ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

UNIVERSITY OF PIRAEUS

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΕΙΡΑΙΩΣ

ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΩΝ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΕΠΙΚΟΙΝΩΝΙΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

ΤΕΛΙΚΗ ΑΠΑΛΛΑΚΤΙΚΗ ΕΡΓΑΣΙΑ

ΕΠΕΞΕΡΓΑΣΙΑ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ

ΓΚΟΛΕΜΙ ΚΡΙΣΤΙΑΝ, Π18029

ΠΕΡΙΕΧΟΜΕΝΑ:

Β.1 Λύση σε άλλη γλώσσα προγραμματισμού

Θέμα 1ο (20 μονάδες)

Ανάπτυξη Λεκτικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Λεκτικό Αναλυτή που διαβάζει μια μικρή ιστορία και μπορεί να παράγει μια λίστα από προτάσεις, κάθε μια από τις οποίες περιέχει μια λίστα από λέξεις. Τεκμηριώστε πειστικά τον κώδικά σας.

Για την επίλυση του παραπάνω ερωτήματος έχει δημιουργηθεί το αρχείο lexical_analyser.py στο οποίο περιέχεται ο πηγαίος κώδικας σε python για την ανάπτυξη του λεκτικού αναλυτή που ζητείται. Παρακάτω παρατίθεται ο κώδικας ώστε να γίνει επεξήγηση του.

```
from nltk.tokenize import sent_tokenize, word_tokenize

# create a string variable containing the story in the 'story.txt' file, after we open it
with open("story.txt", "r") as story_file:
    story = story_file.read().replace('\n', '')

# operate at sentence level using the sentence tokenizer directly
print("SENTENCE TOKENIZER: ")
tokens = sent_tokenize(story)
for token in tokens:
    print("Sentence " + str(tokens.index(token) + 1) + ": " + str(token))

# operate at word level using the word tokenizer
print("\nWORD TOKENIZER: ")
word_tokens = [word_tokenize(t) for t in tokens]
for word_token in word_tokens:
    print("Words in sentence " + str(word_tokens.index(word_token) + 1) + ": " + str(word_token))
```

Ξεκινώντας, κάνουμε import τα packages sent_tokenize και word_tokenize από τη βιβλιοθήκη nltk.tokenize. Τα packages αυτά θα μας βοηθήσουν να ‘’κόψουμε’’ το κείμενο σε προτάσεις και περαιτέρω την κάθε πρόταση στις λέξεις από τις οποίες αυτή αποτελείται. Για να μπορέσουμε να κάνουμε import τα παραπάνω packages είναι απαραίτητο να έχουμε εγκατεστημένη την python στον υπολογιστή μας και έπειτα να εκτελέσουμε την παρακάτω εντολή σε κάποιο terminal: *pip3 install nltk.*

Έχοντας κάνει import τα απαραίτητα πακέτα, ανοίγουμε το αρχείο ‘story.txt’ με δικαιώματα ανάγνωσης και δημιουργούμε μια μεταβλητή τύπου string η οποία περιέχει το περιεχόμενο του αρχείου, δηλαδή την ιστορία που δόθηκε στην πρότυπη λύση του διδάσκοντα, όπως φαίνεται παρακάτω:

```
# create a string variable containing the story in the 'story.txt' file, after we open it
with open("story.txt", "r") as story_file:
    story = story_file.read().replace('\n', '')
```

Στη συνέχεια, καλούμε τη συνάρτηση sent_tokenize() με όρισμα τη μεταβλητή που δημιουργήσαμε προηγουμένως η οποία περιέχει την ιστορία, ώστε να χωριστεί το κείμενο σε προτάσεις. Οι προτάσεις εισάγονται στη λίστα tokens, και τελικά κάθε πρόταση που υπάρχει σε αυτή τη λίστα τυπώνεται. Τα παραπάνω εκτελούνται από τις εξής γραμμές κώδικα:

```
# operate at sentence level using the sentence tokenizer directly
print("SENTENCE TOKENIZER: ")
tokens = sent_tokenize(story)
for token in tokens:
    print("Sentence " + str(tokens.index(token) + 1) + ": " + str(token))
```

