main__"   _   _
15  call   main_factorial   _   _
16  halt   _   _   _
17  end_block   "__main__"   _   _
```

❖ Αρχείο main_fibonacci

Edit Execute

telikos.asm

```
1   L0:
2   b(jump) 24
3   L1:
4   sw ra,(sp)
5   L2:
6   lw t1,-12(sp)
7   li t2,1
8   ble,t1,t2,4
9   L3:
10  b(jump) 6
11  L4:
12  lw t1,-12(sp)
13  lw t0,-8(sp)
14  sw t1,(t0)
15  lw ra,(sp)
16  jr ra
17  L5:
18  b(jump) 16
19  L6:
20  lw t1,-12(sp)
21  li t2,1
22  sub t1,t1,t2
23  L7:
24  addi fp,sp,framelength
25  sw t0,-36(fp)
26  L8:
27  addi fp,sp,framelength
28  addi t0,sp,-20
29  sw t0,-8(fp)
30  L9:
31  lw t0,-4(sp)
32  sw t0,-4(fp)
33  addi sp,sp,36
34  jal _
35  addi sp,sp,36
36  L10:
37  lw t1,-12(sp)
38  li t2,2
39  sub t1,t1,t2
40  L11:
41  addi fp,sp,framelength
42  sw t0,-52(fp)
43  L12:
44  addi fp,sp,framelength
45  addi t0,sp,-28
46  sw t0,-8(fp)
47  L13:
48  lw t0,-4(sp)
49  sw t0,-4(fp)
50  addi sp,sp,36
```

```
1   1   begin_block   fibonacci   _   _
2   2   <=   x   1   4
3   3   jump   _   _   6
4   4   retv   x   _   _
5   5   jump   _   _   16
6   6   -   x   1   %1
7   7   par   %1   CV   _
8   8   par   %2   RET   _
9   9   call   fibonacci   _   _
10  10  -   x   2   %3
11  11  par   %3   CV   _
12  12  par   %4   RET   _
13  13  call   fibonacci   _   _
14  14  +   %2   %4   %5
15  15  retv   %5   _   _
16  16  end_block   fibonacci   _   _
17  17  begin_block   main_fibonacci   _   _
18  18  in   x   _   _
19  19  par   x   CV   _
20  20  par   %6   RET   _
21  21  call   fibonacci   _   _
22  22  out   %6   _   _
23  23  end_block   main_fibonacci   _   _
24  24  begin_block   "__main__"   _   _
25  25  call   main_fibonacci   _   _
26  26  halt   _   _   _
27  27  end_block   "__main__"   _   _
```

```
50  addi sp,sp,36
51  jal _
52  addi sp,sp,36
53  L14:
54  add t1,t1,t2
55  L15:
56  lw t0,-8(sp)
57  sw t1,(t0)
58  lw ra,(sp)
59  jr ra
60  L16:
61  lw ra,(sp)
62  jr ra
63  L17:
64  sw ra,(sp)
65  L18:
66  li a7,5
67  ecall
68  L19:
69  addi fp,sp,framelength
70  lw t0,-12(sp)
71  sw t0,-20(fp)
72  L20:
73  addi fp,sp,framelength
74  addi t0,sp,-16
75  sw t0,-8(fp)
76  L21:
77  sw sp,-4(fp)
78  addi sp,sp,36
79  jal _
80  addi sp,sp,36
81  L22:
82  li a7,1
83  ecall
84  la a0,str_nl
85  li a7,4
86  ecall
87  L23:
88  lw ra,(sp)
89  jr ra
90  L24:
91  addi sp,sp,12
92  mv gp,sp
93  L25:
94  sw sp,-4(fp)
95  addi sp,sp,20
96  jal _
97  addi sp,sp,20
98  L26:
99  li a0,0
100 li a7,93
101 ecall
102 L27:
```

❖ Αρχείο main_primes

Edit Execute

telikos.asm

