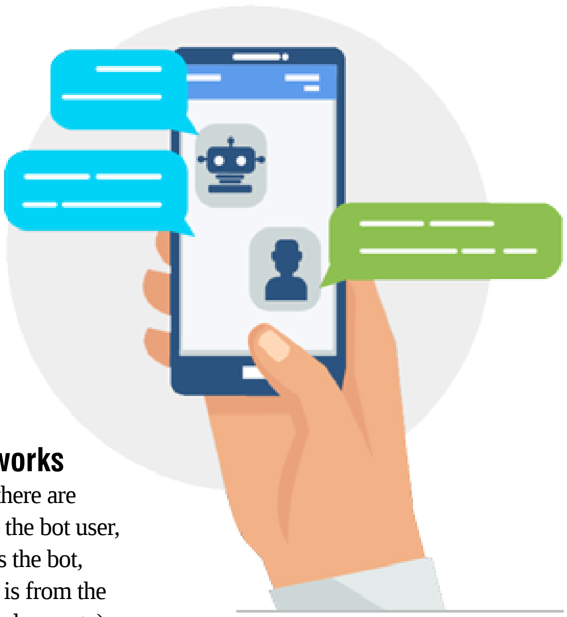
Creating a Very Basic Chatbot in Python on Telegram Messenger

The Internet is full of articles about building chatbots on platforms like Telegram. However, in this article, readers get some insights on how to code for a chatbot in Python. Understanding the package used in this article will enable the adventurous to extend the functionality of this chatbot on their own.



To get started with building the chatbot in real-time, there are some prerequisites, which are as follows:

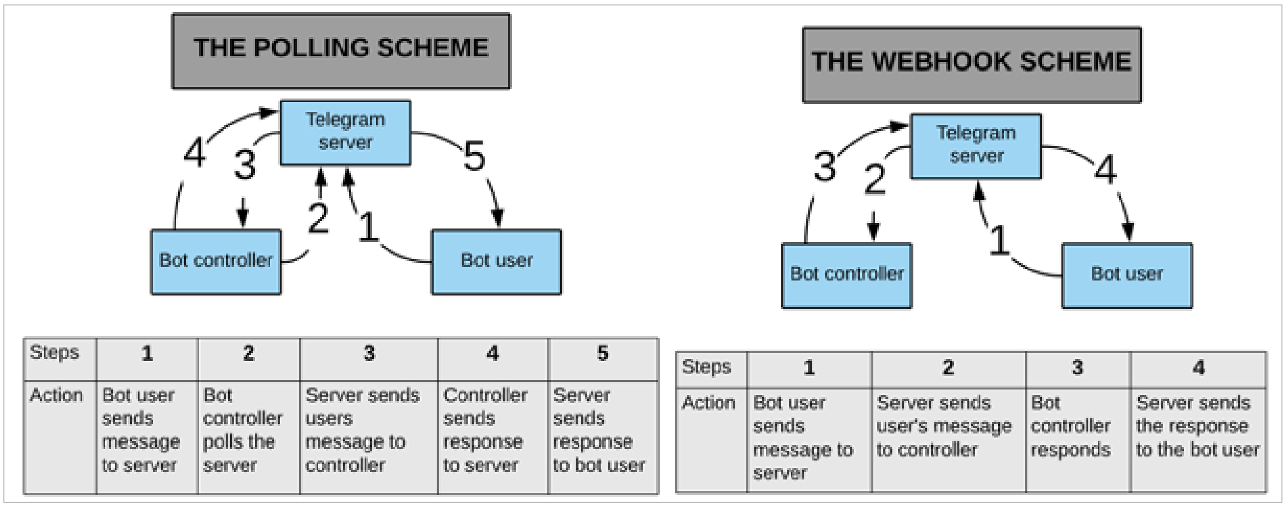
- Basic knowledge of Python and using Pip/Conda to install packages.
- A Python IDE (I use Jupyter Notebook, while you could pick any of the popular options like
- Spyder, PyCharm, Visual Studio Code or Eclipse (with PyDev)).
- You need to install Telegram on a mobile phone and also on your desktop.
- You will need the Python library *python-telegram-bot*.

How the bot scheme works

You need to understand that there are three actors in the scheme — the bot user, the Telegram server that hosts the bot, and the bot controller (which is from the Python script that we write and execute).

Also, note that you can use the following two ways to control the bot.

