ΠΑΝΑΓΙΩΤΗΣ ΑΠΟΣΤΟΛΟΠΟΥΛΟΣ (Π17007)

ΕΡΓΑΣΙΑ ΕΠΕΞΕΡΓΑΣΙΑΣ ΦΥΣΙΚΗΣ ΓΛΩΣΣΑΣ

ΑΚΑΔΗΜΑΪΚΟ ΕΤΟΣ 2019-2020

Περιεχόμενα

[Θέμα 1ο 2](#_Toc46315112)

[1. Εισαγωγή 2](#_Toc46315113)

[2. Παρουσίαση κεντρικής ιδέας 2](#_Toc46315114)

[2.1 Κωδικοποίηση κειμένου 2](#_Toc46315115)

[2.2 Αναπαράσταση 3](#_Toc46315116)

[2.3 Ομοιότητα διανυσμάτων 4](#_Toc46315117)

[2.4 Στατιστικό γλωσσικό μοντέλο 4](#_Toc46315118)

[2.5 Το μοντέλο word2vec 5](#_Toc46315119)

[2.6 Κατηγοριοποίηση κειμένου 6](#_Toc46315120)

[2.7 Συμπέρασμα 6](#_Toc46315121)

[Θέμα 2ο 7](#_Toc46315122)

[Ερώτημα 1ο 7](#_Toc46315123)

[Ερώτημα 2ο 10](#_Toc46315124)

# Θέμα 1ο

# 1. Εισαγωγή

Ένα από τα σύγχρονα θέματα της Επεξεργασίας Φυσικής Γλώσσας (Natural Processing Language) είναι αυτό της κατηγοριοποίησης των διάφορων κειμένων ως θετικά ή αρνητικά. Η ταξινόμηση των κειμένων σε αυτές τις δύο κατηγορίες είναι πολύ σημαντική καθώς μπορούν μεγάλες πλατφόρμες ή μέσα κοινωνικής δικτύωσης να τα χρησιμοποιήσουν με κατάλληλο τρόπο, προκειμένου να παραχθούν χρήσιμα συμπεράσματα. Επιπρόσθετα, να σημειώσουμε ότι η χρήση αντίστοιχων εφαρμογών γίνεται από πλατφόρμες όπως το Facebook, το Twitter, αλλά και από το Imdb για την αποσαφήνιση των κριτικών από τους χρήστες για τις διάφορες ταινίες/σειρές. Για την αντιμετώπιση του συγκεκριμένου θέματος μπορούμε να φτιάξουμε ένα μοντέλο πρόβλεψης. Πιο συγκεκριμένα, ένας ενδεδειγμένος τρόπος προσέγγισης είναι αυτός των **νευρωνικών δικτύων** και **word embeddings**.

# 2. Παρουσίαση κεντρικής ιδέας

Η υλοποίηση του συγκεκριμένου μοντέλου θα μπορούσε να γίνει με την χρήση της γλώσσας προγραμματισμού **Python**, λόγω των έτοιμων βιβλιοθηκών που μας προσφέρει. Ωστόσο, στη συγκεκριμένη παρουσίαση θα δοθεί βάρος στην ανάλυση του θεωρητικού υποβάθρου και όχι του κώδικα, με την παρουσίαση αρκετών χρήσιμων μοντέλων που συμβάλλουν στην επίτευξη του στόχου.

# 2.1 Κωδικοποίηση κειμένου

Αρχικά, για την επίτευξη του στόχου μας θα πρέπει να έχουμε ένα σύνολο από κείμενα(κριτικές), τα οποία να έχουν κάποια ετικέτα, που να τα χαρακτηρίζει ως θετικά ή αρνητικά. Σε αυτό το σημείο δεν μας ενδιαφέρει το σημασιολογικό περιεχόμενο του κειμένου, δηλαδή δεν θα πραγματοποιήσουμε κάποια σημασιολογική ανάλυση. Επιπρόσθετα, δεν θα ληφθούν υπόψη τα οποία σημασιολογικά σχήματα, όπως οι μεταφορές ή οι παρομοιώσεις.

Μια προσέγγιση για τον «χαρακτηρισμό» ενός κειμένου σε θετικό ή αρνητικό θα μπορούσε να ήταν η δημιουργία ενός λεξικού με όλες τις πιθανές λέξεις, ώστε κάθε φορά να μετράμε το βάρος των λέξεων του κειμένου. Για παράδειγμα σε ένα κείμενο 200 λέξεων θα βρίσκαμε τις θετικές και αρνητικές λέξεις του κειμένου και ανάλογα το πρόσημο, δηλαδή την μεταξύ τους διαφορά, θα εξαγάγαμε το συμπέρασμά μας . Ωστόσο, αυτή η τεχνική εμπεριέχει τον κίνδυνο του καθορισμού του βάρους μιας λέξης με υποκειμενικά κριτήρια και όχι με αντικειμενικά ή ακόμα και την ύπαρξη σαρκασμών και ειρωνειών (σημασιολογική αμφισημία).

Έτσι κάθε κείμενο θα το διαχειριζόμαστε ως μια διαφορετική οντότητα, δίχως να ενδιαφερόμαστε εάν ένα κείμενο είναι θετικό ή αρνητικό, και να φτιάξουμε ένα σύστημα το οποίο να μας υπολογίζει την πιθανότητα να ανήκει σε μία από τις δύο κατηγορίες με βάση τις προηγούμενες προσημειώσεις.

