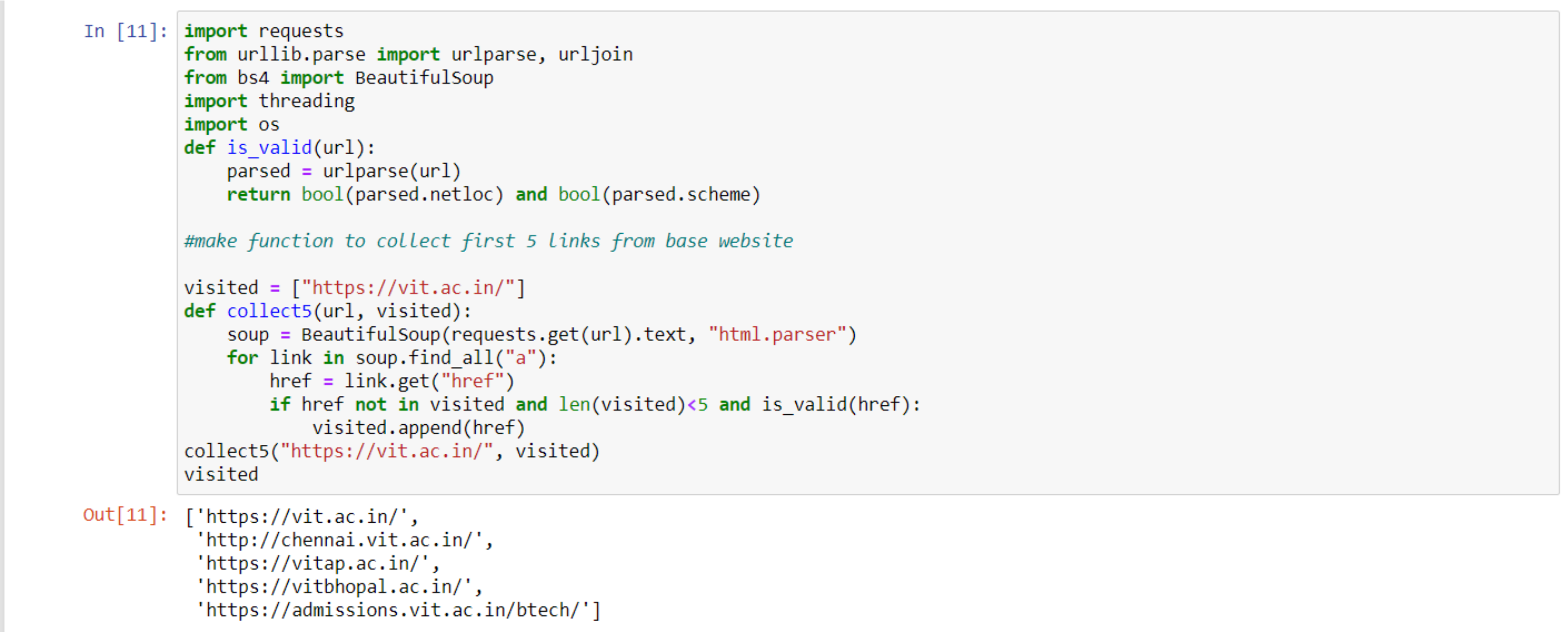
Digital Assignment 2

Devang Mehrotra

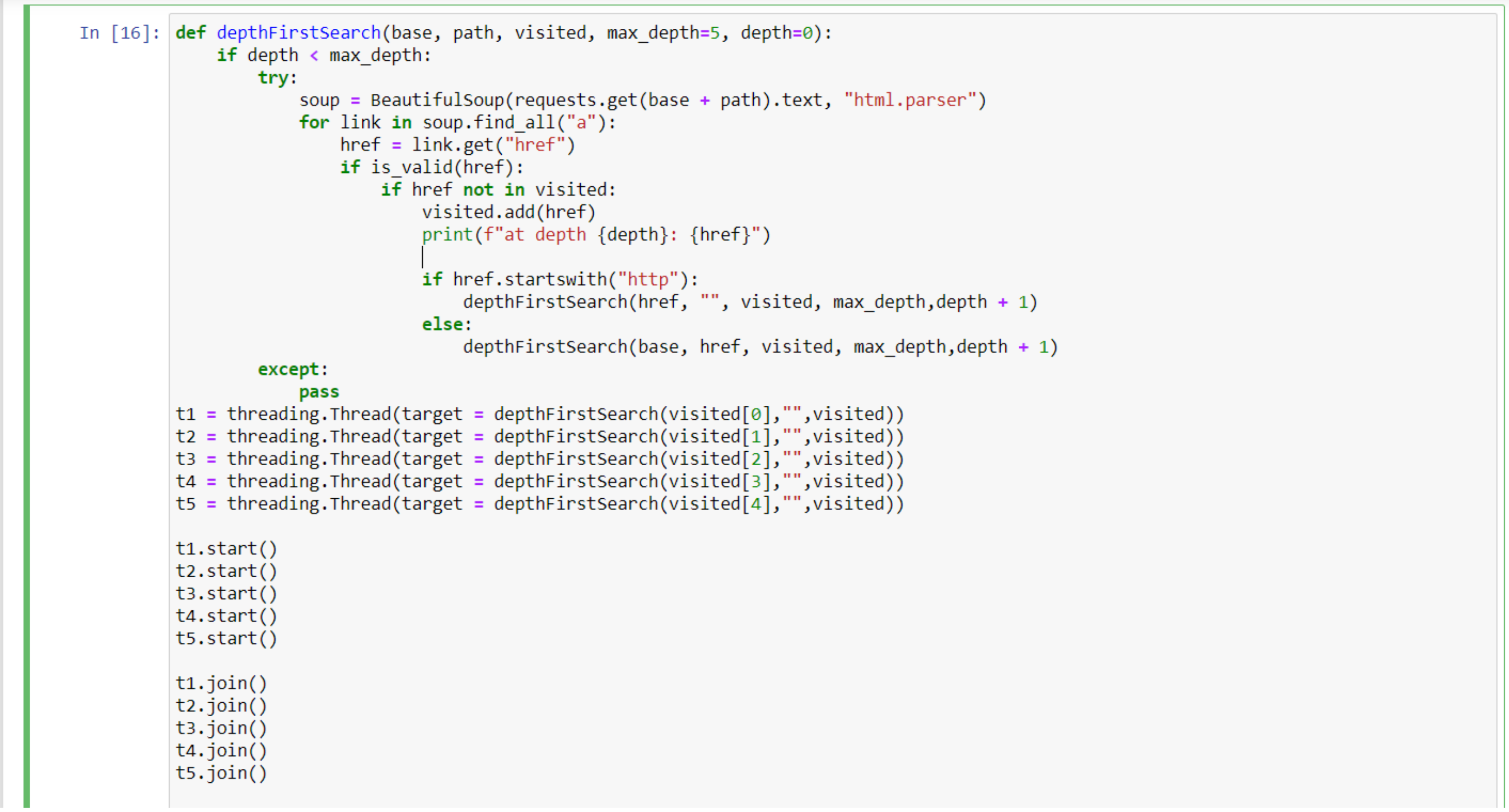
18BCE0763

Q1- Write a python program to

a) show the implementation of a concurrent depth-first crawler (No. of threads = 5 and depth =5).

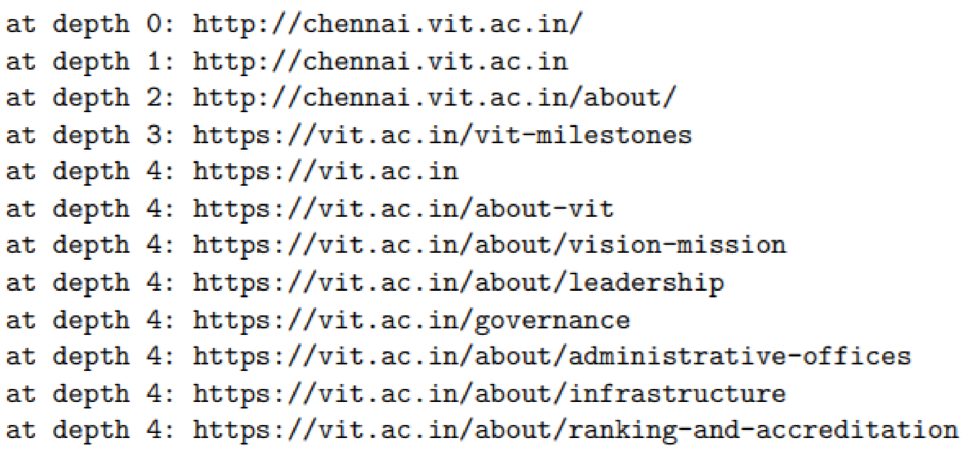


```
In [11]: import requests
         from urllib.parse import urlparse, urljoin
         from bs4 import BeautifulSoup
         import threading
         import os
         def is_valid(url):
             parsed = urlparse(url)
             return bool(parsed.netloc) and bool(parsed.scheme)

         #make function to collect first 5 links from base website

         visited = ["https://vit.ac.in/"]
         def collect5(url, visited):
             soup = BeautifulSoup(requests.get(url).text, "html.parser")
             for link in soup.find_all("a"):
                 href = link.get("href")
                 if href not in visited and len(visited)<5 and is_valid(href):
                     visited.append(href)
         collect5("https://vit.ac.in/", visited)
         visited

Out[11]: ['https://vit.ac.in/',
          'http://chennai.vit.ac.in/',
          'https://vitap.ac.in/',
          'https://vitbhopal.ac.in/',
          'https://admissions.vit.ac.in/btech/']
```



