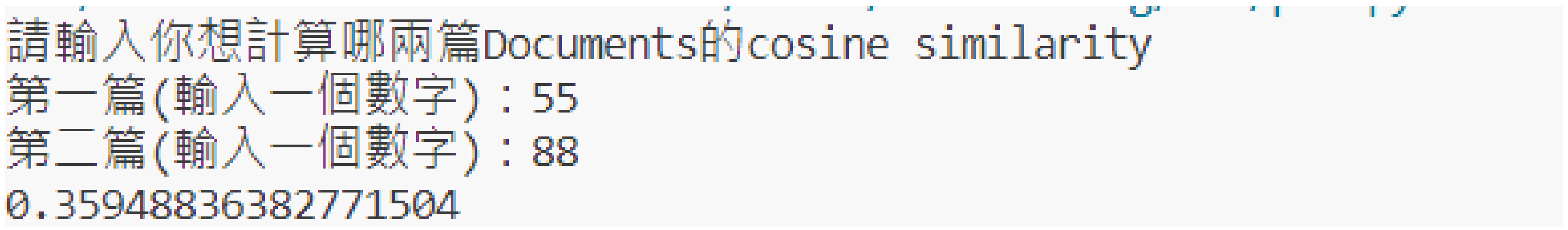
Programming Assignment 2

執行環境: Visual Studio Code
程式語言:Python 3.11.5

資管所碩一 R12725048 楊喻妃

執行方式

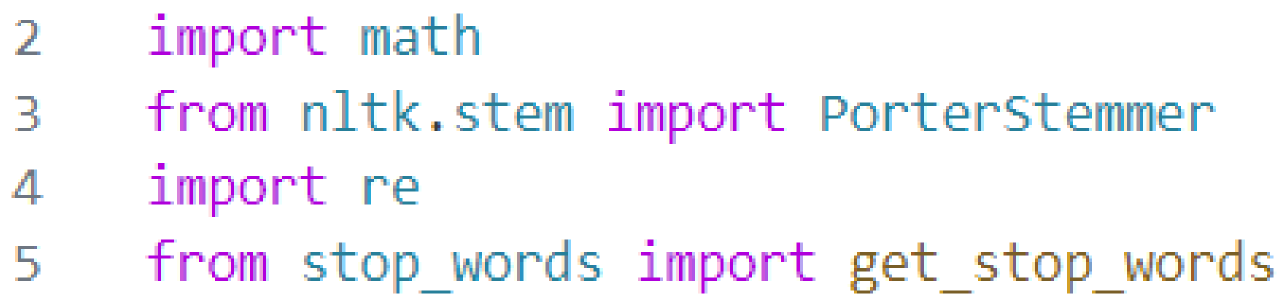
- 使用VS code跑pa2.py檔
- 需要下載的套件有:
  - `pip install stop_words`:刪除不太帶有資訊的單詞所需
  - `pip install nltk`: 使用Porter’s algorithm.所需
- 直接按全部執行即可
- 第三題會需要user輸入兩個數字，分別是想計算兩篇文章的id



```
請輸入你想計算哪兩篇Documents的cosine similarity
第一篇(輸入一個數字)：55
第二篇(輸入一個數字)：88
0.3594883638277150４
```

