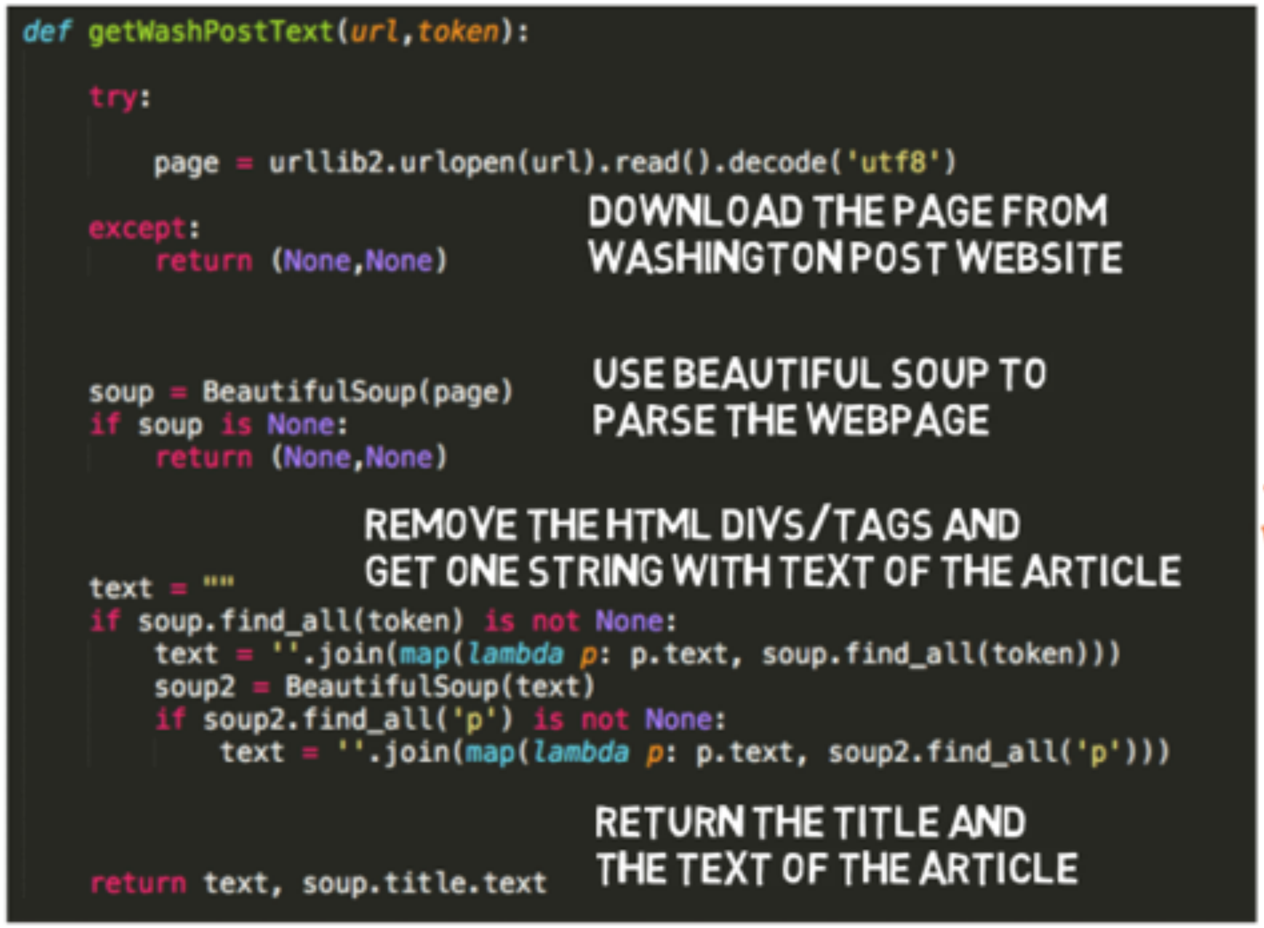
DOWNLOAD AND PARSE AN ARTICLE FROM ITS URL



```
def getWashPostText(url,token):

    try:

        page = urllib2.urlopen(url).read().decode('utf8')

    except:
        return (None,None)


    soup = BeautifulSoup(page)
    if soup is None:
        return (None,None)


    text = ""
    if soup.find_all(token) is not None:
        text = ''.join(map(lambda p: p.text, soup.find_all(token)))
        soup2 = BeautifulSoup(text)
        if soup2.find_all('p') is not None:
            text = ''.join(map(lambda p: p.text, soup2.find_all('p')))


    return text, soup.title.text
```

DOWNLOAD THE PAGE FROM WASHINGTON POST WEBSITE

USE BEAUTIFUL SOUP TO PARSE THE WEBPAGE

REMOVE THE HTML DIVS/TAGS AND GET ONE STRING WITH TEXT OF THE ARTICLE

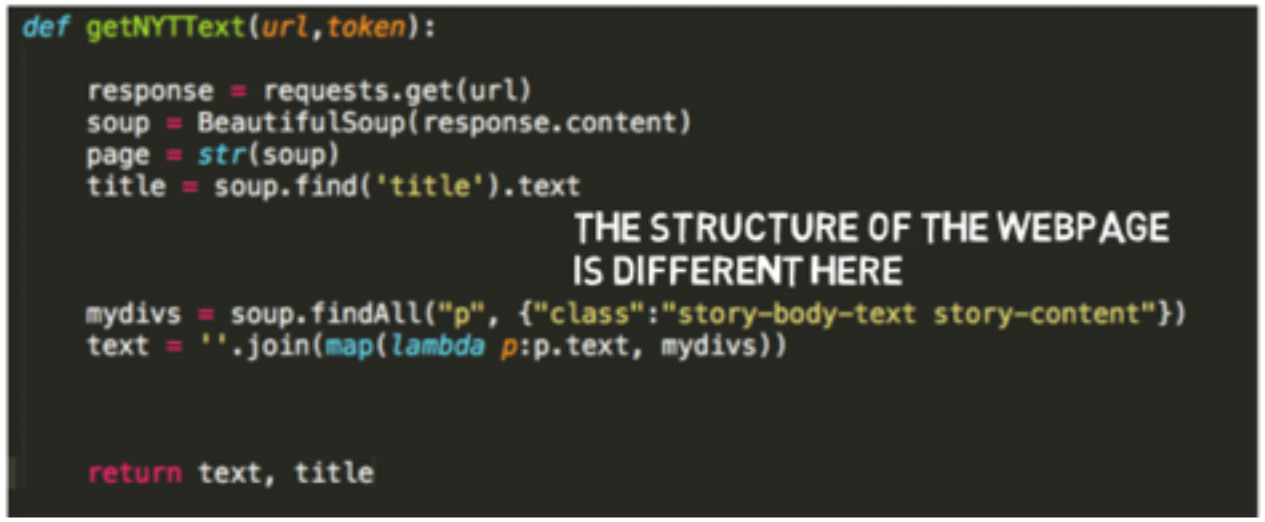
RETURN THE TITLE AND THE TEXT OF THE ARTICLE

THIS FUNCTION TAKES THE URL OF AN ARTICLE IN THE WASHINGTON POST, AND THEN RETURNS THE ARTICLE MINUS ALL OF THE CRUD - HTML, JAVASCRIPT ETC.

THIS WILL ONLY WORK FOR URLS WHERE WE KNOW THE STRUCTURE

(EG SAY ALL ARTICLES IN THE WASHPO ARE ENCLOSED IN <ARTICLE></ARTICLE> TAGS)

HERE IS HOW YOU WOULD DO THE SAME THING FOR THE NEW YORK TIMES



```
def getNYTText(url,token):

    response = requests.get(url)
    soup = BeautifulSoup(response.content)
    page = str(soup)
    title = soup.find('title').text


    mydivs = soup.findAll("p", {"class":"story-body-text story-content"})
    text = ''.join(map(lambda p:p.text, mydivs))



    return text, title
```

