|  |
| --- |
| **(Autonomous Institute Affiliated to VTU)**  **Department of Information Science and Engineering** |
|  |
| A Project Report on |
| **Generating Sentences**  **using N-grams** |
| *Submitted in partial fulfillment of the CIE for the subject*  **Natural Language Processing(ISEA2)** |
| By |
| **Sneha V**  **1MS16IS098**  **&**  **Shrinidhi KR**  **1MS16IS094** |
|  |
|  |
| *Under the guidance*  *of* |
| **Rajeshwari S B** |
| Assistant Professor |
| Department of ISE, MSRIT |
| Bengaluru-560054 |

**TABLE OF CONTENTS**

|  |  |  |
| --- | --- | --- |
| 1. | Idea of Project |  |
| 2. | Concepts used in Project   * Tokenizing words and sentences * Generating n-grams from given sentence |  |
| 3. | Algorithm |  |
| 4. | Output |  |
|  |  |  |
|  |  |  |
|  |  |  |

**IDEA OF THE PROJECT**

The project concentrates on generating sentences based on any given datasets which are paragraphs of related content using the concept of N-grams.

We have improvised the output of sentences generated by increasing the value of n in the N-grams.

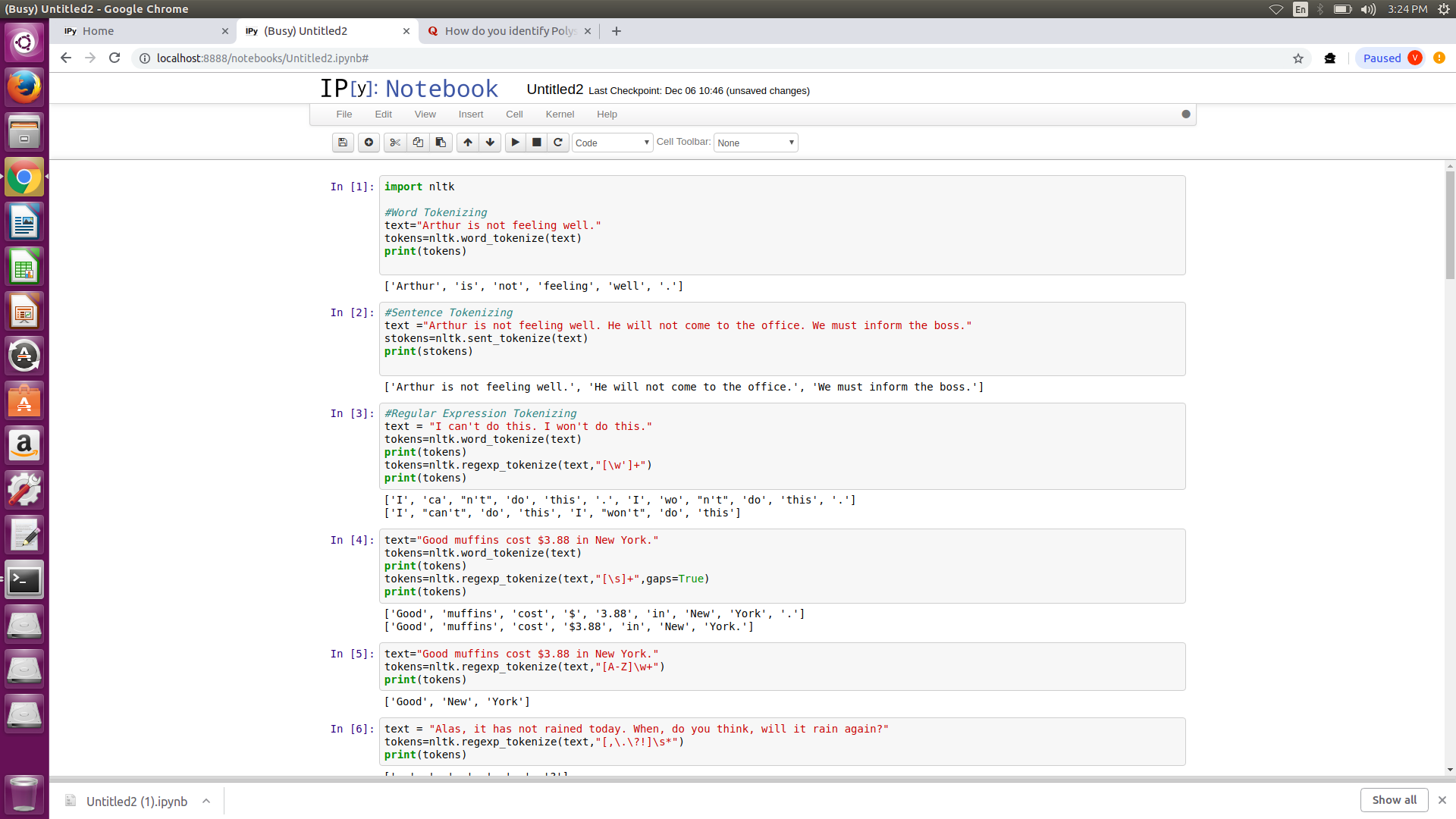
**CONCEPTS USED IN PROJECT**

* **Tokenizing Sentences And Words In NLTK**

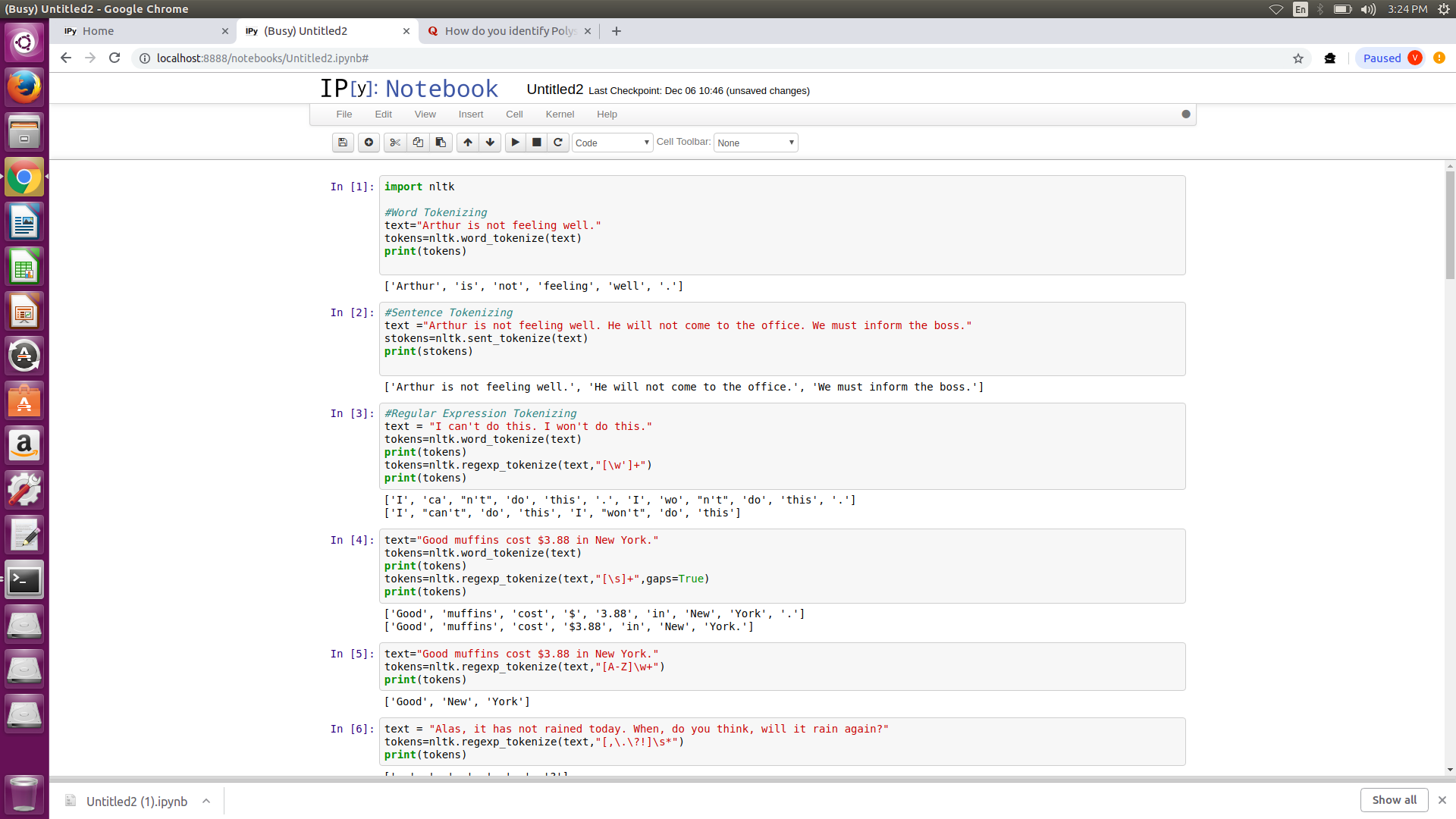
For our language processing, we want to break up the string into words and punctuation. This step is called **tokenization**, and it produces our familiar structure, a list of words and punctuation.