處理邏輯

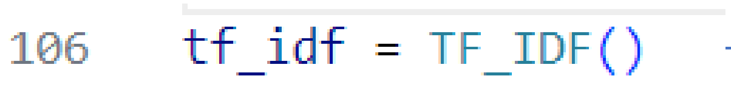
- Step 1 : import 所需套件



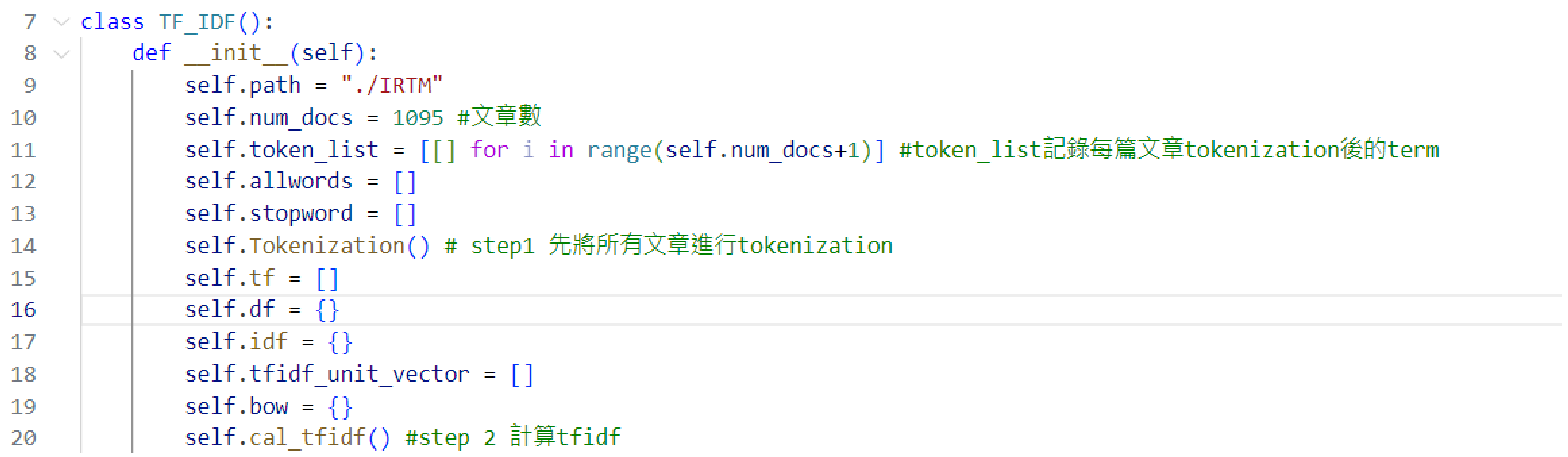
```python
import math
from nltk.stem import PorterStemmer
import re
from stop_words import get_stop_words
```

處理邏輯

- Step 2 : 建立class TF_TFIDF()，並建立其object : tf_idf
  將tf_idf的attributes&methods初始化



```python
tf_idf = TF_IDF()
```

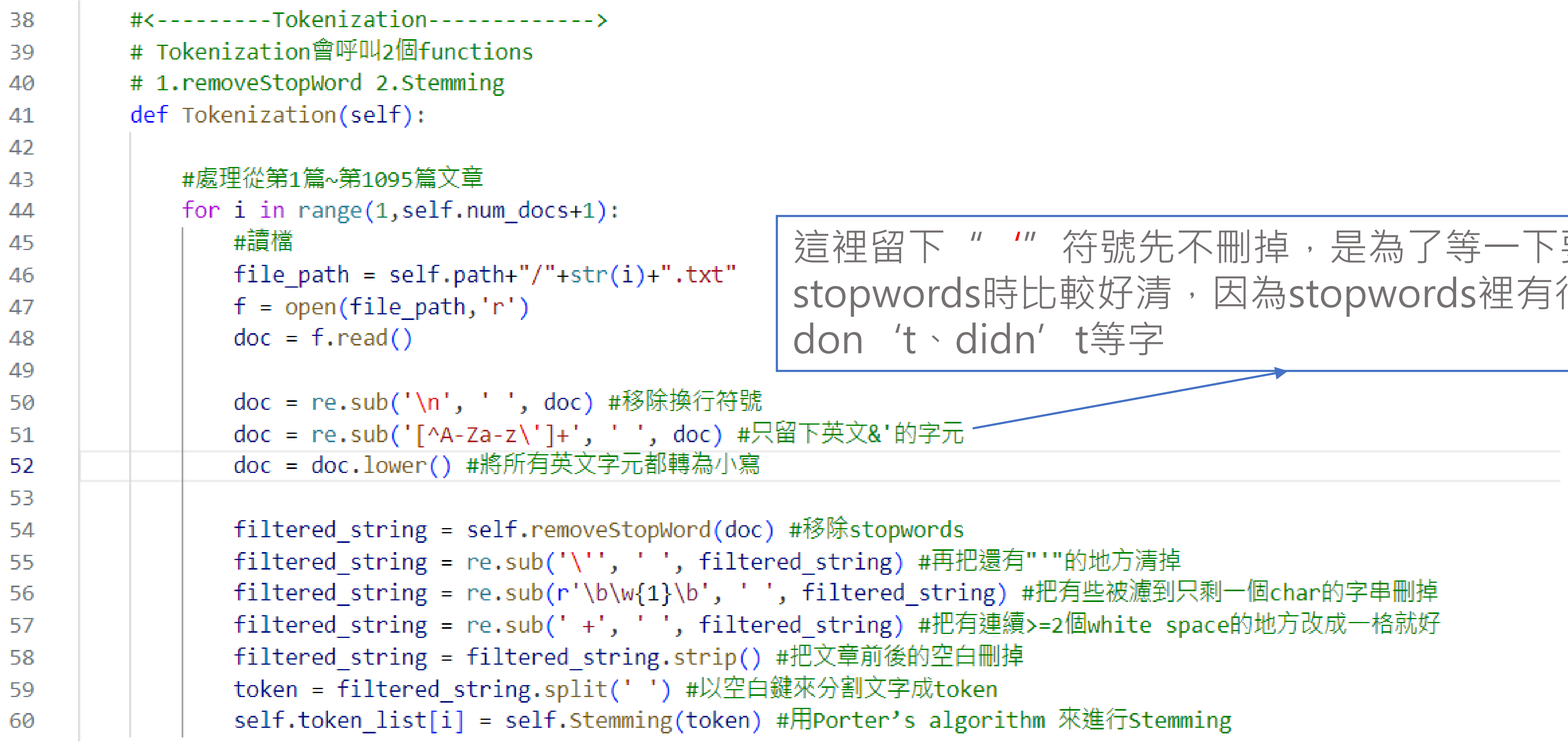


```python
class TF_IDF():
    def __init__(self):
        self.path = "./IRTM"
        self.num_docs = 1095 #文章數
        self.token_list = [[] for i in range(self.num_docs+1)] #token_list記錄每篇文章tokenization後的term
        self.allwords = []
        self.stopword = []
        self.Tokenization() # step1 先將所有文章進行tokenization
        self.tf = []
        self.df = {}
        self.idf = {}
        self.tfidf_unit_vector = []
        self.bow = {}
        self.cal_tfidf() #step 2 計算tfidf
```

