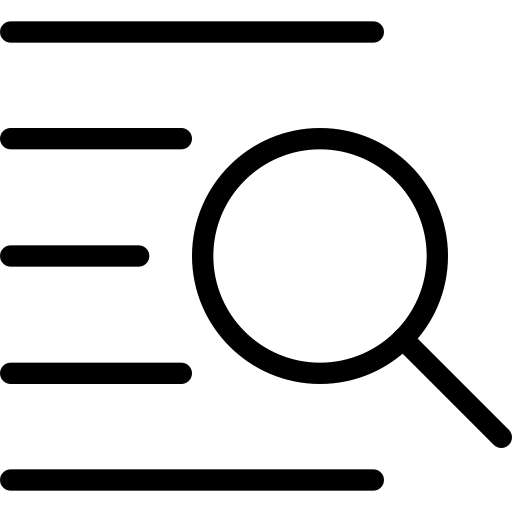


Γλωσσική

Τεχνολογία

**Project**

2021 - 2022



**Στυλιανός Στυλιανάκης**  
**1059713**

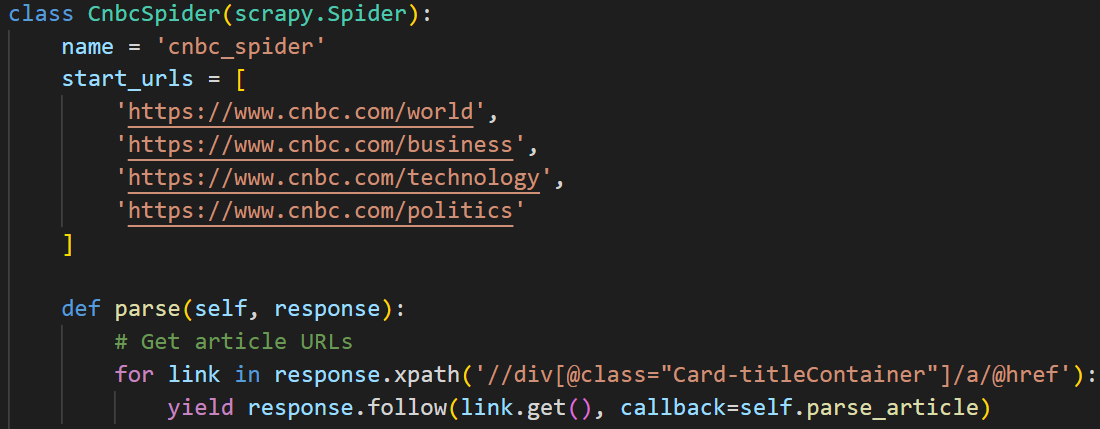
[**Link to code**](https://github.com/Steliostyl/Language_Technology_Project-2021-2022)

## ΜΕΡΟΣ Α

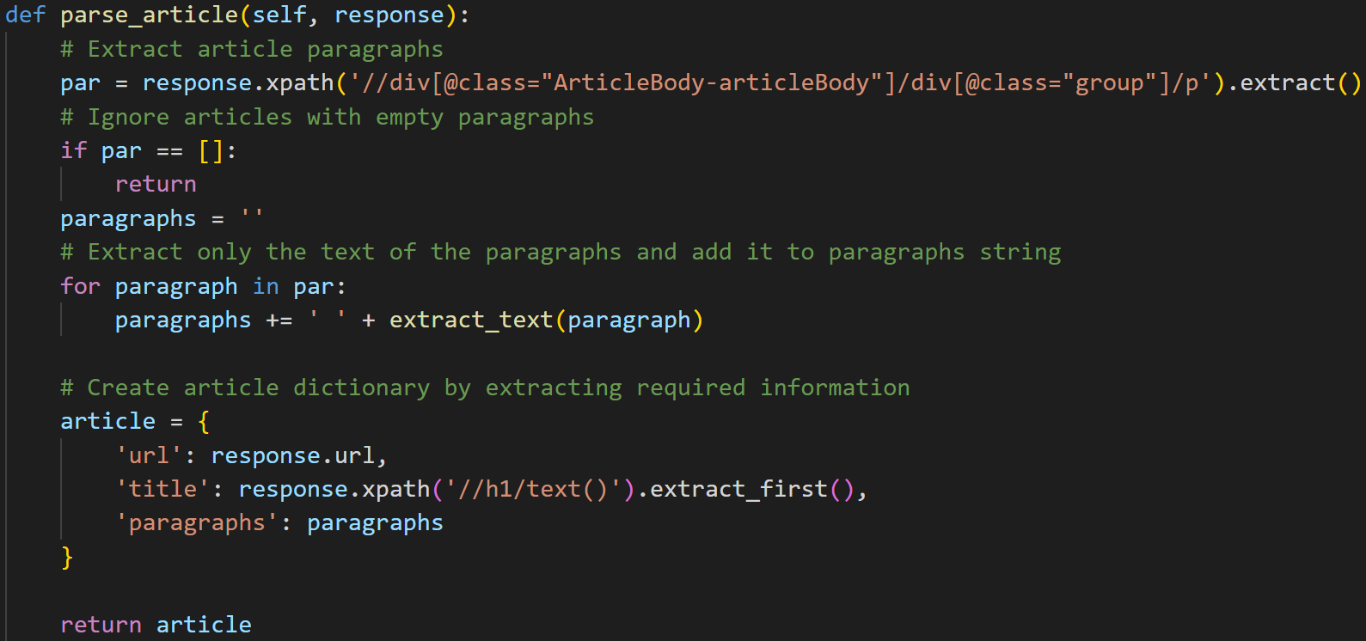
### Προσκομιστής Ιστοσελίδων & Προεπεξεργασία Δεδομένων