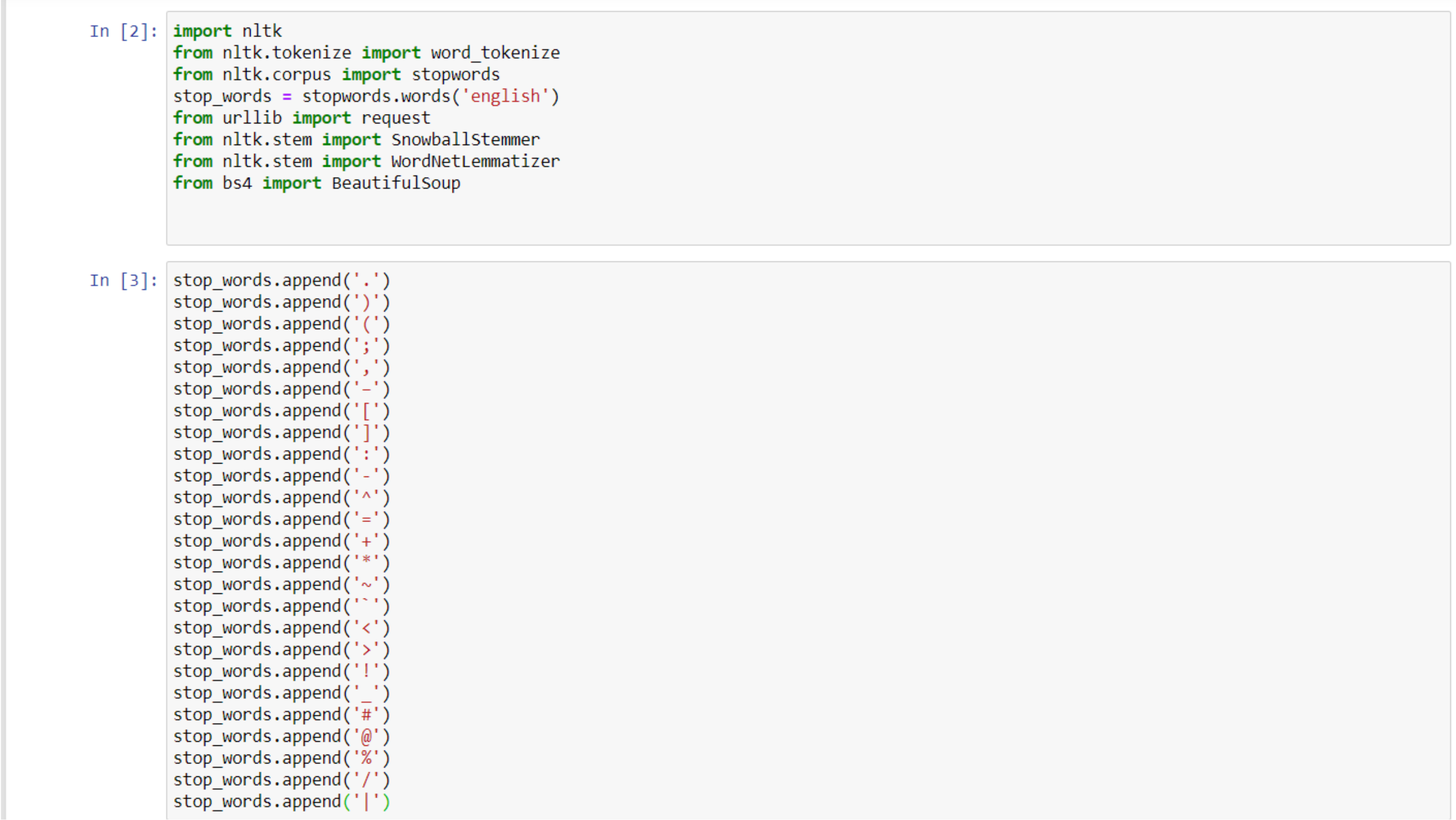
```
In [16]: def depthFirstSearch(base, path, visited, max_depth=5, depth=0):
             if depth < max_depth:
                 try:
                     soup = BeautifulSoup(requests.get(base + path).text, "html.parser")
                     for link in soup.find_all("a"):
                         href = link.get("href")
                         if is_valid(href):
                             if href not in visited:
                                 visited.add(href)
                                 print(f"at depth {depth}: {href}")

                                 if href.startswith("http"):
                                     depthFirstSearch(href, "", visited, max_depth,depth + 1)
                                 else:
                                     depthFirstSearch(base, href, visited, max_depth,depth + 1)
                 except:
                     pass
         t1 = threading.Thread(target = depthFirstSearch(visited[0],"",visited))
         t2 = threading.Thread(target = depthFirstSearch(visited[1],"",visited))
         t3 = threading.Thread(target = depthFirstSearch(visited[2],"",visited))
         t4 = threading.Thread(target = depthFirstSearch(visited[3],"",visited))
         t5 = threading.Thread(target = depthFirstSearch(visited[4],"",visited))

         t1.start()
         t2.start()
         t3.start()
         t4.start()
         t5.start()

         t1.join()
         t2.join()
         t3.join()
         t4.join()
         t5.join()
```

Output-



```
at depth 0: http://chennai.vit.ac.in/
at depth 1: http://chennai.vit.ac.in
at depth 2: http://chennai.vit.ac.in/about/
at depth 3: https://vit.ac.in/vit-milestones
at depth 4: https://vit.ac.in
at depth 4: https://vit.ac.in/about-vit
at depth 4: https://vit.ac.in/about/vision-mission
at depth 4: https://vit.ac.in/about/leadership
at depth 4: https://vit.ac.in/governance
at depth 4: https://vit.ac.in/about/administrative-offices
at depth 4: https://vit.ac.in/about/infrastructure
at depth 4: https://vit.ac.in/about/ranking-and-accreditation
```

b) Develop the crawler program to handle various challenges (such as Parsing, Stemming, Lemmitization, Link Extraction, Canonicalization, Spider Trap etc.) faced by crawler while implementing.



In [2]:

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
stop_words = stopwords.words('english')
from urllib import request
from nltk.stem import SnowballStemmer
from nltk.stem import WordNetLemmatizer
from bs4 import BeautifulSoup
```

In [3]:

```
stop_words.append('.')
stop_words.append(')')
stop_words.append('(')
stop_words.append(';')
stop_words.append(',')
stop_words.append('–')
stop_words.append('[')
stop_words.append(']')
stop_words.append(':')
stop_words.append('-')
stop_words.append('^')
stop_words.append('=')
stop_words.append('+')
stop_words.append('*')
stop_words.append('~')
stop_words.append('`')
stop_words.append('<')
stop_words.append('>')
stop_words.append('!')
stop_words.append('_')
stop_words.append('#')
stop_words.append('@')
stop_words.append('%')
stop_words.append('/')
stop_words.append('|')
```

In [4]:

```
#Taking first 5 docs from visited set.

url1 = "https://vit.ac.in/"
url2 = "http://chennai.vit.ac.in/"
url3 = "https://vitap.ac.in/"
url4 = "https://vit.ac.in/about-vit"
url5 = "https://vit.ac.in/academics/coe"
```

In [5]:

```
#parsing html and getting the content

html1 = request.urlopen(url1).read().decode('utf8')
html2 = request.urlopen(url2).read().decode('utf8')
html3 = request.urlopen(url3).read().decode('utf8')
html4 = request.urlopen(url4).read().decode('utf8')
html5 = request.urlopen(url5).read().decode('utf8')

raw1 = BeautifulSoup(html1, 'html.parser').get_text()
raw2 = BeautifulSoup(html2, 'html.parser').get_text()
raw3 = BeautifulSoup(html3, 'html.parser').get_text()
raw4 = BeautifulSoup(html4, 'html.parser').get_text()
raw5 = BeautifulSoup(html5, 'html.parser').get_text()
```

In [6]:

```
#tokenizing
w1 = word_tokenize(raw1)
w2 = word_tokenize(raw2)
w3 = word_tokenize(raw3)
w4 = word_tokenize(raw4)
w5 = word_tokenize(raw5)
```

In [7]:

```
#stopword removal
filtered1 = [w for w in w1 if not w in stop_words]
filtered2 = [w for w in w2 if not w in stop_words]
filtered3 = [w for w in w3 if not w in stop_words]
filtered4 = [w for w in w4 if not w in stop_words]
filtered5 = [w for w in w5 if not w in stop_words]
```

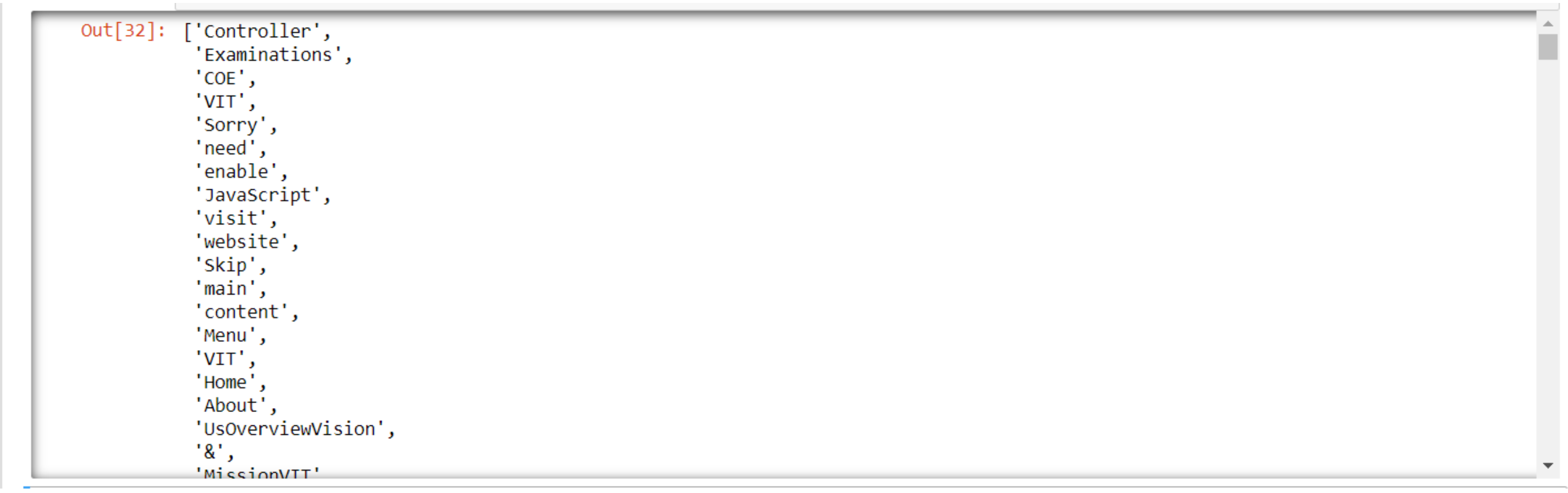
In [8]:

```
#stemming
lemmatizer = WordNetLemmatizer()
lemmatized1 = []
lemmatized2 = []
lemmatized3 = []
lemmatized4 = []
lemmatized5 = []
```

In [9]:

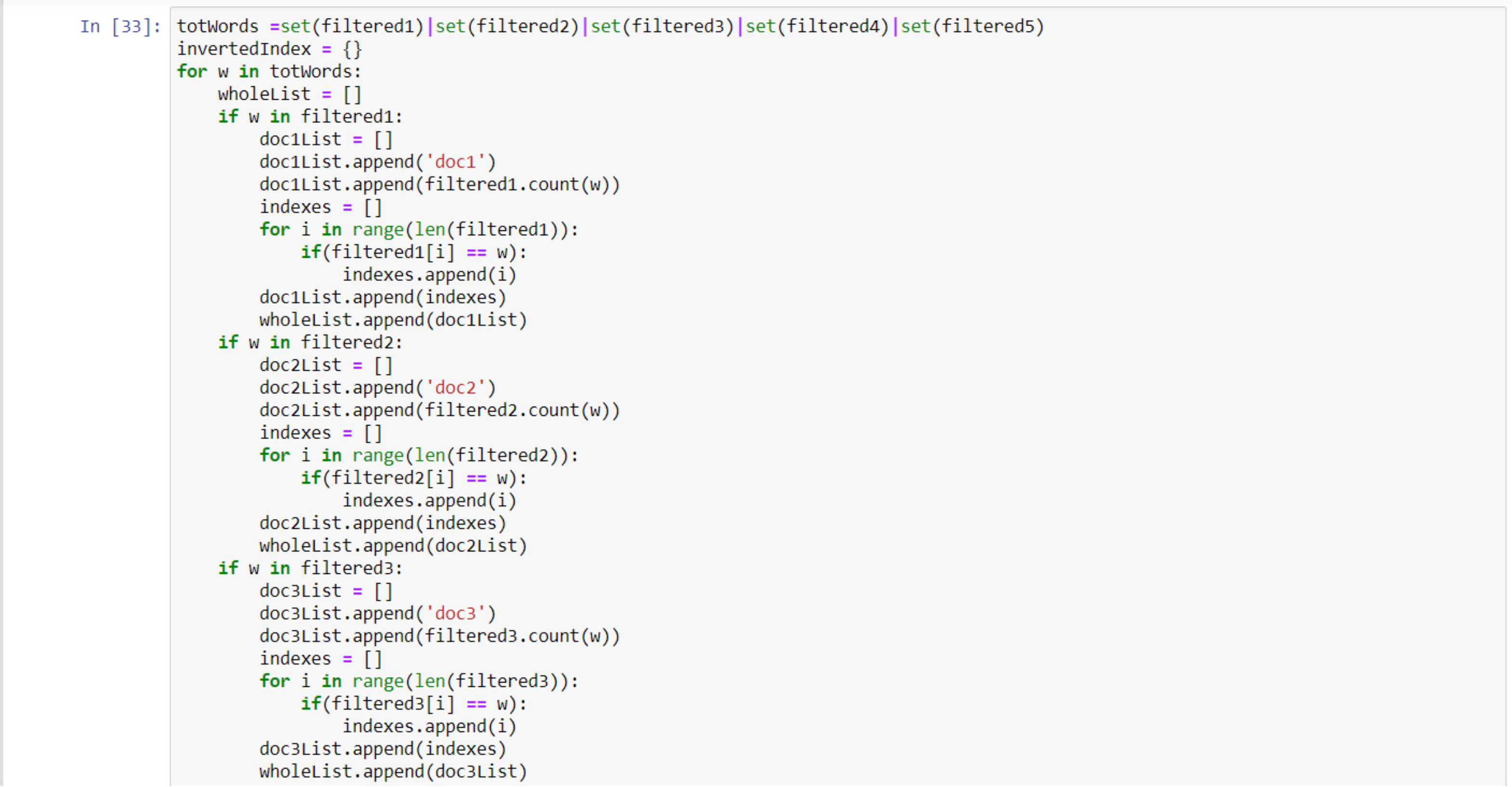
```
for w in filtered1:
    lemmatized1.append(lemmatizer.lemmatize(w))
for w in filtered2:
    lemmatized2.append(lemmatizer.lemmatize(w))
for w in filtered3:
    lemmatized3.append(lemmatizer.lemmatize(w))
for w in filtered4:
    lemmatized4.append(lemmatizer.lemmatize(w))
for w in filtered5:
    lemmatized5.append(lemmatizer.lemmatize(w))
```

Output-

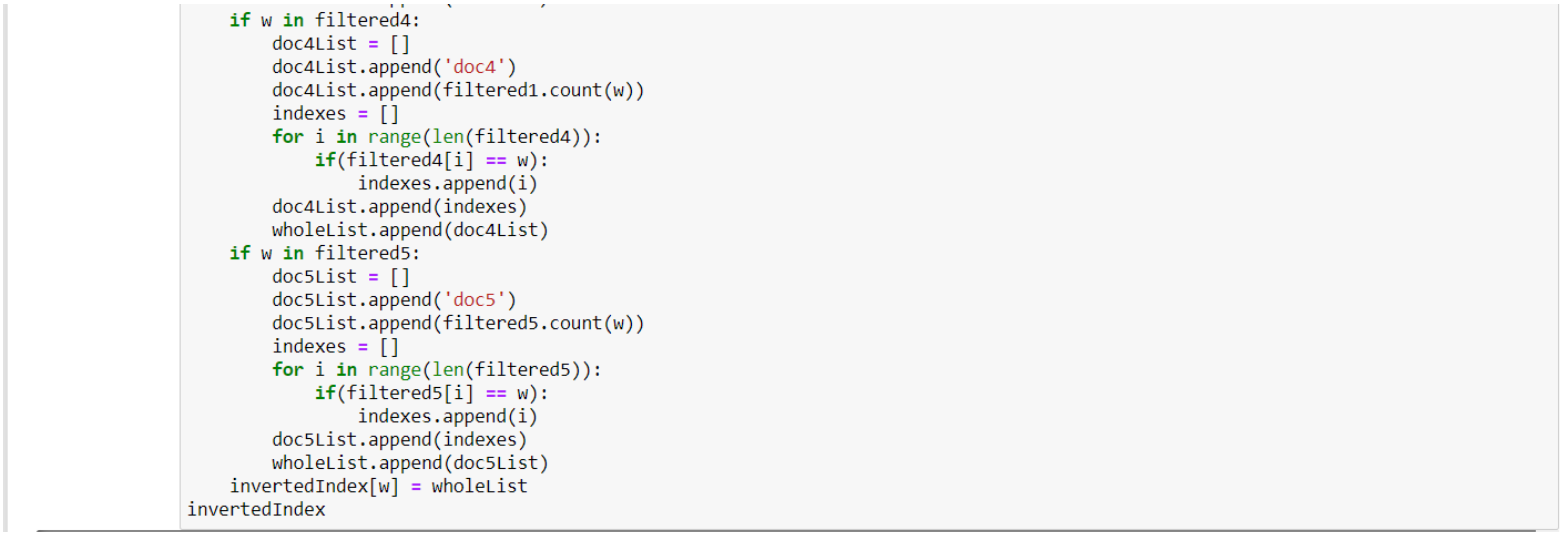


```
Out[32]: ['Controller',
          'Examinations',
          'COE',
          'VIT',
          'Sorry',
          'need',
          'enable',
          'JavaScript',
          'visit',
          'website',
          'Skip',
          'main',
          'content',
          'Menu',
          'VIT',
          'Home',
          'About',
          'UsOverviewVision',
          '&',
          'MissionVIT'
```

c) Based on the contents retrieved, prepare one inverted index file (with proper representation).

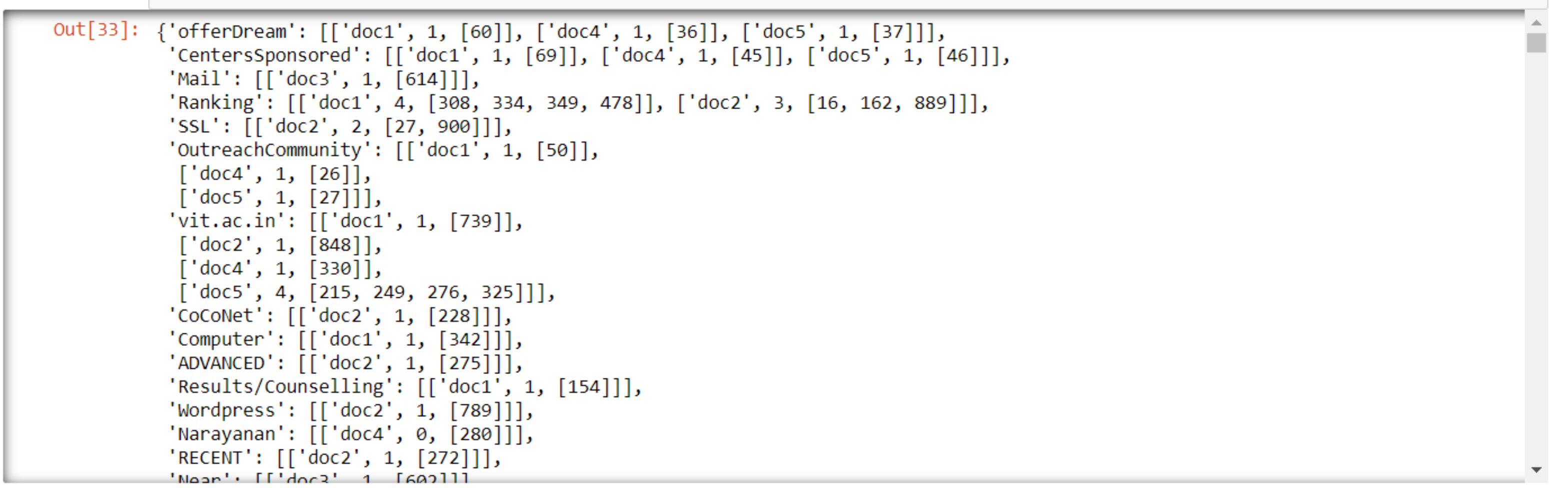


```
In [33]: totWords =set(filtered1)|set(filtered2)|set(filtered3)|set(filtered4)|set(filtered5)
         invertedIndex = {}
         for w in totWords:
             wholeList = []
             if w in filtered1:
                 doc1List = []
                 doc1List.append('doc1')
                 doc1List.append(filtered1.count(w))
                 indexes = []
                 for i in range(len(filtered1)):
                     if(filtered1[i] == w):
                         indexes.append(i)
                 doc1List.append(indexes)
                 wholeList.append(doc1List)
             if w in filtered2:
                 doc2List = []
                 doc2List.append('doc2')
                 doc2List.append(filtered2.count(w))
                 indexes = []
                 for i in range(len(filtered2)):
                     if(filtered2[i] == w):
                         indexes.append(i)
                 doc2List.append(indexes)
                 wholeList.append(doc2List)
             if w in filtered3:
                 doc3List = []
                 doc3List.append('doc3')
                 doc3List.append(filtered3.count(w))
                 indexes = []
                 for i in range(len(filtered3)):
                     if(filtered3[i] == w):
                         indexes.append(i)
                 doc3List.append(indexes)
                 wholeList.append(doc3List)
```



