Assignment 1

Instructions:

- Please read the problem description carefully
- Make sure to complete all requirement (shown as bullets) . In general, it would be much easier if you complete the requirements in the order as shown in the problem description
- Follow the Submission Instruction to submit your assignment

Q1. Define a function to analyze the frequency of words in a string

- Define a function named "**tokenize**" which does the following:
  - has a string as an input
  - splits the string into a list of tokens by space.
    - e.g., "it's a hello world!!!" will be split into two tokens ["it's", "a","hello","world!!!"]
  - if a token starts with or ends with one or more punctuations, remove these punctuations, e.g. "world!!!" -> "world".(hint, you can import module *string*, use *string.punctuation* to get a list of punctuations (say *puncts*), and then use function *strip(puncts)* to remove leading or trailing punctuations)
  - remove the space surrounding each token
  - only keep tokens with 2 or more characters, i.e. *len*(token)>1
  - converts all tokens into lower case
  - returns all the tokens as a list output

In [201…
```
import string

def tokenize(text):

    # initialize a list
    cleaned_tokens = []

    # add your code here
    cleaned_tokens = [text.split()[i].strip(string.punctuation).lower() for i in range(len(text.split())) if len(text.split()[i])>1]

    return cleaned_tokens
```

In [202…
```
# test your code
text = """it's a Hello World!!!
          it is hello world again."""
tokenize(text)
```

Out[202…
```
["it's", 'hello', 'world', 'it', 'is', 'hello', 'world', 'again']
```

Q2. Define a class to analyze a document

- Define a new class called "**Text_Analyzer**" which does the following :
  - has two attributes:
    - **text**, which receives the string value passed by users when creating an object of this class.
    - **token_count**, which is set to {} when an object of this class is created.
  - a function named "**analyze**" that does the following:
    - calls the function "tokenize" to tokenize **text** to get a list of tokens. point)
    - creates a dictionary containing the count of every unique token, e.g. {'it': 5, 'hello':1,...}
    - saves this dictionary to the **token_count** attribute
    - return the **token_count** attribute
  - a function named "**topN**" that returns the top N words in **text** by frequency
    - has a integer parameter *N*
    - returns the top *N* words and their counts as a list of tuples, e.g. [("hello", 5), ("world", 4),...] (hint: By default, a dictionary is sorted by key. However, you need to sort the token_count dictionary by value)
- What kind of words usually have high frequency? Write your analysis.

In [215…
```
class Text_Analyzer(object):

    def __init__(self, doc):
        self.text = doc
        self.token_count = {}

    def analyze(self):
        temp={}
        temp = tokenize(self.text)
        for i in range(len(temp)):
            if temp[i] in self.token_count:
                self.token_count[temp[i]] += 1
            else:
                self.token_count[temp[i]] = 1
        return self.token_count
    def topN(self, N):
        temp_dict = {}
        temp_dict = sorted(self.token_count.items(), key=lambda x: x[-1])
        return [temp_dict[-1-i] for i in range(N)]

print("\nQ2 Analysis:\nIt appears that filler words such as is, and, a, etc tend to be the most frequent inn normal english sentences.")
```

```
Q2 Analysis:
It appears that filler words such as is, and, a, etc tend to be the most frequent inn normal english sentences.
```

In [217…

```
text = """it's a hello world!!!
           it is hello world again.
          """

# create a class object
analyzer = Text_Analyzer(text)

# show attributes after creation
print(analyzer.text)
print(analyzer.token_count)

# call analyze method
print(analyzer.analyze())

# show token_count attribute after calling analyze function
print(analyzer.token_count)

# show top N words
print(analyzer.topN(3))
```

```
it's a hello world!!!
           it is hello world again.

{}
{"it's": 1, 'hello': 2, 'world': 2, 'it': 1, 'is': 1, 'again': 1}
{"it's": 1, 'hello': 2, 'world': 2, 'it': 1, 'is': 1, 'again': 1}
[('world', 2), ('hello', 2), ('again', 1)]
```

Q3. (Bonus) Create Bigrams from a document ## (3 points)

A bigram is any pair of consecutive tokens in a document. Phrases are usually bigrams. Let's add to the class defined in Q2 a function to find phrases.

- Create a new function called "**bigram**" which does the following :
  - takes a **string** and an integer **N** as inputs
  - calls the function "tokenize" to get a list of tokens for the input string
  - slice the list to get any two consecutive tokens as a bigram. For example ["it's", "hello","world"] will generate two bigrams: [["it's", "hello"],["hello","world"]]
  - count the frequency of each unique bigram
  - return top N bigrams and their counts
- Are you able to find good phrases from the top N bigrams? Write down your analysis in a document.