處理邏輯

將一些換行符號、逗點等等以re.sub() function先清除掉
再將剩下的字元全部轉換成小寫
最後用split的方式做tokenize並存入token_list中

- Step 3 : Tokenization



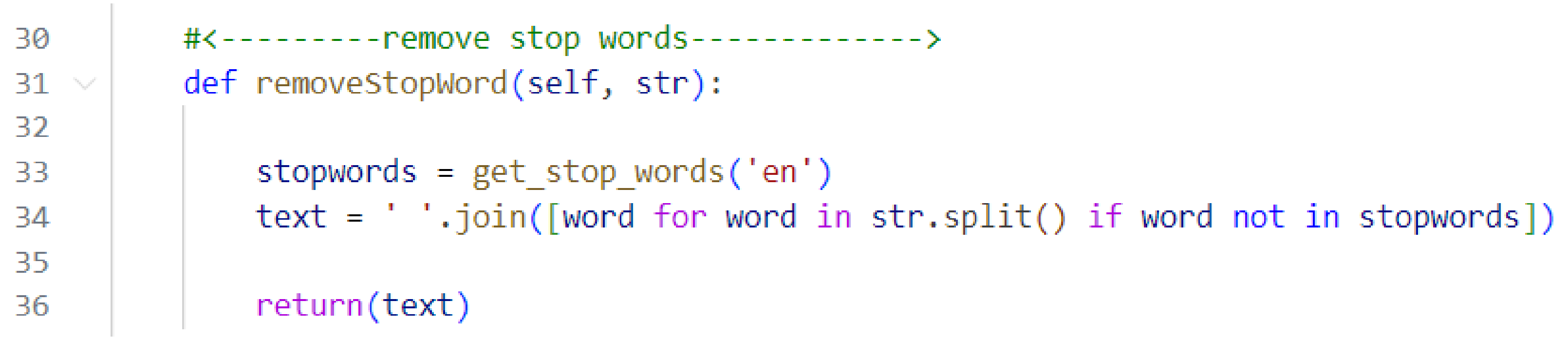
```
    #<---------Tokenization------------->
    # Tokenization會呼叫2個functions
    # 1.removeStopWord 2.Stemming
    def Tokenization(self):

        #處理從第1篇~第1095篇文章
        for i in range(1,self.num_docs+1):
            #讀檔
            file_path = self.path+"/"+str(i)+".txt"
            f = open(file_path,'r')
            doc = f.read()

            doc = re.sub('\n', ' ', doc) #移除換行符號
            doc = re.sub('[^A-Za-z\']+', ' ', doc) #只留下英文&'的字元
            doc = doc.lower() #將所有英文字元都轉為小寫

            filtered_string = self.removeStopWord(doc) #移除stopwords
            filtered_string = re.sub('\'', ' ', filtered_string) #再把還有"'"的地方清掉
            filtered_string = re.sub(r'\b\w{1}\b', ' ', filtered_string) #把有些被濾到只剩一個char的字串刪掉
            filtered_string = re.sub(' +', ' ', filtered_string) #把有連續>=2個white space的地方改成一格就好
            filtered_string = filtered_string.strip() #把文章前後的空白刪掉
            token = filtered_string.split(' ') #以空白鍵來分割文字成token
            self.token_list[i] = self.Stemming(token) #用Porter’s algorithm 來進行Stemming
```