Python has a native tokenizer, the .split() function, which you can pass a separator and it will split the string that the function is called on on that separator. The NLTK tokenizer is more robust. It tokenizes a sentence into words and punctuation.

A sentence or data can be split into words using the method **word\_tokenize():**



A paragraph or data can be split into sentences using the method **sent\_tokenize () where ‘.’ is used to split:**



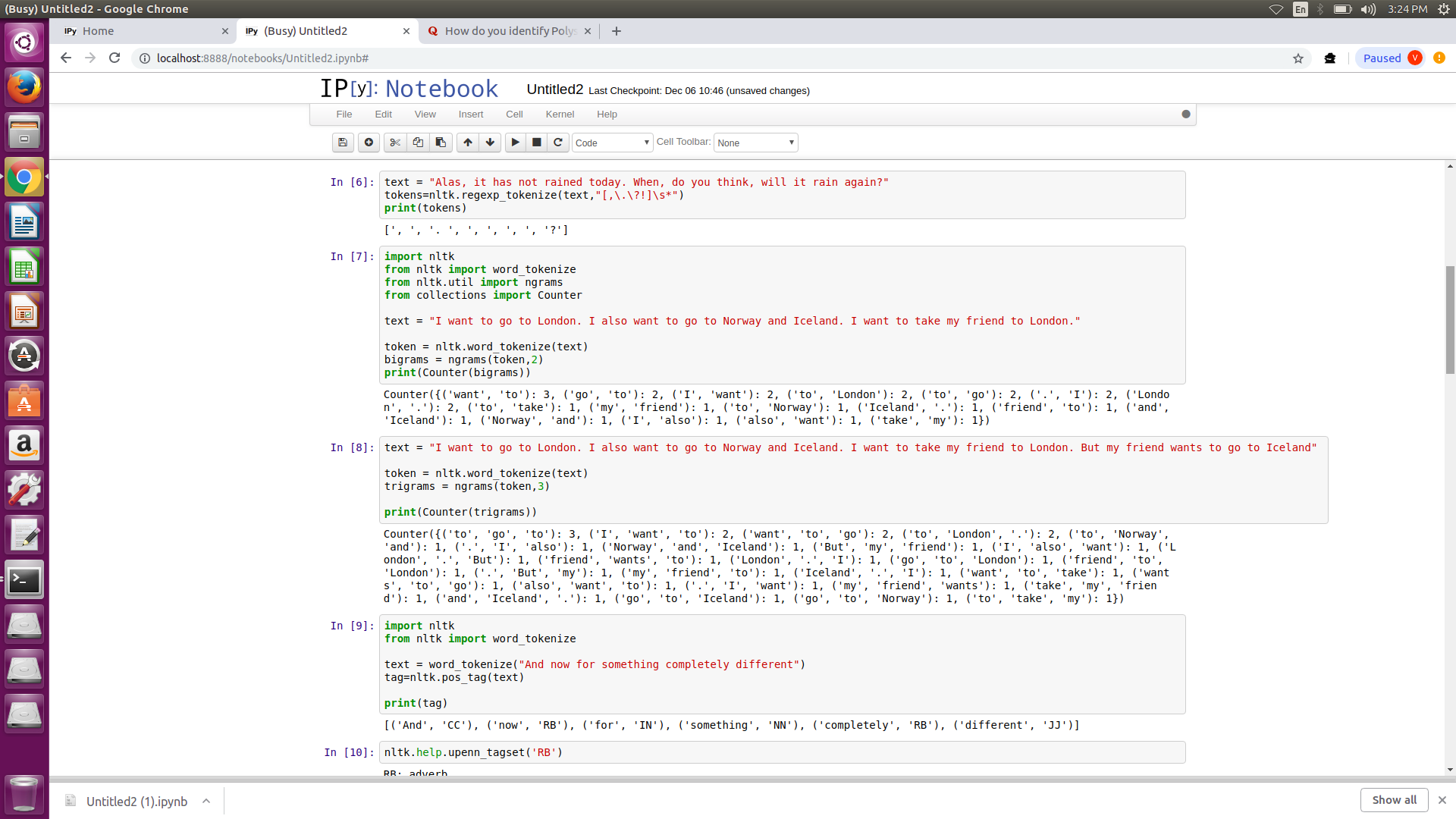
* **Generating N-Grams From Given Sentence**