1. ***The 'polling' scheme***: Here the Python script (which controls the bot) will periodically access, i.e., 'poll' the bot on the Telegram server, check if there are any new messages from a bot user, and respond accordingly.
2. ***The webhook scheme***. This is the more efficient scheme. Here the bot controller script does not 'poll' the bot; rather, it is the bot that informs the bot controller that a new message has been sent by a bot user. This scheme however requires two additional things — a permanent



Steps	1	2	3	4	5
Action	Bot user sends message to server	Bot controller polls the server	Server sends users message to controller	Controller sends response to server	Server sends response to bot user

Steps	1	2	3	4
Action	Bot user sends message to server	Server sends user's message to controller	Bot controller responds	Server sends the response to the bot user

Figute1: The two different schemes for working the bot

IP address of the server where the bot controller is located, and that the server hosting the bot controller must have HTTPS enabled. In this article, I am only describing the 'polling' scheme.

Using @BotFather to create your bot

The steps in creating a Telegram bot are as follows:

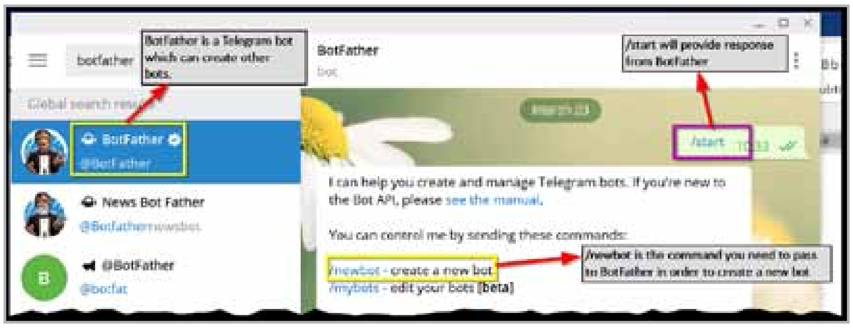
1. Telegram has a bot named @ BotFather which can create a bot for you. So, as a first step, go to your Telegram app and search for @ BotFather.
2. Then send it the message */start*. The scheme works like this — if it is a message without a leading forward slash (/), then it will be interpreted as a text message. However, if it has a leading forward slash, then it will be interpreted as the function / method. A Telegram bot cannot respond on its own (to reduce spamming by bots). It must receive a text message (one without a forward slash) or a command (one with a leading forward slash) to respond to such messages.
3. Now create a new bot using the command */newbot* and give it an appropriate name, which must end in the word *bot.* @BotFather will send you a message, which will include a token. The token should look something like what's shown below:

```
1104497538:AAEbXXXi PegAxUXXXX_
TevmyPG01GnXXLY
```

(Some letters have been replaced with X to preserve privacy.) Figure 3 shows the process on the Telegram app.

4. Note that your newly created bot is hosted on the Telegram server, and you can now access your bot on a Web browser by using a command of the format:

```
https://api.telegram.org/bot<your_bot_
token>/getme
```

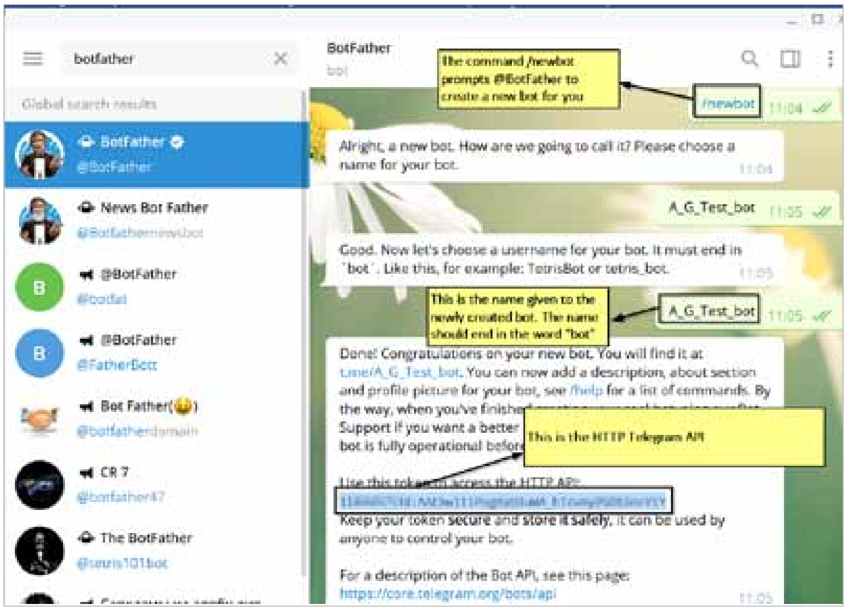

botfather
Global search results
BotFather
@BotFather
News Bot Father
@Botfathernewsbot
@BotFather
@botfat
BotFather
bot
/start
BotFather is a Telegram bot which can create other bots.
/start will provide response from BotFather
/newbot is the command you need to pass to BotFather in order to create a new bot

