**IBM NAAN MUDHALVAN**

**CHATBOT USING PYTHON**

**DEVELOPMENT PART 1:**

## How To Make A Chatbot In Python?

You may have this question in your mind, how to create a chatbot? We’ll take a step by step approach and break down the process of building a Python chatbot.

To build a chatbot in Python, you have to import all the necessary packages and initialize the variables you want to use in your chatbot project. Also, remember that when working with text data, you need to perform data preprocessing on your dataset before designing an ML model.

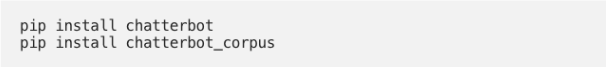
This is where tokenizing helps with text data – it helps fragment the large text dataset into smaller, readable chunks (like words). Once that is done, you can also go for lemmatization that transforms a word into its lemma form. Then it creates a pickle file to store the python objects that are used for predicting the responses of the bot.

Another vital part of the chatbot development process is creating the training and testing datasets.

Now that we’ve covered the basics of chatbot development in Python, let’s dive deeper into the actual process! It will help you understand how to create a chatbot.

### ****1. Prepare the Dependencies****

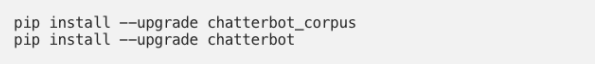
The first step in creating a chatbot in Python with the ChatterBot library is to install the library in your system. It is best if you create and use a new Python virtual environment for the installation. To do so, you have to write and execute this command in your Python terminal:



You can also install ChatterBot’s latest development version directly from GitHub. For this, you will have to write and execute the following command:

pip install git+git://github.com/gunthercox/ChatterBot.git@master

If you wish to upgrade the command, you can do so as well:



Now that your setup is ready, we can move on to the next step to create chatbot using python.

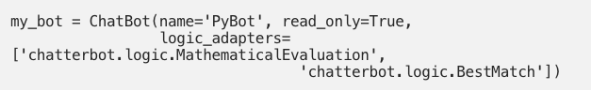
### 2. Import Classes

Importing classes is the second step in the Python chatbot creation process. All you need to do is import two classes – ChatBot from chatterbot and ListTrainer from chatterbot.trainers. To do this, you can execute the following command:

https://d14b9ctw0m6fid.cloudfront.net/ugblog/wp-content/uploads/2019/12/screenshot-docs.google.com-2019.12.12-15_10_57.png

### 3. Create and Train the Chatbot

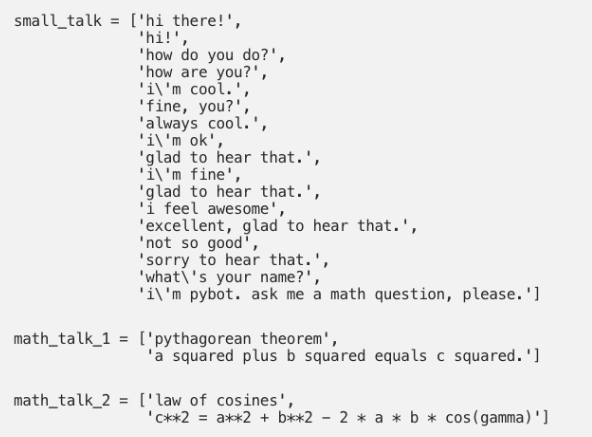
This is the third step on creating chatbot in python. The chatbot you are creating will be an instance of the class “ChatBot.” After creating a new ChatterBot instance, you can train the bot to improve its performance. Training ensures that the bot has enough knowledge to get started with specific responses to specific inputs. You have to execute the following command now:



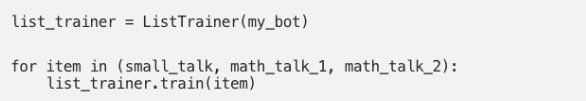
Here, the argument (that corresponds to the parameter name) represents the name of your Python chatbot. If you wish to disable the bot’s ability to learn after the training, you can include the “read\_only=True” command. The command “logic\_adapters” denotes the list of adapters used to train the chatbot.

While the “chatterbot.logic.MathematicalEvaluation” helps the bot to solve math problems, the “chatterbot.logic.BestMatch” helps it to choose the best match from the list of responses already provided.