Mε τη χρήση του **Scrapy** δημιουργήθηκε ο Crawler, με τη βοήθεια του οποίου ανακτώνται άρθρα από τις ειδησεογραφικές ιστοσελίδες *CNN* και *CNBC.* Πιο συγκεκριμένα, μέσα στο αρχείο **spiders.py** ορίστηκαν δύο **spiders**, το **cnbc\_spider** και το **cnn\_spider,** τα οποία είναι υπεύθυνα για την εξαγωγή άρθρων από τις αντίστοιχες ιστοσελίδες.

Ξεκινώντας από τα προκαθορισμένα **start\_urls**, ο crawler ακολουθεί όλους τους συνδέσμους σε άρθρα που βρίσκονται στην διαδρομή του xpath που έχει οριστεί μέσα στη συνάρτηση **parse**, όπως φαίνεται παρακάτω.



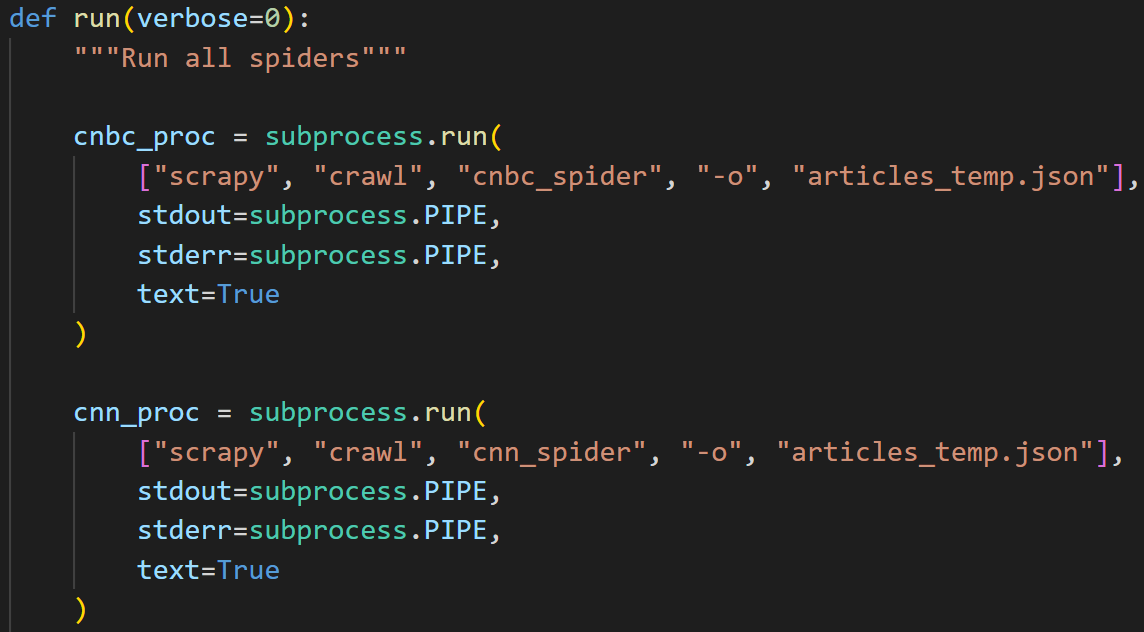
Για κάθε έναν από τους συνδέσμους αυτούς τους που ακολουθεί καλείται η **callback function** parse\_article ώστε να εξαχθούν οι ζητούμενες πληροφορίες.



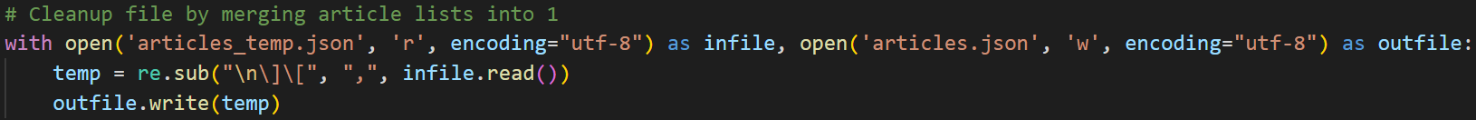
Συγκεκριμένα για την εξαγωγή κειμένου από τις παραγράφους του άρθρου δημιουργείται ένα άδειο string **paragraphs**, καθώς και μία λίστα με όλες τις παραγράφους του άρθρου. Στη συνέχεια, αυτή η λίστα διατρέχεται και από κάθε παράγραφο εξάγεται το καθαρό κείμενο, το οποίο προστίθεται στο string **paragraphs.**

Αντίστοιχα έχει δημιουργηθεί και το **cnn\_spider** με αλλαγές φυσικά στις διαδρομές xpath και άλλες μικρές αλλαγές.