Τέλος, για να ολοκληρωθεί ο λεκτικός αναλυτής, για κάθε πρόταση που χωρίστηκε προηγουμένως καλούμε τη συνάρτηση word_tokenize(), ώστε να χωριστεί σε λέξεις. Έτσι, για κάθε πρόταση, δημιουργείται μια λίστα η οποία περιέχει τις λέξεις από τις οποίες αποτελείται.

```
# operate at word level using the word tokenizer
print("\nWORD TOKENIZER: ")
word_tokens = [word_tokenize(t) for t in tokens]
for word_token in word_tokens:
    print("Words in sentence " + str(word_tokens.index(word_token) + 1) + ": " + str(word_token))
```

Εκτέλεση του αρχείου lexical_analyser.py.

```
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\lexical_analyser.py"
SENTENCE TOKENIZER:
Sentence 1: the dog needs food.
Sentence 2: the cat has the food.
Sentence 3: the dog hates the cat.
Sentence 4: the dog chased the cat.
Sentence 5: the cat is scared.

WORD TOKENIZER:
Words in sentence 1: ['the', 'dog', 'needs', 'food', '.']
Words in sentence 2: ['the', 'cat', 'has', 'the', 'food', '.']
Words in sentence 3: ['the', 'dog', 'hates', 'the', 'cat', '.']
Words in sentence 4: ['the', 'dog', 'chased', 'the', 'cat', '.']
Words in sentence 5: ['the', 'cat', 'is', 'scared', '.']
PS C:\Users\golem\OneDrive - unipi.gr\NLP>
```

Όπως φαίνεται από την παραπάνω εικόνα, μετά την εκτέλεση του πηγαίου κώδικα, το κείμενο έχει χωριστεί σε προτάσεις (5 στο σύνολο) και κάθε μια από αυτές τυπώνεται αριθμητικά. Παρακάτω, η κάθε μια από αυτές τις προτάσεις έχει γίνει tokenized σε λέξεις και για κάθε μια εμφανίζεται μια λίστα η οποία περιέχει τις λέξεις από τις οποίες απαρτίζεται.

Θέμα 2ο (20 μονάδες)

Ανάπτυξη Συντακτικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Συντακτικό Αναλυτή που με βάση τους κανόνες συντακτικής ανάλυσης της πρότυπης λύσης σε Prolog που σας δόθηκε παράγει το συντακτικό δένδρο της πρότασης. Τεκμηριώστε πειστικά τον κώδικά σας.

Για τη δημιουργία του συντακτικού αναλυτή που ζητείται από το παραπάνω ερώτημα, έχει δημιουργηθεί το αρχείο syntax_analyser.py, του οποίου ο κώδικας παρατίθεται παρακάτω ώστε να γίνει επεξήγησή του.

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize

# create a grammar containing only the words needed to analyse the story in the 'story.txt'/ 'story2.txt' files
grammar = nltk.CFG.fromstring("""
S -> NP VP
NP -> PN | DET N | N
VP -> IV | IV ADV | AV ADJ | TV PN NP | V NP
DET -> 'the'
ADJ -> 'scared' | 'bad' | 'sad'
N -> 'cat' | 'dog' | 'food' | 'money' | 'thief' | 'owner' | 'police' | 'shop'
AV -> 'is'
IV ->
TV ->
V -> 'chased' | 'hates' | 'needs' | 'has' | 'gives' | 'calls' | 'enters' | 'arrests'
PN ->
ADV ->
""")

# create a string variable containing the story in the 'story.txt' file, after we open it
with open("story.txt", "r") as story_file:
    story = story_file.read().replace('\n', '')

# tokenize the sentences in the story
tokens = sent_tokenize(story)
stopwords=['.','\'']

# further tokenize the sentences into words
for token in tokens:
    word_tokens = nltk.word_tokenize(token)
    for word in list(word_tokens):
```

