Topic	CHATBOT PHILOSOPHY	
Class Description	**The student will learn to preprocess text for creating an AI-based chatbot.**	
Class	**PRO C119**	
Class time	**45 mins**	
Goal	● Explain AI Based Chatbot Philosophy. ● Learn about NLTK to pre-process text.	
Resources Required	● Teacher Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen ○ Smartphone ● Student Resources: ○ Laptop with internet connectivity ○ Earphones with mic ○ Notebook and pen	
Class structure	**Warm-Up** **Teacher-led Activity 1** **Student-led Activity 1** **Wrap-Up**	**10 mins** **10 mins** **20 mins** **05 mins**
Credit	NLTK by Team NLTK under Apache License Version 2.0.	
WARM-UP SESSION - 10 mins		
Teacher Starts Slideshow **Slide 1 to 5** Refer to speaker notes and follow the instructions on each slide.		

Teacher Action	Student Action
Hey <student's name>. How are you? It's great to see you! Can you tell me what we learned in the previous class? ***Note**: Encourage the student to give answers and be more involved in the discussion.* Amazing! We created a digital diary called 'Digi-Diary'. Also, we were able to save and display diary entries on our webpage. Are you excited to learn something new today? **Following are the WARM-UP session deliverables:** ● Greet the student. ● Revision of previous class activities. ● Quizzes.	**ESR**: Hi, thanks! **ESR**: We created a digital diary. Click on the slide show tab and present the slides
WARM-UP QUIZ Click on In-Class Quiz	
Continue WARM-UP Session **Slide 6 to 13**	
Activity Details **Following are the session deliverables:** ● Appreciate the student. ● Narrate the story by using hand gestures and voice modulation methods to bring in more interest in students.	
Have you used Google assistant or Siri for IOS?	**ESR:** Yes

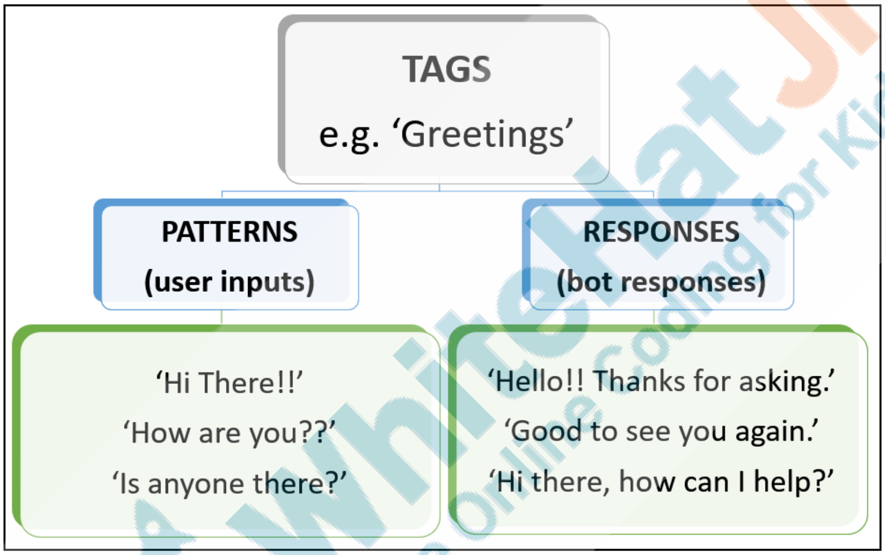
What happens when we search for anything using the Google assistant? Similarly, on some websites, you may see a chat box popping up. Using these chat options, you can get instant help for your queries. We may think that there is some person sitting on the other side to solve your queries. But this is not the case. Programmers have pre-fed data or answers in the system in order to give answers according to questions asked by the user. This is known as a **chatbot**. A chatbot is basically a computer program that can communicate with users either by chat or voice. Chatbots interpret and process users' words and give instant answers. These answers are defined in the program. Refer link for sample chatbot:	**ESR:** It is used to search anything on the internet using our voice. **ESR:** Yes

  Henna's Saloon Aa Hi	
So in today's class, we will learn the philosophy of creating a chatbot. This chatbot will respond to text inputs. Today we'll be creating a chatbot for casual talk. It will give suggestions to spend our day depending upon our hobbies or interests. Similar to text sentiment analysis the text data will be preprocessed and given to the chatbot. So before taking input from the user we need to create a file to store the expected user input and the corresponding response.	

Are you excited to learn about it?	
Teacher Ends Slideshow	
Teacher Initiates Screen Share	
ACTIVITY ● **Introduction to NLTK.** ● **Stemming and Bag of words for creating a chatbot.**	
Teacher Action	**Student Action**
The process of creating a dataset for the chatbot is done in two stages: ● First, define the training data set file. This is done using the **intents.json** file. ● Second, **preprocessing** the json file. The preprocessing is done in a Python file named **data_preprocessing .py.** ***Note:*** *Open Teacher Activity1 for boilerplate code.* Rather than looking at the full sentence entered by the user, only important words or keywords are considered. We need keywords and the respective intents to create a chatbot using an Intent matching algorithm. For example: '**I like to go for a walk**' Chatbot reply: '**You can visit a park today**'. Here, 'like' and 'walk' can be the **keywords** for which chatbot gives rply to go to the park.	

In **intents.json** a dictionary is created with keys and values.

‘**tags**’ represent the **category** for the desired question.
‘patterns’ are the **expected inputs** or **questions** from users
‘responses’ are the list of **corresponding responses** or **answers** for the questions falling under that particular tag.



TAGS
e.g. ‘Greetings’

PATTERNS
(user inputs)

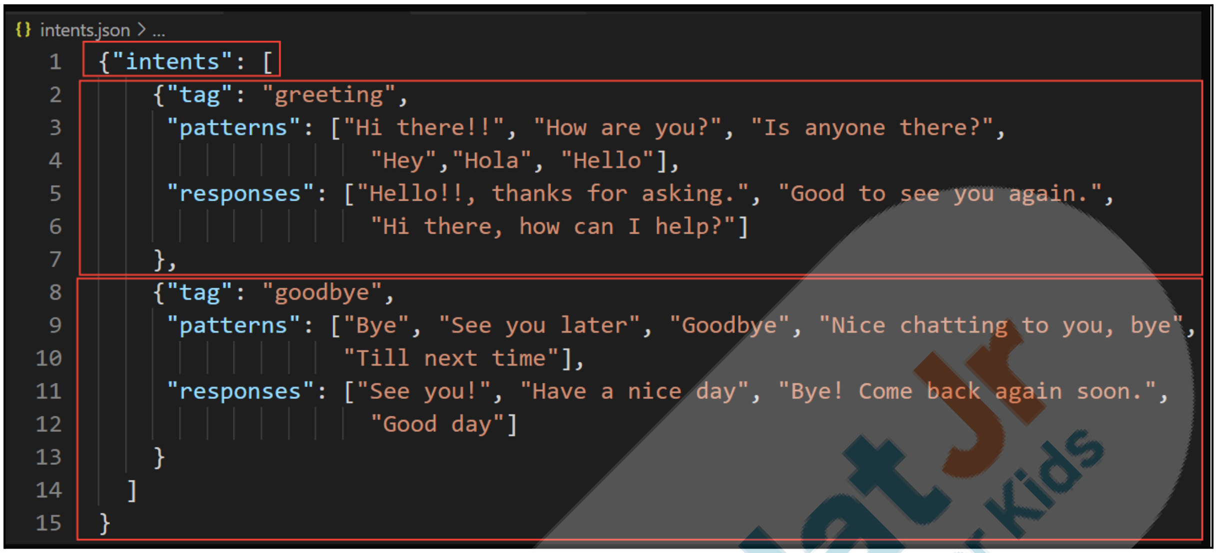
RESPONSES
(bot responses)

‘Hi There!!’
‘How are you??’
‘Is anyone there?’

‘Hello!! Thanks for asking.’
‘Good to see you again.’
‘Hi there, how can I help?’

This will act as our training dataset.

Note: *The teacher can create only two intents and ask the student to create two more such as how can the chatbot help the user, what all suggestions we can get from chatbot etc..*



{} intents.json > ...

```
{"intents": [
        {"tag": "greeting",
         "patterns": ["Hi there!!", "How are you?", "Is anyone there?",
                         "Hey","Hola", "Hello"],
         "responses": ["Hello!!, thanks for asking.", "Good to see you again.",
                         "Hi there, how can I help?"]
        },
        {"tag": "goodbye",
         "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye",
                       "Till next time"],
         "responses": ["See you!", "Have a nice day", "Bye! Come back again soon.",
                         "Good day"]
        }
   ]
}
```

The next step is to create a Python file with the name **data_preprocessing** . This file is responsible for performing **Stemming operation** and **creating Bag of Words.**

Note: *The student will learn Stemming and Bag of words as we move ahead with this class.*

Let's create a python file for preprocessing of this text data. This Python file and JSON file should be in the same folder.

1. Create a **data_preprocessing** file. Let's import all the libraries required to preprocess our data.
2. Start by importing **nltk**.

NLTK is also known as the **Natural Language Toolkit.** It is the Python library used for preprocessing text data. It has methods for cleaning the data and removing repetitive words.

Teacher Activity 2: NLTK

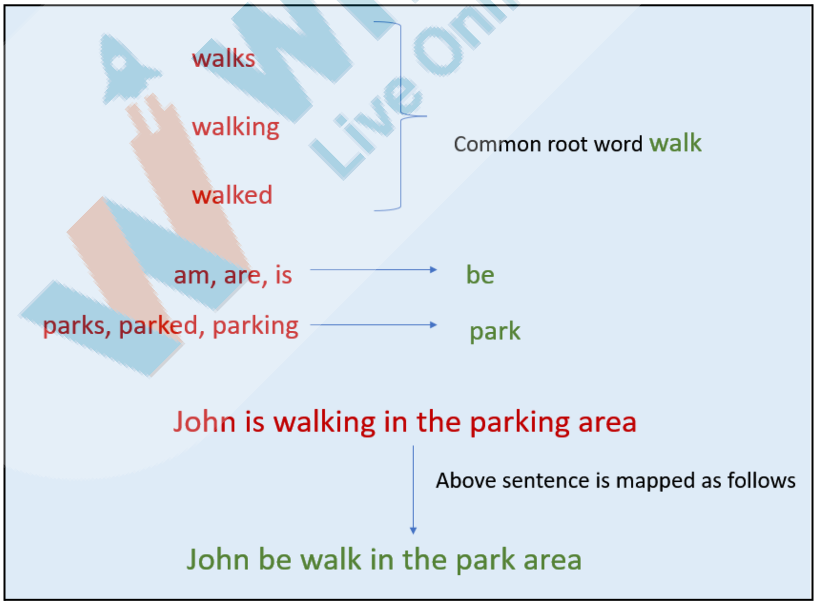
Next, from NLTK we'll import the **PorterStemmer()** class. This class is responsible to give the stem words for given words.
Let's learn the concept of stem words and stemming now.

Stemming is the process of reducing words to their root forms. It maps a group of words to the same stem even if the stem itself is not a valid word.
For example, walks, walking, walked come from the root word walk.

Teacher Activity 3: PorterStemmer

Similarly, all the words in a sentence are reduced to their root words and then the mapping of the sentence is done as shown in the image below.

This way it becomes easier to compare the sentences and give response.

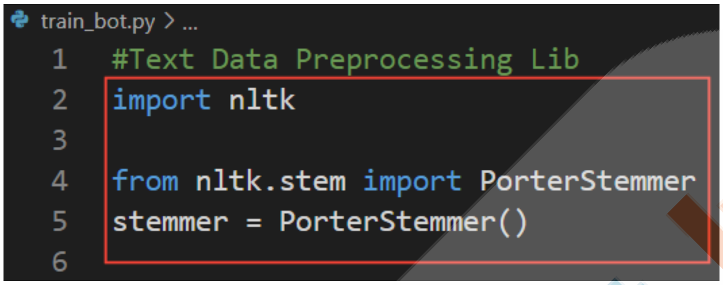


John is walking in the parking area

Above sentence is mapped as follows

John be walk in the park area

3. Import **PorterStemmer** from **nltk.stem**
4. Use variable **stemmer** for defining object for the **PorterStemmer()** class.

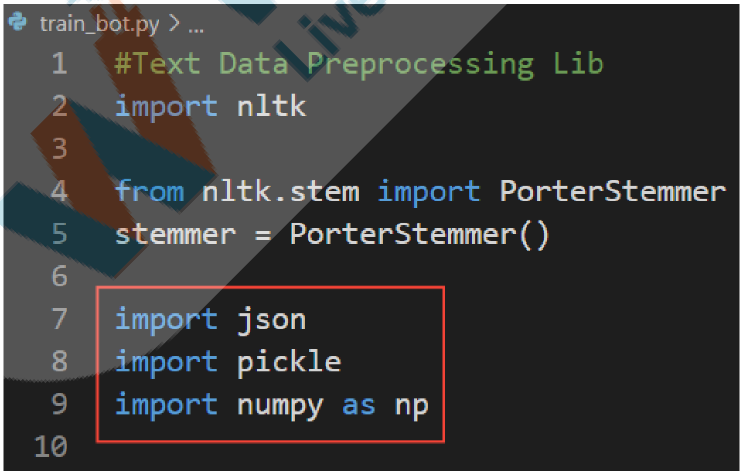


train_bot.py > ...

```python
#Text Data Preprocessing Lib
import nltk

from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

```

5. Next, import all the required libraries such as **json** for reading and processing **JSON** data.
6. Import **pickle**.
   Pickle is the Python library that converts lists, dictionaries, and other objects into streams of zero and one. This will be helpful to store **preprocessed training data**.

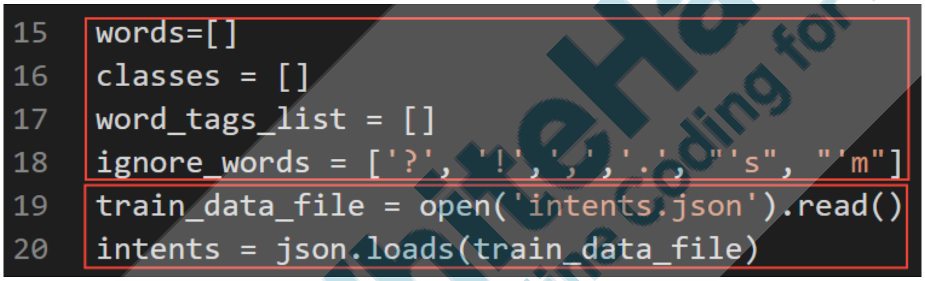
7. Install **numpy** library too as the training dataset has to be Numpy arrays.



train_bot.py > ...

```python
#Text Data Preprocessing Lib
import nltk

from nltk.stem import PorterStemmer
stemmer = PorterStemmer()

import json
import pickle
import numpy as np

```

8. After importing all the desired libraries, our first step is to load our data. For storing **tags, patterns** and **responses** from json file create three lists **words, classes** and **word_tags_list**.
9. Also, create a list of the symbols by name **ignore_words**. These symbols should be avoided for processing text as only words are needed for tokenizing.
10. Create a variable **train_data_file** for reading the **intents.json** file.
11. Load this json file into variable name **intents** using the **json.load()** function.

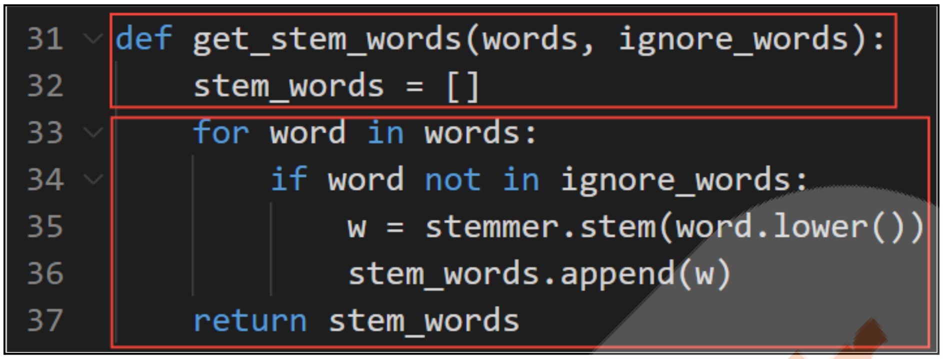


```
words=[]
classes = []
word_tags_list = []
ignore_words = ['?', '!', ',', '.', "'s", "'m"]
train_data_file = open('intents.json').read()
intents = json.loads(train_data_file)
```

Stemming:
Define a function to get the stem word from the list of words. Create an empty list for storing **stem_words**.

Write a **for** loop for looping through each word. Convert the word into lower case to maintain the uniformity between all the words. The words '**Play**' and '**play**' are tokenized differently.
Use the **stem()** function to change the word into its **stem or root** word.

Append this root word into the list **stem_words**. This list will contain all the words converted into their stem words and the punctuation will be removed.

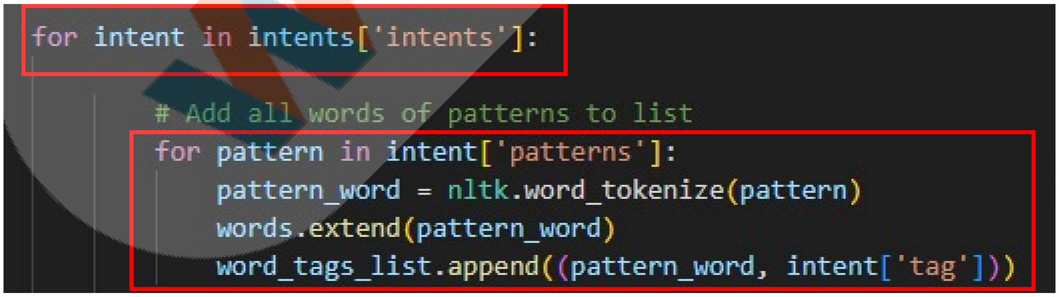


```
def get_stem_words(words, ignore_words):
    stem_words = []
    for word in words:
        if word not in ignore_words:
            w = stemmer.stem(word.lower())
            stem_words.append(w)
    return stem_words
```

Creating chatbot corpus:
Creating a **chatbot corpus** that is set of words that can be expected from user as input. To create a corpus, we first have to extract these words from **intents.json.** We are using a **for** loop to iterate through each intent or tag.

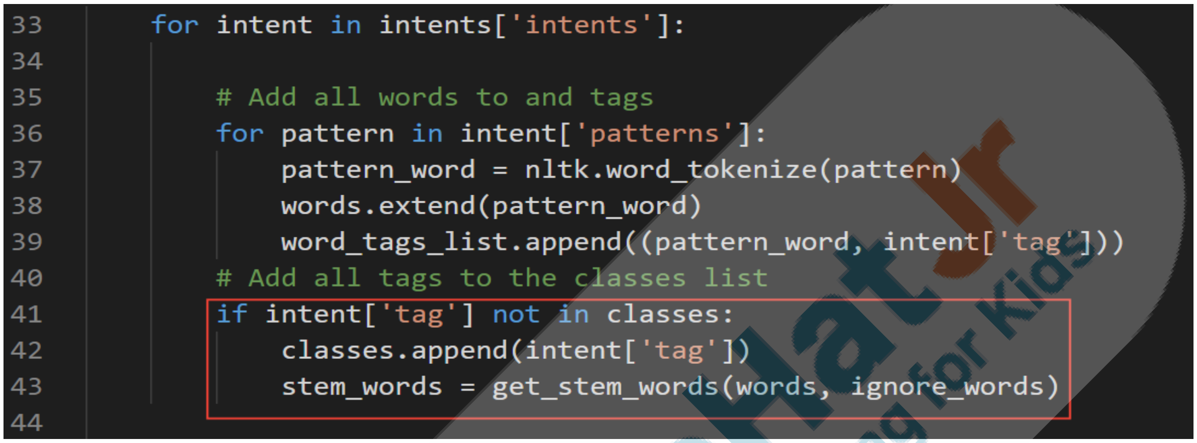
Write a **for** loop for adding words and tags into lists.

1. Tokenize each pattern and store it in the **pattern_word** variable.
2. The **extend()** method will add all the tokenized patterns into the list **words.**
3. Create **word_tags_list,** lists of words and tags in the form **['word', 'tag']**. This is an empty list that is appended by words and tags. In the append these tokenized patterns and respective tags.



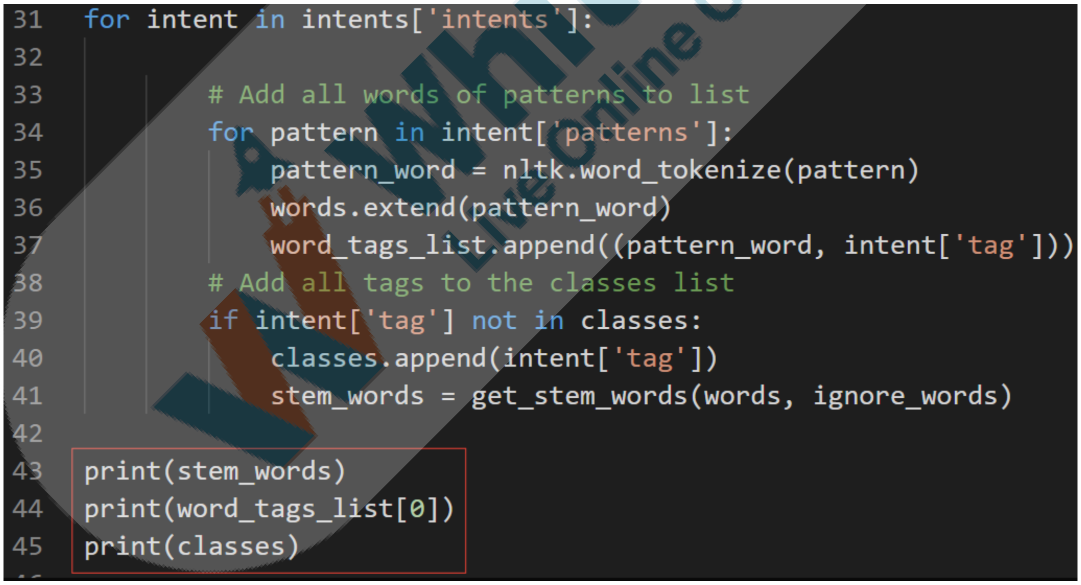
```
for intent in intents['intents']:

        # Add all words of patterns to list
        for pattern in intent['patterns']:
            pattern_word = nltk.word_tokenize(pattern)
            words.extend(pattern_word)
            word_tags_list.append((pattern_word, intent['tag']))
```

4. Now, create a list of tags known as **classes**. Append all the tags to the list.
5. Also, call the **get_stem_words()** function to create the list of stem words while excluding **ignore_words.**



```
    for intent in intents['intents']:

        # Add all words to and tags
        for pattern in intent['patterns']:
            pattern_word = nltk.word_tokenize(pattern)
            words.extend(pattern_word)
            word_tags_list.append((pattern_word, intent['tag']))
        # Add all tags to the classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
            stem_words = get_stem_words(words, ignore_words)
```

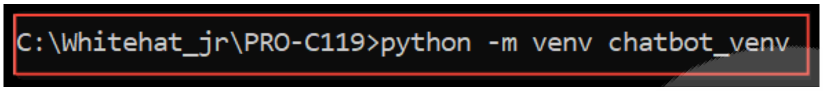
Let's check these lists now for creating word corpus.



```
for intent in intents['intents']:

        # Add all words of patterns to list
        for pattern in intent['patterns']:
            pattern_word = nltk.word_tokenize(pattern)
            words.extend(pattern_word)
            word_tags_list.append((pattern_word, intent['tag']))
        # Add all tags to the classes list
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
            stem_words = get_stem_words(words, ignore_words)

print(stem_words)
print(word_tags_list[0])
print(classes)
```

1. Go to the command prompt and traverse the working folder as we did in class 110.

2. Create a virtual environment **(Windows/Mac)** using **python -m venv <name_of_the _environment>**for testing the model.

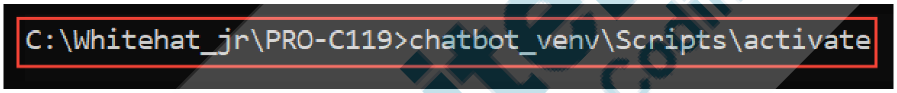


```
C:\Whitehat_jr\PRO-C119>python -m venv chatbot_venv
```

3. Activate the virtual environment using following command:

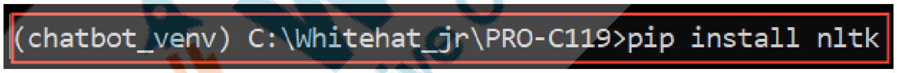
```
<name_of_the_environment>\Scripts\activate
```

*Run the command to create a virtual environment with the name “**cahtbot_venv**”. The environment is user-defined. We can keep the name as we want relevant to our project.*



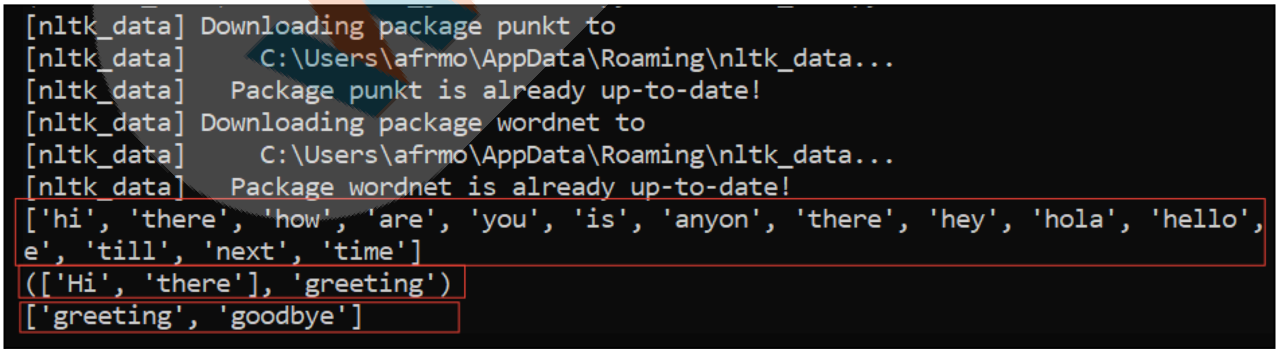
```
C:\Whitehat_jr\PRO-C119>chatbot_venv\Scripts\activate
```

4. Install NLTK using command **pip install nltk.** Also, install all the required libraries in the environment.



```
(chatbot_venv) C:\Whitehat_jr\PRO-C119>pip install nltk
```

5. Run the python file **data_preprocessing .py** and check the output

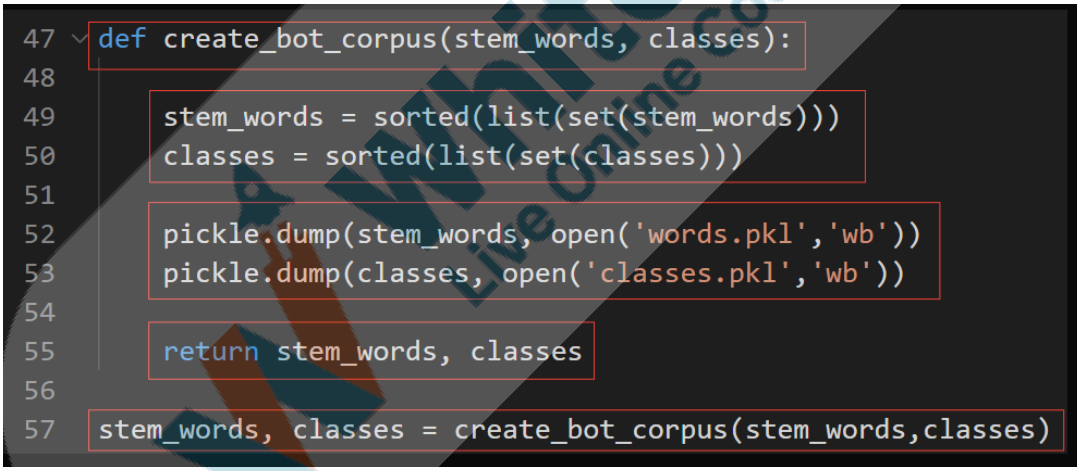


```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
['hi', 'there', 'how', 'are', 'you', 'is', 'anyon', 'there', 'hey', 'hola', 'hello',
e', 'till', 'next', 'time']
(['Hi', 'there'], 'greeting')
['greeting', 'goodbye']
```

1. Define a function **create_bot_corpus** that takes **stem_words** and **classes** parameters for creating a word corpus.
2. List of **stem_words** and **classes** are converted into set to get **unique words** from these lists. Again they are converted into list and stored in **sorted** manner.
3. Create two **pickle** files by name **words.pkl and classes.pkl.** These are then written by **pickle.dump()** method for creating the training dataset. **'wb'** stands for write in binary mode. Thus the data is stored in the binary form(0 and 1) in these files.

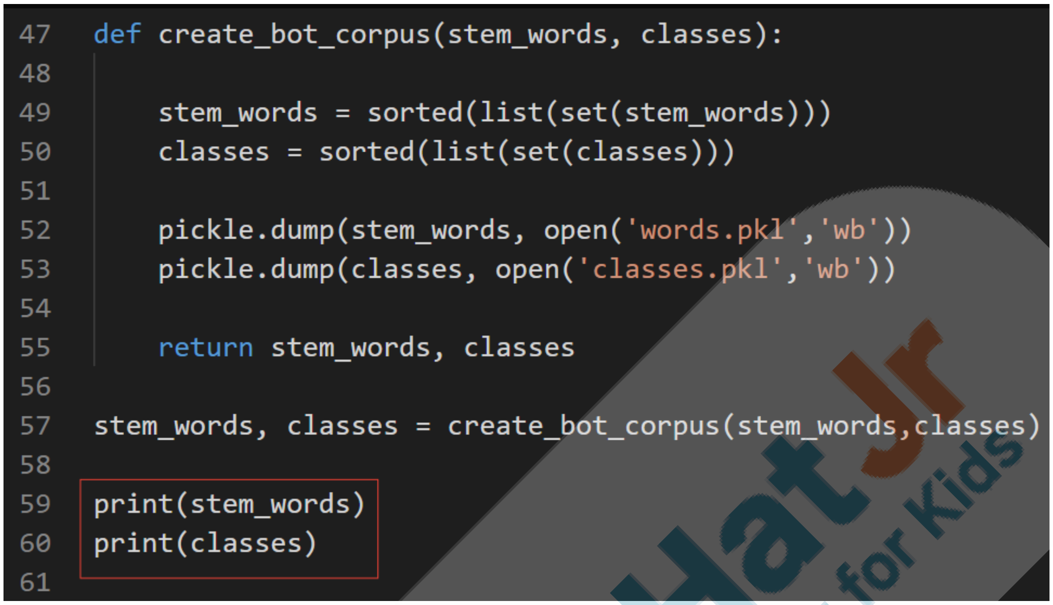
Note: *These files will be created automatically in the same folder when you run the code.*

4. Thus, this function returns the sorted **stem_word** list and sorted list of classes.
5. Call the function to create the corpus.



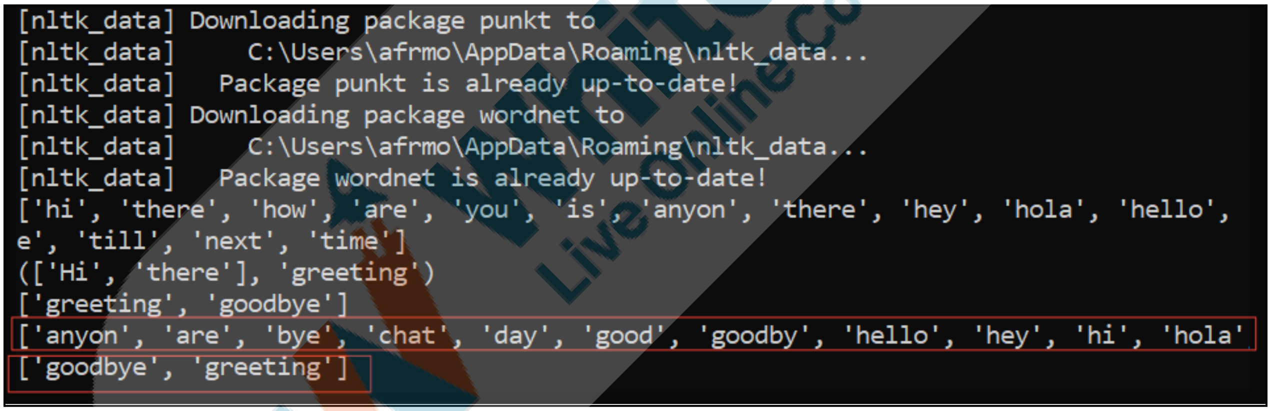
```
def create_bot_corpus(stem_words, classes):

    stem_words = sorted(list(set(stem_words)))
    classes = sorted(list(set(classes)))

    pickle.dump(stem_words, open('words.pkl','wb'))
    pickle.dump(classes, open('classes.pkl','wb'))

    return stem_words, classes

stem_words, classes = create_bot_corpus(stem_words,classes)
```

Let's check **stem_words** and **classes** lists now. You can see both these lists are sorted.



```
def create_bot_corpus(stem_words, classes):

    stem_words = sorted(list(set(stem_words)))
    classes = sorted(list(set(classes)))

    pickle.dump(stem_words, open('words.pkl','wb'))
    pickle.dump(classes, open('classes.pkl','wb'))

    return stem_words, classes

stem_words, classes = create_bot_corpus(stem_words,classes)

print(stem_words)
print(classes)
```

Run this file now and check the output.



```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
['hi', 'there', 'how', 'are', 'you', 'is', 'anyon', 'there', 'hey', 'hola', 'hello',
e', 'till', 'next', 'time']
(['Hi', 'there'], 'greeting')
['greeting', 'goodbye']
['anyon', 'are', 'bye', 'chat', 'day', 'good', 'goodby', 'hello', 'hey', 'hi', 'hola'
['goodbye', 'greeting']
```

Now our next step is to create **Bag Of Words**.

A Bag Of Words is a representation of text that shows the occurrence of words in a sentence.
Two things are required for creating BOW. A vocabulary of known words. This is given by **stem_words** and array to represent the **BOW** representation of the sentence.

Refer to the example given below. Stem words are given on the topmost row. If the word is present in the sentence is assigned value ‘1’ or else ‘0’.

stem_words/ sentences	anyon	are	bye	chat	day	good	goodby	hello	hey
good day	0	0	0	0	1	1	0	0	0
hello anyone	1	0	0	0	0	0	0	1	0

That was interesting! You have to now create the Bag Of Words.

Are you excited?

ESR: Yes.

Teacher Stops Screen Share

Please share your screen with me.

Teacher Starts Slideshow

Slide 14 to 16

Refer to speaker notes and follow the instructions on each slide.

STUDENT-LED ACTIVITY - 20 mins

- **Ask the student to press the ESC key to come back to the panel.**
- **Guide the student to start Screen Share.**
- **The teacher gets into Full Screen.**

Student Initiates Screen Share

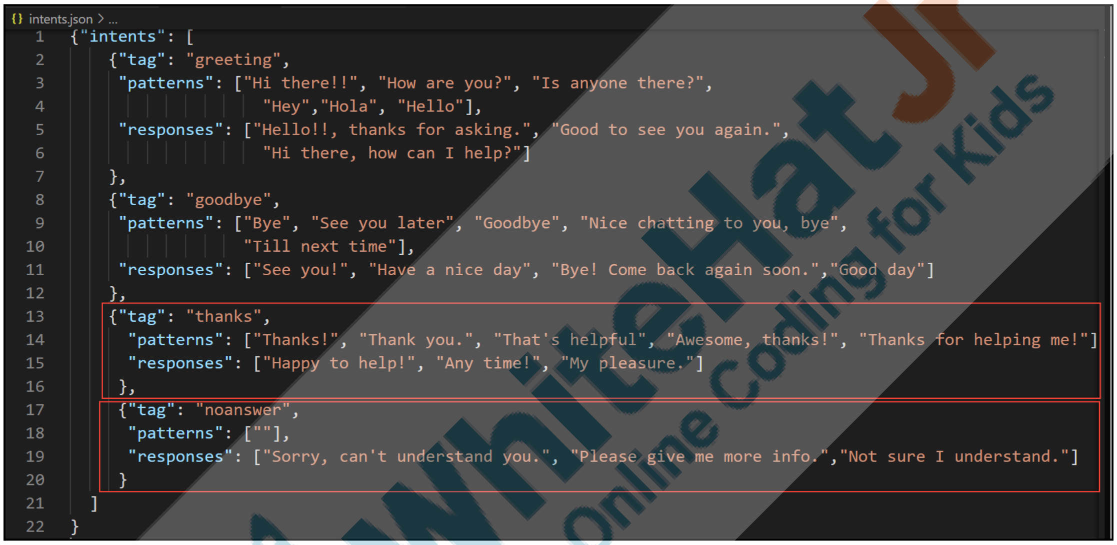
ACTIVITY

- **Creating the Bag Of Words.**

- **Creating the training dataset.**

Note: *Guide the student to open Student Activity1 for boilerplate code. Download the folder and run the file to check the output of stemming.*

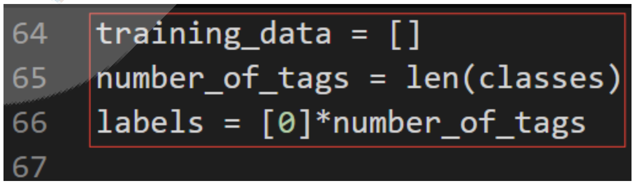
Go to **intents.json**. Now you'll have to add two more intents in this file. Think of the questions you may ask to a chatbot and accordingly add tags, patterns and responses.



```
{"intents": [
    {"tag": "greeting",
     "patterns": ["Hi there!!", "How are you?", "Is anyone there?",
                  "Hey","Hola", "Hello"],
     "responses": ["Hello!!, thanks for asking.", "Good to see you again.",
                  "Hi there, how can I help?"]
    },
    {"tag": "goodbye",
     "patterns": ["Bye", "See you later", "Goodbye", "Nice chatting to you, bye",
                "Till next time"],
     "responses": ["See you!", "Have a nice day", "Bye! Come back again soon.","Good day"]
    },
    {"tag": "thanks",
      "patterns": ["Thanks!", "Thank you.", "That's helpful", "Awesome, thanks!", "Thanks for helping me!"]
      "responses": ["Happy to help!", "Any time!", "My pleasure."]
     },
     {"tag": "noanswer",
      "patterns": [""],
      "responses": ["Sorry, can't understand you.", "Please give me more info.","Not sure I understand."]
     }
  ]
}
```

Start creating Bag Of Words by following these steps:

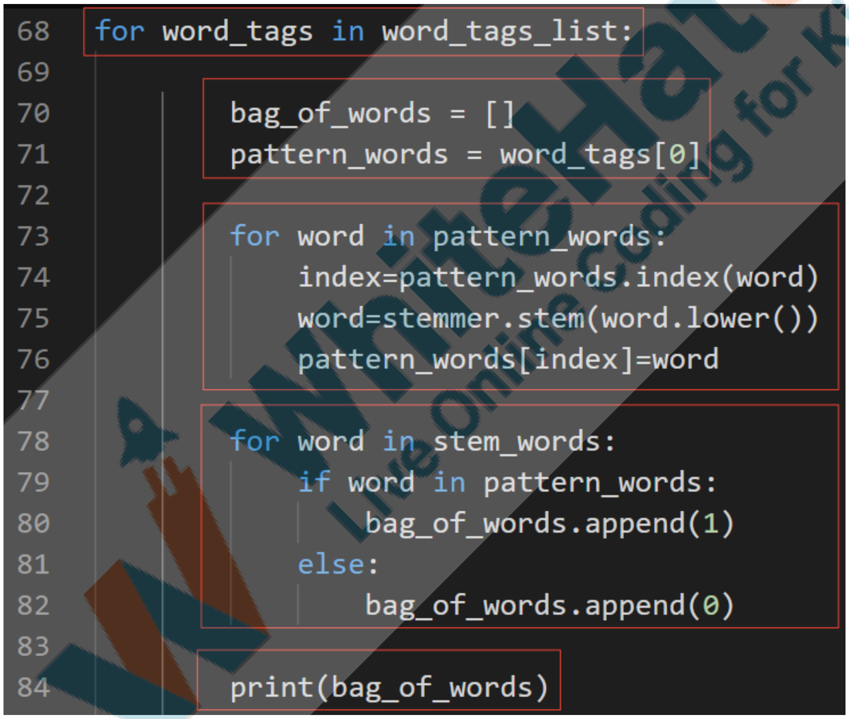
1. Define an empty list by the name **training_data.**
2. Define **number_of_tags** by finding the length of classes list.
3. Define an array of zeroes by name **labels.** Labels will store the corresponding **tag** of the sentence.



```
training_data = []
number_of_tags = len(classes)
labels = [0]*number_of_tags
```

To create Bag Of Words apply a **for** loop to **word_tags_list**. Each element is named as **word_tags.**

1. For each element we'll create an array of zeros and ones.
2. Define an empty array by name **bag_of_words.**
3. Also, define pattern_words as first index of **word_tags**.
4. Apply a **for** loop to **pattern_words** to find their stem words.
5. Then if these words are present in the stem words, **1** is appended in **bag_of_words** else **0** is appended.
6. Print **bag_of_of words** to check the output.



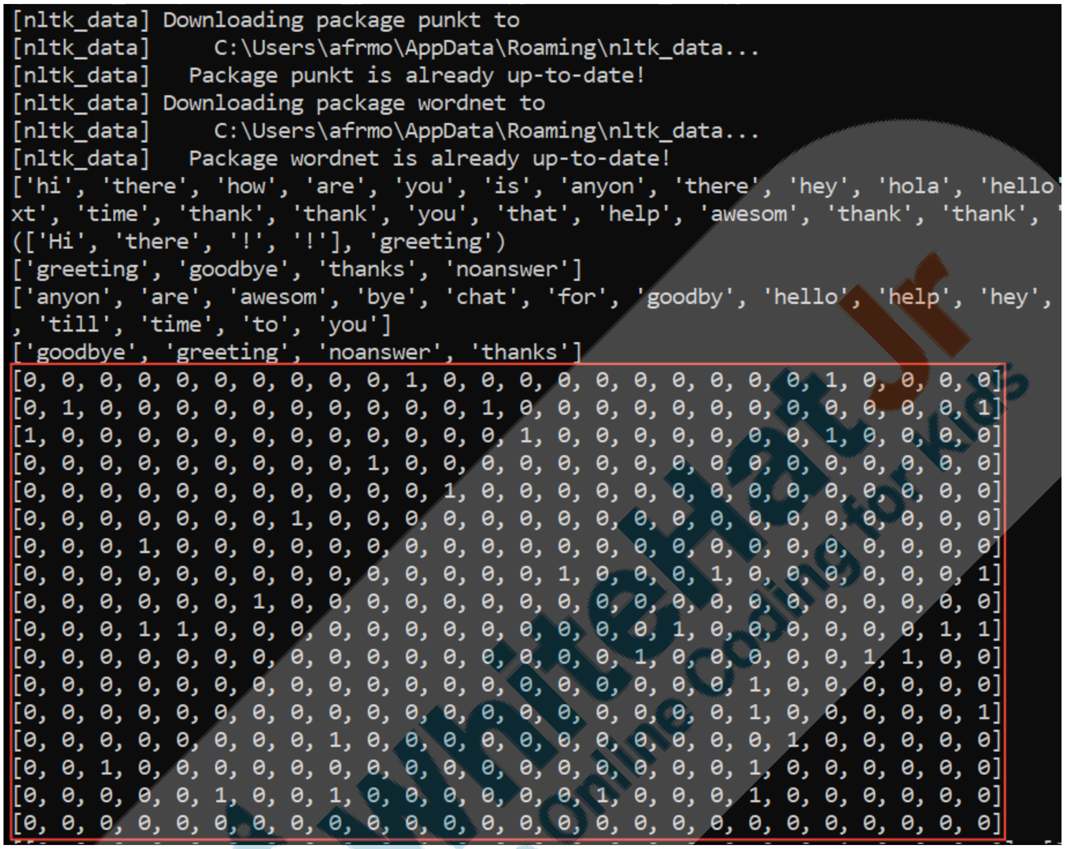
```
for word_tags in word_tags_list:

    bag_of_words = []
    pattern_words = word_tags[0]

    for word in pattern_words:
        index=pattern_words.index(word)
        word=stemmer.stem(word.lower())
        pattern_words[index]=word

    for word in stem_words:
        if word in pattern_words:
            bag_of_words.append(1)
        else:
            bag_of_words.append(0)

    print(bag_of_words)
```

Follow the same steps to check the output.

Note: *Guide the student to open command prompt, create a virtual environment and run the code to check the output.*

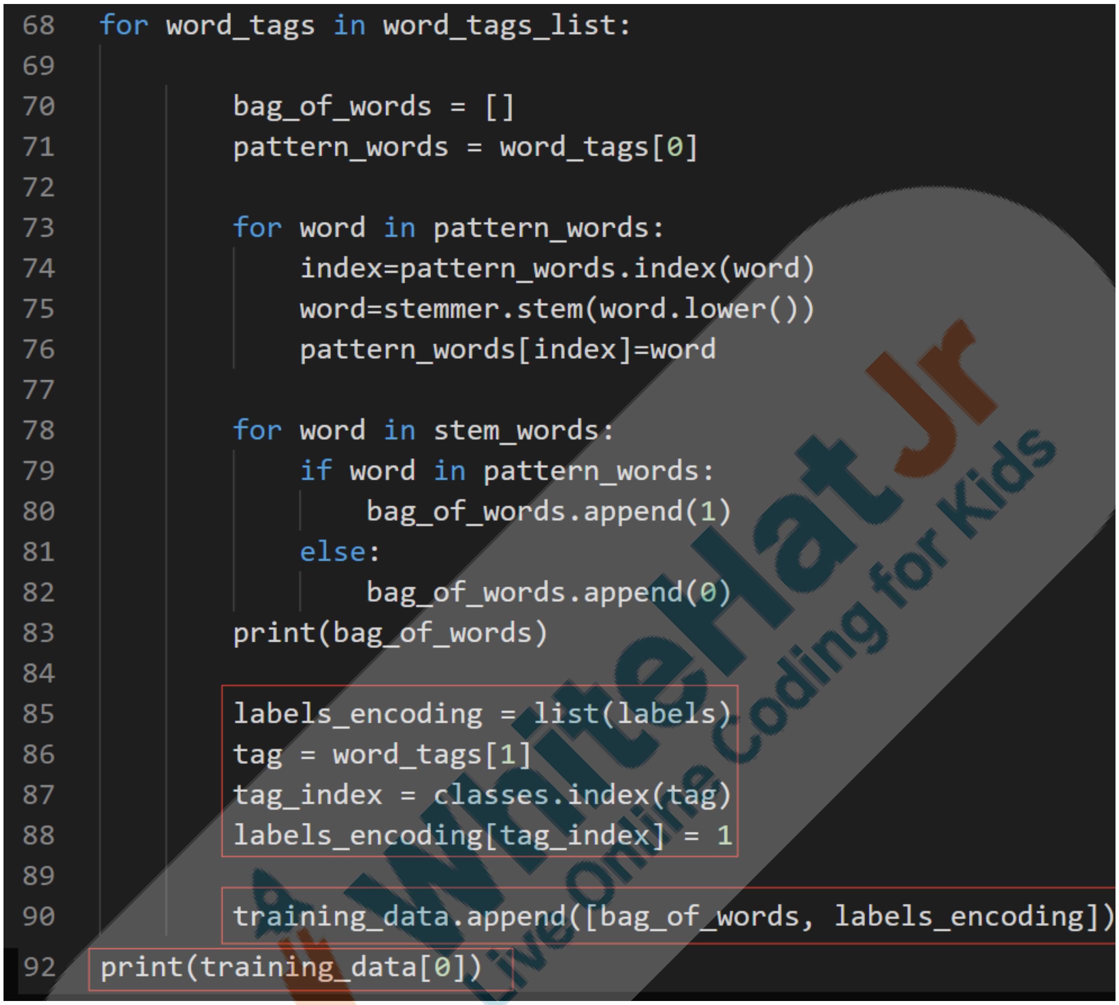
Note: *Since student has added two more intents, the output*

will differ for the student.	


```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\afrmo\AppData\Roaming\nltk_data...
[nltk_data]   Package wordnet is already up-to-date!
['hi', 'there', 'how', 'are', 'you', 'is', 'anyon', 'there', 'hey', 'hola', 'hello
xt', 'time', 'thank', 'thank', 'you', 'that', 'help', 'awesom', 'thank', 'thank', '
(['Hi', 'there', '!', '!'], 'greeting')
['greeting', 'goodbye', 'thanks', 'noanswer']
['anyon', 'are', 'awesom', 'bye', 'chat', 'for', 'goodby', 'hello', 'help', 'hey',
, 'till', 'time', 'to', 'you']
['goodbye', 'greeting', 'noanswer', 'thanks']
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1]
[1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 1]
[0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0]
[0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 1, 0, 0, 1, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0]
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
```

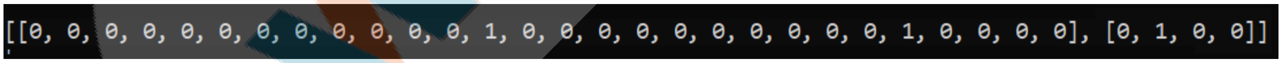
You can see a Bag of Word is printed over here. In the very first list ones are present for the words which are there inthe stem words list and rest zeros are added for the words that are absent.	
Creating label_encoding: 1. **label_encoding** list will have all labels initially as an array of zeros. For four tags it will be initially [**0 0 0 0**]. 2. The **word_tag_list** has all the **tags** at index number 1. Extract these tags and save them in the **tag** variable.	

3. The index of these tags is saved in the **tag_index** variable by using the index function with classes list and passing tag in it. (**classes.index(tag)** (0 to 3 for four tags) 4. Now, For each label, we will append **1** according to its index. For e.g. the first label is goodbye so **labels_encoding** for it will be [1 0 0 0] and so on. ***Note:*** *If requires teacher can print all these variables to make the student understand.* 5. In the **training_data** append the **bag_of_words** of **patterns** with corresponding tags. 6. Print **training_data[0]** to check the data at first index (i.e. 0).	



```
for word_tags in word_tags_list:

        bag_of_words = []
        pattern_words = word_tags[0]

        for word in pattern_words:
            index=pattern_words.index(word)
            word=stemmer.stem(word.lower())
            pattern_words[index]=word

        for word in stem_words:
            if word in pattern_words:
                bag_of_words.append(1)
            else:
                bag_of_words.append(0)
        print(bag_of_words)

        labels_encoding = list(labels)
        tag = word_tags[1]
        tag_index = classes.index(tag)
        labels_encoding[tag_index] = 1

        training_data.append([bag_of_words, labels_encoding])
print(training_data[0])
```

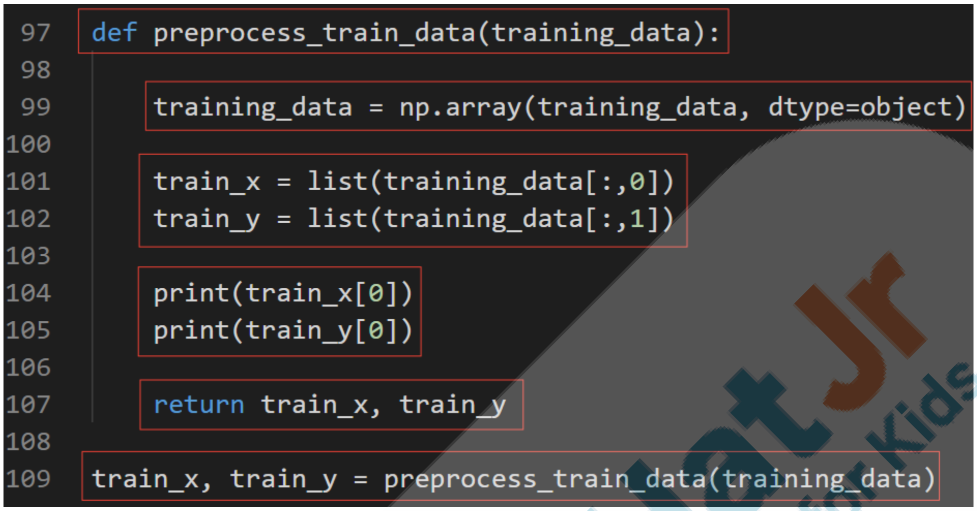
Save and run this and check the output.



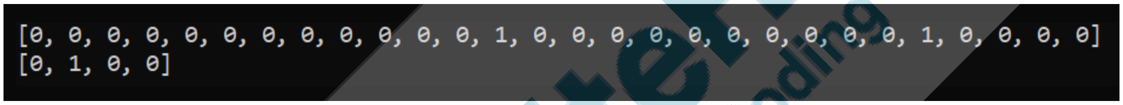
```
[[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0], [0, 1, 0, 0]]
```

The next step is to create a function for training the chatbot.

Define a function **preprocess_train_data.** Here The Bag of words is stored in the NumPy array **train_x** and the corresponding labels are stored in **train_y.** To train the model we need the data in NumPy array. You can print the data at the first index of these arrays. To check the output.



```
def preprocess_train_data(training_data):

    training_data = np.array(training_data, dtype=object)

    train_x = list(training_data[:,0])
    train_y = list(training_data[:,1])

    print(train_x[0])
    print(train_y[0])

    return train_x, train_y

train_x, train_y = preprocess_train_data(training_data)
```



```
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 1, 0, 0, 0, 0]
[0, 1, 0, 0]
```

Well done. So we can see the bag of words and have created the training dataset today. The first list is Bag Of Words (**train_x**) and next list is label (**train_y**).

Next step will be to train the model on this dataset.

You did amazing work today!

Teacher Guides Student to Stop Screen Share

Teacher Starts Slideshow
Slide 17 to 21
<**Note**: Only Applicable for Classes with VA>

Activity details

Following are the WRAP-UP session deliverables:

- Appreciate the student.
- Revise the current class activities.
- Discuss the quizzes.

WRAP-UP QUIZ

Click on In-Class Quiz

Continue WRAP-UP Session

Slide 22 to 27

Activity Details

Following are the session deliverables:

- Explain the facts and trivia
- Next class challenge
- Project for the day
- Additional Activity (Optional)

FEEDBACK

- **Appreciate and compliment the student for trying to learn a difficult concept.**
- **Get to know how they are feeling after the session.**
- **Review and check their understanding.**

Teacher Action	Student Action
You get Hats off for your excellent work! In the next class, we will train the model to create the chatbot.	*Make sure you have given at least 2 Hats Off during the class for:* Creatively Solved Activities +10 Great Question +10

PROJECT OVERVIEW DISCUSSION Refer the document below in Activity Links Sections	
Teacher Clicks	

ACTIVITY LINKS		
Activity Name	**Description**	**Links**
Teacher Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C119-Teacher-Boilerplate-Code
Teacher Activity 2	NLTK	https://www.nltk.org/
Teacher Activity 3	PorterStemmer	https://www.nltk.org/_modules/nltk/stem/porter.html
Teacher Activity 4	Reference Code	https://github.com/procodingclass/PRO-C119-Reference-Code
Teacher Reference 1	Project	https://s3-whjr-curriculum-uploads.whjr.online/bf8d4f82-408f-4803-a803-8071f435b1eb.pdf
Teacher Reference 2	Project Solution	https://github.com/procodingclass/PRO-C119-Project-Solution.git
Teacher Reference 3	Visual-Aid	https://s3-whjr-curriculum-uploads.whjr.online/7046213e-1a94-4b90-986f-a50e1072e440.html
Teacher Reference 4	In-Class Quiz	https://s3-whjr-curriculum-uploads.whjr.online/27c4dd10-e730-4d8f-882a-d62fa0d63e15.pdf
Student Activity 1	Boilerplate Code	https://github.com/procodingclass/PRO-C119-Student-Boilerplate-Code