```
1   L0:
2   b(jump) 42
3   L1:
4   sw ra,(sp)
5   L2:
6   lw t1,-16(sp)
7   lw t2,-12(sp)
8   div t1,t1,t2
9   L3:
10  lw t2,-12(sp)
11  mul t1,t1,t2
12  L4:
13  lw t1,-16(sp)
14  beq,t1,t2,6
15  L5:
16  b(jump) 8
17  L6:
18  li t1,1
19  lw t0,-8(sp)
20  sw t1,(t0)
21  lw ra,(sp)
22  jr ra
23  L7:
24  b(jump) 9
25  L8:
26  li t1,0
27  lw t0,-8(sp)
28  sw t1,(t0)
29  lw ra,(sp)
30  jr ra
31  L9:
32  lw ra,(sp)
33  jr ra
34  L10:
35  sw ra,(sp)
36  L11:
37  li t1,2
38  sw t1,-16(sp)
39  L12:
40  lw t1,-16(sp)
41  lw t2,-12(sp)
42  blt,t1,t2,14
43  L13:
44  b(jump) 25
45  L14:
46  addi fp,sp,framelength
47  lw t0,-16(sp)
48  sw t0,-28(fp)
49  L15:
50  addi fp,sp,framelength
51  lw t0,-12(sp)
52  sw t0,-32(fp)
53  L16:
54  addi fp,sp,framelength
55  addi t0,sp,-20
56  sw t0,-8(fp)
57  L17:
58  sw sp,-4(fp)
59  addi sp,sp,28
60  jal _
61  addi sp,sp,28
62  L18:
63  li t2,1
64  beq,t1,t2,20
65  L19:
66  b(jump) 22
67  L20:
68  li t1,0
69  lw t0,-8(sp)
70  sw t1,(t0)
71  lw ra,(sp)
72  jr ra
73  L21:
74  b(jump) 22
75  L22:
76  lw t1,-16(sp)
77  li t2,1
78  add t1,t1,t2
79  L23:
80  sw t1,-16(sp)
```

```
80  sw t1,-16(sp)
81  L24:
82  b(jump) 12
83  L25:
84  li t1,1
85  lw t0,-8(sp)
86  sw t1,(t0)
87  lw ra,(sp)
88  jr ra
89  L26:
90  lw ra,(sp)
91  jr ra
92  L27:
93  sw ra,(sp)
94  L28:
95  li t1,2
96  sw t1,-12(sp)
97  L29:
98  lw t1,-12(sp)
99  li t2,30
100 ble,t1,t2,31
101 L30:
102 b(jump) 41
103 L31:
104 addi fp,sp,framelength
105 lw t0,-12(sp)
106 sw t0,-28(fp)
107 L32:
108 addi fp,sp,framelength
109 addi t0,sp,-16
110 sw t0,-8(fp)
111 L33:
112 sw sp,-4(fp)
113 addi sp,sp,28
114 jal _
115 addi sp,sp,28
116 L34:
117 li t2,1
118 beq,t1,t2,36
119 L35:
120 b(jump) 38
121 L36:
122 lw t1,-12(sp)
123 li a7,1
124 ecall
125 la a0,str_nl
126 li a7,4
127 ecall
128 L37:
129 b(jump) 38
130 L38:
131 lw t1,-12(sp)
132 li t2,1
133 add t1,t1,t2
134 L39:
135 sw t1,-12(sp)
136 L40:
137 b(jump) 29
138 L41:
139 lw ra,(sp)
140 jr ra
141 L42:
142 addi sp,sp,12
143 mv gp,sp
144 L43:
145 sw sp,-4(fp)
146 addi sp,sp,24
147 jal _
148 addi sp,sp,24
149 L44:
150 li a0,0
151 li a7,93
152 ecall
153 L45:
154
```

❖ Αρχείο main_countdigits

Edit | Execute

telikos.asm

```
1   L0:
2   b(jump) 13
3   L1:
4   sw ra,(sp)
5   L2:
6   li a7,5
7   ecall
8   L3:
9   li t1,0
10  sw t1,-16(sp)
11  L4:
12  lw t1,-12(sp)
13  li t2,0
14  bgt,t1,t2,6
15  L5:
16  b(jump) 11
17  L6:
18  lw t1,-12(sp)
19  li t2,10
20  div t1,t1,t2
21  L7:
22  sw t1,-12(sp)
23  L8:
24  lw t1,-16(sp)
25  li t2,1
26  add t1,t1,t2
27  L9:
28  sw t1,-16(sp)
29  L10:
30  b(jump) 4
31  L11:
32  lw t1,-16(sp)
33  li a7,1
34  ecall
35  la a0,str_nl
36  li a7,4
37  ecall
38  L12:
39  lw ra,(sp)
40  jr ra
41  L13:
42  addi sp,sp,12
43  mv gp,sp
44  L14:
45  sw sp,-4(fp)
46  addi sp,sp,28
47  jal _
48  addi sp,sp,28
49  L15:
50  li a0,0
51  li a7,93
52  ecall
53  L16:
54
```

```
1   1   begin_block   main_countdigits   _   _
2   2   in   x   _   _
3   3   =   0   _   count
4   4   >   x   0   6
5   5   jump   _   _   11
6   6   //   x   10   %1
7   7   =   %1   _   x
8   8   +   count   1   %2
9   9   =   %2   _   count
10  10  jump   _   _   4
11  11  out   count   _   _
12  12  end_block   main_countdigits   _   _
13  13  begin_block   "__main__"   _   _
14  14  call   main_countdigits   _   _
15  15  halt   _   _   _
16  16  end_block   "__main__"   _   _
```


Ευχαριστούμε!