# 2.2 Αναπαράσταση

Για να μπορέσουμε να κατασκευάσουμε ένα μοντέλο πρόβλεψης θα πρέπει να αναπαραστήσουμε τις λέξεις σε μορφή διανύσματος, ώστε να αυτά να δίδονται σαν είσοδο σε κάποιον αλγόριθμο μηχανικής μάθησης, όπως για παράδειγμα σε ένα νευρωνικό δίκτυο. Πιο συγκεκριμένα, αφού σκανάρουμε όλα τα διαθέσιμα κείμενα, βρίσκουμε τις μοναδικές εμφανίσεις της κάθε λέξης και τις αποθηκεύουμε σε μια λίστα-λεξικό (dictionary), με διαστάσεις **v**, όσες και οι λέξεις. Για κάθε λέξη σε αυτό το λεξικό δημιουργούμε ένα διάνυσμα **v** θέσεων, στο οποίο στη θέση στην οποία αντιστοιχεί η θέση της λέξης, βάζουμε την τιμή **1** και σε όλες τις υπόλοιπες. Για παράδειγμα εάν ο λεξικό μας έχει τις λέξεις [“the”, “big”, “dog”], το διάνυσμα που αντιστοιχεί στη λέξη “the” είναι το [1,0,0], στη λέξη “big” είναι το [0,1,0] και στη λέξη “dog” είναι το [0,0,1]. Συνήθως, κατά την διαδικασία «χτισίματος του λεξικού» για να κρατήσουμε τον αριθμό των λέξεων μικρό μετατρέπουμε τα κείμενα σε lower case και παραλείπουμε τα στοιχεία στίξης, αλλά και αφαιρούμε τις λέξεις με λιγότερες από **k** εμφανίσεις, όπου **k** είναι μια παράμετρος που την θέτουμε εμείς.

# 2.3 Ομοιότητα διανυσμάτων

Σημαντικό βήμα είναι η αξιοποίηση των διανυσμάτων που δημιουργήσαμε, δηλαδή να είμαστε σε θέση να συγκρίνουμε πόσο δύο διανύσματα στον χώρο είναι «κοντά», δηλαδή σημασιολογικά. Επειδή δύναται να έχουμε διανύσματα με τελείως διαφορετικά μήκη δεν θα χρησιμοποιήσουμε το εσωτερικό γινόμενο, αλλά η γωνία συνημίτονου (Εικόνα 1).

# 2.4 Στατιστικό γλωσσικό μοντέλο

Με τον όρο αυτό αναφερόμαστε σε μια κατανομή, είναι δηλαδή μια ακολουθία κατανομής λέξεων. Αυτό που μας ενδιαφέρει, είναι η εύρεση της κατανομής της λέξης η οποία «ακολουθεί» μια συγκεκριμένη ακολουθία λέξεων. Η συγκεκριμένη μοντελοποίηση είναι ευρέως γνωστή σε εφαρμογές όπως το Word (spell checking), η αναγνώριση φωνής ακόμα και το autocomplete. Aαναλυτικότερα για την πρώτη εφαρμογή, για μια λέξη που δεν είναι στο λεξικό, αναζητούνται κάποια μέτρα απόστασης, ώστε να βρεθούν οι λέξεις που ταιριάζουν, ταξινομημένες πάντα σύμφωνα με τις πιθανότητές τους. Ωστόσο, στη φυσική γλώσσα υπάρχει η πιθανότητα να δημιουργηθεί μια καινούργια πρόταση, η οποία να μην υπάρχει σε καμία συλλογή οπότε η αντιμετώπιση είναι πιο περίπλοκη.

# 2.5 Το μοντέλο word2vec

Για την αντιμετώπιση του παραπάνω προβλήματος υιοθετήθηκε η πρόταση του Mikolov, γνωστό και ως το μοντέλο **word2vec**. Κάθε φορά που εκπαιδεύουμε ένα τέτοιο μοντέλο, επιθυμούμε είτε να του δώσουμε k λέξεις προσπαθώντας να βρούμε την επόμενη, είτε να του δίνουμε μια λέξη και θα προσπαθούμε να βρούμε σε ένα «παράθυρο», έστω k λέξεων ποιες είναι οι πιο πιθανές λέξεις που το συνοδεύουν. Η πρώτη αρχιτεκτονική ονομάζεται **CBOW** και η δεύτερη **Skip-gram** (Εικόνα)**.** Το μοντέλο αυτό για να είναι λειτουργικό και αξιόπιστο θα πρέπει αφού το «χτίσουμε», να το εκπαιδεύσουμε. Για την επίτευξη του στόχου μας η Python προσφέρει μια σειρά από βιβλιοθήκες και συναρτήσεις, οι οποίες συμβάλλουν σε αυτό.

**Εικόνα που περιέχει χάρτης

Περιγραφή που δημιουργήθηκε αυτόματα**

Εικόνα 1- Μοντέλα word2vec

# 2.6 Κατηγοριοποίηση κειμένου

Έχοντας στην φαρέτρα μας όλα τα παραπάνω εργαλεία είμαστε σε θέση να τα συνδυάσουμε, ώστε να κατηγοριοποιήσουμε ένα νέο-άγνωστο κείμενο σε μία ευρύτερη κατηγορία. Αρχικά, κάθε κείμενο θα αναπαρίσταται ως ο μέσος όρος των διανυσμάτων των λέξεων του συγκεκριμένου κειμένου[[1]](#footnote-1). Για να είμαστε σίγουροι ότι τα αποτελέσματα της εκπαίδευσης του μοντέλου μας θα είναι τέτοια, ώστε να μπορεί να ανταπεξέλθει σε ένα νέο κείμενο, θα πρέπει τα δεδομένα μας, δηλαδή τα κείμενα τα οποία είναι ήδη κατηγοριοποιημένα να τα χωρίσουμε σε δύο κατηγορίες: αυτά που θα χρησιμοποιηθούν για την εκπαίδευση και αυτά για τον έλεγχο. Η αναλογία συνήθως είναι 75% για την πρώτη κατηγορία και 25% για τη δεύτερη. Επιπρόσθετα, για τον αλγόριθμο μηχανικής μάθησης χρησιμοποιούμε **Λογιστική Παλινδρόμηση,** δηλαδή το μοντέλο θα μάθει την πιθανότητα η ετικέτα να είναι ένα (θετική) δεδομένου ενός κειμένου. Συνεπώς, ο αλγόριθμος προσπαθεί να βρει τα βάρη, ώστε να ελαχιστοποιήσει το σφάλμα στα δεδομένα εκπαίδευσης. Με αυτή τη τεχνική είμαστε σε θέση να γνωρίζουμε σε τι ποσοστό ταξινομεί ορθά τα κείμενα το μοντέλο.