THE STRUCTURE OF THE WEBPAGE IS DIFFERENT HERE

WE'LL USE THESE INSIDE ANOTHER FUNCTION THAT TAKES IN THE URL OF AN ENTIRE SECTION OF A NEWSPAPER AND PARSES ALL OF THE URLS FOR ARTICLES LINKED OFF THAT SECTION

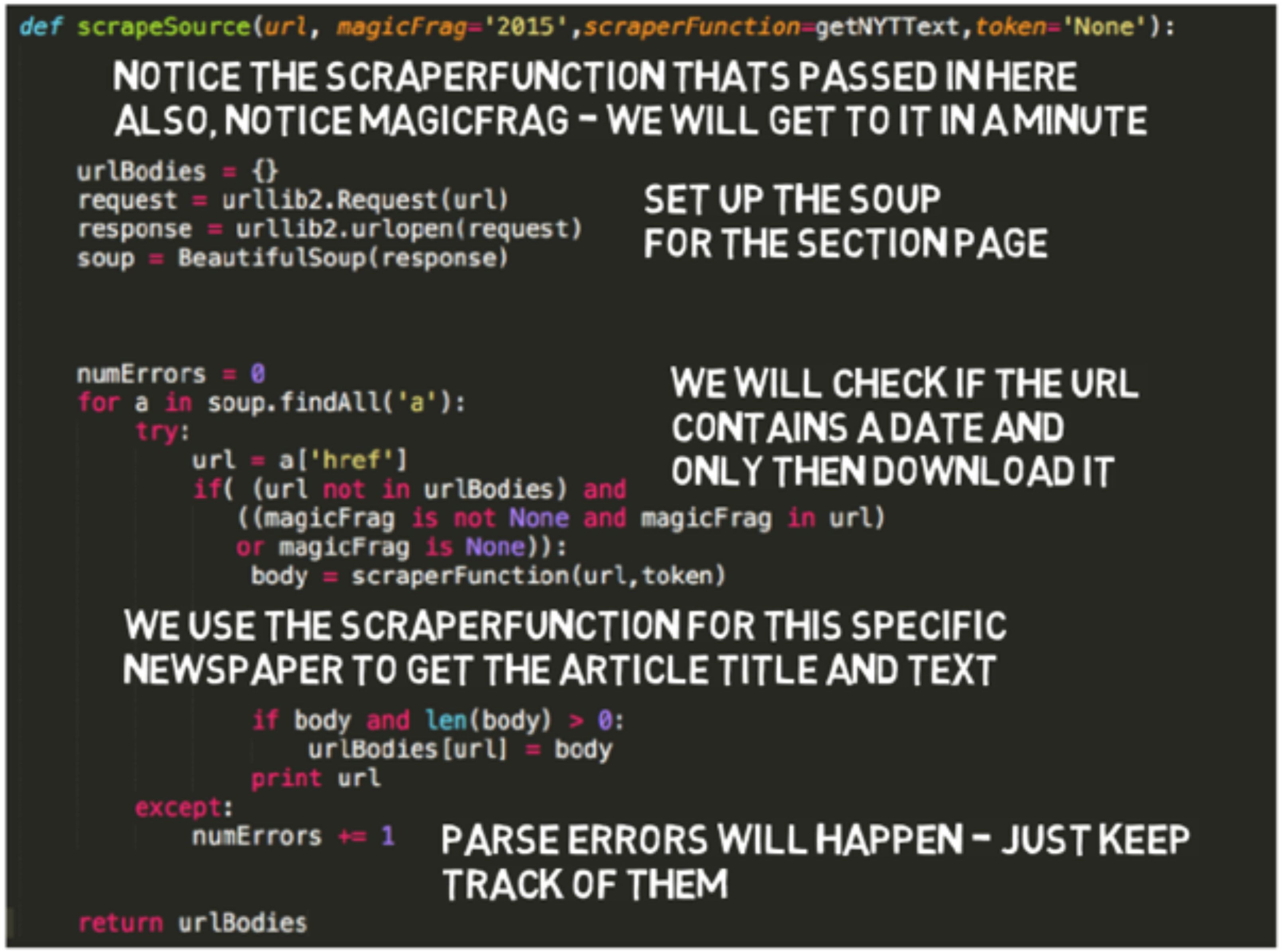
DOWNLOAD ALL ARTICLES IN A SECTION OF A NEWSPAPER

THIS FUNCTION TAKES IN THE URL SCRAPING FUNCTION FROM THE LAST STEP

THESE SECTIONS USUALLY COME WITH A LOT OF NON-NEWS LINKS, SO WE WILL EMPLOY A HACK. WE WILL CONSIDER SOMETHING TO BE A URL ONLY IF IT HAS A DATELINE

THIS FUNCTION RETURNS A DICTIONARY THE URLS AS KEYS AND THE CORRESPONDING ARTICLE TITLE, TEXT AS VALUES

WE'LL USE THIS TO DOWNLOAD ALL THE ARTICLES FOR SPORTS (NON-TECH) AND TECH NEWS SECTIONS OF BOTH WASHINGTON POST AND NEW YORK TIMES. THIS WILL BE OUR CORPUS FOR TRAINING THE MODEL.



```
def scrapeSource(url, magicFrag='2015',scraperFunction=getNYTText,token='None'):
```

NOTICE THE SCRAPERFUNCTION THATS PASSED IN HERE
ALSO, NOTICE MAGICFRAG - WE WILL GET TO IT IN A MINUTE

```
    urlBodies = {}
    request = urllib2.Request(url)
    response = urllib2.urlopen(request)
    soup = BeautifulSoup(response)
```

SET UP THE SOUP FOR THE SECTION PAGE

```
    numErrors = 0
    for a in soup.findAll('a'):
        try:
            url = a['href']
            if( (url not in urlBodies) and
               ((magicFrag is not None and magicFrag in url)
               or magicFrag is None)):
                body = scraperFunction(url,token)
```

WE WILL CHECK IF THE URL CONTAINS A DATE AND ONLY THEN DOWNLOAD IT

WE USE THE SCRAPERFUNCTION FOR THIS SPECIFIC NEWSPAPER TO GET THE ARTICLE TITLE AND TEXT

```
                if body and len(body) > 0:
                    urlBodies[url] = body
                print url
        except:
            numErrors += 1
    return urlBodies
```

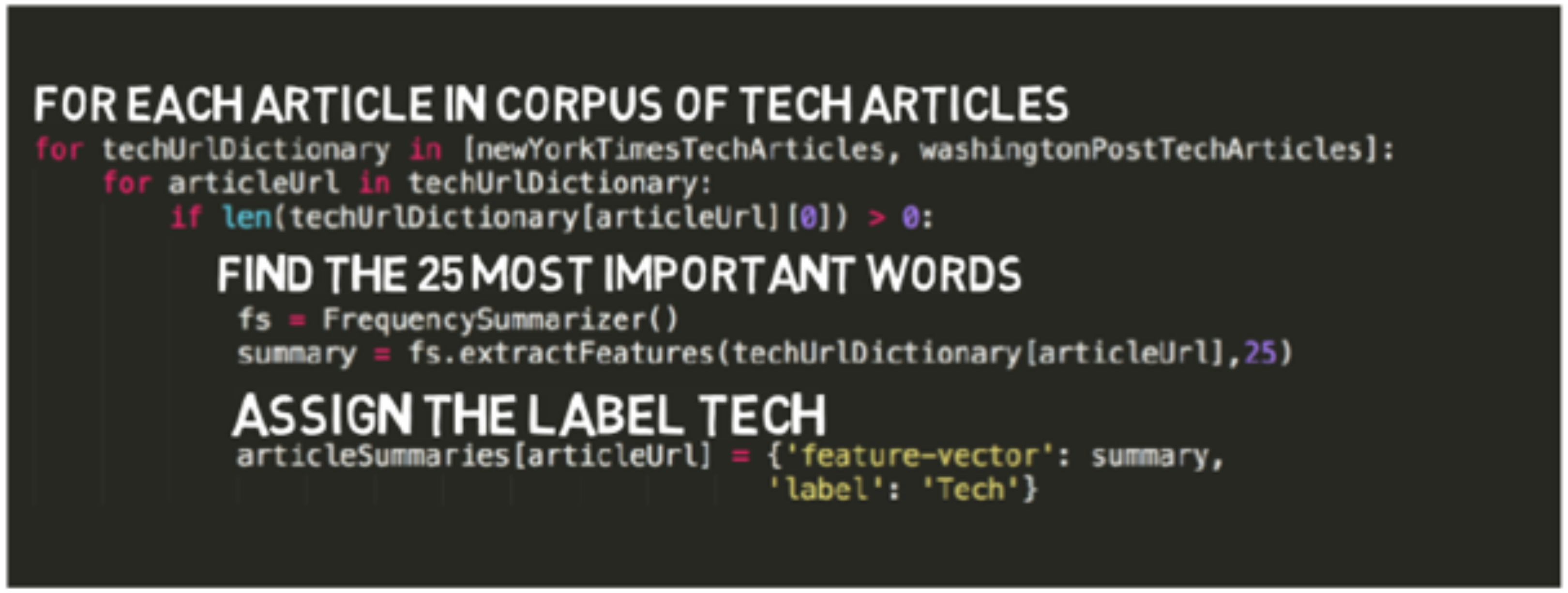
PARSE ERRORS WILL HAPPEN - JUST KEEP TRACK OF THEM

SET UP THE TRAINING DATASET

AS THIS IS A SUPERVISED LEARNING APPROACH WE NEED TO SET UP THE TRAINING DATA THAT THE ALGORITHM WILL 'LEARN' FROM

THE TRAINING DATA IS SET UP AS TUPLES - ARTICLES AND THE CORRESPONDING LABELS (TECH/NON-TECH)

EACH ARTICLE IS REPRESENTED BY THE LIST OF MOST IMPORTANT WORDS IN THE ARTICLE (FEATURE VECTOR)



FOR EACH ARTICLE IN CORPUS OF TECH ARTICLES

```
for techUrlDictionary in [newYorkTimesTechArticles, washingtonPostTechArticles]:
    for articleUrl in techUrlDictionary:
        if len(techUrlDictionary[articleUrl][0]) > 0:
```