Since you have to provide a list of responses, you can do it by specifying the lists of strings that can be later used to train your Python chatbot, and find the best match for each query. Here’s an example of responses you can train your chatbot using python to learn:



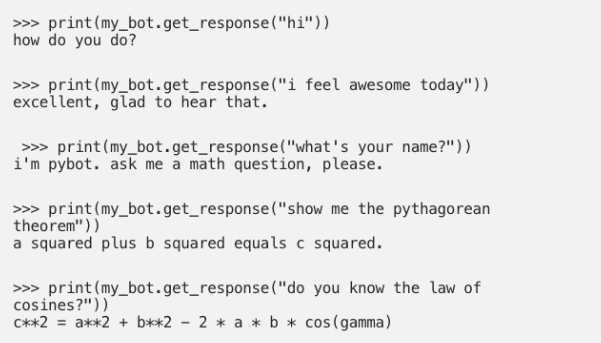
You can also create and train the bot by writing an instance of “ListTrainer” and supplying it with a list of strings like so:



|  |
| --- |
|  |
|  |

### 4. Communicate with the Python Chatbot

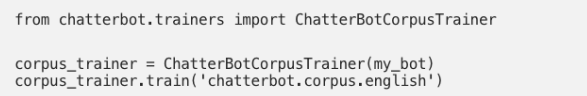
To interact with your Python chatbot, you can use the .get\_response() function. This is how it should look while communicating:



However, it is essential to understand that the chatbot using python might not know how to answer all your questions. Since its knowledge and training is still very limited, you have to give it time and provide more training data to train it further.

### 5. Train your Python Chatbot with a Corpus of Data

In this last step of how to make a chatbot in Python, for training your python chatbot even further, you can use an existing corpus of data. Here’s an example of how to train your Python chatbot with a corpus of data provided by the bot itself:



In the previous step, you built a chatbot that you could interact with from your command line. The chatbot started from a clean slate and wasn’t very interesting to talk to.

In this step, you’ll train your chatbot using ListTrainer to make it a little smarter from the start. You’ll also learn about built-in trainers that come with ChatterBot, including their limitations.

Your chatbot doesn’t have to start from scratch, and ChatterBot provides you with a quick way to train your bot. You’ll use [ChatterBot’s ListTrainer](https://chatterbot.readthedocs.io/en/stable/training.html#training-via-list-data) to provide some conversation samples that’ll give your chatbot more room to grow:

1# bot.py

2

3from chatterbot import ChatBot

4from chatterbot.trainers import ListTrainer

5

6chatbot = ChatBot("Chatpot")

7

8trainer = ListTrainer(chatbot)

9trainer.train([

10 "Hi",

11 "Welcome, friend 🤗",

12])

13trainer.train([

14 "Are you a plant?",

15 "No, I'm the pot below the plant!",

16])

17

18exit\_conditions = (":q", "quit", "exit")

19while True:

20 query = input("> ")

21 if query in exit\_conditions:

22 break

23 else:

24 print(f"🪴 {chatbot.get\_response(query)}")

In line 4, you import ListTrainer, to which you pass your chatbot on line 8 to create trainer.

In lines 9 to 12, you set up the first training round, where you pass a list of two strings to trainer.train(). Using .train() injects entries into your database to build upon the graph structure that ChatterBot uses to choose possible replies.

You can run more than one training session, so in lines 13 to 16, you add another statement and another reply to your chatbot’s database.

If you now run the interactive chatbot once again using python bot.py, you can elicit somewhat different responses from it than before:

> hi

🪴 Welcome, friend 🤗

> hello

🪴 are you a plant?

> me?

🪴 are you a plant?

> yes

🪴 hi

> are you a plant?

🪴 No, I'm the pot below the plant!

> cool

🪴 Welcome, friend 🤗

The conversation isn’t yet fluent enough that you’d like to go on a second date, but there’s additional context that you didn’t have before! When you train your chatbot with more data, it’ll get better at responding to user inputs.

The ChatterBot library comes with [some corpora](https://github.com/gunthercox/chatterbot-corpus) that you can use to train your chatbot. However, at the time of writing, there are some issues if you try to use these resources straight out of the box.

While the provided corpora might be enough for you, in this tutorial you’ll skip them entirely and instead learn how to adapt your own conversational input data for training with ChatterBot’s ListTrainer.

To train your chatbot to respond to industry-relevant questions, you’ll probably need to work with custom data, for example from existing support requests or chat logs from your company.

Moving forward, you’ll work through the steps of converting chat data from a WhatsApp conversation into a format that you can use to train your chatbot. If your own resource is WhatsApp conversation data, then you can use these steps directly. If your data comes from elsewhere, then you can adapt the steps to fit your specific text format.

To start off, you’ll learn how to export data from a WhatsApp chat conversation.

## ****Conclusion****

What we’ve illustrated here is just one among the many ways of **how to make a chatbot in Python.** You can also use NLTK, another resourceful Python library to create a Python chatbot. And although what you learned here is a very basic**chatbot in Python** having hardly any cognitive skills, it should be enough to help you understand the anatomy of chatbots.

Once you understand the design of a **chatbot using python** fully well, you can experiment with it using different tools and commands to make it even smarter.