```
    if word in stopwords:
        word_tokens.remove(word)

print(word_tokens)
parser = nltk.ChartParser(grammar)
for tree in parser.parse(word_tokens):
    print(tree)
    print("\n")
    tree.draw()
```

Όπως και στο προηγούμενο ερώτημα, ξεκινώντας κάνουμε import τα απαραίτητα πακέτα από τη βιβλιοθήκη nltk, για την εγκατάσταση της οποίας όπως είπαμε προηγουμένως πρέπει να εκτελεστεί η εντολή *pip install nltk* σε κάποιο terminal.

Στη συνέχεια, δημιουργούμε σε μορφή string μεταβλητής μια γραμματική όπως αυτή που δόθηκε από τον διδάσκοντα στη πρότυπη λύση σε prolog. Η γραμματική είναι της μορφης CFG (Context Free Grammar) και περιέχει μόνο όσες λέξεις χρειάζονται ώστε να γίνει συντακτική ανάλυση των δύο ιστοριών που έχουμε αποθηκεύσει στα αρχεία 'story.txt' και 'story2.txt'. Στη γραμματική ορίζονται κανόνες για τις προτάσεις π.χ. S -> NP VP που σημαίνει ότι μια πρόταση αποτελείται από μια noun phrase και από μια verb phrase και παράλληλα ορίζονται οι λέξεις ως ουσιαστικά, ρήματα, αντικείμενα, επίθετα κ.α.

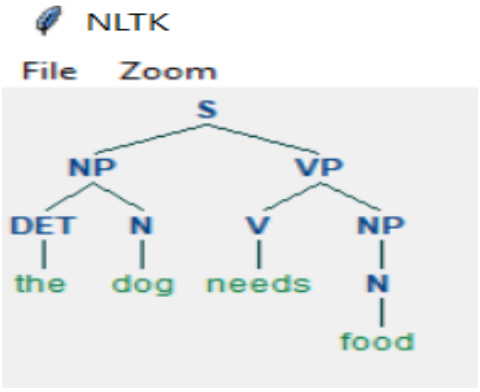
Εν συνεχεία, όπως και στο θέμα 1, ανοίγουμε το αρχείο που περιέχει την ιστορία που θέλουμε και αρχικά χωρίζουμε το κείμενο σε προτάσεις και έπειτα χωρίζουμε σε word tokens την κάθε πρόταση χρησιμοποιώντας τις συναρτήσεις sent_tokenize(), word_tokenize() αντίστοιχα.

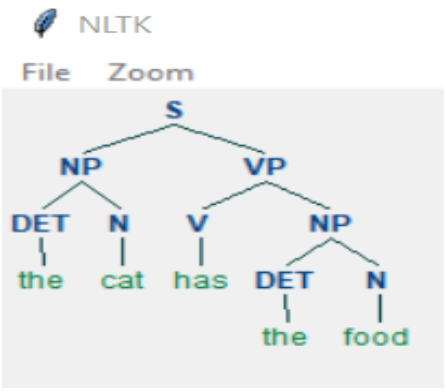