這裡留下 “ ‘” 符號先不刪掉，是為了等一下要移除stopwords時比較好清，因為stopwords裡有很多像是don ‘t、didn’ t等字

處理邏輯

- Step 3(1) : Stopwords removal

將參數str中含有stopwords的字過濾掉，再傳回去

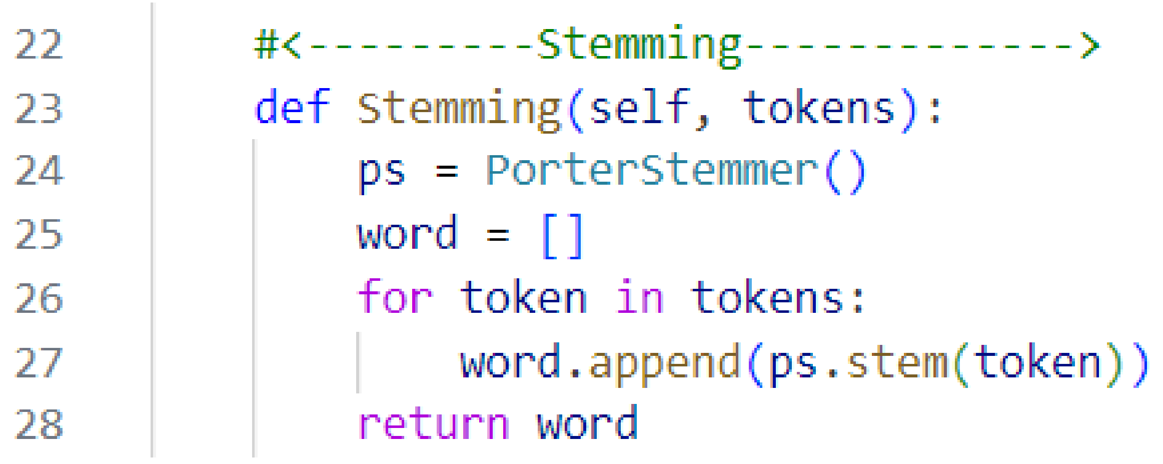


```python
    #<---------remove stop words------------->
    def removeStopWord(self, str):

        stopwords = get_stop_words('en')
        text = ' '.join([word for word in str.split() if word not in stopwords])

        return(text)
```

處理邏輯

- Step 3(2) : Stemming using Porter’ s algorithm

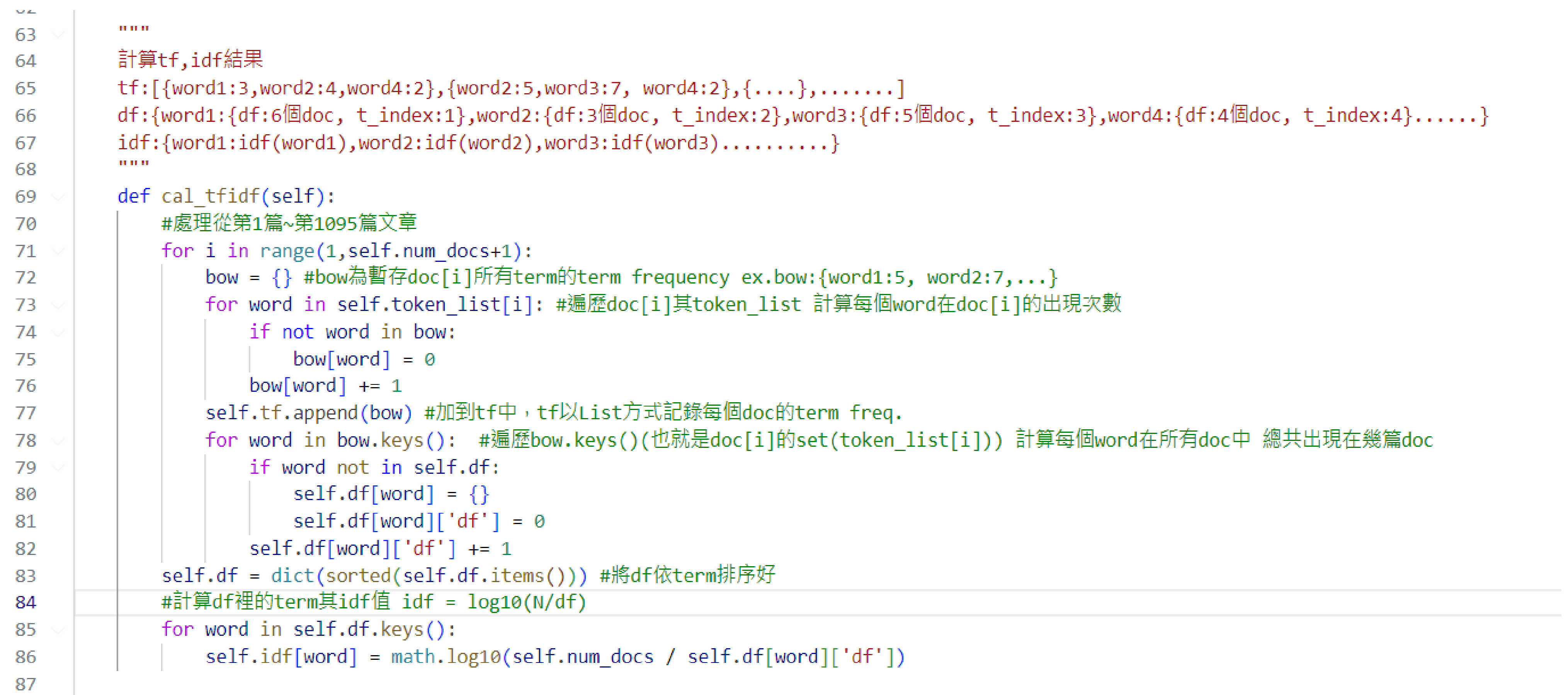
利用PorterStemmer套件進行stemming
並將處理完的token放入word，以List的格式回傳
最後stemming完的字會被放到token_list[i]中，而i為document id



```python
    #<---------Stemming------------->
    def Stemming(self, tokens):
        ps = PorterStemmer()
        word = []
        for token in tokens:
            word.append(ps.stem(token))
        return word
```