```
    if w in filtered4:
        doc4List = []
        doc4List.append('doc4')
        doc4List.append(filtered1.count(w))
        indexes = []
        for i in range(len(filtered4)):
            if(filtered4[i] == w):
                indexes.append(i)
        doc4List.append(indexes)
        wholeList.append(doc4List)
    if w in filtered5:
        doc5List = []
        doc5List.append('doc5')
        doc5List.append(filtered5.count(w))
        indexes = []
        for i in range(len(filtered5)):
            if(filtered5[i] == w):
                indexes.append(i)
        doc5List.append(indexes)
        wholeList.append(doc5List)
    invertedIndex[w] = wholeList
invertedIndex
```

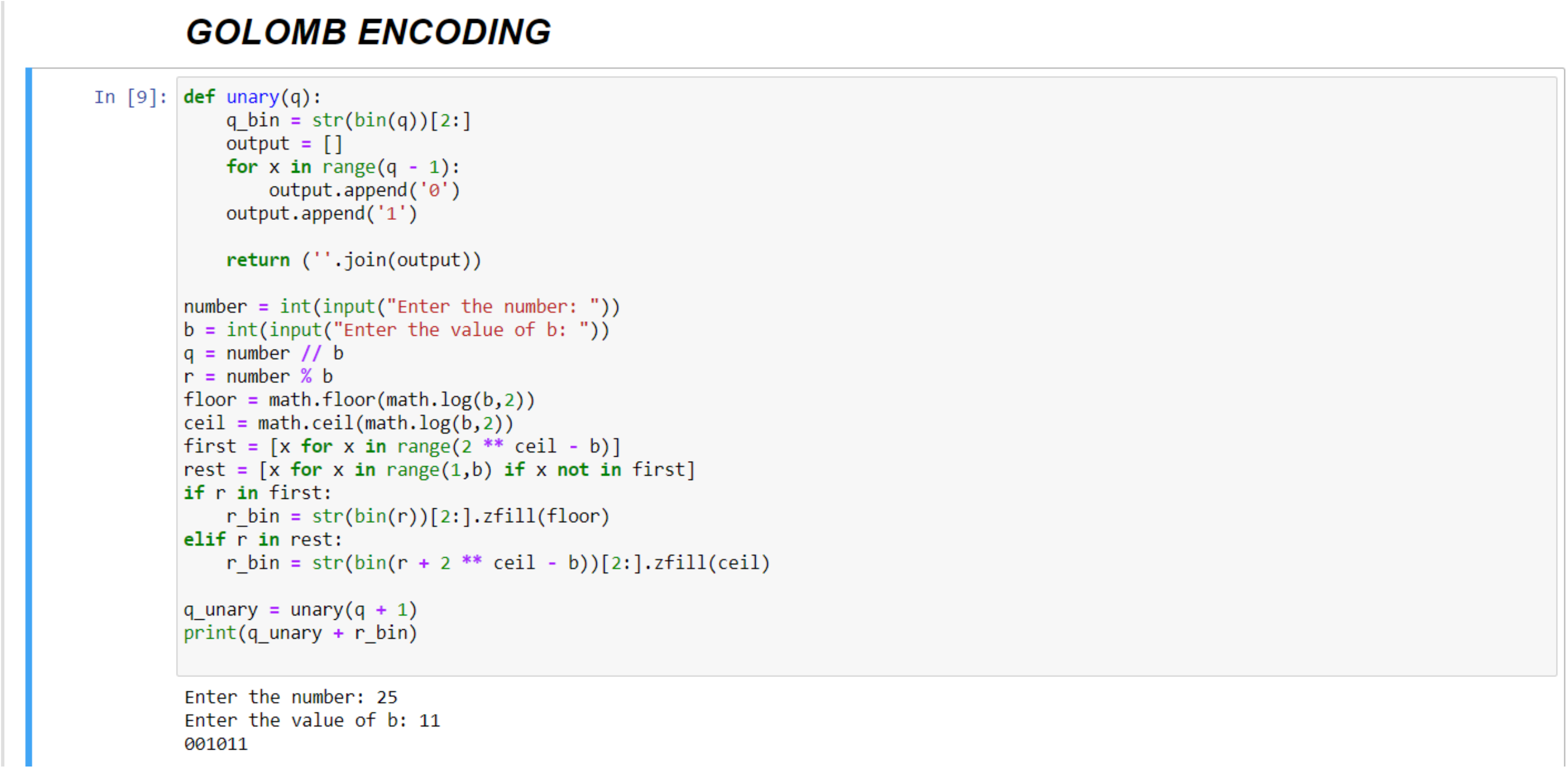
Output-



```
Out[33]: {'offerDream': [['doc1', 1, [60]], ['doc4', 1, [36]], ['doc5', 1, [37]]],
          'CentersSponsored': [['doc1', 1, [69]], ['doc4', 1, [45]], ['doc5', 1, [46]]],
          'Mail': [['doc3', 1, [614]]],
          'Ranking': [['doc1', 4, [308, 334, 349, 478]], ['doc2', 3, [16, 162, 889]]],
          'SSL': [['doc2', 2, [27, 900]]],
          'OutreachCommunity': [['doc1', 1, [50]],
           ['doc4', 1, [26]],
           ['doc5', 1, [27]]],
          'vit.ac.in': [['doc1', 1, [739]],
           ['doc2', 1, [848]],
           ['doc4', 1, [330]],
           ['doc5', 4, [215, 249, 276, 325]]],
          'CoCoNet': [['doc2', 1, [228]]],
          'Computer': [['doc1', 1, [342]]],
          'ADVANCED': [['doc2', 1, [275]]],
          'Results/Counselling': [['doc1', 1, [154]]],
          'Wordpress': [['doc2', 1, [789]]],
          'Narayanan': [['doc4', 0, [280]]],
          'RECENT': [['doc2', 1, [272]]],
          'Near': [['doc3', 1, [602]]],
```

Q2- Write a python program to show the implementation of Golomb Encoding-decoding technique.

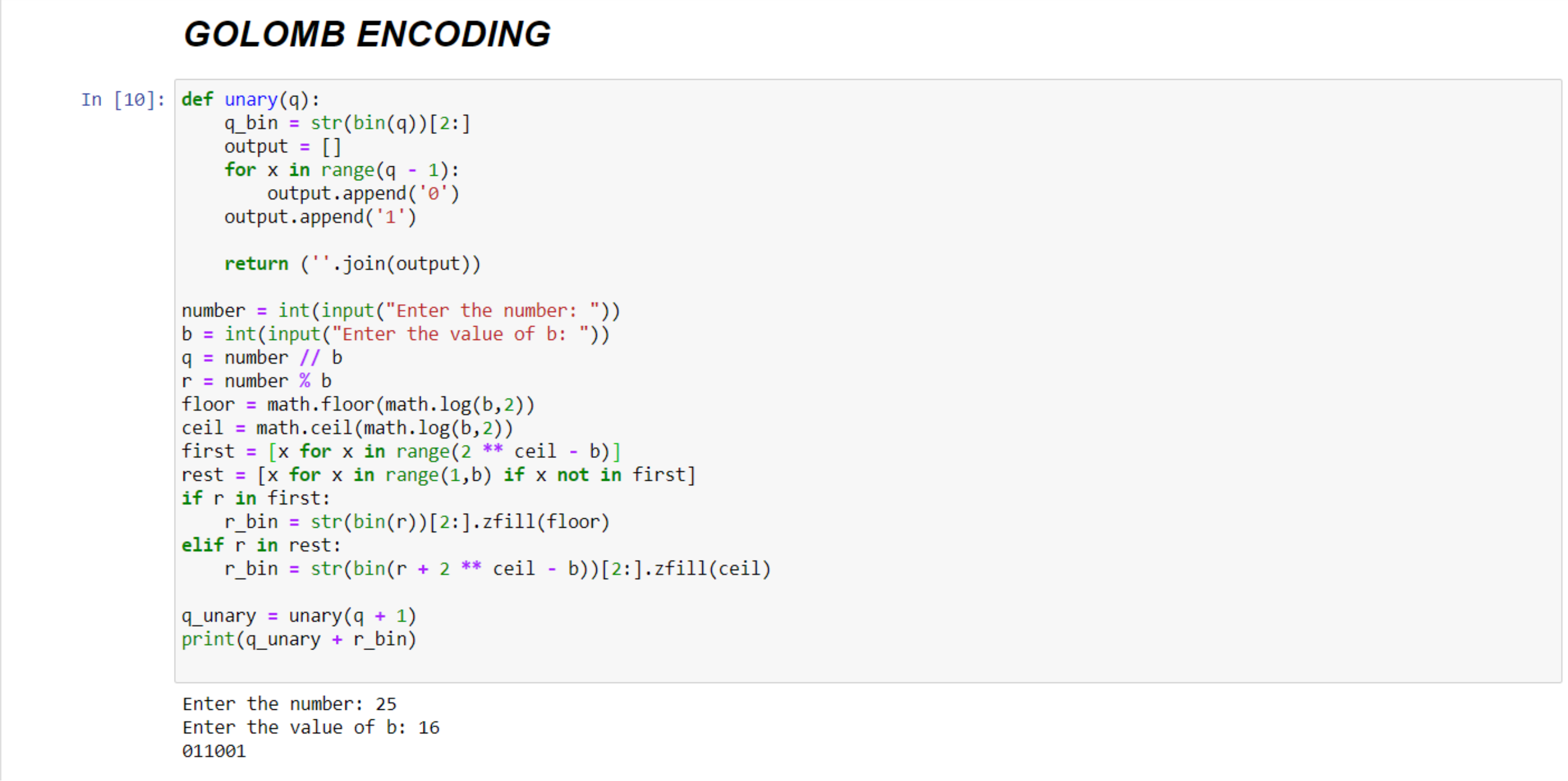
a) Encode x=25, 37, with b=11 and b=16.



GOLOMB ENCODING

```
In [9]: def unary(q):
            q_bin = str(bin(q))[2:]
            output = []
            for x in range(q - 1):
                output.append('0')
            output.append('1')

            return (''.join(output))

        number = int(input("Enter the number: "))
        b = int(input("Enter the value of b: "))
        q = number // b
        r = number % b
        floor = math.floor(math.log(b,2))
        ceil = math.ceil(math.log(b,2))
        first = [x for x in range(2 ** ceil - b)]
        rest = [x for x in range(1,b) if x not in first]
        if r in first:
            r_bin = str(bin(r))[2:].zfill(floor)
        elif r in rest:
            r_bin = str(bin(r + 2 ** ceil - b))[2:].zfill(ceil)

        q_unary = unary(q + 1)
        print(q_unary + r_bin)
```

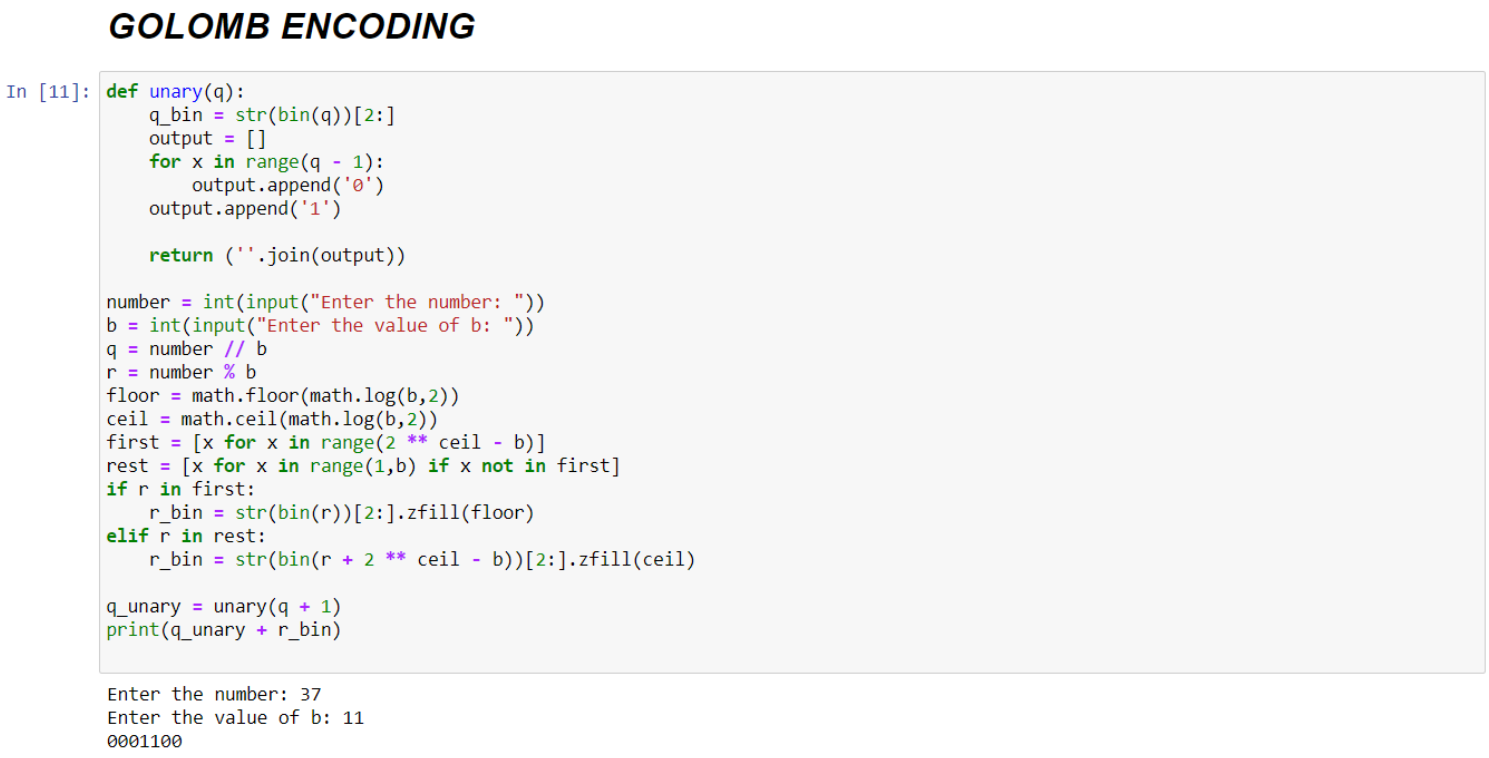
```
Enter the number: 25
Enter the value of b: 11
001011
```



GOLOMB ENCODING

```
In [10]: def unary(q):
             q_bin = str(bin(q))[2:]
             output = []
             for x in range(q - 1):
                 output.append('0')
             output.append('1')

             return (''.join(output))

         number = int(input("Enter the number: "))
         b = int(input("Enter the value of b: "))
         q = number // b
         r = number % b
         floor = math.floor(math.log(b,2))
         ceil = math.ceil(math.log(b,2))
         first = [x for x in range(2 ** ceil - b)]
         rest = [x for x in range(1,b) if x not in first]
         if r in first:
             r_bin = str(bin(r))[2:].zfill(floor)
         elif r in rest:
             r_bin = str(bin(r + 2 ** ceil - b))[2:].zfill(ceil)

         q_unary = unary(q + 1)
         print(q_unary + r_bin)
```

```
Enter the number: 25
Enter the value of b: 16
011001
```

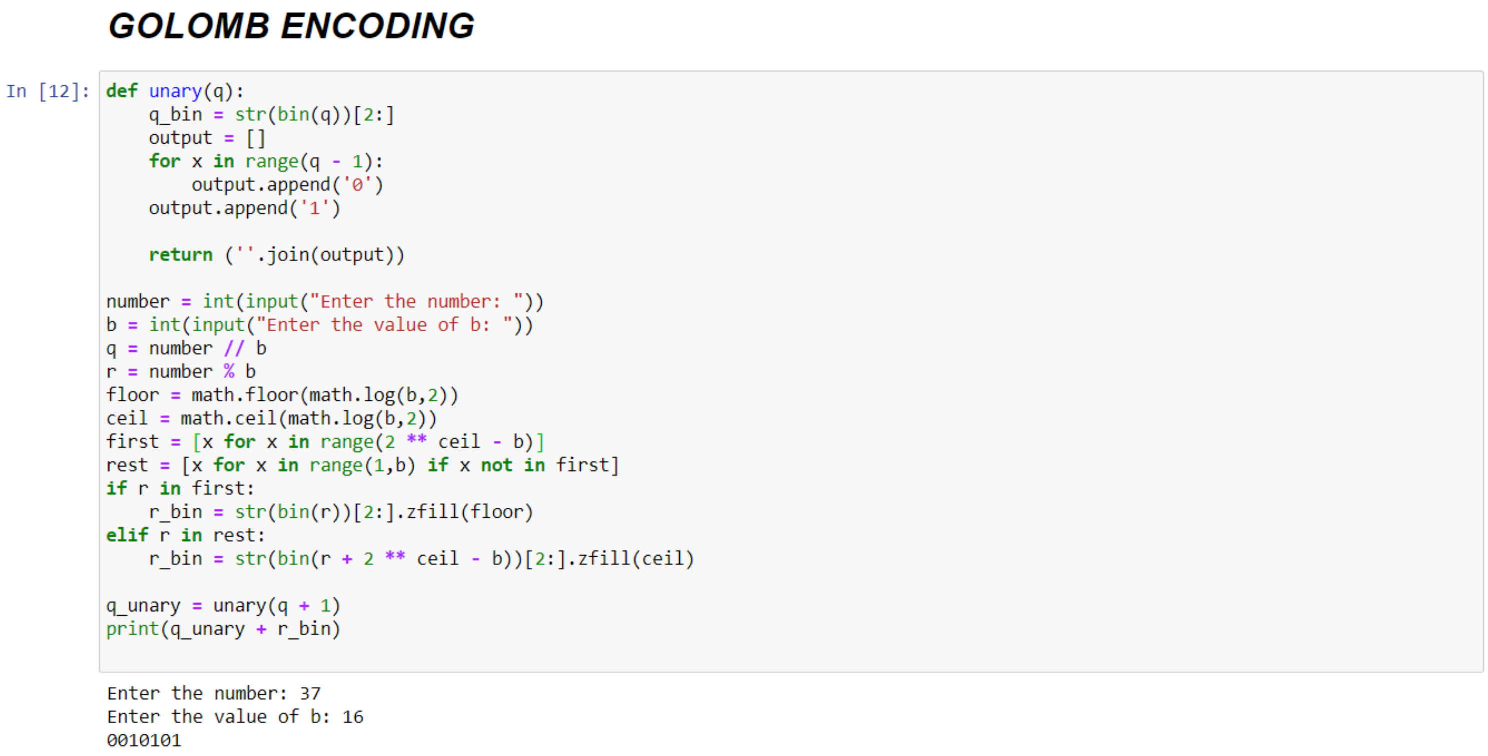


GOLOMB ENCODING

In [11]:

```
def unary(q):
    q_bin = str(bin(q))[2:]
    output = []
    for x in range(q - 1):
        output.append('0')
    output.append('1')

    return (''.join(output))

number = int(input("Enter the number: "))
b = int(input("Enter the value of b: "))
q = number // b
r = number % b
floor = math.floor(math.log(b,2))
ceil = math.ceil(math.log(b,2))
first = [x for x in range(2 ** ceil - b)]
rest = [x for x in range(1,b) if x not in first]
if r in first:
    r_bin = str(bin(r))[2:].zfill(floor)
elif r in rest:
    r_bin = str(bin(r + 2 ** ceil - b))[2:].zfill(ceil)

q_unary = unary(q + 1)
print(q_unary + r_bin)
```

```
Enter the number: 37
Enter the value of b: 11
0001100
```



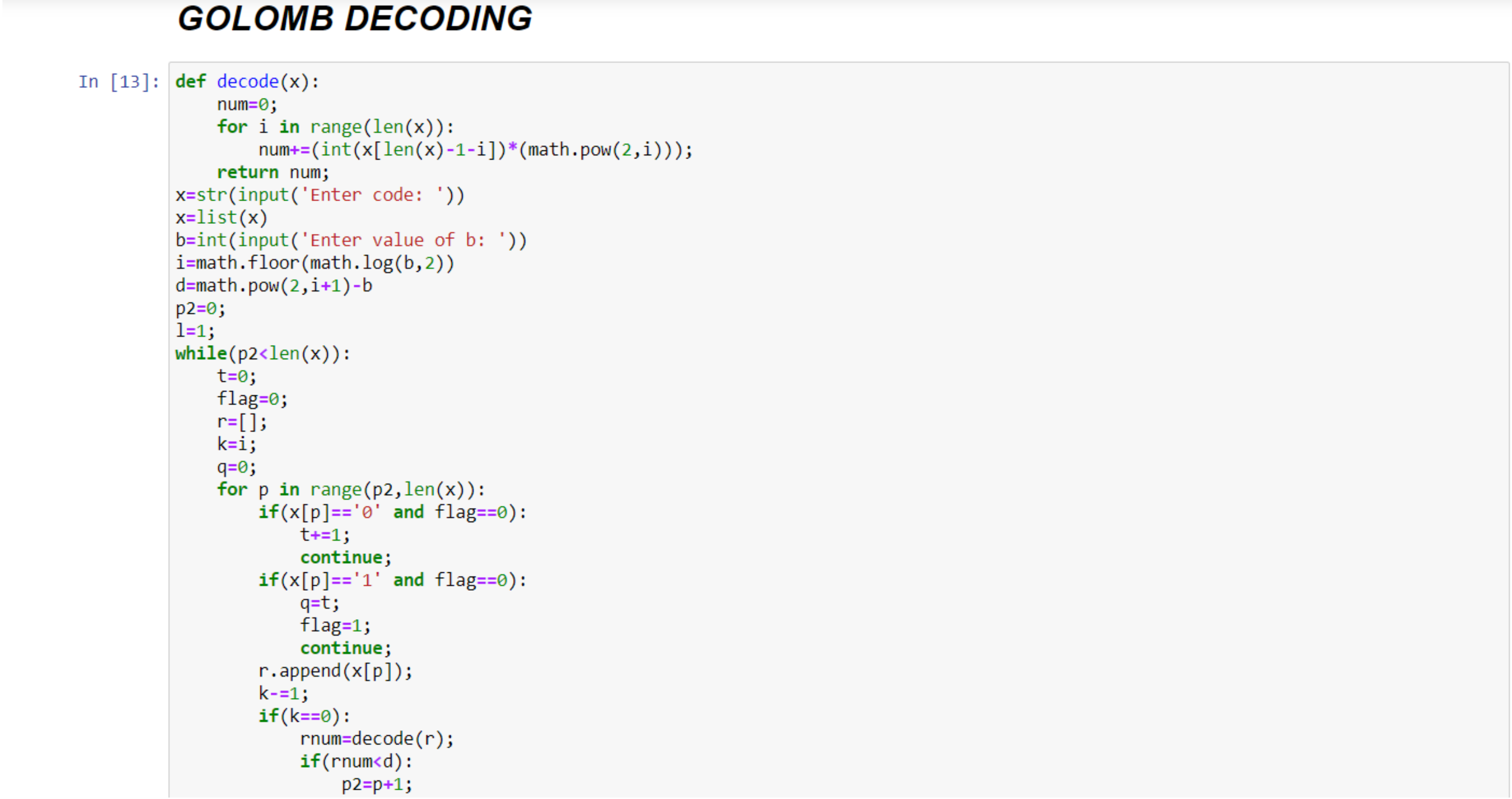
GOLOMB ENCODING

In [12]:

```
def unary(q):
    q_bin = str(bin(q))[2:]
    output = []
    for x in range(q - 1):
        output.append('0')
    output.append('1')

    return (''.join(output))

number = int(input("Enter the number: "))
b = int(input("Enter the value of b: "))
q = number // b
r = number % b
floor = math.floor(math.log(b,2))
ceil = math.ceil(math.log(b,2))
first = [x for x in range(2 ** ceil - b)]
rest = [x for x in range(1,b) if x not in first]
if r in first:
    r_bin = str(bin(r))[2:].zfill(floor)
elif r in rest:
    r_bin = str(bin(r + 2 ** ceil - b))[2:].zfill(ceil)

q_unary = unary(q + 1)
print(q_unary + r_bin)
```

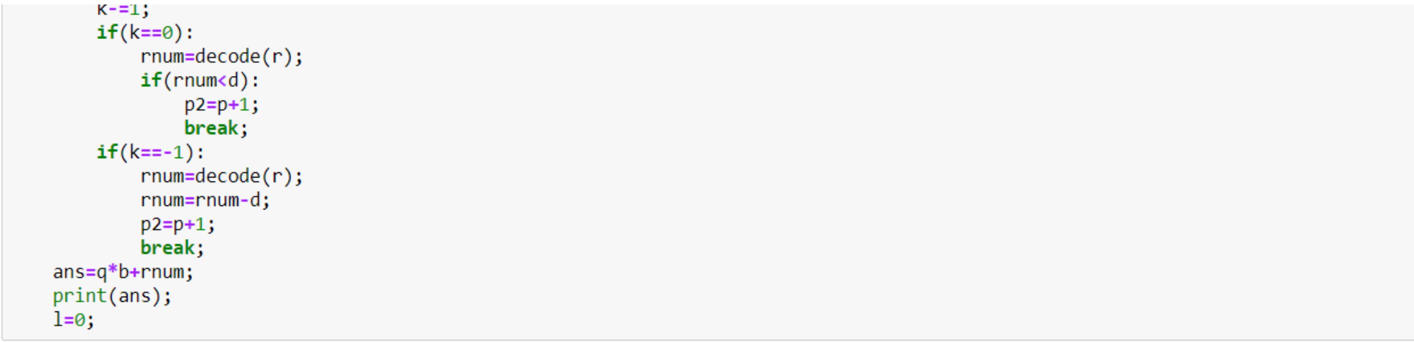
```
Enter the number: 37
Enter the value of b: 16
0010101
```

b) Decode the Golomb encoded sequence 1111111110010001101 with b = 10.



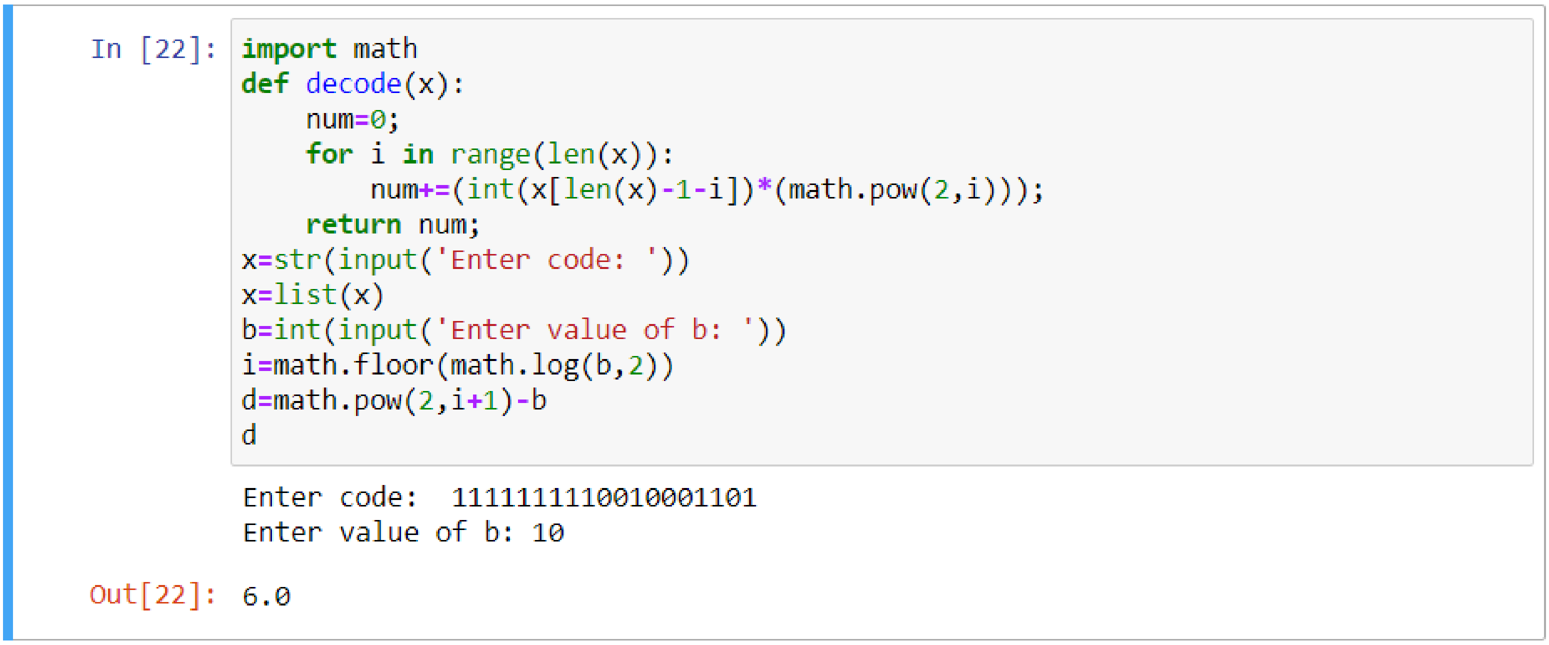
GOLOMB DECODING

In [13]:
```
def decode(x):
    num=0;
    for i in range(len(x)):
        num+=(int(x[len(x)-1-i])*(math.pow(2,i)));
    return num;
x=str(input('Enter code: '))
x=list(x)
b=int(input('Enter value of b: '))
i=math.floor(math.log(b,2))
d=math.pow(2,i+1)-b
p2=0;
l=1;
while(p2<len(x)):
    t=0;
    flag=0;
    r=[];
    k=i;
    q=0;
    for p in range(p2,len(x)):
        if(x[p]=='0' and flag==0):
            t+=1;
            continue;
        if(x[p]=='1' and flag==0):
            q=t;
            flag=1;
            continue;
        r.append(x[p]);
        k-=1;
        if(k==0):
            rnum=decode(r);
            if(rnum<d):
                p2=p+1;
```



```
        k-=1;
        if(k==0):
            rnum=decode(r);
            if(rnum<d):
                p2=p+1;
                break;
        if(k==-1):
            rnum=decode(r);
            rnum=rnum-d;
            p2=p+1;
            break;
    ans=q*b+rnum;
    print(ans);
    l=0;
```

Output-



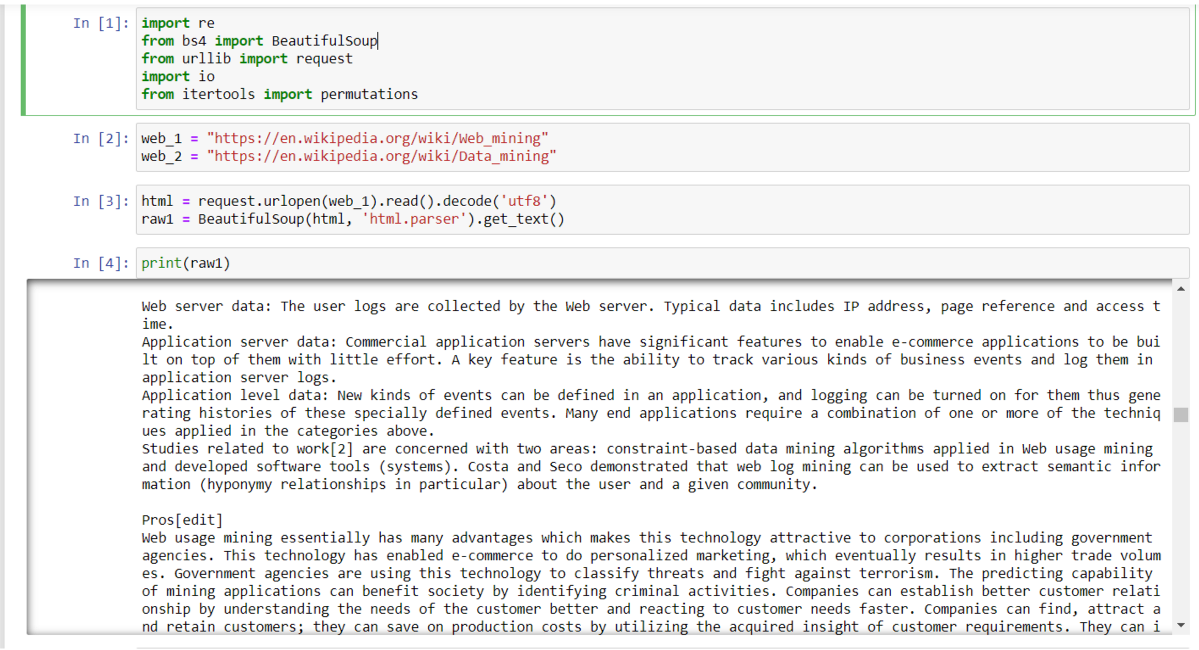
```
In [22]: import math
         def decode(x):
             num=0;
             for i in range(len(x)):
                 num+=(int(x[len(x)-1-i])*(math.pow(2,i)));
             return num;
         x=str(input('Enter code: '))
         x=list(x)
         b=int(input('Enter value of b: '))
         i=math.floor(math.log(b,2))
         d=math.pow(2,i+1)-b
         d

         Enter code:  1111111110010001101
         Enter value of b: 10

Out[22]: 6.0
```

Q3- Write a python program to extract the contents (excluding any tags) from two websites

https://en.wikipedia.org/wiki/Web_mining

https://en.wikipedia.org/wiki/Data_mining

Save the content in two separate files. Construct a trie based on the content retrieved in using HashMap / B-Tree / Dictionary. Write a program to show the implementation of Predictive Typing and Auto-Correct using the trie prepared.



```
In [1]: import re
        from bs4 import BeautifulSoup
        from urllib import request
        import io
        from itertools import permutations

In [2]: web_1 = "https://en.wikipedia.org/wiki/Web_mining"
        web_2 = "https://en.wikipedia.org/wiki/Data_mining"

In [3]: html = request.urlopen(web_1).read().decode('utf8')
        raw1 = BeautifulSoup(html, 'html.parser').get_text()

In [4]: print(raw1)

        Web server data: The user logs are collected by the Web server. Typical data includes IP address, page reference and access t
        ime.
        Application server data: Commercial application servers have significant features to enable e-commerce applications to be bui
        lt on top of them with little effort. A key feature is the ability to track various kinds of business events and log them in
        application server logs.
        Application level data: New kinds of events can be defined in an application, and logging can be turned on for them thus gene
        rating histories of these specially defined events. Many end applications require a combination of one or more of the techniq
        ues applied in the categories above.
        Studies related to work[2] are concerned with two areas: constraint-based data mining algorithms applied in Web usage mining
        and developed software tools (systems). Costa and Seco demonstrated that web log mining can be used to extract semantic infor
        mation (hyponymy relationships in particular) about the user and a given community.

        Pros[edit]
        Web usage mining essentially has many advantages which makes this technology attractive to corporations including government
        agencies. This technology has enabled e-commerce to do personalized marketing, which eventually results in higher trade volum
        es. Government agencies are using this technology to classify threats and fight against terrorism. The predicting capability
        of mining applications can benefit society by identifying criminal activities. Companies can establish better customer relati
        onship by understanding the needs of the customer better and reacting to customer needs faster. Companies can find, attract a
        nd retain customers; they can save on production costs by utilizing the acquired insight of customer requirements. They can i
```

```
In [5]: html = request.urlopen(web_2).read().decode('utf8')
        raw2 = BeautifulSoup(html, 'html.parser').get_text()
```

```
In [6]: print(raw2)
```

```


Related articles
List of datasets for machine-learning research
Outline of machine learning

vte
Data mining is a process of discovering patterns in large data sets involving methods at the intersection of machine learnin
g, statistics, and database systems.[1] Data mining is an interdisciplinary subfield of computer science and statistics with
an overall goal to extract information (with intelligent methods) from a data set and transform the information into a compre
hensible structure for further use.[1][2][3][4] Data mining is the analysis step of the "knowledge discovery in databases" pr
ocess, or KDD.[5] Aside from the raw analysis step, it also involves database and data management aspects, data pre-processin
g, model and inference considerations, interestingness metrics, complexity considerations, post-processing of discovered stru
ctures, visualization, and online updating.[1]
The term "data mining" is a misnomer, because the goal is the extraction of patterns and knowledge from large amounts of dat
a, not the extraction (mining) of data itself.[6] It also is a buzzword[7] and is frequently applied to any form of large-sca
le data or information processing (collection, extraction, warehousing, analysis, and statistics) as well as any application
of computer decision support system, including artificial intelligence (e.g., machine learning) and business intelligence. Th
e book Data mining: Practical machine learning tools and techniques with Java[8] (which covers mostly machine learning materi
al) was originally to be named just Practical machine learning, and the term data mining was only added for marketing reason
s.[9] Often the more general terms (large scale) data analysis and analytics or, when referring to actual methods, artificial
```

```
In [7]: with io.open('web_mining.doc', mode = 'w+',encoding="utf-8") as file_1:
            file_1.write(raw1)
```

```
In [8]: with io.open('data_mining.doc', mode = 'w+',encoding="utf-8")as file_2:
            file_2.write(raw2)
```

```
In [9]: content_1=raw1
        content_2=raw2
```

```
In [10]: words_1 = content_1.lower().split()
         words_2 = content_2.lower().split()
         all_words = words_1 + words_2
         temp_unique_words = list(set(all_words))
```

```
In [11]: from nltk.corpus import stopwords
         unique_words = []
         for x in temp_unique_words:
             if x.isalpha() and x not in stopwords.words("english"):
                 unique_words.append(x)
```

```
In [12]: from nltk.stem import PorterStemmer
         stem_words = {}
         stemmer = PorterStemmer()
```

```
In [13]: final_word_list = []
         for x in unique_words:
             temp = stemmer.stem(x)
             if x not in final_word_list:
                 final_word_list.append(temp)
         found_words = []
```

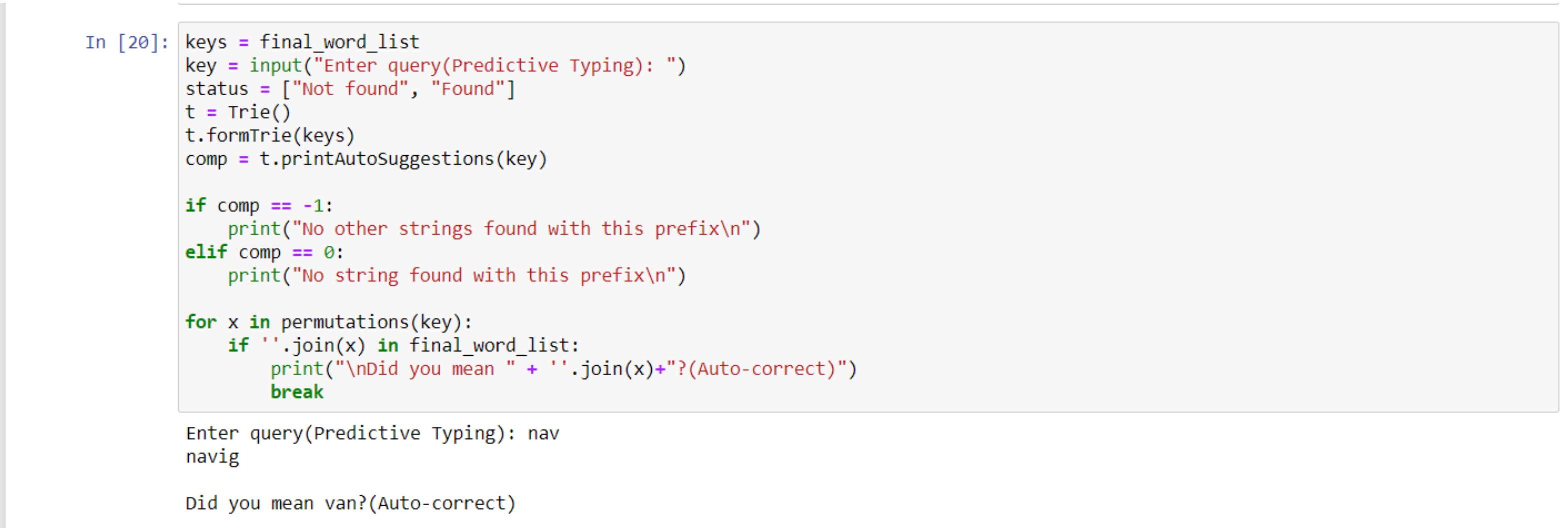
```
In [14]: class TrieNode():
             def __init__(self):
                 self.children = {}
                 self.last = False
```

In [15]:

```python
class Trie():

    def __init__(self):
        self.root = TrieNode()
        self.word_list = []


    def formTrie(self, keys):
        for key in keys:
            self.insert(key)

    def insert(self, key):
        node = self.root

        for a in list(key):
            if not node.children.get(a):
                node.children[a] = TrieNode()
            node = node.children[a]
        node.last = True

    def search(self, key):
        node = self.root
        found = True
        for a in list(key):
            if not node.children.get(a):
                found = False
                break

            node = node.children[a]
        return node and node.last and found
```

```python
    def suggestionsRec(self, node, word):
        if node.last:
            self.word_list.append(word)
        for a,n in node.children.items():
            self.suggestionsRec(n, word + a)

    def printAutoSuggestions(self, key):
        node = self.root
        not_found = False
        temp_word = ''

        for a in list(key):
            if not node.children.get(a):
                not_found = True
                break
            temp_word += a
            node = node.children[a]
        if not_found:
            return 0
        elif node.last and not node.children:
            return -1
        self.suggestionsRec(node, temp_word)
        for s in self.word_list:
            print(s)
            found_words.append(s)
        return 1
```

Output-



In [20]:

```
keys = final_word_list
key = input("Enter query(Predictive Typing): ")
status = ["Not found", "Found"]
t = Trie()
t.formTrie(keys)
comp = t.printAutoSuggestions(key)

if comp == -1:
    print("No other strings found with this prefix\n")
elif comp == 0:
    print("No string found with this prefix\n")

for x in permutations(key):
    if ''.join(x) in final_word_list:
        print("\nDid you mean " + ''.join(x)+"?(Auto-correct)")
        break
```

```
Enter query(Predictive Typing): nav
navig

Did you mean van?(Auto-correct)
```

Q4- Write a python program to extract the contents (excluding any tags) from the following five websites

https://en.wikipedia.org/wiki/Web_mining

https://en.wikipedia.org/wiki/Data_mining

https://en.wikipedia.org/wiki/Artificial_intelligence

https://en.wikipedia.org/wiki/Machine_learning

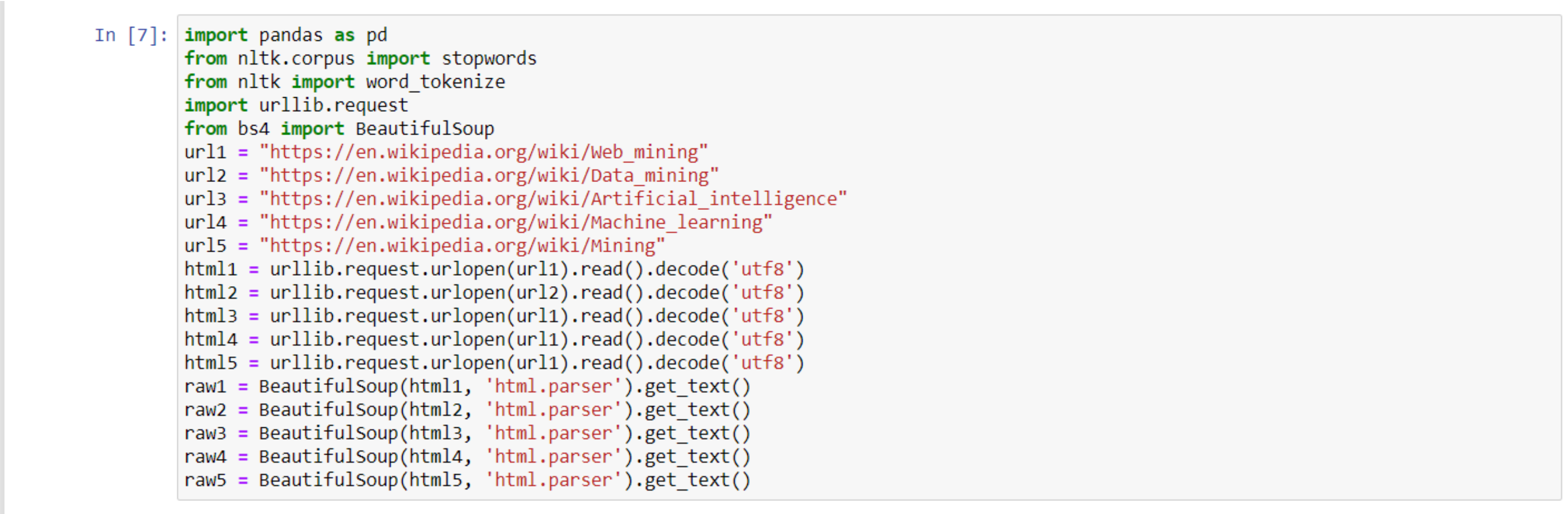
https://en.wikipedia.org/wiki/Mining

Refined the contents by applying stopword removal and lemmatization process. Save the refined tokenized content in five separate files. Considering a vector space model and do the following operations according to the query “Mining large volume of data”.

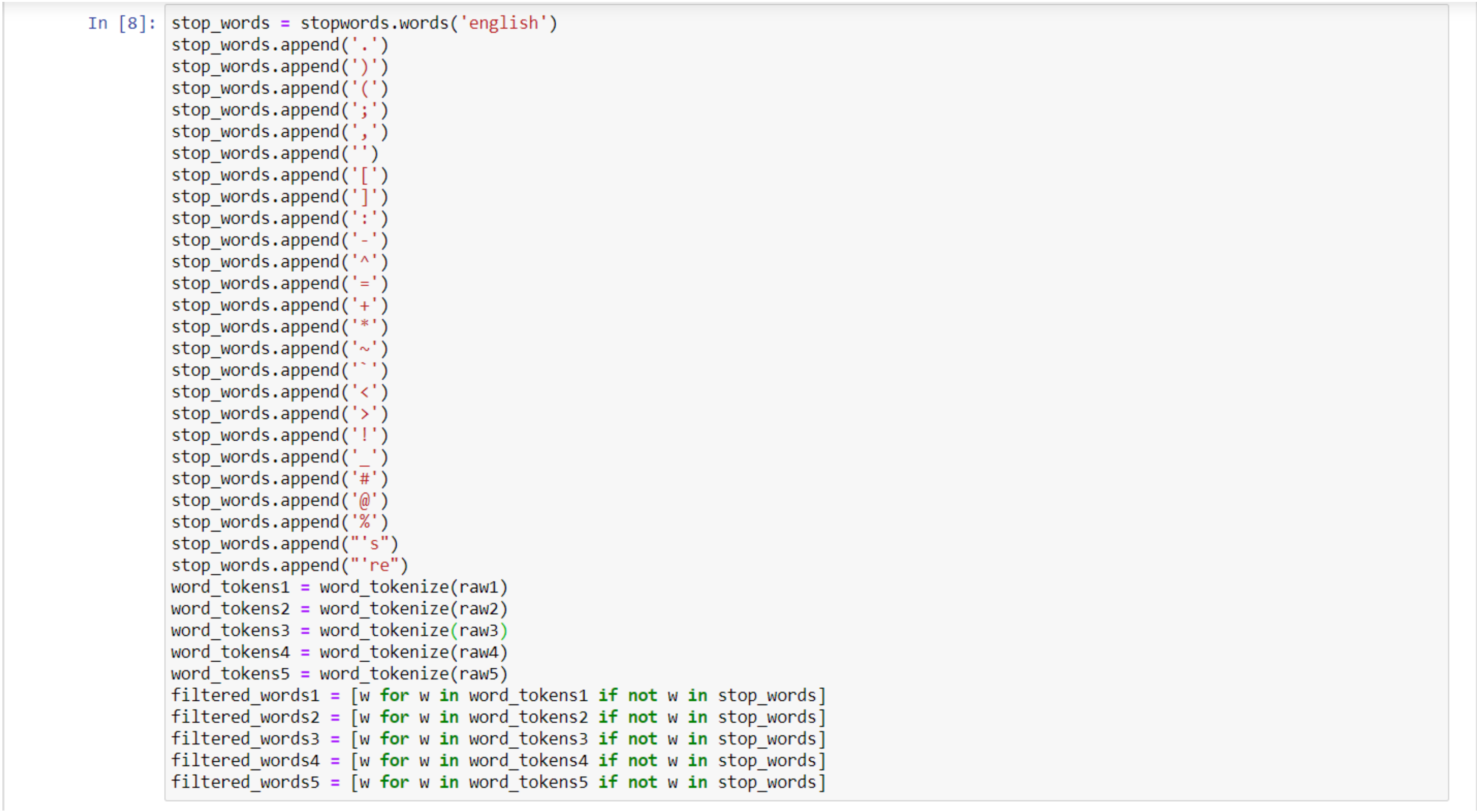
- Bag-of-Words (Document corpus)
- TF (Document corpus)

- IDF (Document corpus)
- TF-IDF (Document corpus)
- TF-IDF (Query)
- Normalized (Query)
- Normalized - TF-IDF (Document corpus)
- Cosine Similarity

- Euclidean Distance
- Document Ranking (Display Order)
- Document Similarity (Among Documents)



