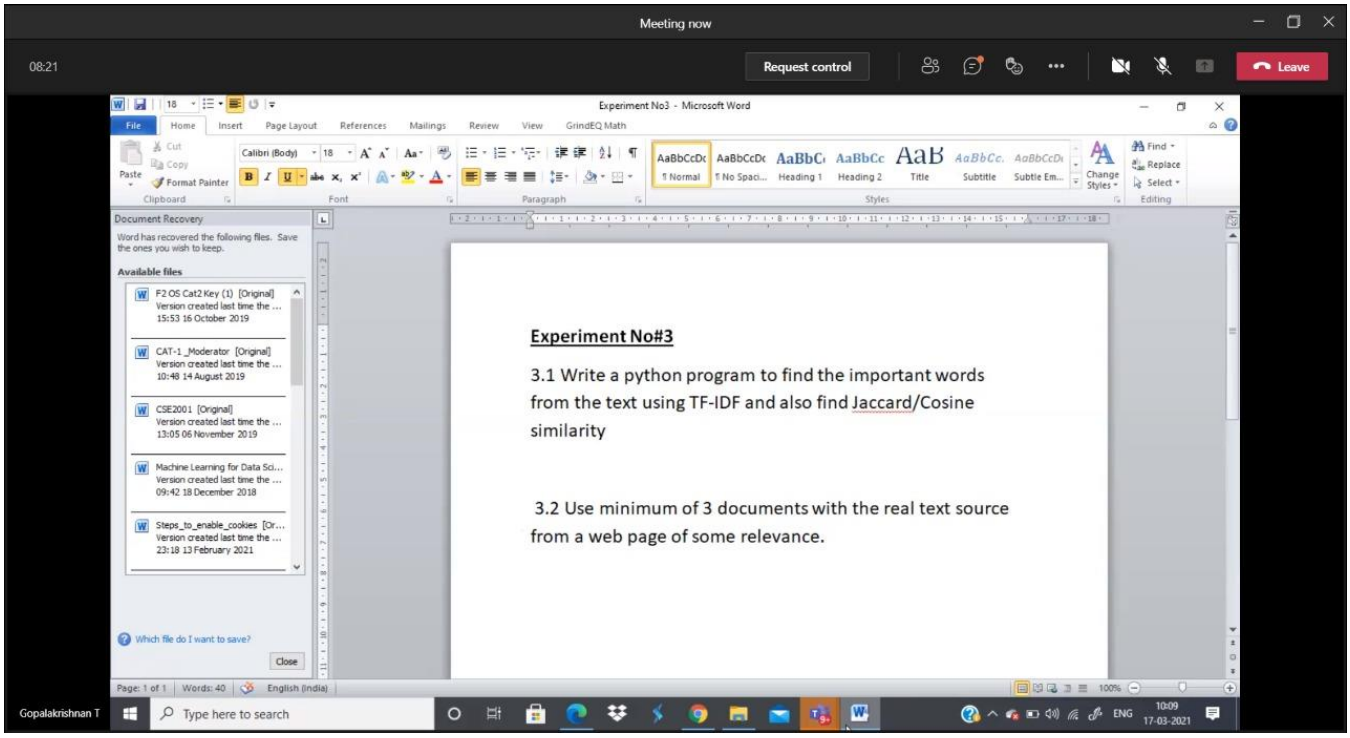


Experiment 3:

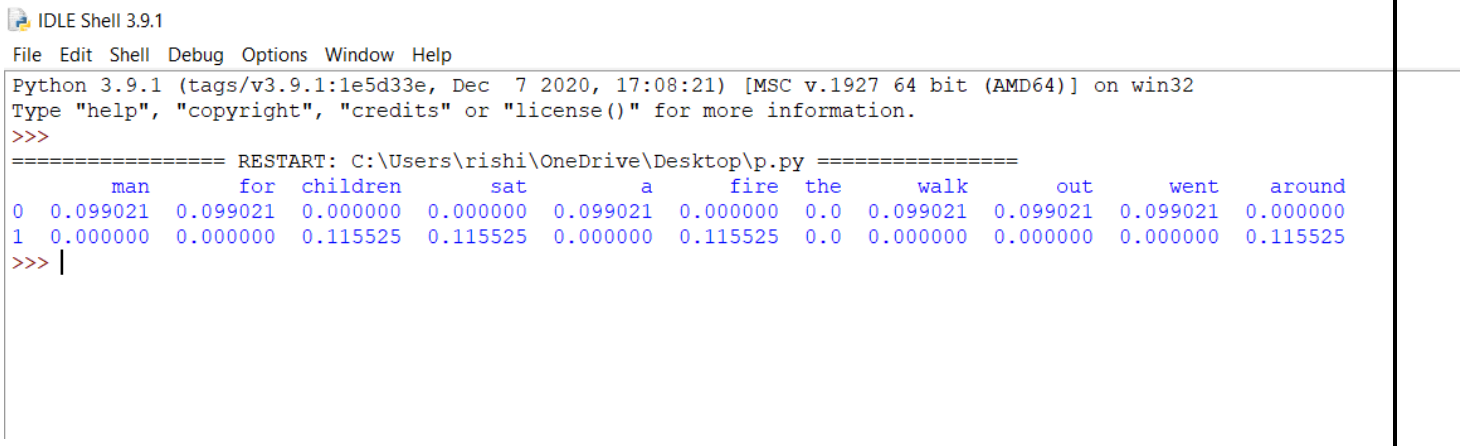
Q1.(a)TF-IDF , Jaccardian and Cosine Similarity using python.

Solution:- (TF-IDF)

```
import pandas as pd

documentA = 'the man went out for a walk'
documentB = 'the children sat around the fire'
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1


def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict


tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)
```

```
def computeIDF(documents):
    import math
    N = len(documents)

    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict


idfs = computeIDF([numOfWordsA, numOfWordsB])


def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf


tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
print(
    df.to_string())  # Here the '0' row represents the TF-IDF of Document 1 and '1'
row represents the TF-IDF of Document 2.
```

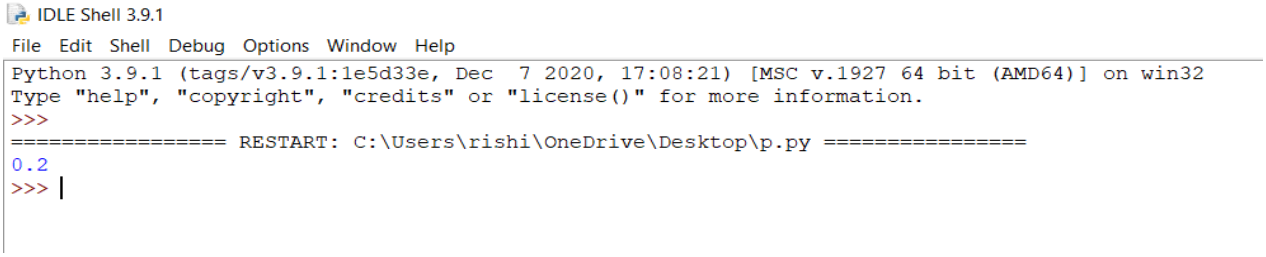
Output:



Solution: (Jaccardian Similarity)

```
str1="AI is our friend and it has been friendly"
str2="AI and humans have always been friendly"
a = set(str1.split())
b = set(str2.split())
c = a.intersection(b)
print(len(a))
print(len(b))
print(len(c))
print((len(c)/(len(a)+len(b)+len(c))))
```

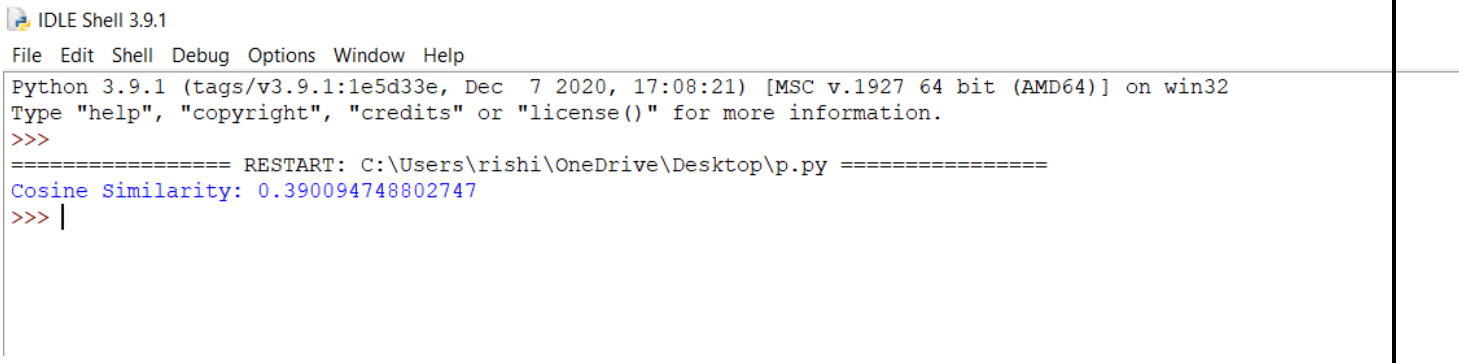
Output:



Solution: (Cosine Similarity)

```
import math
import re
from collections import Counter
WORD = re.compile(r"\w+")
def get_cosine(vec1, vec2):
    intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])
    sum1 = sum([vec1[x] ** 2 for x in list(vec1.keys())])
    sum2 = sum([vec2[x] ** 2 for x in list(vec2.keys())])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
return float(numerator) / denominator
def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)
text1 = "Web mining is the application of data mining techniques to discover
patterns from the World Wide Web"
text2 = "As the name proposes, this is information gathered by mining the web"
vector1 = text_to_vector(text1)
vector2 = text_to_vector(text2)
cosine = get_cosine(vector1, vector2)
print("Cosine Similarity:", cosine)
```

Output:



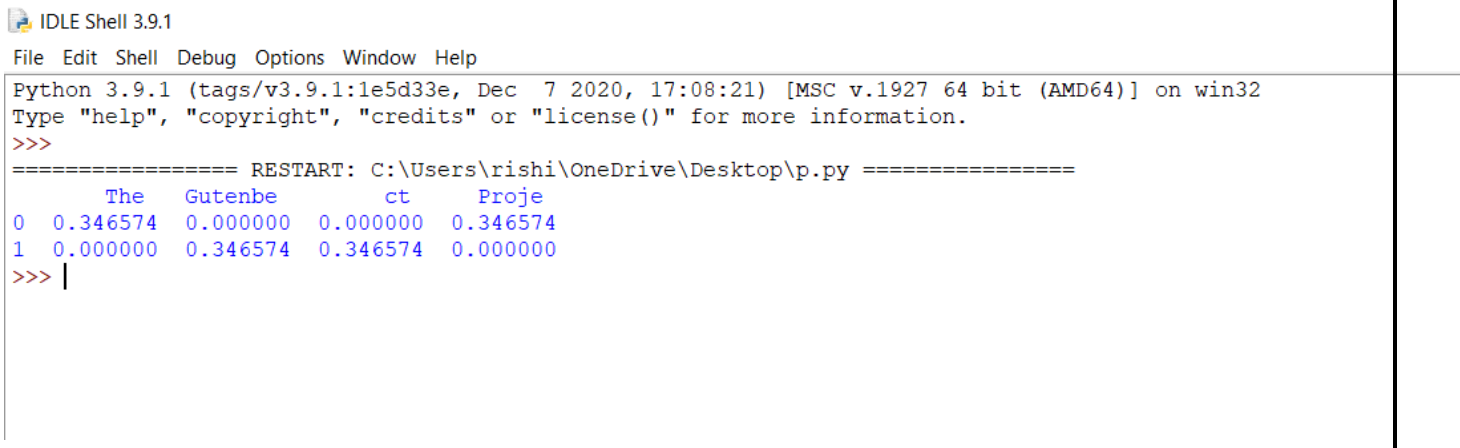
Q1(b). Implement TF-IDF , Jaccardian and Cosine Similarity using python on webpages.