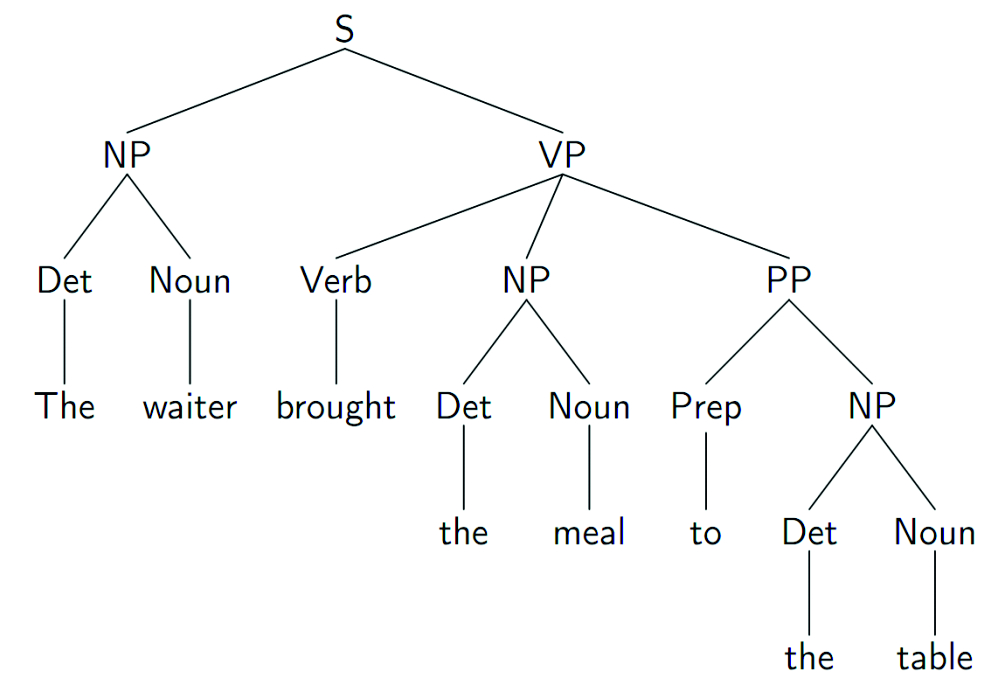
# 2.7 Συμπέρασμα

Σύμφωνα με όλα τα παραπάνω, αντιλαμβανόμαστε ότι αυτή η διαδικασία προϋποθέτει την σωστή εκπαίδευση του μοντέλου μας, εισάγοντάς του δεδομένα-κείμενα τα οποία είναι ήδη προσημειωμένα. Επίσης, όσο πιο μεγάλα είναι τα κείμενά μας τόσο μεγαλύτερη ακρίβεια θα υπάρχει σε νέα κείμενα. Το θέμα αυτό είναι από τα πιο σύγχρονα στην Επεξεργασία Φυσικής Γλώσσας και αδιαμφισβήτητα έχει πολλές προεκτάσεις.

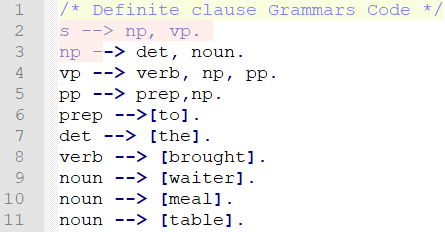
# Θέμα 2ο

# Ερώτημα 1ο

Προκειμένου να αναγνωρίσουμε την πρόταση : [the, waiter, brought, the, meal, to, the, table], σύμφωνα με το παρακάτω σχήμα (**Εικόνα** ), θα πρέπει να χρησιμοποιήσουμε την ακόλουθη γραμματική , η οποία είναι σε μορφή DCG (Defining Clause Grammars)[[2]](#footnote-2).

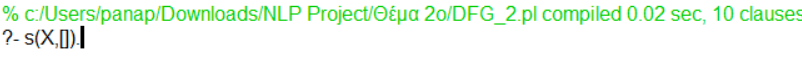


Εικόνα 2- Συντακτικό δέντρο



Εικόνα 3- Γραμματική για την αναγνώριση της πρότασης

Συνεπώς, παρατηρούμε ότι μια πρόταση χωρίζεται σε μία *noun\_phrase* και μια *verb\_phrase*, οι οποίες στη συνέχεια θα αναλυθούν περαιτέρω. Αυτό που φαίνεται παραπάνω είναι η ανάλυση μιας *noun\_phrase* σε *det* και *noun*, ενώ μια *verb\_phrase* σε *verb, noun\_phrase* και *preposition\_phrase.* H τελευταία με τη σειρά της αναλύεται σε *prep (preposition)* και *noun\_phrase.* Επόμενο βήμα είναι η ανάλυση των *prep,det,verb* και *noun*, δηλαδή ο καθορισμός του συνόλου των δυνατών τιμών που μπορούν να λάβουν. Αυτό θα έχει και ως αποτέλεσμα την αναγνώριση της ζητούμενης πρότασης. Για την επιβεβαίωση της ορθότητας της γραμματικής μας έχουμε δύο δυνατούς τρόπους: ο πρώτος είναι να δημιουργήσουμε όλες τις δυνατές προτάσεις και να βρούμε την ζητούμενη και ο δεύτερος να «ρωτήσουμε» αν η συγκεκριμένη πρόταση είναι true or false. Παρακάτω παρατίθενται και οι δύο τρόποι.