處理邏輯

- Step 4 : 計算tf、df、idf

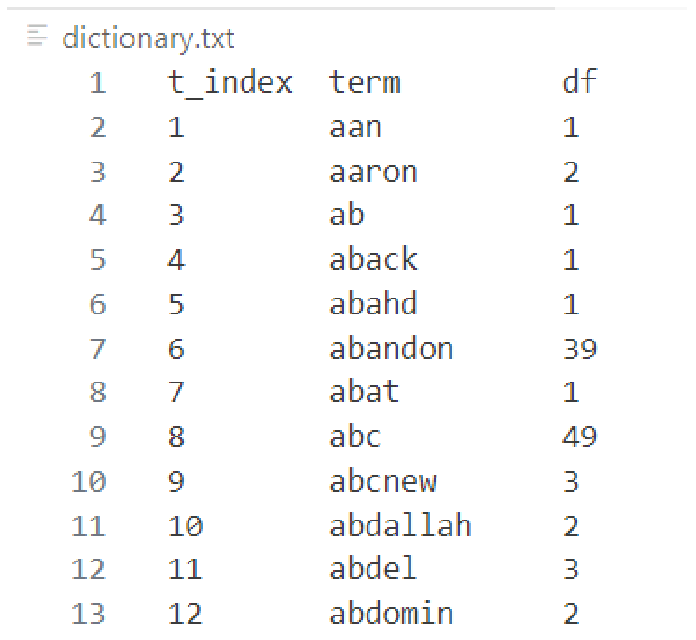


```python
    """
    計算tf,idf結果
    tf:[{word1:3,word2:4,word4:2},{word2:5,word3:7, word4:2},{....},.......]
    df:{word1:{df:6個doc, t_index:1},word2:{df:3個doc, t_index:2},word3:{df:5個doc, t_index:3},word4:{df:4個doc, t_index:4}......}
    idf:{word1:idf(word1),word2:idf(word2),word3:idf(word3).........}
    """
    def cal_tfidf(self):
        #處理從第1篇~第1095篇文章
        for i in range(1,self.num_docs+1):
            bow = {} #bow為暫存doc[i]所有term的term frequency ex.bow:{word1:5, word2:7,...}
            for word in self.token_list[i]: #遍歷doc[i]其token_list 計算每個word在doc[i]的出現次數
                if not word in bow:
                    bow[word] = 0
                bow[word] += 1
            self.tf.append(bow) #加到tf中，tf以List方式記錄每個doc的term freq.
            for word in bow.keys():  #遍歷bow.keys()(也就是doc[i]的set(token_list[i])) 計算每個word在所有doc中 總共出現在幾篇doc
                if word not in self.df:
                    self.df[word] = {}
                    self.df[word]['df'] = 0
                self.df[word]['df'] += 1
        self.df = dict(sorted(self.df.items())) #將df依term排序好
        #計算df裡的term其idf值 idf = log10(N/df)
        for word in self.df.keys():
            self.idf[word] = math.log10(self.num_docs / self.df[word]['df'])
```