Τέλος, με τις παρακάτω γραμμές κώδικα καλούμε έναν chart parser με βάση τη γραμματική που φτιάξαμε, ο οποίος θα αναλύσει τις προτάσεις που του δίδονται κάνοντας συντακτική ανάλυση. Τελικά, η κάθε πλέον συντακτικά αναλυμένη πρόταση τυπώνεται σε μορφή δέντρου μέσω της nltk.draw().

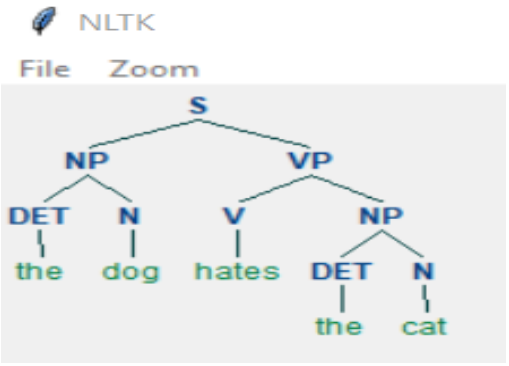
```
print(word_tokens)
parser = nltk.ChartParser(grammar)
for tree in parser.parse(word_tokens):
    print(tree)
    print("\n")
    tree.draw()
```

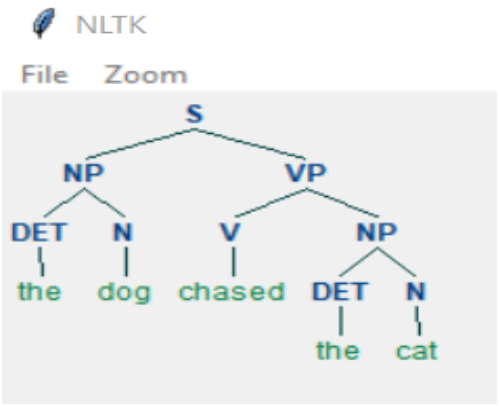
Εκτέλεση του syntax_analyser.py για τα story.txt, story2.txt.

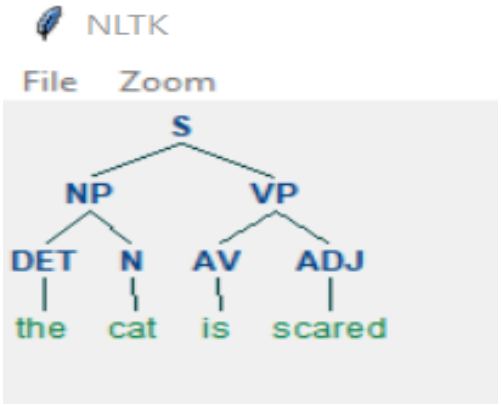
story.txt









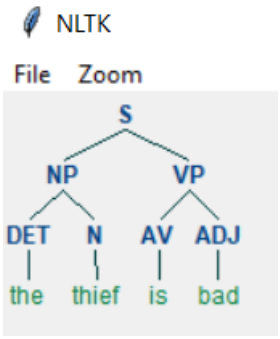


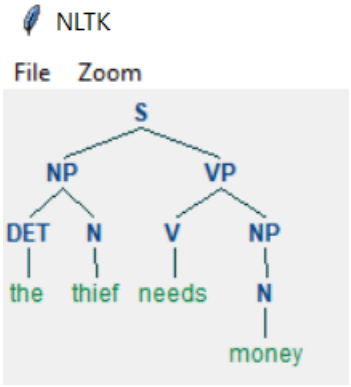
Εκτύπωση των συντακτικά αναλυμένων προτάσεων σε μορφή δέντρων. Όπως φαίνεται στα παραπάνω δένδρα, οι προτάσεις έχουν αναλυθεί συντακτικά και η κάθε λέξη βρίσκεται πλέον κάτω από συγκεκριμένο κλαδί που υποδεικνύει τι μέρος του λόγου είναι. Για παράδειγμα, στην ανάλυση της τελευταίας πρότασης, βλέπουμε πώς η λέξη “cat” είναι μέρος της noun phrase και είναι noun ενώ η λέξη ‘’scared’’ είναι adjective και μέρος της verb phrase.

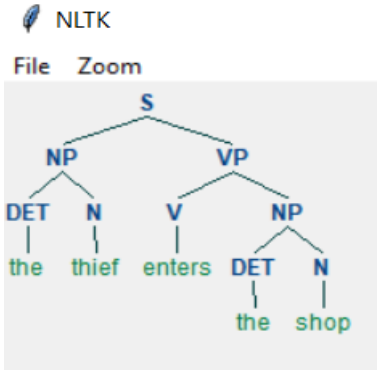
```
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\syntax_analyser.py"
['the', 'dog', 'needs', 'food']
(S (NP (DET the) (N dog)) (VP (V needs) (NP (N food))))


['the', 'cat', 'has', 'the', 'food']
(S (NP (DET the) (N cat)) (VP (V has) (NP (DET the) (N food))))


['the', 'dog', 'hates', 'the', 'cat']
(S (NP (DET the) (N dog)) (VP (V hates) (NP (DET the) (N cat))))


['the', 'dog', 'chased', 'the', 'cat']
(S (NP (DET the) (N dog)) (VP (V chased) (NP (DET the) (N cat))))


['the', 'cat', 'is', 'scared']
(S (NP (DET the) (N cat)) (VP (AV is) (ADJ scared)))
```

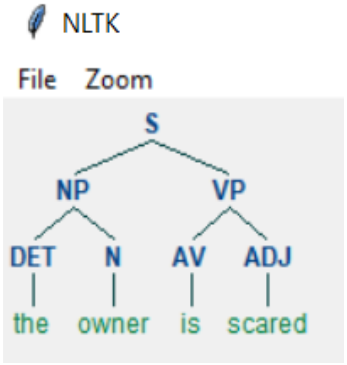
Εκτύπωση των συντακτικά αναλυμένων προτάσεων σε μορφή κειμένου.

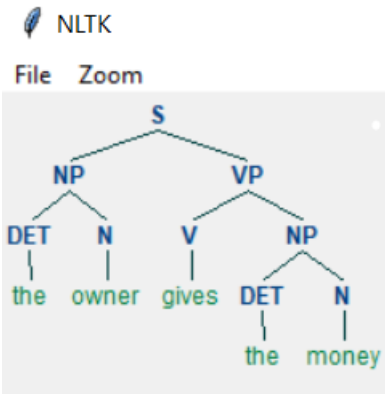
story2.txt

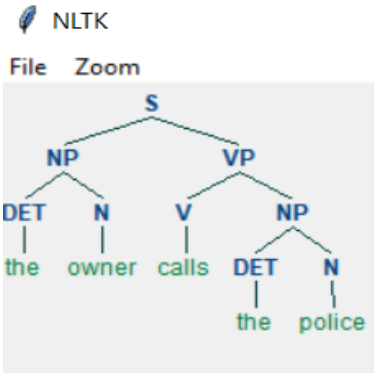


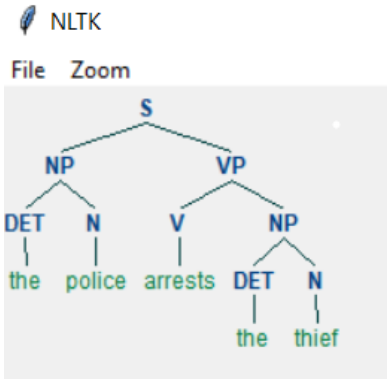


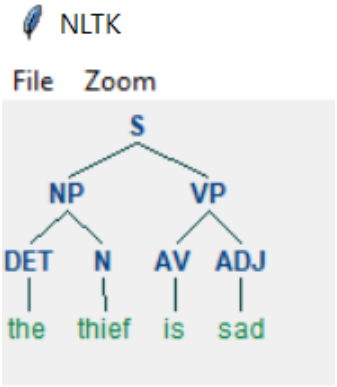












Όπως και στην προηγούμενη ιστορία που αναλύθηκε, οι προτάσεις αναλύονται συντακτικά και εμφανίζονται με τη μορφή δένδρου μέσω της nltk.draw() στο αντίστοιχο ‘’κλαδί’’ ανάλογα με το μέρος του λόγου που αντιπροσωπούν. Εκτός από την εμφάνιση των προτάσεων συντακτικά αναλυμένων με τη μορφή δένδρων, γίνεται print στο command line της ανάλυσης σε μορφή κειμένου, όπως φαίνεται στην παρακάτω εικόνα:

```
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive
['the', 'thief', 'is', 'bad']
(S (NP (DET the) (N thief)) (VP (AV is) (ADJ bad)))


['the', 'thief', 'needs', 'money']
(S (NP (DET the) (N thief)) (VP (V needs) (NP (N money))))


['the', 'thief', 'enters', 'the', 'shop']
(S (NP (DET the) (N thief)) (VP (V enters) (NP (DET the) (N shop))))


['the', 'owner', 'is', 'scared']
(S (NP (DET the) (N owner)) (VP (AV is) (ADJ scared)))


['the', 'owner', 'gives', 'the', 'money']
(S (NP (DET the) (N owner)) (VP (V gives) (NP (DET the) (N money))))


['the', 'owner', 'calls', 'the', 'police']
(S (NP (DET the) (N owner)) (VP (V calls) (NP (DET the) (N police))))


['the', 'police', 'arrests', 'the', 'thief']
(S
  (NP (DET the) (N police))
  (VP (V arrests) (NP (DET the) (N thief))))


['the', 'thief', 'is', 'sad']
(S (NP (DET the) (N thief)) (VP (AV is) (ADJ sad)))
```

Θέμα 3ο (30 μονάδες)

Ανάπτυξη Σημασιολογικού Αναλυτή σε άλλη γλώσσα Προγραμματισμού. Αναζητείστε στο διαδίκτυο ή αναπτύξτε εσείς Σημασιολογικό Αναλυτή που με βάση τους κανόνες σημασιολογικής ανάλυσης της πρότυπης λύσης σε Prolog που σας δόθηκε παράγει τα σημαινόμενα της πρότασης (σχέσεις μεταξύ ρημάτων, ουσιαστικών, επιθέτων, κ.λπ). Τεκμηριώστε πειστικά τον κώδικά σας.

Θέμα 4ο (30 μονάδες)

Πρόγραμμα στην γλώσσα προγραμματισμού της επιλογής σας, για την ενημέρωση και πραγματοποίηση ερωταποκρίσεων σε Βάση Γνώσης που έχετε αναπτύξει γιαα αυτό τον σκοπό. Οι ερωτήσεις και απαντήσεις θα δίδονται σε φυσική γλώσσα.

Για την επίλυση των παραπάνω θεμάτων έχουν δημιουργηθεί τα εξής αρχεία: semantic_analyser.py, grammar_vocabulary.fcfg, καθώς και μια βάση δεδομένων σε MySQL Server μέσω του προγράμματος MySQL Workbench η οποία περιέχει τη βάση γνώσης που απαιτείται όπως αυτή δόθηκε από τον διδάσκοντα στην πρότυπη λύση σε Prolog.

Το αρχείο semantic_analyser.py περιέχει τον πηγαίο κώδικα ο οποίος σε συνδυασμό με τη βάση δεδομένων και το αρχείο grammar_vocabulary.py το οποίο περιέχει τη γραμματική και το λεξιλόγιο που έχουμε δημιουργήσει, δημιουργεί τον σημασιολογικό αναλυτή καθώς και τις ερωτοαποκρίσεις που ζητούνται.

Ξεκινώντας είναι απαραίτητο να δημιουργηθεί η βάση δεδομένων. Για τον λόγο αυτό θα πρέπει να εγκαταστηθούν ο MySQL Server και το MySQL Workbench. Έχοντας εγκαταστήσει τα παραπάνω, μέσω του Workbench, δημιουργούμε ένα νέο connection σε localhost με τα εξής στοιχεία: username: root, password: test123, name: nlp_db (τα παραπάνω στοιχεία είναι ενδεικτικά, ωστόσο παρατίθενται καθώς αυτά ήταν τα στοιχεία που χρησιμοποιήθηκαν στη συγκεκριμένη περίπτωση). Αφού δημιουργήσουμε το connection επιλέγουμε create schema και δημιουργούμε μια βάση με όνομα

nlp_db. Στη συνέχεια πατώντας το κουμπί ''Open a SQl script file'' που βρίσκεται στο toolbar, επιλέγουμε το αρχείο nlp_db.sql. Το αρχείο αυτό περιέχει τον κώδικα που χρησιμοποιήθηκε για τη δημιουργία του πίνακα nlp_db.data ο οποίος περιέχει τη βάση γνώσης με τα απαραίτητα columns αλλά και το script για την εισαγωγή των δεδομένων. Εκτελώντας, λοιπόν το συγκεκριμένο αρχείο δημιουργείται η βάση γνώσης μας και θα πρέπει να έχουν πλέον εισαχθεί και τα απαραίτητα δεδομένα για να γίνουν αργότερα δοκιμές.

Παρακάτω φαίνεται η εικόνα του πίνακα που έχουμε δημιουργήσει στο MySQL Workbench με τα δεδομένα που έχουν εισαχθεί, εκτελώντας το script ''select * from nlp_db.data'':