Εικόνα που περιέχει κείμενο, εφημερίδα

Περιγραφή που δημιουργήθηκε αυτόματα

Εικόνα 4- Εμφάνιση όλων των δυνατών προτάσεων



Εικόνα 5- 2ος τρόπος αναγνώρισης

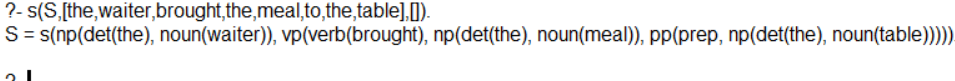
# Ερώτημα 2ο

Στη Prolog μπορούμε να παράγουμε συντακτικά δέντρα που είναι ισοδύναμα με αυτό της εικόνας 1, στην μορφή **functor**. Προκειμένου, να επιτευχθεί αυτό θα πρέπει να αλλάξουμε την παραπάνω γραμματική, όπως φαίνεται παρακάτω[[3]](#footnote-3).

Εικόνα που περιέχει οθόνη, τηλέφωνο, άνδρας, ιδιοκτησία

Περιγραφή που δημιουργήθηκε αυτόματα

Ουσιαστικά, αυτό που παρατηρούμε είναι ότι η φιλοσοφία της ανάλυσης της κάθε φράσης είναι ίδια. Οι αλλαγές που έγιναν αφορούν μόνο το κομμάτι της εμφάνισης του συντακτικού δέντρου σε μορφή **functor.**



Εικόνα 6- Εμφάνιση συντακτικού δέντρου σε μορφή **functor**

Στην παραπάνω εικόνα φαίνεται το συντακτικό δέντρο σε μορφή **functor**, για την αναγνώριση της πρότασης [the, waiter, brought, the, meal, to, the, table].

# Θέμα 4ο

# 1. Παρουσίαση κεντρικής ιδέας

Σκοπός αυτής της άσκησης είναι η ανάπτυξη ενός Λεκτικού Αναλυτή σε άλλη γλώσσα προγραμματισμού, η οποία θα διαβάζει μια μικρή ιστορία και θα είναι σε θέση να παράγει μια λίστα από προτάσεις, κάθε μία από τις οποίες περιέχει μια λίστα από λέξεις. Η υλοποίηση της συγκεκριμένης άσκησης έγινε στη γλώσσα προγραμματισμού **Python version 3.8.**  Αξίζει να σημειωθεί ότι χρησιμοποιήθηκε η βιβλιοθήκη **nltk**, η οποία αφορά την Επεξεργασία Φυσικής Γλώσσας. Έτσι στη συγκεκριμένη εφαρμογή θα ζητάμε από τον χρήστη να εισάγει το όνομα του αρχείου όπου επιθυμεί να διαβαστεί προκειμένου να υποστεί την συγκεκριμένη διαδικασία, δηλαδή τον χωρισμό σε προτάσεις.

# 2. Επεξήγηση Κώδικα

# 2.1 Συνάρτηση sentence\_split()

Όπως αναφέρθηκε και παραπάνω το πρόγραμμα έχει υλοποιηθεί σε **Python.**  Η κύρια συνάρτηση, η οποία και επιτυγχάνει την δημιουργία λιστών από προτάσεις, όπου η καθεμία περιέχει μια λίστα από λέξεις, είναι η **sentence\_split()**. Η συγκεκριμένη συνάρτηση έχει σαν όρισμα το κείμενο, το οποίο θα υποστεί την παραπάνω διαδικασία. Αρχικά, για κάθε μια πρόταση θα εφαρμοστεί η μέθοδος **tokenize*(sent\_tokenize)***. Στη συνέχεια για κάθε λέξη της πρότασης που βρίσκεται στην πρόταση, εφαρμόζεται και για αυτή η μέθοδος **tokenize*(word\_tokenize)****.* Οι λέξεις αυτές αποθηκεύονται σε μία λίστα, την **word.**

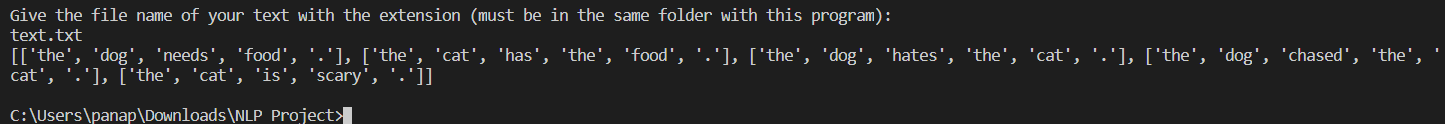
# 2.2 Εκτέλεση διαδικασίας

Για την εκτέλεση της διαδικασίας, αφού πρώτα έχει ανοιχτεί και διαβαστεί το αρχείο με την βοήθεια των συναρτήσεων **.open()** και **.read()**, αντίστοιχα, θα κληθεί η συνάρτηση **sentence\_split**, με όρισμα το κείμενο που έχει διαβαστεί από το αρχείο.