N-grams are contiguous sequences of n-items in a sentence. N can be 1, 2 or any other positive integers, although usually we do not consider very large N because those n-grams rarely appears in many different places.

When performing machine learning tasks related to natural language processing, we usually need to generate n-grams from input sentences. For example, in text classification tasks, in addition to using each individual token found in the corpus, we may want to add bi-grams or tri-grams as features to represent our documents.

**Bigrams and trigrams generation along with their frequency of occurence-**

**Syntax-** ngrams(tokens,n) where n is the number of items to be sequenced

****

**ALGORITHM TO GENERATE SENTENCES**

* A Function Accepting N-1 Parameters

token = nltk.word\_tokenize(text)

Ngrams = ngrams(token,N)

def generate\_Ngrams(first\_string, second\_string, third\_string. . .(N-1)th\_string):

m=0

l=[]

for x in range(0, len(firstwords of Ngrams)):

if firstword of Ngrams == first\_string and secondword == second\_string and thirdword==third\_string and so on till

(N-1)thword == (N-1)th\_string:

if countNgrams >m:

del l[:]

m = count(Ngrams)

l.append(Nth\_word)

elif countNgrams==m:

m = countngrams[x]

l.append(Nth\_word)

randomno=random.randint(0,len(l)-1)

predict=l[randomno]

// shifting of Ngrams leads to printing the second string

print(second\_string)

return second\_string,third\_string. . . nth\_string

l1=”hello” //you can set the beginning word of the sentence

while f!= 2://here 2 is the number of sentences to generate until fullstop

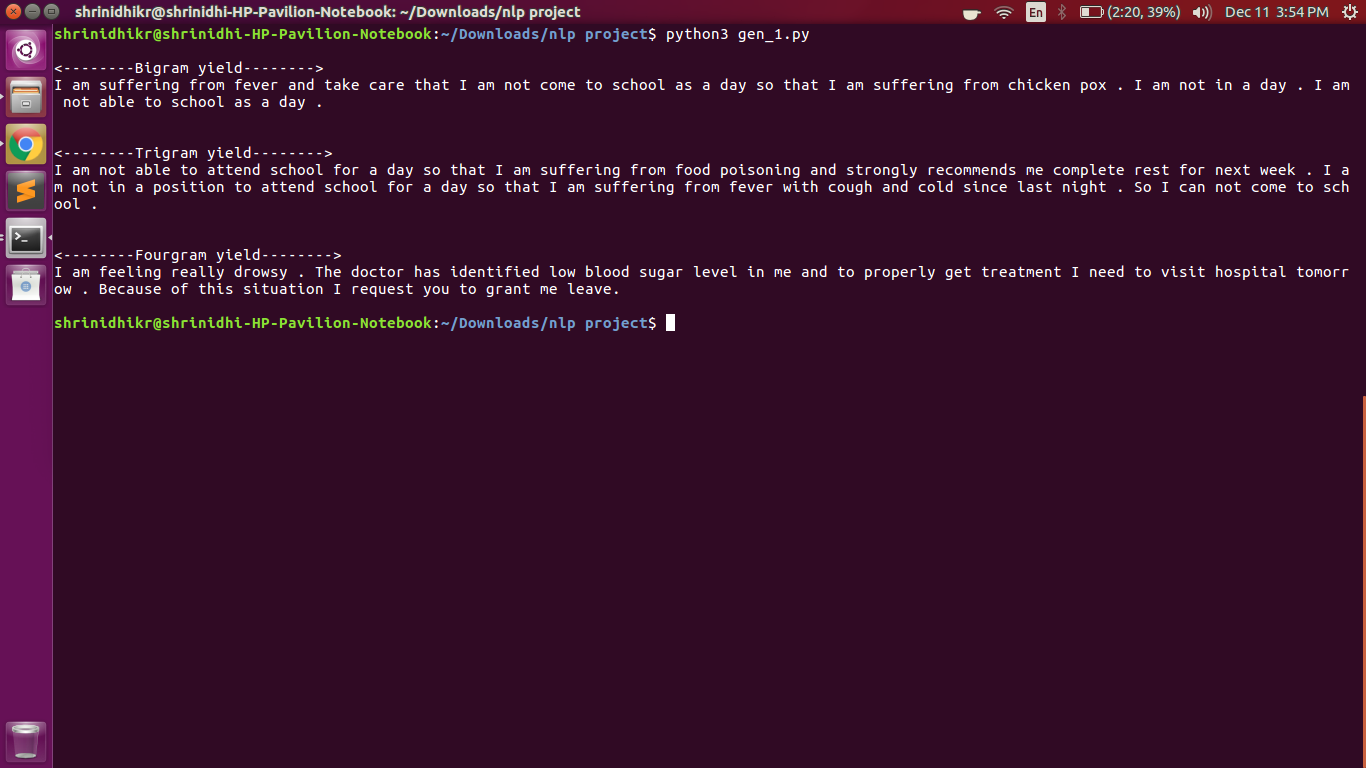
l2,l3,l4…lN= generate\_fourgrams(l2,l3,l4…lN)