I can help you create and manage Telegram bots. If you're new to the Bot API, please see the manual.

You can control me by sending these commands:

/newbot - create a new bot
/mybots - edit your bots [beta]

Figure 2: Using the command '/start' on @BotFather



/newbot 11:04

Alright, a new bot. How are we going to call it? Please choose a name for your bot. 11:04

A_G_Test_bot 11:05

Good. Now let's choose a username for your bot. It must end in `bot`. Like this, for example: TetrisBot or tetris_bot. 11:05

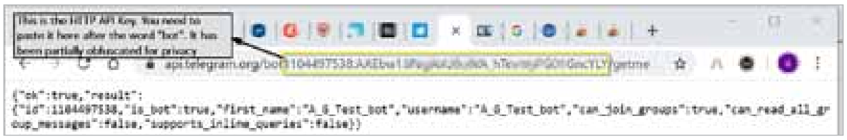
A_G_Test_bot 11:05

Done! Congratulations on your new bot. You will find it at t.me/A_G_Test_bot. You can now add a description, about section and profile picture for your bot, see /help for a list of commands. By the way, when you've finished [illegible] Support if you want a better [illegible] bot is fully operational before [illegible]

Use this token to access the HTTP API:
[illegible]
Keep your token secure and store it safely, it can be used by anyone to control your bot.

For a description of the Bot API, see this page:
https://core.telegram.org/bots/api 11:05

Figure 3: Creating a new bot using @BotFather


This is the HTTP API Key. You need to paste it here after the word "bot". It has been partially obfuscated for privacy

api.telegram.org/bot1104497538:AAEba1[illegible]/getme

```
{"ok":true,"result":
{"id":1104497538,"is_bot":true,"first_name":"A_G_Test_bot","username":"A_G_Test_bot","can_join_groups":true,"can_read_all_gr
oup_messages":false,"supports_inline_queries":false}}
```

Figure 4: A screenshot of the output on giving a command

The above URL has three parts: (1) *https://api.telegram.org/bot* ; (2) *<your_bot_token>* ; and (3) */getme*. The first two parts of the URL will allow your browser to access your bot (which resides on the Telegram server) and the last part will give the appropriate command to your bot. Here the command is */getme* and so it will provide details of your bot in the form of a JSON object (similar to a Python dictionary). Figure 4 shows this.

Now there are two ways to interact with the bot. The first is of course the method we used earlier, where we used a Web browser and a URL of the format *https://api.telegram.org/bot<your_bot_token>/getme*.

There is however a second method in which we can use a Python script to access the messages/ commands coming from the bot user to our bot hosted on the Telegram server. In the next section, we will look at creating a bot controller through a script written in Python.

Writing a Python script to create a bot controller using *python-library-telegram*

To create a bot controller script, we shall use a Python library *python-telegram-bot*; so download this using Pip:

```
pip install python-telegram-bot
```

This is shown in Figure 5.

To write the Python script, you can use any popular IDE. Some common ones are PyCharm, Visual Studio Code and Eclipse (with PyDev). You need to know some of the classes and methods of this library in order to truly understand how the code works. Remember, we are using the *polling* and not the *webhook* technique.

We shall use instances of (i.e., we will create objects of) two classes from the *telegram.ext* sub-module, namely, *telegram.ext.Updater* and *telegram.ext.Dispatcher*. The Updater class gets updates from Telegram and passes them on to the Dispatcher class.

When you create an Updater object, it will create a Dispatcher object for you and link them together with a Queue. This Dispatcher object can then be used to sort the updates fetched by the Updater according to the handlers you registered, and deliver them to a callback function that you defined.

The above para may be a bit confusing, so I have elaborated on it, as follows.