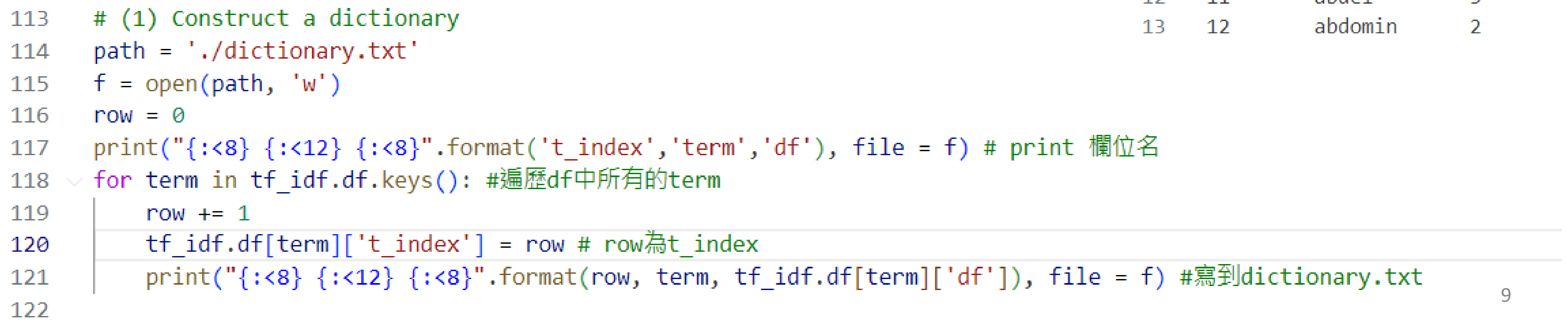
處理邏輯

- Step 5 : 回答第一題 建立dictionary.txt

Result:

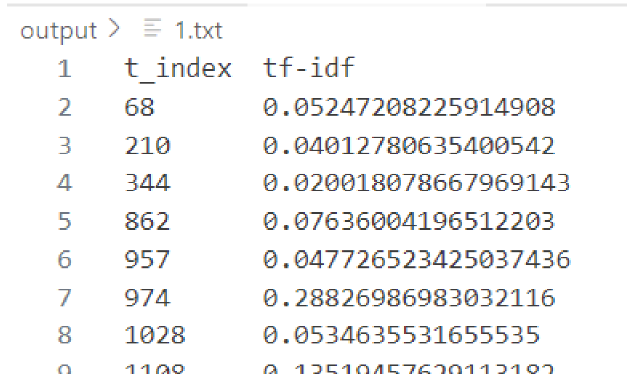

dictionary.txt

```
t_index  term         df
1        aan          1
2        aaron        2
3        ab           1
4        aback        1
5        abahd        1
6        abandon      39
7        abat         1
8        abc          49
9        abcnew       3
10       abdallah     2
11       abdel        3
12       abdomin      2
```



```python
# (1) Construct a dictionary
path = './dictionary.txt'
f = open(path, 'w')
row = 0
print("{:<8} {:<12} {:<8}".format('t_index','term','df'), file = f) # print 欄位名
for term in tf_idf.df.keys(): #遍歷df中所有的term
    row += 1
    tf_idf.df[term]['t_index'] = row # row為t_index
    print("{:<8} {:<12} {:<8}".format(row, term, tf_idf.df[term]['df']), file = f) #寫到dictionary.txt