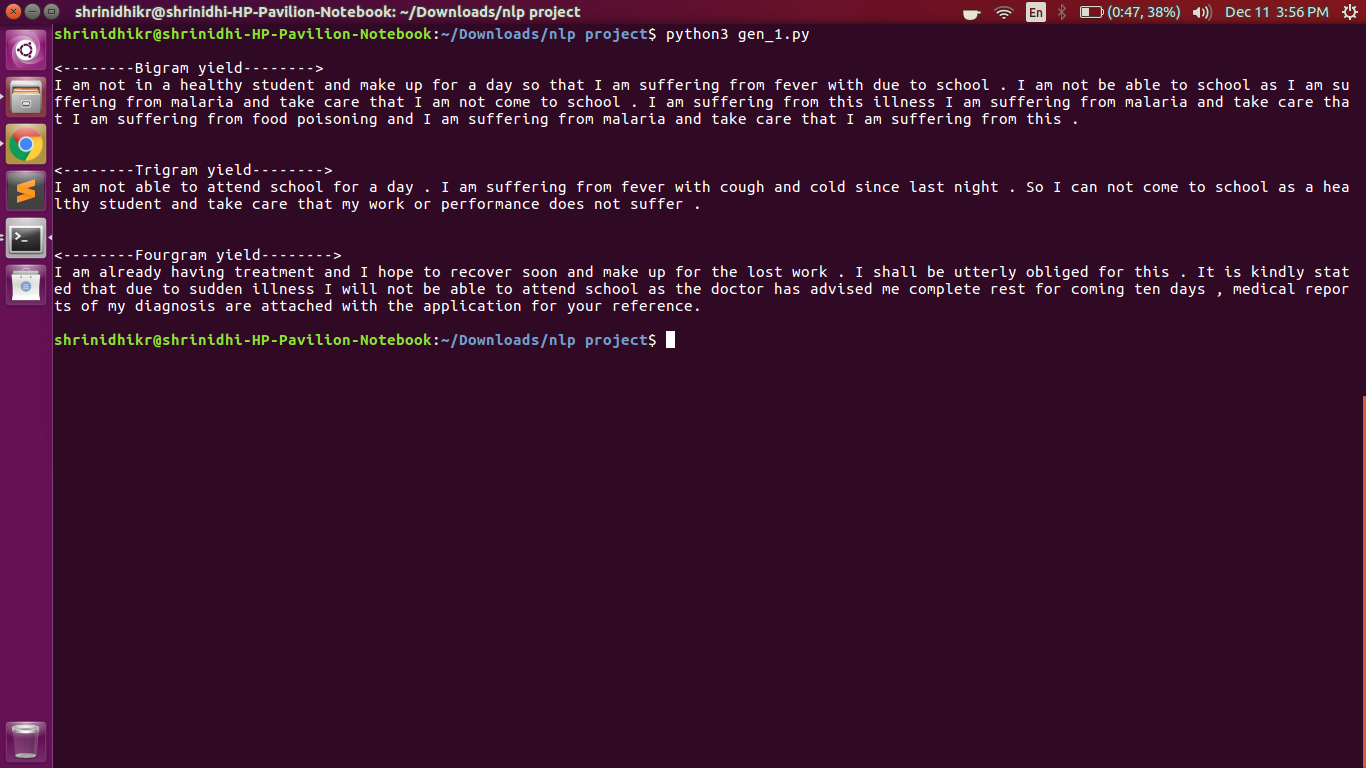
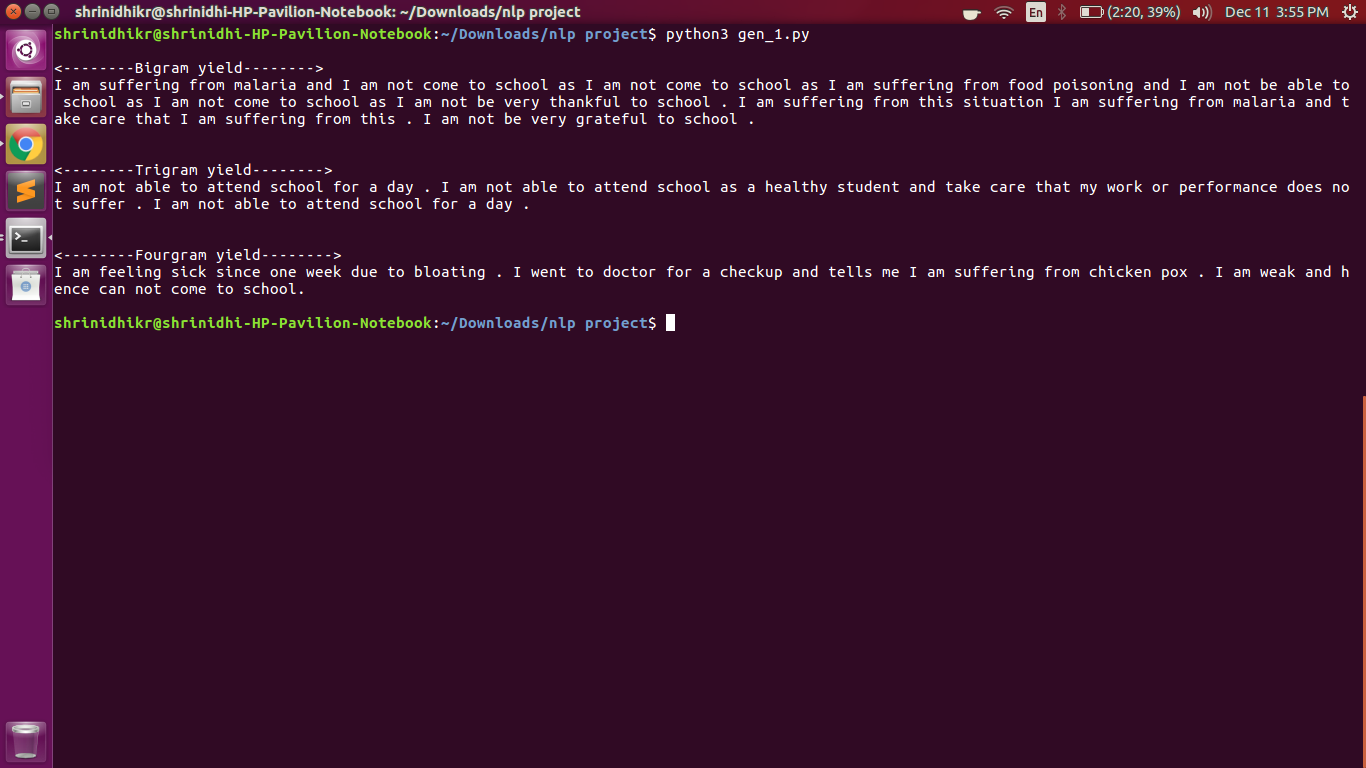
if lN == ".":

f = f+1

**OUTPUT**

The given dataset is a collection of templates of sick leave letters.

****

****

**BIBLIOGRAPHY**

* <http://www.nltk.org/_modules/nltk/model/ngram.html>

**THANK YOU**