FIND THE 25 MOST IMPORTANT WORDS

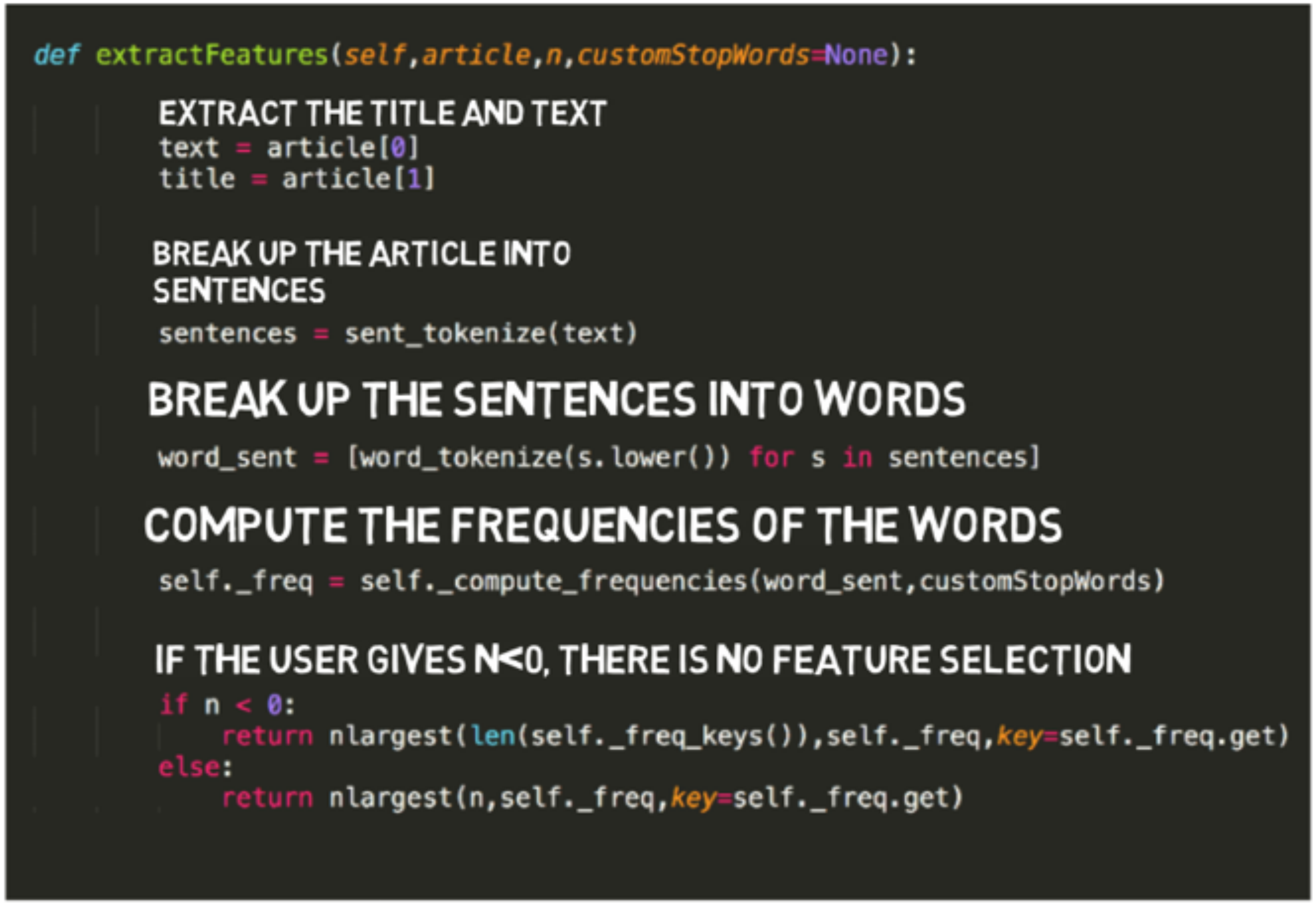
```
            fs = FrequencySummarizer()
            summary = fs.extractFeatures(techUrlDictionary[articleUrl],25)
```

ASSIGN THE LABEL TECH

```
            articleSummaries[articleUrl] = {'feature-vector': summary,
                                            'label': 'Tech'}
```

DO THIS FOR BOTH TECH AND NON-TECH ARTICLES

EXTRACT FEATURES FROM AN ARTICLE



```
def extractFeatures(self,article,n,customStopWords=None):
```

EXTRACT THE TITLE AND TEXT

```
        text = article[0]
        title = article[1]
```

BREAK UP THE ARTICLE INTO SENTENCES

```
        sentences = sent_tokenize(text)
```

BREAK UP THE SENTENCES INTO WORDS

```
        word_sent = [word_tokenize(s.lower()) for s in sentences]
```

COMPUTE THE FREQUENCIES OF THE WORDS

```
        self._freq = self._compute_frequencies(word_sent,customStopWords)
```

IF THE USER GIVES N<0, THERE IS NO FEATURE SELECTION

```
        if n < 0:
            return nlargest(len(self._freq_keys()),self._freq,key=self._freq.get)
        else:
            return nlargest(n,self._freq,key=self._freq.get)
```

THIS FUNCTION FINDS THE N MOST IMPORTANT WORDS IN AN ARTICLE

THIS IS FEATURE SELECTION - INSTEAD OF USING ALL THE WORDS, WE ARE USING THE MOST IMPORTANT WORDS AS FEATURES

IMPORTANCE IS COMPUTED AS THE FREQUENCY OF THE WORD IN THE ARTICLE