Step 1: Create an object (we will call it *my_updater*) of class *telegram.ext.Updater*. To do this, you need to import the Updater class from *telegram.ext*. Also, when you create your Updater object, namely *my_updater*, you will need the HTTP API key for the bot you created earlier. The code to create this object will be like what follows:

```
my_updater = Updater(token='your_bot_
token', use_context=True)
```

Note: In the example code on the website (Ref 2 & 3), this variable is often called Dispatcher, but I have used my_dispatcher to specify that this is not some method of a class but an object of class Updater, which I have created.

Step 2: Now, this Updater class has an attribute called Dispatcher. This attribute returns an object of class *telegram.ext.Dispatcher*. Don't get confused by this scheme. This is a typical method by which many libraries of Python work. So you can create an object of Dispatcher class (we will call it *my_dispatcher*) by using the *dispatcher* attribute of the object of the Updater class (which here, is *my_updater*). The following code snippet shows this:

```
my_dispatcher = my_updater.dispatcher
```

Step 3: Here, we will write a function named *start*. So, whenever the bot user types a message */start*, this function gets triggered. The way the Telegram package is written is such that when you write a function (like */start* here), the Telegram server will pass it two attributes. So you must have two attributes in your function definition and by convention, these are called *update* and *context*. I don't want to get too technical but for those interested, the attribute *update* will get an object of class *telegram.update.Update*. While the attribute *context* will get an object of class *telegram.ext.callbackcontext.CallbackContext.*

The attribute *update* is actually a Python dictionary. You can confirm this by putting a *print(update)* command somewhere in your *get()* function. This dictionary has a *key* named *effective_chat* and the *value* of this key is another dictionary, which contains *keys* like *ID*, *first_name*, *last_name*, etc. So, using a format of type *update.effective_chat.id*, you can get the ID of the chat. In our bot code, we will 'extract' three parameters of the bot user — (1) *id*, (2) *first_name*, and (3) *last_name*.

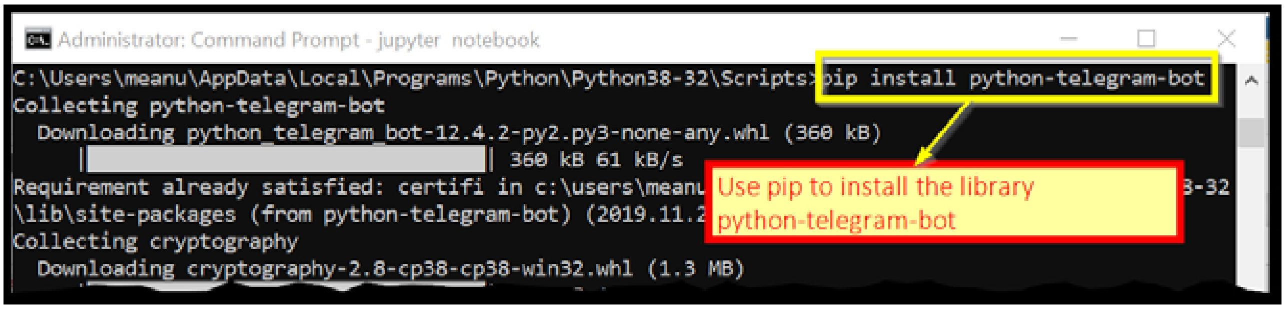
In this step, we will do the following things:

- Extract *id*, *first_name* and *last_name* from the parameter *update*.
- We will use the *first_name* and *last_name* to send a reply back to the bot user.
- To send a reply back to the bot user, we need to use the following command:

```
context.bot.send_message(chat_id, text=
out_text)
```

To make the code more readable and easier to follow, I have split this in two lines as:

```
my_bot = context.bot
my_bot.send_message(chat_id, text=
out_text)
```


Administrator: Command Prompt - jupyter notebook
Use pip to install the library python-telegram-bot

```
C:\Users\meanu\AppData\Local\Programs\Python\Python38-32\Scripts>pip install python-telegram-bot
Collecting python-telegram-bot
  Downloading python_telegram_bot-12.4.2-py2.py3-none-any.whl (360 kB)
     |████████████████████████████████| 360 kB 61 kB/s
Requirement already satisfied: certifi in c:\users\meanu                                   3-32
\lib\site-packages (from python-telegram-bot) (2019.11.2
Collecting cryptography
  Downloading cryptography-2.8-cp38-cp38-win32.whl (1.3 MB)
```

Figure 5: Downloading the Python library *python-telegram-bot*

For the technically inclined, you may note that *my_bot* (which is the return value of *context.bot*) is an object of class *telegram.Bot*. Further, this object, i.e., *my_bot*, being an object of class *telegram.Bot*, has a method *send_message()*. So you can use this method to send a message to the bot user.