```
5 •  select * from nlp_db.data;
```

Result Grid | Filter Rows: | Edit: | Export/Import:

id	name	verb	noun	adjective	intransitive_verb	transitive_verb	adverb	receiver
1	mary	NULL	dog	NULL	NULL	give	NULL	john
2	mary	NULL	book	NULL	NULL	give	NULL	tomy
3	mary	NULL	NULL	NULL	run	NULL	quickly	NULL
4	mary	NULL	NULL	tall	NULL	NULL	NULL	NULL
5	mary	NULL	NULL	slim	NULL	NULL	NULL	NULL
6	mary	NULL	NULL	blonde	NULL	NULL	NULL	NULL
7	mary	love	book	NULL	NULL	NULL	NULL	NULL
8	dog	need	food	NULL	NULL	NULL	NULL	NULL
9	cat	have	food	NULL	NULL	NULL	NULL	NULL
10	dog	hate	cat	NULL	NULL	NULL	NULL	NULL
11	dog	chase	cat	NULL	NULL	NULL	NULL	NULL
12	cat	NULL	NULL	scary	NULL	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

Έχοντας δημιουργήσει τη βάση γνώσης που θα χρειαστούμε, προχωρούμε στη δημιουργία της γραμματικής και του λεξιλογίου που βρίσκεται στο αρχείο grammar_vocabulary.fcfg. Μέσω του αρχείου αυτού (Feature Context Free Grammar) ορίζονται ταυτόχρονα τα σημαινόμενα της κάθε πρότασης, δηλαδή οι σχέσεις μεταξύ των μερών του λόγου κ.α. που θα χρειαστούν για την ανάπτυξη του σημασιολογικού αναλυτή. Το συγκεκριμένο αρχείο, λοιπόν, αποτελεί την απάντηση του θέματος 3, στο οποίο ζητείται η ανάπτυξη του σημασιολογικού αναλυτή με τα σημαινόμενα των προτάσεων, τη γραμματική και το λεξιλόγιο.

Τέλος, για τη δημιουργία των ερωτο-αποκρίσεων χρησιμοποιώντας τη βάση γνώσης και το αρχείο grammar_vocabulary.fcfg που δημιουργήσαμε, χρησιμοποιείται πηγαίος κώδικας σε python μέσω του αρχείου semantic_analyser.py το οποίο θα αναλύσουμε παρακάτω.

Ξεκινώντας, όπως και στα προηγούμενα ερωτήματα γίνονται import τα απαραίτητα πακέτα μέσω της βιβλιοθήκης NLTK και στη συνέχεια ‘’φορτώνουμε’’ το αρχείο grammar_vocabulary.fcfg και δημιουργούμε έναν parser πάνω σε αυτό ώστε να μπορούμε αργότερα να κάνουμε συντακτική ανάλυση. Τα παραπάνω γίνοντα από τις εξής γραμμές κώδικα στις οποίες περιέχονται επεξηγηματικά σχόλια, σε όσα σημεία κρίνεται απαραίτητο:

```
from mysql.connector import connection
import nltk
import mysql.connector
from nltk import load_parser

# load the grammar and vocabulary we have created in the 'grammar_vocabulary.fcfg' file
grammar = nltk.data.load('grammar_vocabulary.fcfg')
# allows us to read the grammar into NLTK, ready for use in parsing
chart_parser = load_parser('grammar_vocabulary.fcfg')
```

Εν συνεχεία, προχωρούμε στον ορισμό κάποιων ερωτήσεων σε μορφή φυσικής γλώσσας (natural language query – nl_query) ως παραδείγματα για τον έλεγχο της λειτουργικότητας του κώδικα.

```
# natural language query entered by the user
# example queries shown below
'''
nl_query = "who loves book"
nl_query = "who needs food"
nl_query = "who has food"
nl_query = "who chases cat"
'''
nl_query = "who runs quickly"
```

Έπειτα, αφού ο χρήστης επιλέξει να κάνει μια ερώτηση σε φυσική γλώσσα, επεξεργαζόμαστε το ερώτημα αυτό σε συνδυασμό με το αρχείο γραμματικής και λεξιλογίου που έχουμε, ώστε τελικά να το φέρουμε σε μορφή SQL query ώστε να το υποβάλλουμε στη βάση δεδομένων που περιέχει τη βάση γνώσης και να πάρουμε απάντηση. Παρακάτω φαίνονται τα βήματα που οδηγούν από το ερώτημα φυσικής γλώσσας (nl_query), σε ένα ερώτημα σε μορφή SQL (query_db – database query) το οποίο θα υποβληθεί στη βάση. Π.χ.

Natural language query: who needs food

SQL query: SELECT Name FROM data WHERE Verb='need' AND Noun='food';

Στον κώδικα υπάρχουν σχόλια που επεξηγούν τη λειτουργία της κάθε γραμμής, όπου αυτό είναι απαραίτητο.

```
# using the .fcfg file we created and the user's query,
# create an SQL query in order to submit it to the database and get the wanted answer
# split the nl_query into tokens
tokens = nl_query.split()
# parse the tokens using the chart parser we created earlier
parsed = chart_parser.parse(tokens)
trees = list(parsed)

query = trees[0].label()['SEM']
query = [i for i in query if i]
# final form of the query that will be submitted to the database
query_db = ' '.join(query) + ';'
```

Τελικά, αφού από το ερώτημα φυσικής γλώσσας καταφέραμε να δημιουργήσουμε ένα ερώτημα σε μορφή SQL, είμαστε σε θέση πλέον να συνδεθούμε μέσω του πηγαίου κώδικα με τη βάση και να υποβάλουμε το ερώτημα μας ώστε να πάρουμε την απάντηση.

```
# establish connection with the database 'nlp_db' giving our credentials
connection = mysql.connector.connect(
   host = "localhost",
   user = "root",
   password = "test123",
   database = "nlp_db"
)

# create a cursor object in order to submit the query created to the database
cursor = connection.cursor()
# execute the query
cursor.execute(query_db)
# fetch the answers from the database
records = cursor.fetchall()
for row in records:
   print("The answer to your question is: " + row[0])

connection.close()
```

Όπως φαίνεται από τον παραπάνω κώδικα, συνδεόμαστε αρχικά με τη βάση εισάγοντας τα στοιχεία μας, όπως αυτά αναφέρθηκαν προηγουμένως, και στη συνέχεια μέσω ενός αντικειμένου τύπου cursor, υποβάλλουμε το ερώτημα στη βάση ώστε να πάρουμε την τελική απάντηση.

Εκτέλεση του semantic_analyser.py με διάφορα παραδείγματα ερωτήσεων:

```
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\semantic_analyser.py"
Your question: who runs quickly
The answer to your question is: mary
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\semantic_analyser.py"
Your question: who chases cat
The answer to your question is: dog
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\semantic_analyser.py"
Your question: who loves book
The answer to your question is: mary
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\semantic_analyser.py"
Your question: who has food
The answer to your question is: cat
PS C:\Users\golem\OneDrive - unipi.gr\NLP> python -u "c:\Users\golem\OneDrive - unipi.gr\NLP\semantic_analyser.py"
Your question: who needs food
The answer to your question is: dog
```