```
In [7]: import pandas as pd
        from nltk.corpus import stopwords
        from nltk import word_tokenize
        import urllib.request
        from bs4 import BeautifulSoup
        url1 = "https://en.wikipedia.org/wiki/Web_mining"
        url2 = "https://en.wikipedia.org/wiki/Data_mining"
        url3 = "https://en.wikipedia.org/wiki/Artificial_intelligence"
        url4 = "https://en.wikipedia.org/wiki/Machine_learning"
        url5 = "https://en.wikipedia.org/wiki/Mining"
        html1 = urllib.request.urlopen(url1).read().decode('utf8')
        html2 = urllib.request.urlopen(url2).read().decode('utf8')
        html3 = urllib.request.urlopen(url1).read().decode('utf8')
        html4 = urllib.request.urlopen(url1).read().decode('utf8')
        html5 = urllib.request.urlopen(url1).read().decode('utf8')
        raw1 = BeautifulSoup(html1, 'html.parser').get_text()
        raw2 = BeautifulSoup(html2, 'html.parser').get_text()
        raw3 = BeautifulSoup(html3, 'html.parser').get_text()
        raw4 = BeautifulSoup(html4, 'html.parser').get_text()
        raw5 = BeautifulSoup(html5, 'html.parser').get_text()
```



```
In [8]: stop_words = stopwords.words('english')
        stop_words.append('.')
        stop_words.append(')')
        stop_words.append('(')
        stop_words.append(';')
        stop_words.append(',')
        stop_words.append('')
        stop_words.append('[')
        stop_words.append(']')
        stop_words.append(':')
        stop_words.append('-')
        stop_words.append('^')
        stop_words.append('=')
        stop_words.append('+')
        stop_words.append('*')
        stop_words.append('~')
        stop_words.append('`')
        stop_words.append('<')
        stop_words.append('>')
        stop_words.append('!')
        stop_words.append('_')
        stop_words.append('#')
        stop_words.append('@')
        stop_words.append('%')
        stop_words.append("'s")
        stop_words.append("'re")
        word_tokens1 = word_tokenize(raw1)
        word_tokens2 = word_tokenize(raw2)
        word_tokens3 = word_tokenize(raw3)
        word_tokens4 = word_tokenize(raw4)
        word_tokens5 = word_tokenize(raw5)
        filtered_words1 = [w for w in word_tokens1 if not w in stop_words]
        filtered_words2 = [w for w in word_tokens2 if not w in stop_words]
        filtered_words3 = [w for w in word_tokens3 if not w in stop_words]
        filtered_words4 = [w for w in word_tokens4 if not w in stop_words]
        filtered_words5 = [w for w in word_tokens5 if not w in stop_words]
```

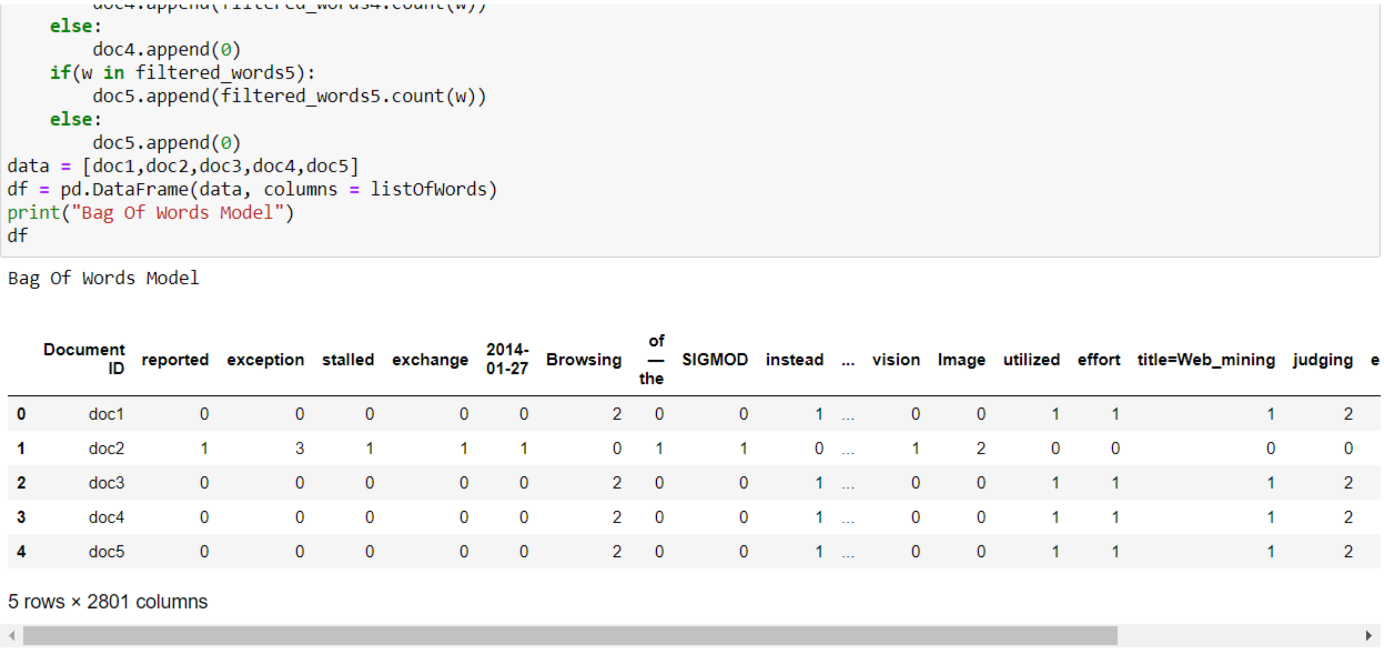
In [9]:

```
from nltk.stem import WordNetLemmatizer
lemmatizer = WordNetLemmatizer()
lemmatized = []
for w in filtered_words1:
    w = lemmatizer.lemmatize(w)
for w in filtered_words2:
    w = lemmatizer.lemmatize(w)
for w in filtered_words3:
    w = lemmatizer.lemmatize(w)
for w in filtered_words4:
    w = lemmatizer.lemmatize(w)
for w in filtered_words5:
    w = lemmatizer.lemmatize(w)
```

In [13]:

```
set1 = set(filtered_words1)
set2 = set(filtered_words2)
set3 = set(filtered_words3)
set4 = set(filtered_words4)
set5 = set(filtered_words5)
totalSet = set1|set2|set3|set4|set5
totalList = list(totalSet)
listOfWords = ['Document ID']
for i in range(len(totalList)):
    listOfWords.append(totalList[i])
doc1 = ['doc1']
doc2 = ['doc2']
doc3 = ['doc3']
doc4 = ['doc4']
doc5 = ['doc5']
for w in listOfWords:
    if(w == "Document ID"):
        continue
    if (w in filtered_words1):
        doc1.append(filtered_words1.count(w))
    else:
        doc1.append(0)
    if (w in filtered_words2):
        doc2.append(filtered_words2.count(w))
    else:
        doc2.append(0)
    if (w in filtered_words3):
        doc3.append(filtered_words3.count(w))
    else:
        doc3.append(0)
    if (w in filtered_words4):
        doc4.append(filtered_words4.count(w))
    else:
        doc4.append(0)
    if(w in filtered_words5):
        doc5.append(filtered_words5.count(w))
    else:
        doc5.append(0)
```

Output-



```
        doc4.append(filtered_words4.count(w))
    else:
        doc4.append(0)
    if(w in filtered_words5):
        doc5.append(filtered_words5.count(w))
    else:
        doc5.append(0)
data = [doc1,doc2,doc3,doc4,doc5]
df = pd.DataFrame(data, columns = listOfWords)
print("Bag Of Words Model")
df
```

```
Bag Of Words Model
```

Out[13]:

	Document ID	reported	exception	stalled	exchange	2014-01-27	Browsing	of — the	SIGMOD	instead	...	vision	Image	utilized	effort	title=Web_mining	judging	e
0	doc1	0	0	0	0	0	2	0	0	1	...	0	0	1	1	1	2	
1	doc2	1	3	1	1	1	0	1	1	0	...	1	2	0	0	0	0	
2	doc3	0	0	0	0	0	2	0	0	1	...	0	0	1	1	1	2	
3	doc4	0	0	0	0	0	2	0	0	1	...	0	0	1	1	1	2	
4	doc5	0	0	0	0	0	2	0	0	1	...	0	0	1	1	1	2	

5 rows × 2801 columns