# 3. Αποτελέσματα εκτέλεσης

Στην παρακάτω εικόνα βλέπουμε το αποτέλεσμα της εκτέλεσης του κώδικα για το αρχείου ’text.txt’, το οποίο περιλαμβάνει το εξής κείμενο:

|  |
| --- |
| the dog needs food. the cat has the food. the dog hates the cat.  the dog chased the cat. the cat is scary. |



Εικόνα 7-Αποτέλσμα εκτέλεσης προγράμματος για το κείμενο του αρχείου 'text.txt'

# 4. Απαιτήσεις συστήματος και εκτέλεση

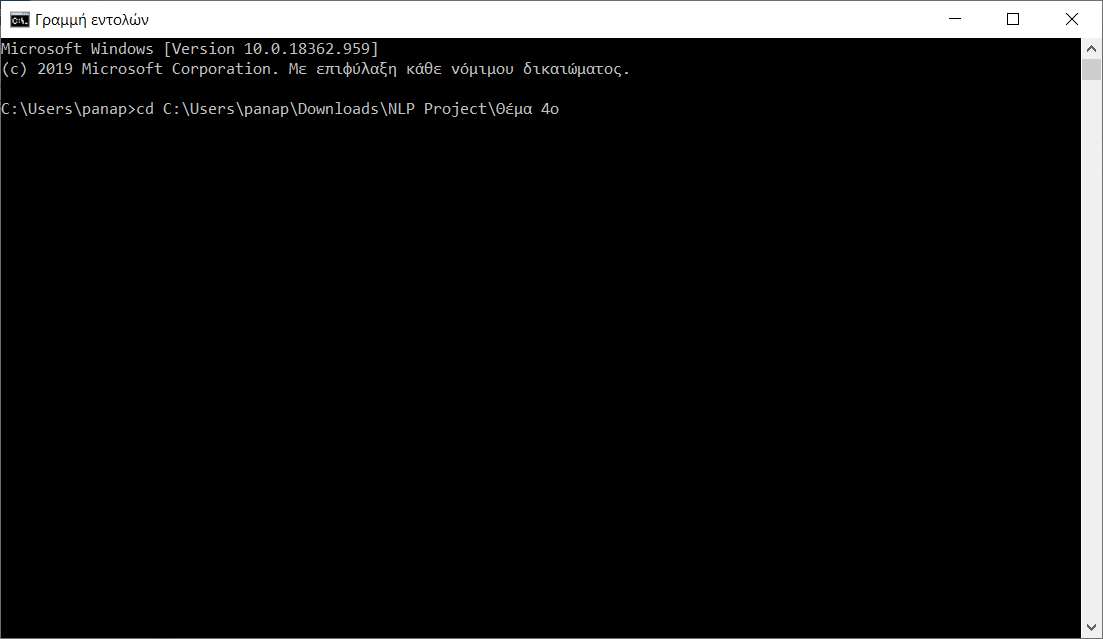
Η παραπάνω άσκηση υλοποιήθηκε στην γλώσσα προγραμματισμού ***Python, version 3.8****.* Για να μπορεί να γίνει η εκτέλεση του προγράμματος είναι απαραίτητο να είναι εγκατεστημένες οι βιβλιοθήκες:

* **os**
* **nltk**

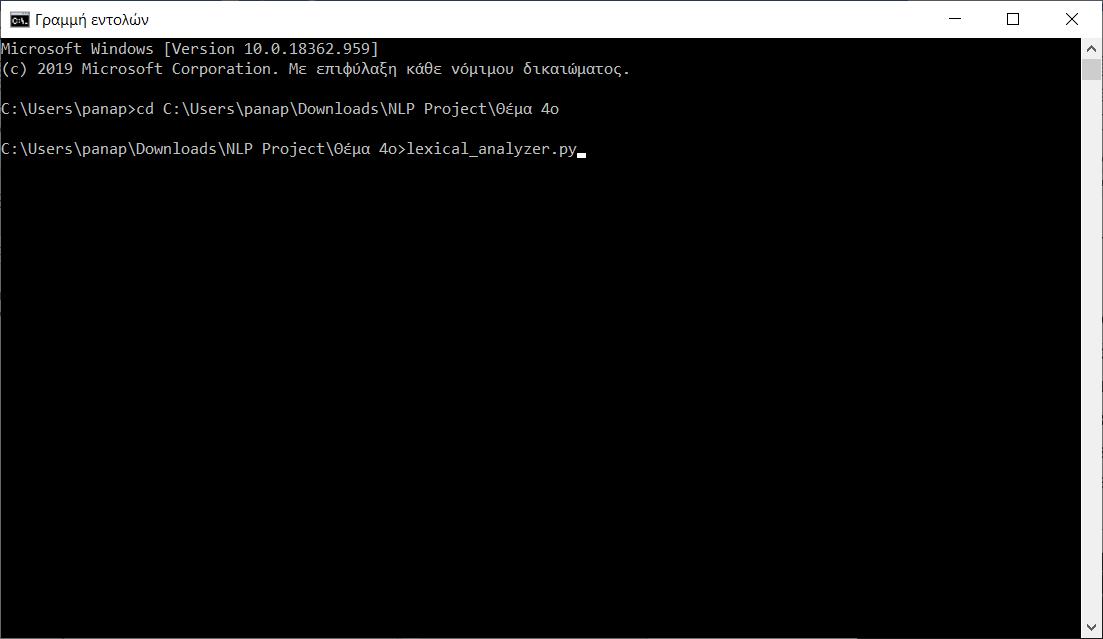
Η επίλυση της άσκησης βρίσκεται στον φάκελο **Θέμα 4** και συγκεκριμένα στα αρχείο **lexical\_analyzer.py**.