In [223…

```
class Text_Analyzer(object):

    def __init__(self, doc):
        self.text = doc
        self.token_count = {}

    def analyze(self):
        temp={}
        temp = tokenize(self.text)
        for i in range(len(temp)):
            if temp[i] in self.token_count:
                self.token_count[temp[i]] += 1
            else:
                self.token_count[temp[i]] = 1
        return self.token_count

    def topN(self, N):
        temp_dict = {}
        temp_dict = sorted(self.token_count.items(), key=lambda x: x[-1])
        return [temp_dict[-1-i] for i in range(N)]

    def bigram(self, N):
        temp_dict = {}
        bi_dict = []
        token_count = tokenize(self.text)
        #print(token_count)
        for i in range(len(token_count) - 1):
            bi_dict.append(token_count[i:i+2])
        #print(str(bi_dict[2]))
        for i in range(len(bi_dict)):
            if str(bi_dict[i]) in temp_dict:
                temp_dict[str(bi_dict[i])] += 1
            else:
                temp_dict[str(bi_dict[i])] = 1
        temp_dict = sorted(temp_dict.items(), key=lambda x: x[-1])
        return [temp_dict[-1-i] for i in range(N)]

print("\nQ3 Analysis:\n Immune System was an interesting biagram that I found in the test case below.")
```

```
Q3 Analysis:
```

```
Immune System was an interesting biagram that I found in the test case below.
```

In [224…

```
text = """it's a hello world!!!
           it is hello world again.
           """

# create a class object
analyzer = Text_Analyzer(text)
analyzer.bigram(2)
```

Out[224…

```
[("['hello', 'world']", 2), ("['world', 'again']", 1)]
```

Put everything together and test using main block

When you submit, remember to remove the test code for each individual function or class above. Just keep the definitions of classes and functions

In [226…

```
# best practice to test your class
# if your script is exported as a module,
# the following part is ignored
# this is equivalent to main() in Java

if __name__ == "__main__":

    # Test Question 1
    text='''What does "immunity" really mean?
            To scientists, immunity means a resistance to a disease gained
            through the immune system’s exposure to it, either by infection
            or through vaccination. But immunity doesn’t always mean complete
            protection from the virus.

            How does the body build immunity?
            The immune system has two ways to provide lasting protection:
            T cells that remember the pathogen and trigger a rapid response,
            and B cells that produce antibodies — proteins the body makes
            to fight off a specific pathogen.
            So-called “memory T cells” also stick around. Ideally, they live up
            to their name and recognize a previously encountered pathogen
            and either help coordinate the immune system or kill infected cells.
        '''
    print("Test Question 1")
    print(tokenize(text))


    # Test Question 2
    print("\nTest Question 2")
    analyzer=Text_Analyzer(text)
    analyzer.analyze()
    print(analyzer.token_count)
    print(analyzer.topN(5))


    #3 Test Question 3
    print("\nTest Question 3")
    print(analyzer.bigram(10))
```

```
Test Question 1
['what', 'does', 'immunity', 'really', 'mean', 'to', 'scientists', 'immunity', 'means', 'resistance', 'to', 'disease', 'gained', 'through', 'th
e', 'immune', 'system’s', 'exposure', 'to', 'it', 'either', 'by', 'infection', 'or', 'through', 'vaccination', 'but', 'immunity', 'doesn’t', 'a
lways', 'mean', 'complete', 'protection', 'from', 'the', 'virus', 'how', 'does', 'the', 'body', 'build', 'immunity', 'the', 'immune', 'system',
'has', 'two', 'ways', 'to', 'provide', 'lasting', 'protection', 'cells', 'that', 'remember', 'the', 'pathogen', 'and', 'trigger', 'rapid', 'res
ponse', 'and', 'cells', 'that', 'produce', 'antibodies', 'proteins', 'the', 'body', 'makes', 'to', 'fight', 'off', 'specific', 'pathogen', 'so-
called', '“memory', 'cells”', 'also', 'stick', 'around', 'ideally', 'they', 'live', 'up', 'to', 'their', 'name', 'and', 'recognize', 'previousl
y', 'encountered', 'pathogen', 'and', 'either', 'help', 'coordinate', 'the', 'immune', 'system', 'or', 'kill', 'infected', 'cells']

Test Question 2
{'what': 1, 'does': 2, 'immunity': 4, 'really': 1, 'mean': 2, 'to': 6, 'scientists': 1, 'means': 1, 'resistance': 1, 'disease': 1, 'gained': 1,
'through': 2, 'the': 7, 'immune': 3, 'system’s': 1, 'exposure': 1, 'it': 1, 'either': 2, 'by': 1, 'infection': 1, 'or': 2, 'vaccination': 1, 'b
ut': 1, 'doesn’t': 1, 'always': 1, 'complete': 1, 'protection': 2, 'from': 1, 'virus': 1, 'how': 1, 'body': 2, 'build': 1, 'system': 2, 'has':
1, 'two': 1, 'ways': 1, 'provide': 1, 'lasting': 1, 'cells': 3, 'that': 2, 'remember': 1, 'pathogen': 3, 'and': 4, 'trigger': 1, 'rapid': 1, 'r
esponse': 1, 'produce': 1, 'antibodies': 1, 'proteins': 1, 'makes': 1, 'fight': 1, 'off': 1, 'specific': 1, 'so-called': 1, '“memory': 1, 'cell
s”': 1, 'also': 1, 'stick': 1, 'around': 1, 'ideally': 1, 'they': 1, 'live': 1, 'up': 1, 'their': 1, 'name': 1, 'recognize': 1, 'previously':
1, 'encountered': 1, 'help': 1, 'coordinate': 1, 'kill': 1, 'infected': 1}
[('the', 7), ('to', 6), ('and', 4), ('immunity', 4), ('pathogen', 3)]

Test Question 3
[("['the', 'immune']", 3), ("['pathogen', 'and']", 2), ("['cells', 'that']", 2), ("['immune', 'system']", 2), ("['the', 'body']", 2), ("['infec
ted', 'cells']", 1), ("['kill', 'infected']", 1), ("['or', 'kill']", 1), ("['system', 'or']", 1), ("['coordinate', 'the']", 1)]
```