```

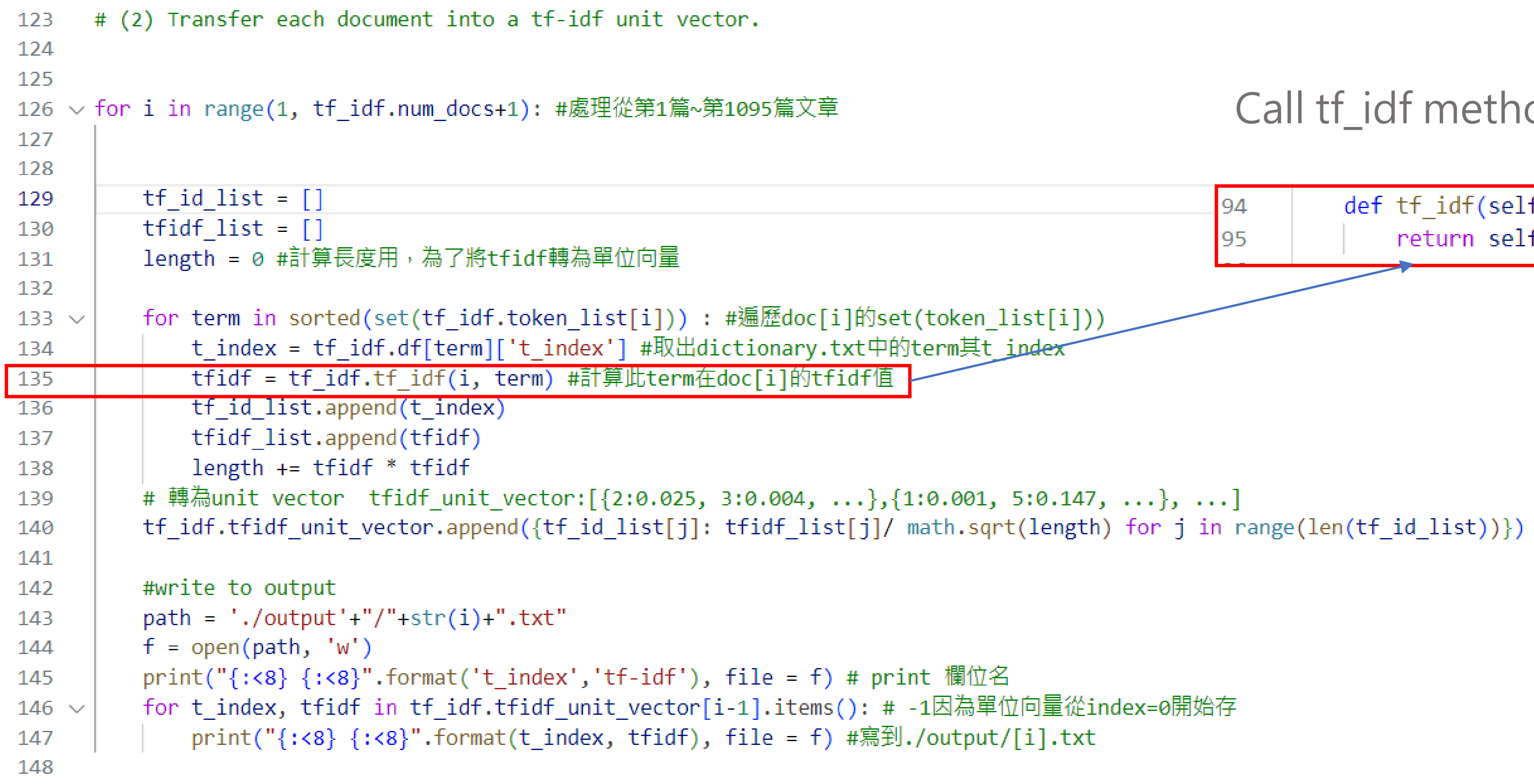
Result:


output > 1.txt

```
t_index  tf-idf
68       0.05247208225914908
210      0.04012780635400542
344      0.020018078667969143
862      0.07636004196512203
957      0.047726523425037436
974      0.28826986983032116
1028     0.0534635531655535
1108     0.13519457629113182
```

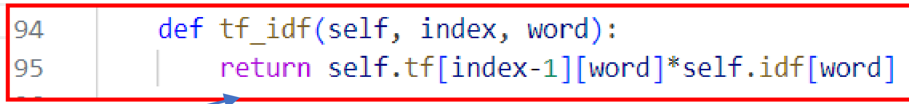
處理邏輯

- Step 6 : 回答第二題 建立DocID.txt 計算tfidf單位向量



```python
# (2) Transfer each document into a tf-idf unit vector.


for i in range(1, tf_idf.num_docs+1): #處理從第1篇~第1095篇文章


    tf_id_list = []
    tfidf_list = []
    length = 0 #計算長度用，為了將tfidf轉為單位向量

    for term in sorted(set(tf_idf.token_list[i])) : #遍歷doc[i]的set(token_list[i]))
        t_index = tf_idf.df[term]['t_index'] #取出dictionary.txt中的term其t_index
        tfidf = tf_idf.tf_idf(i, term) #計算此term在doc[i]的tfidf值
        tf_id_list.append(t_index)
        tfidf_list.append(tfidf)
        length += tfidf * tfidf
    # 轉為unit vector  tfidf_unit_vector:[{2:0.025, 3:0.004, ...},{1:0.001, 5:0.147, ...}, ...]
    tf_idf.tfidf_unit_vector.append({tf_id_list[j]: tfidf_list[j]/ math.sqrt(length) for j in range(len(tf_id_list))})

    #write to output
    path = './output'+"/"+str(i)+".txt"
    f = open(path, 'w')
    print("{:<8} {:<8}".format('t_index','tf-idf'), file = f) # print 欄位名
    for t_index, tfidf in tf_idf.tfidf_unit_vector[i-1].items(): # -1因為單位向量從index=0開始存
        print("{:<8} {:<8}".format(t_index, tfidf), file = f) #寫到./output/[i].txt
```