Πιο συγκεκριμένα, για να εκτελέσουμε τον κώδικα ακολουθούμε τα παρακάτω βήματα:

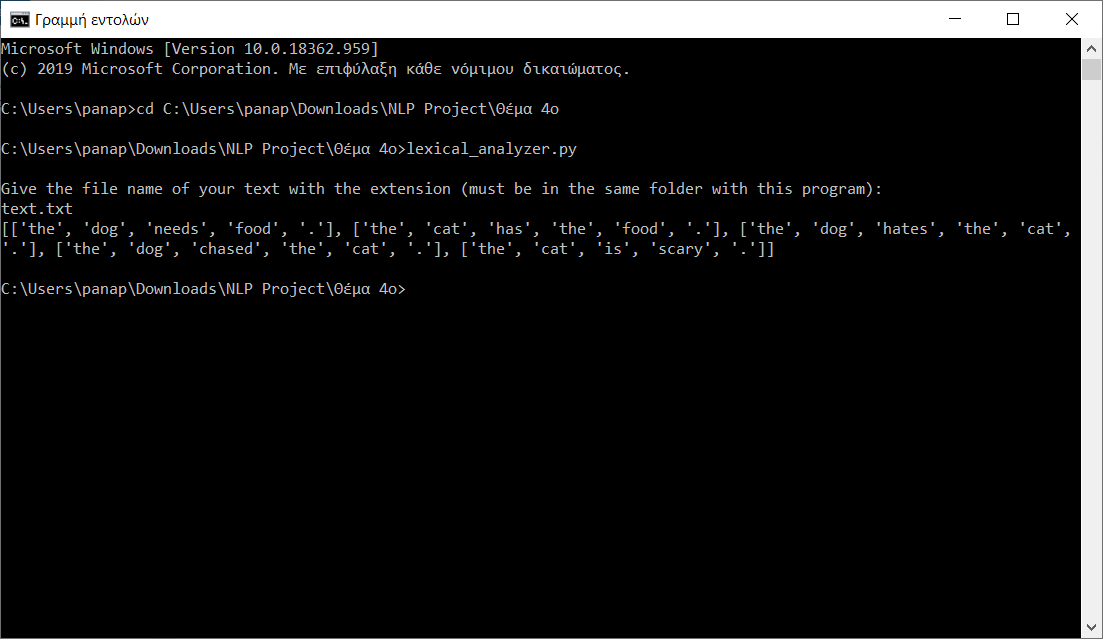
1. Ανοίγουμε τη γραμμή εντολών και μεταβαίνουμε στον αντίστοιχο φάκελο**: \NLP Project \Θέμα 4**.(***Εικόνα 8***)
2. Πληκτρολογούμε το όνομα του αρχείου **lexical\_analyzer.py**, ώστε να ανοίξει το εκτελέσιμο.(***Εικόνα 9***)



Εικόνα 8- Μετάβαση στο φάκελο, στον οποίο βρίσκεται το αρχείο του προγράμματος



Εικόνα 9-Εισαγωγή του ονόματος αρχείου προς εκτέλεση



Εικόνα 11- Εμφάνιση αποτελέσματος

# Θέμα 5ο

# 1. Παρουσίαση κεντρικής ιδέας

Σκοπός αυτής της άσκησης είναι η ανάπτυξη ενός Συντακτικού Αναλυτή σε άλλη γλώσσα προγραμματισμού, ο οποίος με βάση τους συντακτικούς κανόνες της πρότυπης λύσης θα παράγει το συντακτικό δέντρο μιας πρότασης. Η υλοποίηση της συγκεκριμένης άσκησης έγινε στη γλώσσα προγραμματισμού **Python version 3.8.**  Αξίζει να σημειωθεί ότι χρησιμοποιήθηκε η βιβλιοθήκη **nltk**, η οποία αφορά την Επεξεργασία Φυσικής Γλώσσας. Έτσι στη συγκεκριμένη εφαρμογή ανάλογα την πρόταση που έχουμε στον κώδικά μας θα εμφανίζεται το αντίστοιχο συντακτικό δέντρο, το οποίο θα είναι σε μορφή **functor.**

# 2. Επεξήγηση Κώδικα

# 2.1 Συντακτικοί κανόνες (Grammar)

Όπως αναφέρθηκε και παραπάνω το πρόγραμμα έχει υλοποιηθεί σε **Python.**  Απαραίτητο βήμα για την δημιουργία ενός συντακτικού δέντρου είναι ο καθορισμός των συντακτικών κανόνων[[4]](#footnote-4) (γραμματική). Αρχικά, θα πρέπει η κάθε πρόταση να αναλύεται σε επιμέρους στοιχεία. Στη συγκεκριμένη γραμματική η πρόταση αναλύεται σε **noun\_phrase** και **verb\_phrase.** Το κάθε στοιχείο-φράση αναλύεται περαιτέρω. Παρακάτω ακολουθούν πίνακες με περισσότερες πληροφορίες.

|  |  |  |  |
| --- | --- | --- | --- |
| Σύμβολο | Σημασία | Παράδειγμα | Αναλύεται σε: |
| S | Sentence | *the man walked* | NP VP |
| NP | Noun Phrase | *a dog* | PN | Det N |N |
| VP | Verb Phrase | *saw a park* | IV | IV Adv | AV Adj |  TV PN NP | V NP |
| PN | Proper Nouns |  | mary, john, tomy |
| N | Noun | *food* | food, cat, cats, dog, dogs,  book, books, feather, feathers  baby, babies, boy, boys, girl,  girls, icecream, icecreams |
| Det | Determiner | *the* | the, a , an |
| AV | Auxiliary Verbs | *is* | is, does, do, are |
| IV | Intransitive Verbs | *runs* | runs, run, running, hurts, hurt,  hurting, walks, walk, walking,  jumps, jump, jumping, shoots,  shoot, shooting |
| TV | Transitive Verbs | *gives* | gives, give, gave, giving |
| Adv | Adverb | *quickly* | quickly, slowly, independently |
| Adj | Adjective | *scary* | scary, tall, short, blonde, slim,  fat |

# 2.2 Καθορισμός πρότασης

Το πρόγραμμα για να μπορέσει να δημιουργήσει το συντακτικό ΄δέντρο θα πρέπει να γνωρίζει για ποια πρόταση θα εργαστεί. Συνεπώς, θα εκχωρήσουμε στην μεταβλητή ‘***sentence***’ την πρότασή μας. Η συγκεκριμένη πρόταση θα χωριστεί-split σε μία λίστα με το όνομα **sent.** Όπως θα δούμε και παρακάτω αυτή η λίστα θα μας χρειαστεί στην παραγωγή του συντακτικού δέντρου.

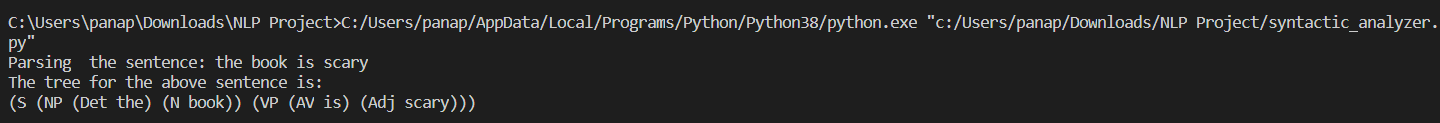
# 2.3 Ορισμός του parser και δημιουργία συντακτικού δέντρου

Προκειμένου να ολοκληρωθεί η διαδικασία επιτυχώς θα πρέπει να ορίσουμε τον parser, δηλαδή να του «εισάγουμε» την γραμματική , σύμφωνα με την οποία θα λειτουργεί. Η python και η βιβλιοθήκη **nltk,** μας δίνουν τη δυνατότητα να φτιάξουμε έναν απλό ***simple-down CFG parser.*** Πιο συγκεκριμένα, η κατάλληλη συνάρτηση για αυτό είναι η **RecursiveDescentParser(),** η οποία δέχεται σαν είσοδο τη γραμματική που δημιουργήσαμε παραπάνω. Το τελικό αποτέλεσμα, θα «σημιουρφηθεί» μετά την εκτέλεση ενός for-loop. Αυτό που εκτελεί το συκγκετιμενο είναι για κάθε λέξη που υπάρχει στην λίστα **sent**(βλ. ενότητα 2.2) να την αντιστοιχίζει με κάποιο κανόνα της γραμματικής. Η διαδικασία αυτή είναι και αναδρομική.

# 3. Αποτελέσματα εκτέλεσης

Στην παρακάτω εικόνα βλέπουμε το αποτέλεσμα της εκτέλεσης του κώδικα για την πρόταση:

|  |
| --- |
| “the book is scary” |



Εικόνα 12-Αποτέλσμα εκτέλεσης προγράμματος για την παραπάνω πρόταση

# 4. Απαιτήσεις συστήματος και εκτέλεση

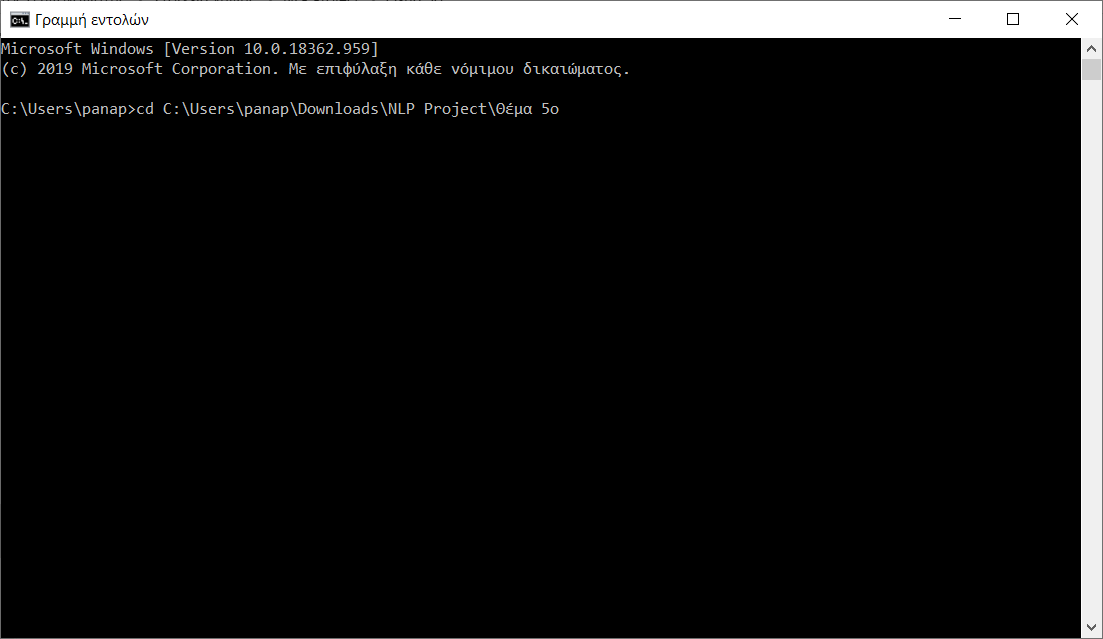
Η παραπάνω άσκηση υλοποιήθηκε στην γλώσσα προγραμματισμού ***Python, version 3.8****.* Για να μπορεί να γίνει η εκτέλεση του προγράμματος είναι απαραίτητο να είναι εγκατεστημένες οι βιβλιοθήκες:

* **nltk**

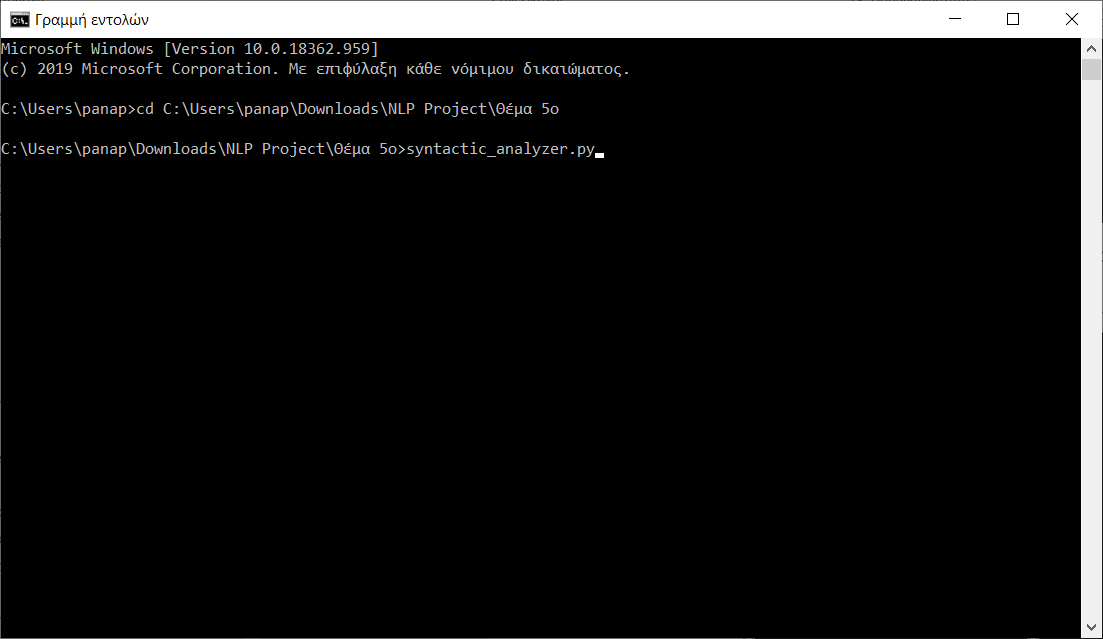
Η επίλυση της άσκησης βρίσκεται στον φάκελο **Θέμα 5** και συγκεκριμένα στα αρχείο **syntactic\_analyzer.py**.

Πιο συγκεκριμένα, για να εκτελέσουμε τον κώδικα ακολουθούμε τα παρακάτω βήματα:

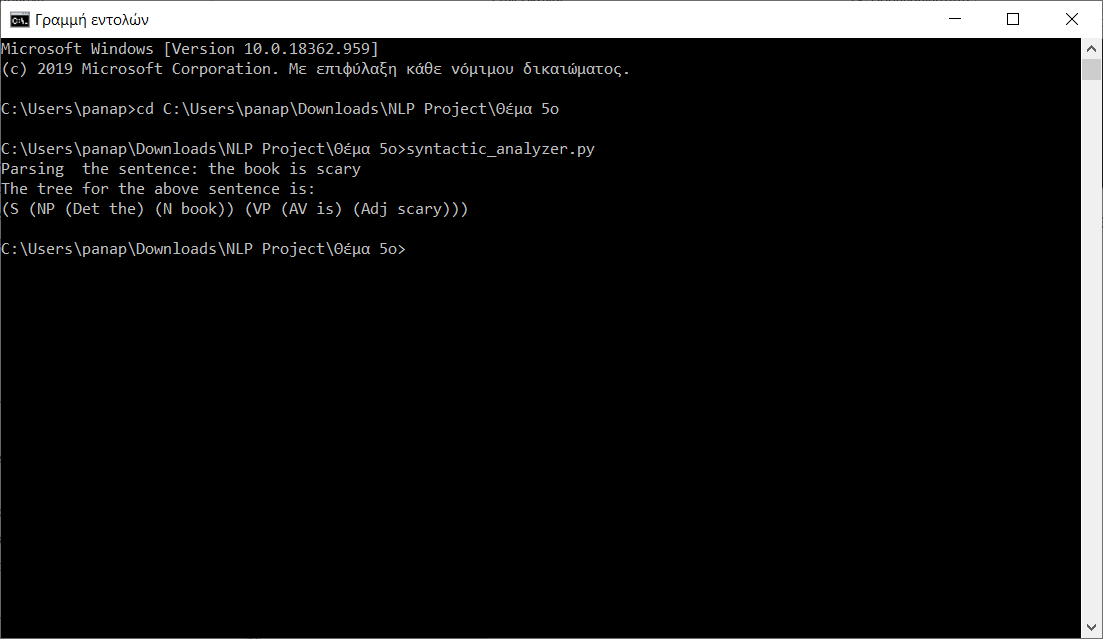
1. Ανοίγουμε τη γραμμή εντολών και μεταβαίνουμε στον αντίστοιχο φάκελο**: \NLP Project \Θέμα 5**.(***Εικόνα 8***)
2. Πληκτρολογούμε το όνομα του αρχείου **syntactic\_analyzer.py**, ώστε να ανοίξει το εκτελέσιμο.(***Εικόνα 9***)



Εικόνα 14- Μετάβαση στο φάκελο, στον οποίο βρίσκεται το αρχείο του προγράμματος



Εικόνα 15-Εισαγωγή του ονόματος αρχείου προς εκτέλεση



Εικόνα 17- Εμφάνιση αποτελέσματος

1. Να σημειωθεί ότι η αναπαράσταση της κάθε λέξης έχει περιεγράφηκε στην ενότητα 2.2. [↑](#footnote-ref-1)
2. H συγκεκριμένη γραμματική υπάρχει και στο αρχείο ‘DFG\_2.pl’ στον φάκελο **Θέμα 2ο.** [↑](#footnote-ref-2)
3. Η συγκεκριμένη γραμματική υπάρχει και στο αρχείο ‘syntax\_generator’ στον φάκελο **Θέμα 2ο.** [↑](#footnote-ref-3)
4. Να σημειωθεί ότι οι συντακτικοί κανόνες είναι ίδιοι με αυτούς της πρότυπης λύσης. [↑](#footnote-ref-4)