Note: You can confirm that the return of context.bot is actually an object of class telegram.Bot by looking at the source code for this in the file callbackcontext.py. You can find this file on GitHub at *https://github.com/python-telegram-bot/python-telegram-bot/blob/7b3b278c7c09d2961a696e58b955914486d96313/telegram/ext/callbackcontext.py* . You will find the definition of *bot()* around lines 155-157. Also the class telegram.Bot has a method *send_message()* by looking at the source code of the file bot.py available at *https://github.com/python-telegram-bot/python-telegram-bot/blob/master/telegram/bot.py* . *You will get the code for the method send_message() around lines 265-275.*

But in the above scheme, there is one problem — you need to register this *command*, i.e., *get()* in the dispatcher. To do this, the module has a class *Handler*, and from this class many other classes are inherited such as *CommandHandler*, *MessageHandler*, etc. Here, since *get()* is a command, we will use the specific *Handler* meant to handle commands, which is *Commandhandler*. Note that the class *Commandhandler* sub-classes the class Handler, i.e., it inherits from the class Handler. So the scheme works like this.

You first create an object of class *CommandHandler* (I have named it *start_handler*) and pass to it the name of the command as the first parameter and the name of the callback function as the second parameter. So the line of code will be as follows:

```
start_handler = CommandHandler('start', start)
```

Note: *You can see the source code for the Dispatcher class in the file dispatcher.py at https://github.com/python-telegram-bot/python-telegram-bot/blob/7b3b278c7c09d2961a696e58b955914486d96313/telegram/ext/dispatcher.py.*

Having created an object of type *CommandHandler*, you can add it to the *Dispatcher* class object (which in our code is *my_dispatcher*), using the *method add_handler()* of the *Dispatcher* class. So, the code will be as follows:

```
from telegram.ext import Updater, CommandHandler
my_updater = Updater(token='Your_token_here', use_context=True)
my_dispatcher = my_updater.dispatcher

def start(update, context):
chat_id = update.effective_chat.id # Get id of the chat.
f_name = update.effective_chat.first_name # First name of bot user
l_name = update.effective_chat.last_name # Last name of bot user
full_name = f_name + ‘’ + l_name  # Full name of bot user
out_text = ‘Hi {0} {1} from bot’.format(f_name, l_name)
 # The print functions are to show the contents of the variables
 # You can remove the print functions
 print(update)
  print(‘effective_chat ->’,update.effective_chat)
 print(‘chat_id->’, chat_id)
 print(‘bot->’, context.bot)

  my_bot = context.bot  # my_bot is an object of type telegram.Bot
 # print(my_bot)
 my_bot.send_message(chat_id, text=out_text)
start_handler = CommandHandler(‘start’, start)
my_dispatcher.add_handler(start_handler)
# my_updater.start_polling() starts the bot, and the bot begins
# to start polling Telegram for any chat updates.
# The bot has its own separate threads so it wont halt your Python script.
my_updater.start_polling()
# my_updater.idle() command is used to block the script until
# the user sends a command to break from the Python script such as ctrl-c
on windows
my_updater.idle()
```

The complete code for this bot is available on GitHub at *https://github.com/ag999git/jupyter_notebooks/blob/master/telegram-bot*. You can scan the given QR code to go to the website directly.

Further extensions

You can improvise and extend the functionality of the bot. This bot accepts only one command, namely, */start*. You can create a message handler. For that you will need an object of class *MessageHandler*. You can also try using *Webhooks* or install your bot on a remote server like Heroku, PythonAnywhere, etc.

References

[1] Complete documentation of a Python Telegram bot is available at *https://readthedocs.org/projects/python-telegram-bot/downloads/pdf/stable/*
[2] *https://python-telegram-bot.readthedocs.io/en/latest/telegram.ext.updater.html#telegram.ext.Updater*
[3] *https://python-telegram-bot.readthedocs.io/en/latest/telegram.ext.dispatcher.html#telegram.ext.Dispatcher*

By: Anurag Gupta

The author is an Engineer by education but a police officer by profession. He is a Python enthusiast who has recently co-authored a book on Python named Python Programming: Problem Solving, Packages and Libraries