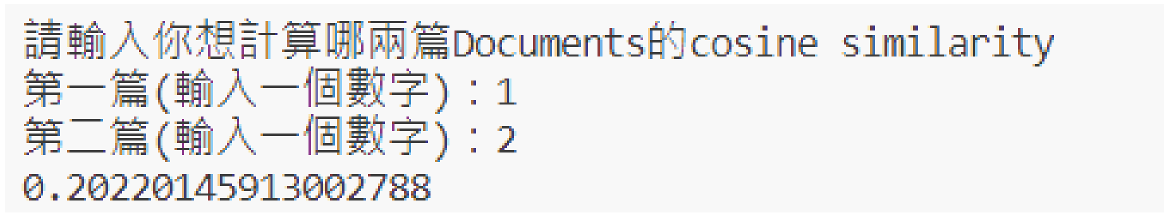
Call tf_idf method 計算tf-idf



```python
    def tf_idf(self, index, word):
        return self.tf[index-1][word]*self.idf[word]
```

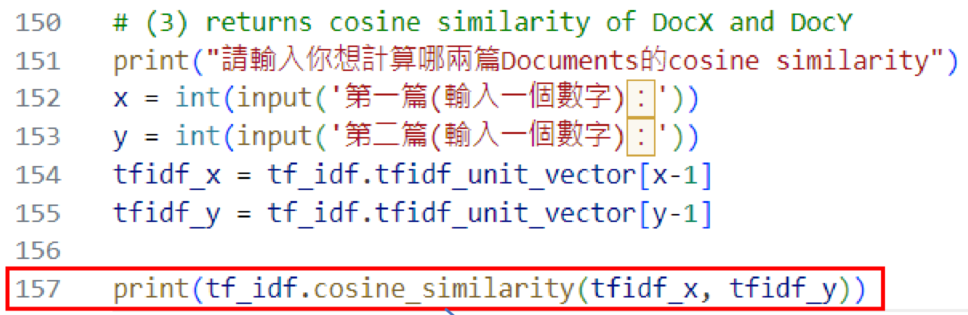
處理邏輯

Result:

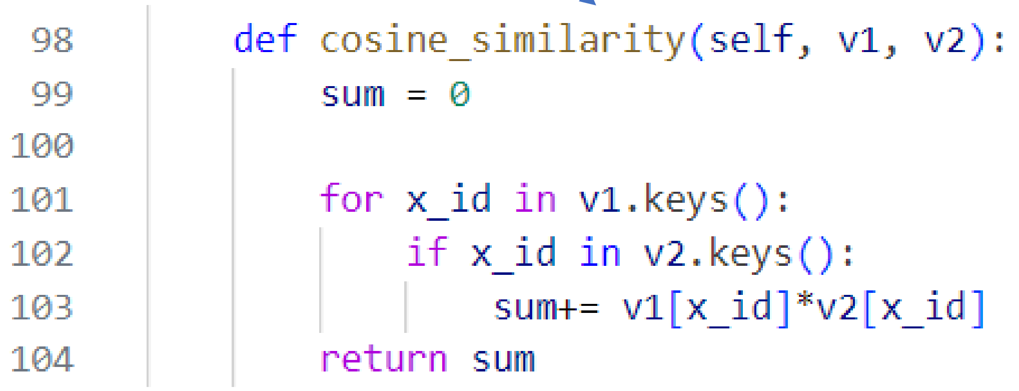


```
請輸入你想計算哪兩篇Documents的cosine similarity
第一篇(輸入一個數字)：1
第二篇(輸入一個數字)：2
0.20220145913002788
```

- Step 7 : 回答第三題 計算cosine similarity



```
# (3) returns cosine similarity of DocX and DocY
print("請輸入你想計算哪兩篇Documents的cosine similarity")
x = int(input('第一篇(輸入一個數字)：'))
y = int(input('第二篇(輸入一個數字)：'))
tfidf_x = tf_idf.tfidf_unit_vector[x-1]
tfidf_y = tf_idf.tfidf_unit_vector[y-1]

print(tf_idf.cosine_similarity(tfidf_x, tfidf_y))
```



```
    def cosine_similarity(self, v1, v2):
        sum = 0

        for x_id in v1.keys():
            if x_id in v2.keys():
                sum+= v1[x_id]*v2[x_id]
        return sum
```

Call cosine_similarity method 計算DocX&DocY的cosine similarity

邏輯:
假設DocX和DocY的tfidf unit vector 分別為下表
則只需要從將有同時出現在兩者的term其tf-idf互乘即可

t_index	tf-idf
25	0.025
55	0.15
69	0.003
102	0.155
455	0.044

DocX

t_index	tf-idf
21	0.0008
48	0.158
55	0.06
158	0.005
455	0.157

DocY

ans=
0.15*0.06+0.044+0.157