Τα spiders αυτά τρέχουν μέσα από τη συνάρτηση **run** του αρχείου **run\_Spiders.py**. Στο αρχείο αυτό δημιουργείται ένα subprocess για το κάθε spider και τρέχει ουσιαστικά το shell command “scrapy crawl *spider\_name* -o articles\_temp.json", δηλαδή κάνει crawl το κάθε spider και αποθηκεύει την έξοδο του στο αρχείο *articles\_temp.json*.



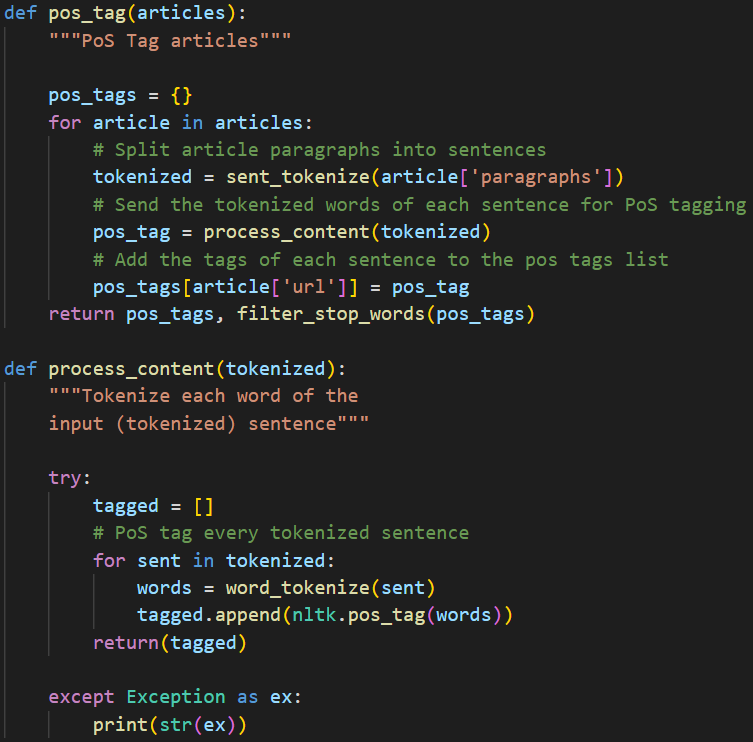
Στη συνέχεια, επειδή το περιεχόμενο του αρχείου περιέχει **δύο λίστες** με άρθρα (μία από κάθε διεργασία), αντιγράφεται το αρχείο χωρίς τα σημεία «ένωσης» των λιστών. Χρησιμοποιώντας τη συνάρτηση re.sub("\n\]\[", ",", infile.read()), αντικαθίσταται η σειρά χαρακτήρων “\n][“ με ένα κόμμα, ενώνοντας έτσι τις λίστες σε **μία** και στη συνέχεια αποθηκεύεται στο καινούριο αρχείο *articles.json*.



### Μορφοσυντακτική Ανάλυση

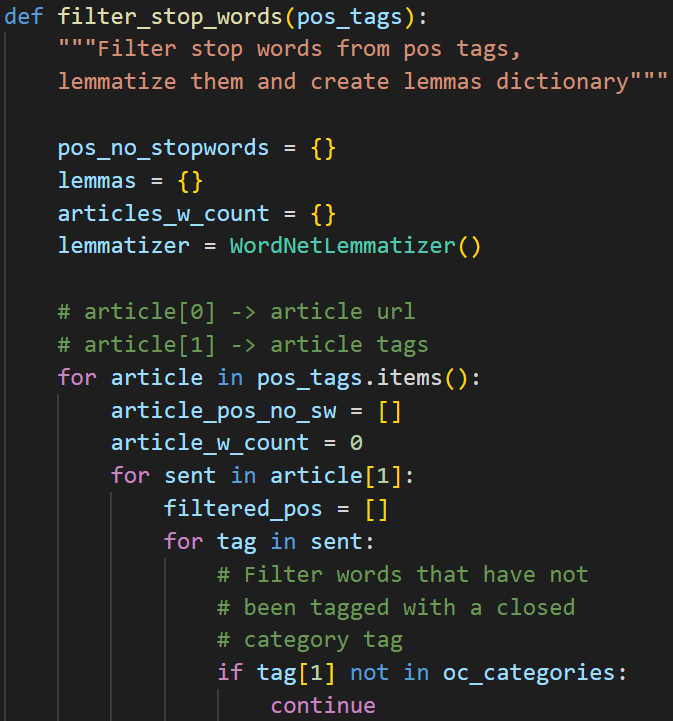
Μετά που θα κατέβουν τα άρθρα και θα δημιουργηθεί το αρχείο *articles.json*, διαβάζονται και φορτώνονται σε ένα **dictionary** μέσω της συνάρτησης readJSON, ώστε να υποστούν στη συνέχεια επεξεργασία. Για τον **PoSTagger** χρησιμοποιήθηκε έτοιμη υλοποίηση της βιβλιοθήκης **nltk**, ενώ οι συναρτήσεις που χρησιμοποιούν τη βιβλιοθήκη αυτή βρίσκονται όλες στο αρχείο *nltk\_functions.py*.

Παρακάτω φαίνεται πώς γίνονται PoSTag τα άρθρα (χωρίς να αφαιρεθούν τα stop words).



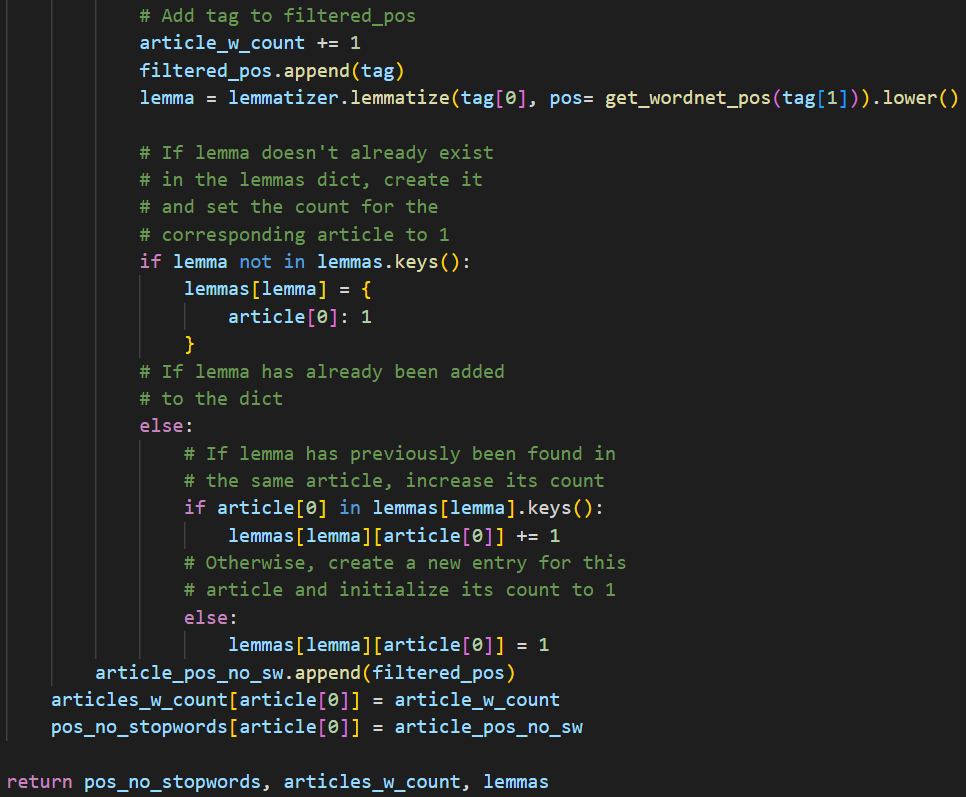
### Αναπαράσταση ιστοσελίδων στο Μοντέλο Διανυσματικού Χώρου.

Μόλις τελειώσει η διαδικασία του tagging, καλείται η συνάρτηση filter\_stop\_words, η οποία παίρνει σαν όρισμα τα **tags** που έχουν προκύψει από την προηγούμενη συνάρτηση, και από αυτά κρατάει μόνο εκείνα που **δεν είναι stop words**. Στη συνάρτηση αυτή γίνεται και η αρχικοποίηση, αλλά και η συμπλήρωση του λεξικού των **λημμάτων**.

Εδώ να σημειωθεί ότι θα ήταν πιο αποδοτικό αυτό το φιλτράρισμα να γινόταν κατά τη διάρκεια του αρχικού tagging, αλλά έγινε διαχωρισμός των δύο λειτουργιών επειδή στην εκφώνηση του πρότζεκτ παρουσιάζονται ως διαφορετικά υποσυστήματα.

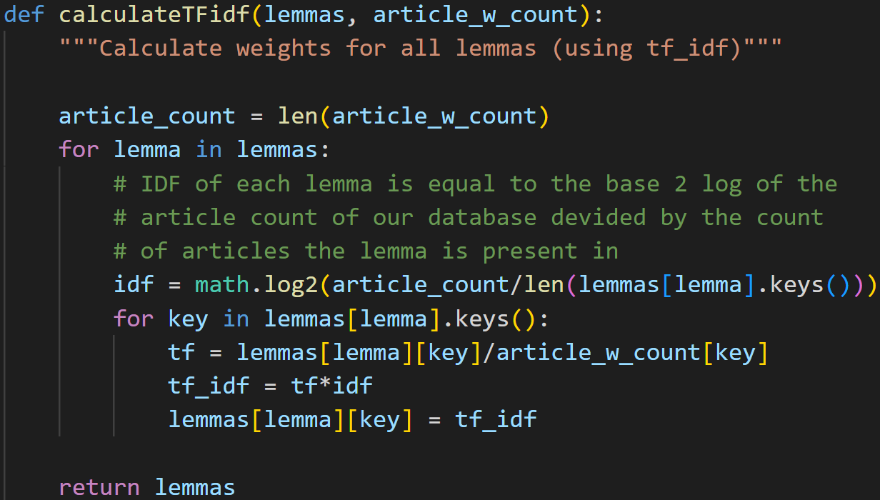
Έπειτα από το **φιλτράρισμα** των stop words, τα εναπομείναντα tags προστίθενται στη λίστα με τα φιλτραρισμένα tags και μετά εξάγουμε το **λήμμα** της λέξης. Ταυτόχρονα, ελέγχουμε και τη **συχνότητα εμφάνισης** του λήμματος **στο άρθρο** μας (με τους κατάλληλους ελέγχους και ενέργειες αρχικοποίησης), καθώς και τον **αριθμό λέξεων** του κάθε **άρθρου**.

Τέλος, από τη συνάρτηση επιστρέφονται τα tags χωρίς τα stop words, το λεξικό του πλήθους λέξεων των άρθρων, αλλά και το λεξικό των λημμάτων.



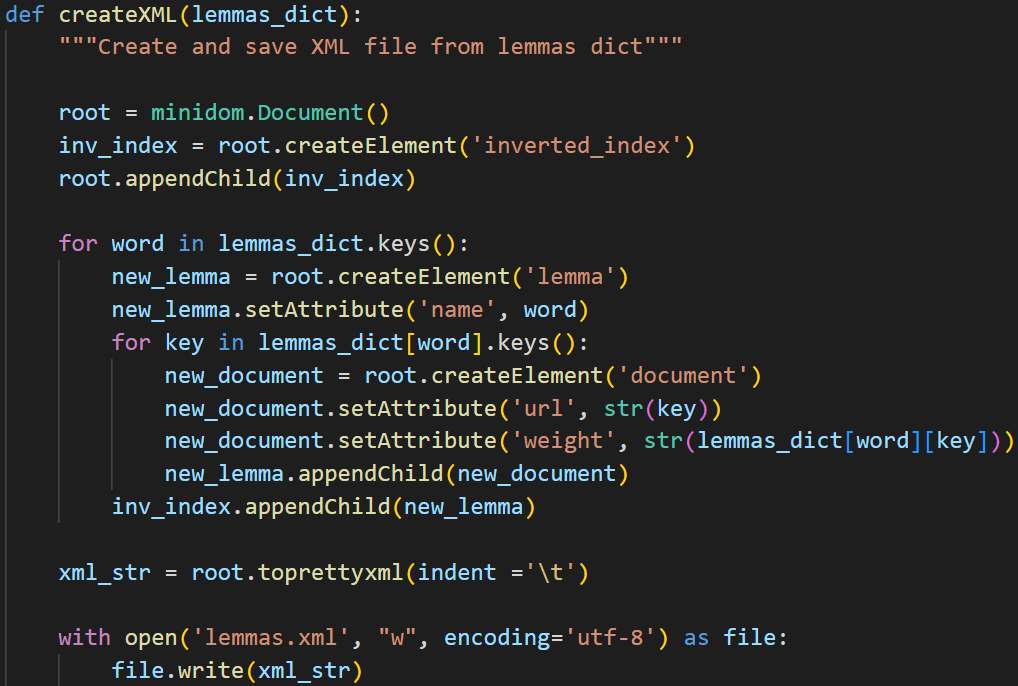
### Δημιουργία του ευρετηρίου

Σε αυτή τη φάση έχει ουσιαστικά φτιαχτεί ένα σημαντικό **μέρος του ευρετηρίου**, απλώς αντί για το βάρος του λήμματος στα αντίστοιχα άρθρα έχουμε προς το παρόν μόνο τη **συχνότητα εμφάνισής** τους σε αυτά. Για τη **μετατροπή** από συχνότητα εμφάνισης σε βάρη υπάρχει η συνάρτηση calculateTFidf η οποία παίρνει σαν είσοδο το λεξικό των λημμάτων και το επιστρέφει μετά από τη μετατροπή. Η διαδικασία αυτή φαίνεται στο παρακάτω snapshot του κώδικα:



### Αποθήκευση και επαναφόρτωση ευρετηρίου

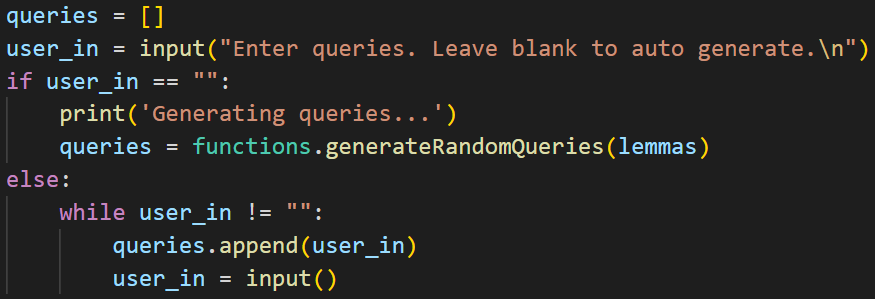
Η **αποθήκευση** του ευρετηρίου γίνεται με τη συνάρτηση createXML του αρχείου *functions.py*, η οποία παρουσιάζεται παρακάτω



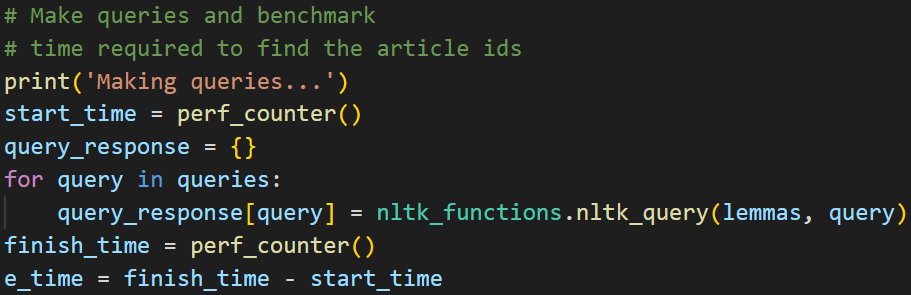
Αντίστοιχα, η **επαναφόρτωση** του ευρετηρίου από το αρχείο γίνεται με τη μέθοδο readXML του ίδιου αρχείου.

### Αξιολόγηση Ευρετηρίου

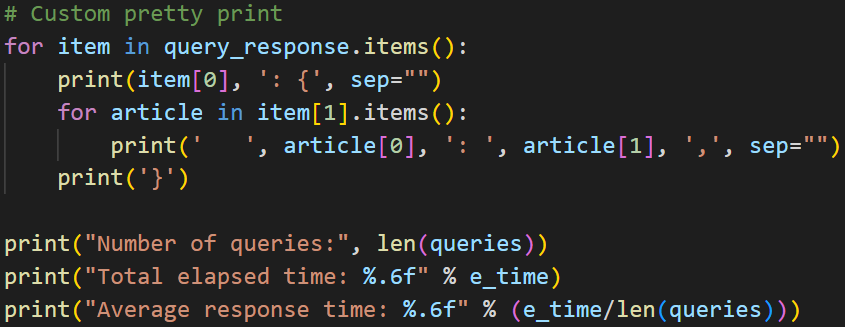
Ο **χρήστης** έχει την επιλογή να **εισάγει** ο ίδιος ερωτήματα ή να **δημιουργηθούν** αυτόματα, όπως φαίνεται από τον παρακάτω κώδικα:



Στη συνέχεια, τα ερωτήματα υποβάλλονται και το σύστημα αξιολογείται ως προς το **χρόνο απόκρισης** του. Για αυτόν το σκοπό χρησιμοποιήθηκε η βιβλιοθήκη time και συγκεκριμένα η συνάρτηση perf\_counter. Ακριβώς πριν την κλήση της συνάρτησης υποβολής ερωτημάτων, αποθηκεύεται ο **χρόνος έναρξης** στη μεταβλητή start\_time και αντίστοιχα μόλις απαντηθούν όλα τα ερωτήματα στη μεταβλητή finish\_time. Είναι εμφανές ότι η διαφορά τους είναι και ο **συνολικός χρόνος απόκρισης** του ευρετηρίου μας. Για να βρούμε τον μέσο χρόνο απόκρισης, διαιρούμε τον συνολικό χρόνο με το πλήθος των ερωτημάτων που υποβλήθηκαν.

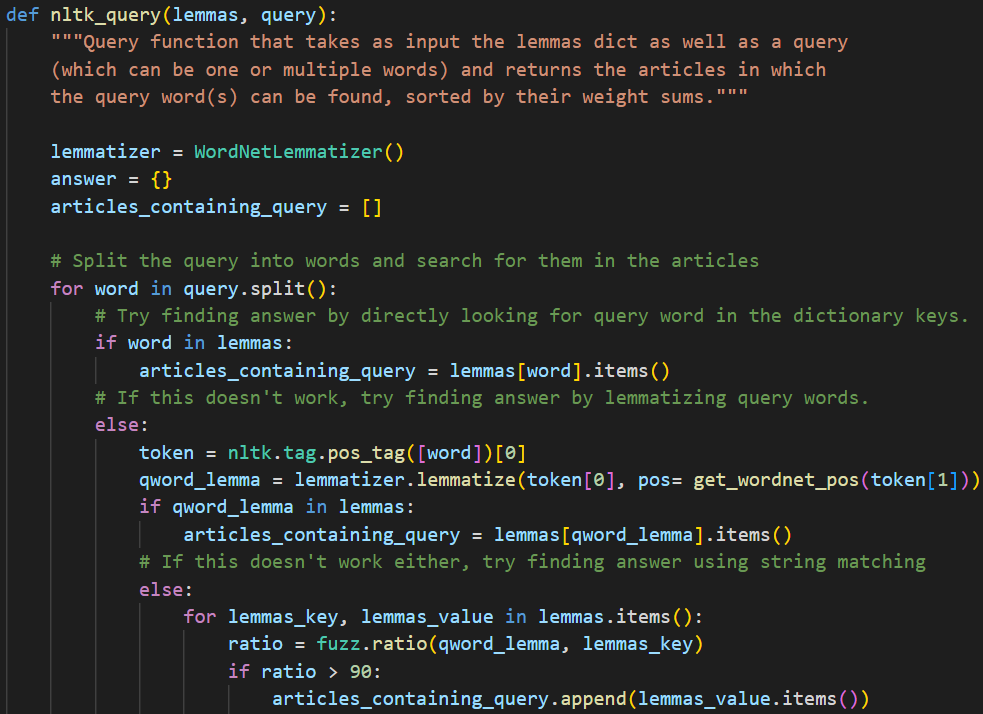


Τέλος, εκτυπώνονται τα ερωτήματα με τις απαντήσεις τους, καθώς και οι μετρικές απόδοσης του ευρετηρίου.

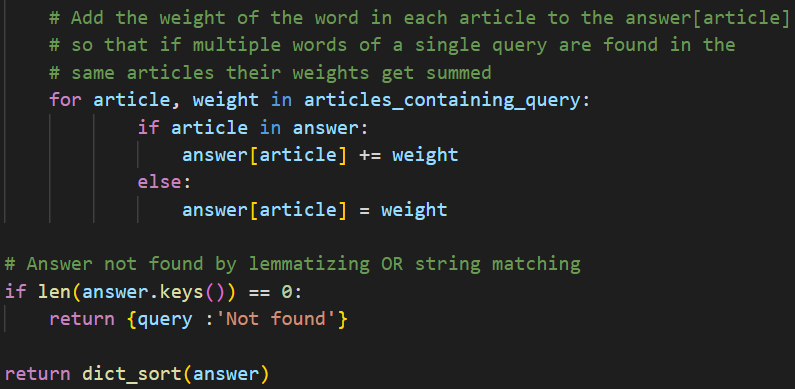


Η συνάρτηση των ερωτημάτων nltk\_query δέχεται ως είσοδο το **λεξικό των λημμάτων**, καθώς και το **ερώτημα** και επιστρέφει τα άρθρα στα οποία βρίσκονται οι λέξεις του ερωτήματος, **ταξινομημένα** ως προς το (συνολικό) βάρος τους.

Αρχικά, «σπάει» το ερώτημα σε **λέξεις** (σε περίπτωση που είναι πολλών λέξεων) και στη συνέχεια τις ψάχνει μία μία σε **τρία στάδια**. Ξεκινάει την αναζήτηση ελέγχοντας αν η λέξη του ερωτήματος υπάρχει **όπως είναι** στο λεξικό των λημμάτων. Σε δεύτερο στάδιο, αν δεν υπάρχει δηλαδή, τη μετατρέπει σε **λήμμα** και έπειτα το αναζητεί στο λεξικό των λημμάτων. Αν ούτε αυτό δουλέψει, ως τελευταία (και πολύ χρονοβόρα) προσπάθεια γίνεται **σύγκριση** (string-matching) **όλων των λημμάτων** με τη λέξη και όποια λήμματα ταιριάζουν αρκετά θεωρούνται ως σωστή απάντηση.



Αφού βρει σε ποια άρθρα περιέχεται η λέξη του ερωτήματος, η συνάρτηση διατρέχει αυτά τα άρθρα και δημιουργεί την **τελική απάντηση**. Τέλος, πριν επιστραφεί η απάντηση, γίνεται **ταξινόμηση** των άρθρων με βάση το βάρος τους, μέσω της συνάρτησης dict\_sort.

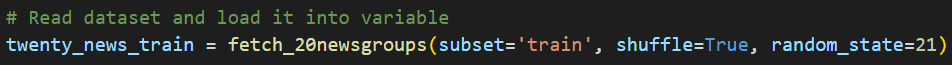


## Μέρος Β

Για το μέρος Β χρησιμοποιήθηκε η βιβλιοθήκη **sklearn**, καθώς διαθέτει υλοποιημένες συναρτήσεις με τις οποίες πραγματοποιήθηκε το κομμάτι αυτό της εργασίας.

### Προ-επεξεργασία των συλλογών Ε και Α

Για την φόρτωση των εγγράφων σε τυχαία σειρά χρησιμοποιήθηκε η παρακάτω συνάρτηση του sklearn.datasets.

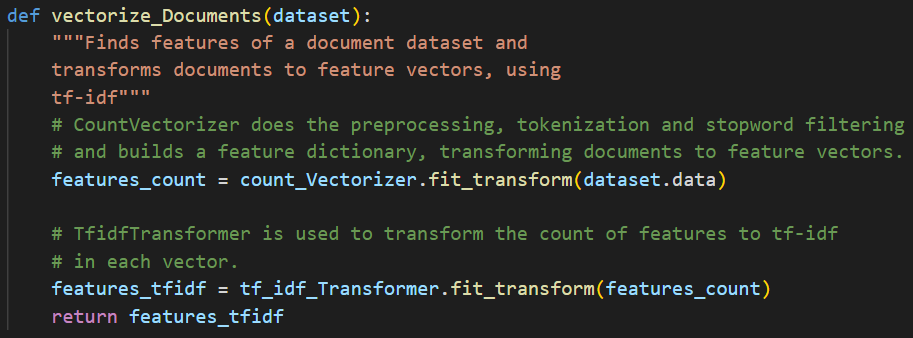


### Δημιουργία χώρου χαρακτηριστικών

Η δημιουργία του χώρου χαρακτηριστικών, καθώς και των διανυσμάτων έγινε με τη συνάρτηση vectorize\_Documents, η οποία παίρνει σαν όρισμα το training dataset που φορτώσαμε.

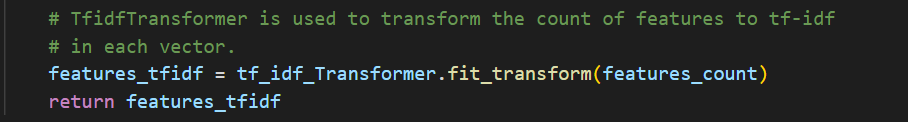


Όπως φαίνεται παρακάτω, χρησιμοποιείται ένας CountVectorizer και πιο συγκεκριμένα η συνάρτηση fit\_transform ώστε να γίνει tokenization και φιλτράρισμα των λέξεων των κειμένων. Στη συνέχεια, επιλέγονται τα χαρακτηριστικά και τέλος μετατρέπονται τα κείμενα σε διανύσματα πλήθους.



### Δημιουργία διανυσμάτων χαρακτηριστικών

Επειδή όμως ως συνήθως το απλό πλήθος δεν είναι χρήσιμη μετρική, οι διαστάσεις των διανυσμάτων μετατρέπονται σε **tf\_idf** αξιοποιώντας έναν TfidfTransformer, όπως φαίνεται παρακάτω.



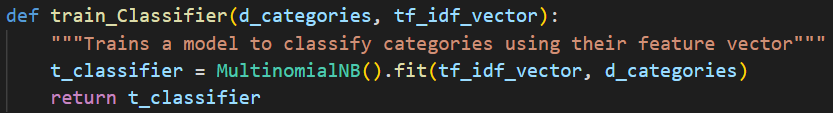
### Σύγκριση διανυσμάτων χαρακτηριστικών

#### Με χρήση Multinomial Classifier

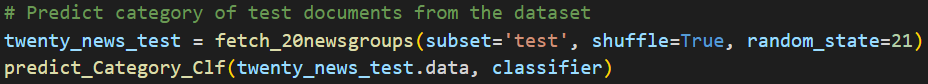
Ένας τρόπος κατηγοριοποίησης είναι να χρησιμοποιήσουμε υλοποιημένο classifier. Πριν τη σύγκριση των διανυσμάτων χαρακτηριστικών από το test dataset, θα πρέπει να φτιάξουμε και να εκπαιδεύσουμε έναν **classifier**.



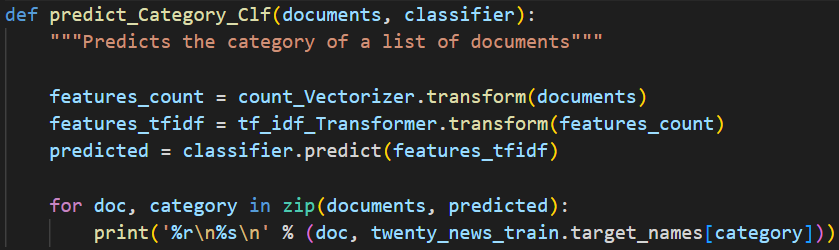
Αυτό επιτυγχάνεται με τη συνάρτηση fit, όπως φαίνεται παρακάτω.



Συγκρίνουμε τα κείμενα μέσω της συνάρτησης predict\_Category\_Clf.



Η συνάρτηση αυτή, δημιουργεί διανύσματα με τα tf\_idf χαρακτηριστικά του κάθε κειμένου και χρησιμοποιεί τον **classifier** που έχουμε εκπαιδεύσει για τη σύγκριση, ώστε να κατηγοριοποιήσει το κάθε κείμενο.



Ο χρόνος που χρειάζεται για να κατηγοριοποιήσει και τα 7532 άρθρα και να εκτυπώσει είναι περίπου 36.5”. Χωρίς την εκτύπωση, ο χρόνος μειώνεται δραματικά στα περίπου 5.3”. Τα πειράματα εκπονήθηκαν σε λάπτοπ με επεξεργαστή Intel Core i7 8550u (15w) με base frequency 1.8GHz.