Solution: (TF-IDF)

```
import pandas as pd
from urllib import request

url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
documentA = raw[:10]
documentB = raw[10:20]
bagOfWordsA = documentA.split(' ')
bagOfWordsB = documentB.split(' ')
uniqueWords = set(bagOfWordsA).union(set(bagOfWordsB))
numOfWordsA = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsA:
    numOfWordsA[word] += 1
numOfWordsB = dict.fromkeys(uniqueWords, 0)
for word in bagOfWordsB:
    numOfWordsB[word] += 1


def computeTF(wordDict, bagOfWords):
    tfDict = {}
    bagOfWordsCount = len(bagOfWords)
    for word, count in wordDict.items():
        tfDict[word] = count / float(bagOfWordsCount)
    return tfDict


tfA = computeTF(numOfWordsA, bagOfWordsA)
tfB = computeTF(numOfWordsB, bagOfWordsB)


def computeIDF(documents):
    import math
    N = len(documents)
    idfDict = dict.fromkeys(documents[0].keys(), 0)
    for document in documents:
        for word, val in document.items():
            if val > 0:
                idfDict[word] += 1

    for word, val in idfDict.items():
        idfDict[word] = math.log(N / float(val))
    return idfDict


idfs = computeIDF([numOfWordsA, numOfWordsB])


def computeTFIDF(tfBagOfWords, idfs):
    tfidf = {}
    for word, val in tfBagOfWords.items():
        tfidf[word] = val * idfs[word]
    return tfidf


tfidfA = computeTFIDF(tfA, idfs)
tfidfB = computeTFIDF(tfB, idfs)
df = pd.DataFrame([tfidfA, tfidfB])
print(
```

```
    df.to_string())  # Here the '0' row represents the TF-IDF of Document 1 and '1'
row represents the TF-IDF of Document 2.
```

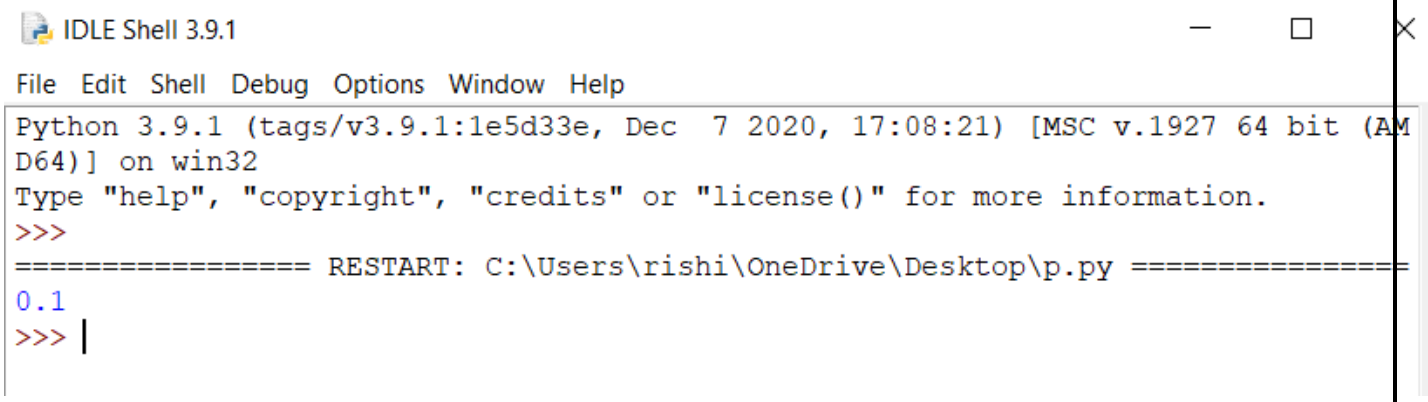
Output:



Solution: (Jaccardian Similarity)

```
from urllib import request
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
str1=raw[:10]
str2=raw[:40]
a = set(str1.split())
b = set(str2.split())
c = a.intersection(b)
print((len(c)/(len(a)+len(b)+len(c))))
```

Output:



Solution: (Cosine Similarity)

```
import math
import re
from collections import Counter
from urllib import request
WORD = re.compile(r"\w+")
def get_cosine(vec1, vec2):
    intersection = set(vec1.keys()) & set(vec2.keys())
    numerator = sum([vec1[x] * vec2[x] for x in intersection])
    sum1 = sum([vec1[x] ** 2 for x in list(vec1.keys())])
    sum2 = sum([vec2[x] ** 2 for x in list(vec2.keys())])
    denominator = math.sqrt(sum1) * math.sqrt(sum2)
    if not denominator:
        return 0.0
    else:
        return float(numerator) / denominator
def text_to_vector(text):
    words = WORD.findall(text)
    return Counter(words)
url = "http://www.gutenberg.org/files/2554/2554-0.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
text1 = raw[:10]
text2 = raw[:40]
vector1 = text_to_vector(text1)
vector2 = text_to_vector(text2)
cosine = get_cosine(vector1, vector2)
print("Cosine Similarity:", cosine)
```

Output: