Gareth Dwyer FOLLOW
Pythoneer and Writer

Building a Chatbot using Telegram and Python (Part 1)

Published Nov 10, 2016

Chatbots are all the rage at the moment, with some predicting that they will be bigger than mobile apps. The main idea of chatbots is that instead of having to dig through awkward mobile menus and learn UIs, you'll simply have a conversation with a bot through a familiar
instant messaging interface. If you want to order a Pizza, you start a conversation with the Domino's Pizza bot and have the same conversation with it that you might have with a human.

There are a few different platforms that allow you to build your own chatbot. One of these, which is arguably the simplest to use and is also growing steadily in popularity, is Telegram.

In this tutorial, we'll walk through building a simple Telegram Bot using Python. At first, our bot will simply echo back any message we send it, but then we'll extend it to add a database and persist information across chat sessions.

We'll use Python to power our Bot and SQLite to store information persistently across sessions. In summary, this is a tutorial series that will:

- Show you how to write a simple Echo Bot from scratch using Python and the Telegram Bot API (Part 1)

- Show how to run our Bot from a VPS and allow it to scale to more users (Part 3).

Although creating an Echo Bot is simple enough, and you can find various scripts and frameworks online that will give you this as a starting point—we will do everything from scratch and explain every piece of code we write. We'll also look at some subtleties in the Telegram API, and talk about what these mean for us as developers. If you just want to create a Telegram bot as quickly as possible, this tutorial is probably not what you're looking for, but if you want to gain a deeper understanding of how chatbots work and how to build one from scratch, then you're in the right place.

What you need

You'll need to have:

- Some basic Python knowledge to follow this tutorial
- You should be comfortable with running commands in a Linux Shell, a MacOS Terminal, or a Windows Command Prompt
- You should be able to install Python packages using the pip package manager (or conda if you're more comfortable with that)
- Ideally, you should have written at least a basic SQL statement before, but this is not strictly necessary (and will only be relevant in Part 2).

All of the code is aimed at Python 3.5, but it should be easily adaptable to other versions of Python.

Why Python?

You can write a Telegram chat bot in any language you want. Some of the main options apart from Python would be Java, PHP, or Ruby. If you are more familiar with a different high-level programming language, then you might prefer to use that instead, but Python is a good choice for several reasons:

- Python can make HTTP requests very concisely and simply through the requests module. Getting the content from a URL (which is how we'll be controlling our Telegram Bot) would need many more lines of Java than the Python equivalent.
- Python is the most popular language for natural language processing and machine learning: although we won't be using either of these for our simple bot, both of them would be necessary for a more advanced Bot. Thus, if you want to extend the Bot, it's good to get comfortable with Python.

technologies such as WSGI to reach "web scale".

- Python is portable—we can easily run the same code on Linux, MacOS, or Windows.

Setting up

Nearly everything we do will be achievable using only the standard Python libraries, but we'll also be using the third-party `requests` module which provides a number of improvements to Python's `urllib` , and allows us to make HTTP requests very simply and concisely. Install this through pip using a command similar to the following (you may need to use `pip` instead of `pip3` and/or add the `--user` flag, based on how you usually install Python libraries).

```
pip3 install requests
```

If you usually use a virtual environment for new Python projects, then set one of those up first, and install requests inside that.

Creating a Telegram Bot

The first step is to tell Telegram that you want to create a new bot. All the messages that our Bot sends and receives will go through Telegram's infrastructure. Our code will periodically make a request to retrieve all new messages to our Bot from Telegram's servers, and will then send responses to each message as necessary. In order to register a bot with Telegram, you first need to create a personal Telegram account. Visit web.telegram.org and enter your phone number. Telegram will send you a text message (SMS), and you can then create an account by following the instructions on the screen. If you already have a Telegram account, then you can simply use that one, and you can also use any of the Telegram Desktop and Mobile apps available from telegram.org, instead of the Web app that we'll be using for all examples in this tutorial.

Once you have a Telegram account, you can register a new Telegram Bot by using Bot Father. Visit telegram.me/botfather to start a conversation with Telegram's bot that creates other bots. Telegram bots can receive *messages* or *commands*. The former are simply text that you send as if you were sending a message to another person, while the latter are prefixed with a `/` character. To create a new bot, send the following command to Bot Father as a chat (exactly as if you were talking to another

```
/newbot
```

You should get a reply instantly that asks you to choose a name for your Bot. We'll call our Bot `To Do Bot` because, by the end of this tutorial, it'll function as a simple "to do" list. Send the following message to Bot Father when it prompts you for a name:

```
To Do Bot
```

Bot Father will now ask you to pick a username for your Bot. This username has to end in `bot` , and be globally unique. As Telegram has grown more popular, it has become more difficult to find a short and relevant username for your Bot. In this tutorial, we'll be using `exampletodo_bot` , but for the rest of this tutorial, we'll indicate the Bot's username with `<your-bot-username>` , so you'll have to substitute your chosen username wherever relevant from now on. Send your chosen username to Bot Father:

```
<your-bot-username>
```

Now Bot Father will send you a "Congratulations" message, which will include a token. The token should look something like this:

```
2483457814:AAHrlCx234_VskzWEJdWjTsdfuwejHyu5mI
```

For the rest of this tutorial, we'll indicate where you need to put your token by using `<your-bot-token>` .

Take note of the token, as we'll need it in the code that we're about to write.

Interacting with our Bot through our web browser

We can control our Bot by sending HTTPS requests to Telegram. This means that the simplest way to interact with our Bot is through a web browser. By visiting different URLs, we send different commands to our Bot. The simplest command is one where we get information about our Bot. Visit the following URL in your browser (substituting the bot token that you got before)

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

The first part of the URL indicates that we want to communicate with the Telegram API (api.telegram.org). We follow this with `/bot` to say that we want to send a

Enjoy this post?

we want to send the command to and to prove that we own it. Finally, we specify the command that we want to send (`/getme`) which in this case just returns basic information about our Bot using JSON. The response should look similar to the following:

```
{"ok":true,"result":{"id":248718785,"first_name":"To Do Bot","username":"e
```

Retrieving messages sent to our Bot

The simplest way for us to retrieve messages sent to our Bot is through the `getUpdates` call. If you visit `https://api.telegram.org/bot<your-bot-token>/getUpdates` , you'll get a JSON response of all the new messages sent to your Bot. Our Bot is brand new and probably hasn't received any messages yet, so if you visit this now, you should see an empty response.

Telegram Bots can't talk to users until the user first initiates a conversation (this is to reduce spam). In order to try out the `getUpdates` call, we'll first send a message to our Bot from our own Telegram account. Visit `telegram.me/<your-bot-username>` to open a conversation with your Bot in the web client (or search for `@<your-bot-username>` in any of the Telegram clients). You should see your Bot displayed with a `/start` button at the bottom of the screen. Click this button to start chatting with your Bot. Send your Bot a short message, such as "hello".

Now visit the `https://api.telegram.org/bot<your-bot-token>/getUpdates` URL again, and you should see a JSON response showing the messages that your bot has received (including one from when you pressed the start button). Let's take a look at an example of this and highlight the import data that we'll be writing code to extract in the next section.

```
{"ok":true,"result":[{"update_id":625407400,
"message":{"message_id":1,"from":{"id":24860000,"first_name":"Gareth","last
"message":{"message_id":2,"from":{"id":24860000,"first_name":"Gareth","last
```

The `result` section of the JSON is a list of updates that we haven't acknowledged yet

Enjoy this post?

it is part of, and the contents of the message. The two pieces of information that we'll focus on for now are the chat ID, which will allow us to send a reply message and the message text which contains the text of the message. In the next section, we'll see how to extract these two pieces of data using Python.

Sending a message from our Bot

The final API call that we'll try out in our browser is that used to send a message. To do this, we need the chat ID for the chat where we want to send the message. There are a bunch of different IDs in the JSON response from the `getUpdates` call, so make sure you get the right one. It's the `id` field which is inside the `chat` field (24860000 in the example above, but yours will be different). Once you have this ID, visit the following URL in your browser, substituting `<chat-id>` for your chat ID.

```
https://api.telegram.org/bot<your-bot-token>/sendMessage?chat_id=<chat-id>&
```

Once you've visited this URL, you should see a message from your Bot sent to your which says "TestReply".

Now that we know how to send and receive messages using the Telegram API, we can get going with automating this process by writing some logic in Python.

Writing the Python code for our Bot

Now we can get to writing Python. Create the file `echobot.py` and add the following code:

Enjoy this post?

```python
import json
import requests

TOKEN = "<your-bot-token>"
URL = "https://api.telegram.org/bot{}/".format(TOKEN)


def get_url(url):
    response = requests.get(url)
    content = response.content.decode("utf8")
    return content


def get_json_from_url(url):
    content = get_url(url)
    js = json.loads(content)
    return js


def get_updates():
    url = URL + "getUpdates"
    js = get_json_from_url(url)
    return js


def get_last_chat_id_and_text(updates):
    num_updates = len(updates["result"])
    last_update = num_updates - 1
    text = updates["result"][last_update]["message"]["text"]
    chat_id = updates["result"][last_update]["message"]["chat"]["id"]
    return (text, chat_id)


def send_message(text, chat_id):
    url = URL + "sendMessage?text={}&chat_id={}".format(text, chat_id)
    get_url(url)


text, chat = get_last_chat_id_and_text(get_updates())
send_message(text, chat)
```

Let's pull apart what this code does:

- In lines 1 and 2, we import the `requests` and `json` modules. The first is to make web requests using Python and we'll use it to interact with the Telegram API (similarly to what we were using our web browser for earlier). We'll use the JSON module to parse the JSON responses from Telegram into Python dictionaries so that we can extract the pieces of data that we need.
- The next two lines are global variables, where we define our Bot's token that we need to authenticate with the Telegram API, and we create the basic URL that we'll be using in all our requests to the API.
- The `get_url` function simply downloads the content from a URL and gives us a string. We add the `.decode("utf8")` part for extra compatibility as this is necessary for some Python versions on some platforms. Normally, we'd do some exception handling here as this request could fail if our internet connection were down, if Telegram's service were down, or if there were an issue with our Token. However for simplicity, here we'll simply assume that everything Always Works (TM).
- The `get_json_from_url` function gets the string response as above and parses this into a Python dictionary using `json.loads()` (`loads` is short for Load String). We'll always use this one as Telegram will always give us a JSON response.
- `get_updates` calls the same API command that we used in our browser earlier, and retrieves a list of "updates" (messages sent to our Bot).
- `get_last_chat_id_and_text` provides a simple but inelegant way to get the chat ID and the message text of the most recent message sent to our Bot. Because get_updates will always send all the messages that were recently sent to our bot, this is not ideal, as we will always download a whole bunch of messages when we only want the last one. We'll discuss later in more detail how to do this more elegantly. For now, this function returns a tuple of the `chat_id` which identifies the specific chat between our Bot and the person who sent the message, and the `text` , which is the message itself.
- `send_message` takes the text of the message we want to send (`text`) and the chat ID of the chat where we want to send the message (`chat_id`). It then calls the `sendMessage` API command, passing both the text and the chat ID as URL parameters, thus asking Telegram to send the message to that chat.

The final two lines bring everything we have written together to actually receive and send a message. First, we get the text and the chat ID from the most recent message sent to our Bot. Then, we call `send_message` using the same text that we just received, effectively "echoing" the last message back to the user.

At the moment, our script doesn't listen for new messages and immediately reply

Enjoy this post?

and echo it. We can test it out by sending our bot a message, and then running the script. Give this a go!

Flaws with our bot

The most obvious problem with our Bot is that we have to run a Python script manually every time we want to interact with it. Also, as mentioned before, we always download the entire message history that Telegram provides. This is both inefficient and unreliable, as we don't want to unnecessarily download the entire message history if we only want a single message, and because Telegram only keeps this list of updates for 24 hours. Another issue is that we pass our message as a string, but because this is converted to a URL before being sent to Telegram, you'll notice that some unexpected things happen if you send messages to the bot with special characters (for example, the + symbol will disappear from all echoed messages). Finally, the Bot throws an index error if we try to run it when there are no new messages to receive.

We'll now update our bot to:

- Constantly listen for new messages and reply to each.
- Acknowledge each message as it receives it and tells Telegram to not send us that message again.
- Use Long Polling so that we don't have to make too many requests.
- Correctly encode our messages to account for URL formatting.

Listening for new messages

We don't want to manually start our Bot every time that we want it to reply to the latest message, so the first thing to do is to wrap our code that receives new messages and echoes them back in a loop. We'll also put this in a `main` function and use the Pythonic `if __name__ == '__main__'` statement so that we could import our functions into another script without running anything. We don't want to ask for new updates as fast as possible, so we'll also put a small delay between requests (this is kinder to Telegram's servers and better for our own network resources, too).

At the top of the file, add a new import for the Python `time` module

```
import time
```

Enjoy this post?

```
def main():
    last_textchat = (None, None)
    while True:
        text, chat = get_last_chat_id_and_text(get_updates())
        if (text, chat) != last_textchat:
            send_message(text, chat)
            last_textchat = (text, chat)
        time.sleep(0.5)


if __name__ == '__main__':
    main()
```

This code now gets the most recent messages from Telegram every half second. We now also need to remember the most recent message that we replied to (we save this in the `last_textchat` variable) so that we don't keep on sending the echoes every second to messages that we've already processed. This is again a very crude way of achieving what we want (for example, if we send the same message to our bot twice in a row, it won't reply to the second one), but we'll see a more elegant way to achieve this below. For now, you can run this code and now instead of the script terminating, you'll see that it keeps running. You can now send your Bot a series of messages, and (as long as you don't send more than one per half second), you'll see each of them getting echoed back again.

Acknowledging the messages we've already seen

Instead of asking Telegram for all our recent messages with every call, and then trying to figure out which ones we are interested in, we can tell Telegram that we've already processed certain messages and that we want to stop receiving them as part of the `getUpdates` calls. Each update has an `update_id` field, and these are incremental (later messages have higher numbers). When we make the `getUpdates` API call, we can optionally pass an `offset` argument and give an `update_id` as the value. This tells Telegram that we've already seen and processed that message and that we don't want it again. This also means that Telegram will never send us any of the previous messages (messages with a lower `update_id`) again either, so we need to make sure that we really are finished with all of the messages before doing this.

Modify our bot as follows:

- Add an optional `offset` parameter to our getUpdates function. If this is specified,

```python
def get_updates(offset=None):
    url = URL + "getUpdates"
    if offset:
        url += "?offset={}".format(offset)
    js = get_json_from_url(url)
    return js
```

- Add a function that calculates the highest ID of all the updates we receive from getUpdates. This should look as follows.

```python
def get_last_update_id(updates):
    update_ids = []
    for update in updates["result"]:
        update_ids.append(int(update["update_id"]))
    return max(update_ids)
```

This simply loops through each of the updates that we get from Telegram and then returns the biggest ID. We need this so that we can call `getUpdate` s again, passing this ID, and indicate which messages we've already seen.

- Add a function to send an echo reply for each message that we receive. This should look as follows:

```python
def echo_all(updates):
    for update in updates["result"]:
        try:
            text = update["message"]["text"]
            chat = update["message"]["chat"]["id"]
            send_message(text, chat)
        except Exception as e:
            print(e)
```

- Update the code in `main()` so that it looks like this:

Enjoy this post?

```
def main():
    last_update_id = None
    while True:
        updates = get_updates(last_update_id)
        if len(updates["result"]) > 0:
            last_update_id = get_last_update_id(updates) + 1
            echo_all(updates)
        time.sleep(0.5)
```

Our main code no longer needs to worry about duplicate messages, as each time we get new messages, we send the biggest update_id along with the next request, ensuring that we only ever receive messages that we haven't seen before.

Note that we have to check if there are new updates (which we do in the third line of `main()`), and that we have to always send an update ID which is one bigger than the previous one we've seen (i.e. we're actually telling Telegram which ID we're expecting, not which one we've seen).

Try out the changes by restarting the Python script and sending some messages to your Bot—you should see that it works as before, but now it doesn't matter if you send duplicate messages or send messages too quickly, both of which are big improvements.

Using Long Polling

The last major problem with our Echo Bot is that it has to make a web request every 0.5 seconds. This is not great for Telegram's servers (they explicitly ask people not to do this outside of testing scenarios) and not great for our resources either. Long Polling takes advantage of the fact that most of the time, we are receiving "empty" responses. Because our Bot is probably not going to be receiving messages every half second, most of the time when we ask for updates, there aren't any. With Long Polling, instead of Telegram telling us that there aren't updates, it simply keeps the connection open until there are updates, and then sends these down the open pipe. Of course, it's impractical to keep a connection open forever, so we can specify the number of seconds that we want to wait for. This is done by passing another optional argument to the `getUpdates` call, namely `timeout` .

To make our code use Long Polling, simply update our `get_updates` method as follows:

```
def get_updates(offset=None):
    url = URL + "getUpdates?timeout=100"
    if offset:
        url += "&offset={}".format(offset)
    js = get_json_from_url(url)
    return js
```

Now we always pass along the `timeout` argument. Because we now have two arguments, we also need to change where we previously had `?offset={}` to `&offset={}` (in URLs, we specify that the argument list is starting with a `?` but further arguments are separated with `&`).

Run the bot again, and it should run exactly as before, but now it'll be making far fewer requests and using less of your machine's resources. If you want to check that this is working, simply add a line like `print("getting updates")` directly below the `while True` in the `main` function and run the bot with and without the `timeout` argument that we just added. Without the timeout, you'll see that the bot checks for updates every 0.5 seconds. While with the timeout, it will only initiate a new check every 100 seconds, or whenever a new message is received.

Correctly encoding our message text

The final problem of our echo bot is that it acts strangely if we send it messages containing special characters. For example, all + signs disappear from our messages, and all text after an & sign disappears, too. This is caused by these symbols having special meanings in the context of URLs. To fix this, we need to encode any special characters in our message. Luckily, the standard Python `urllib` has a function that handles this for us, so we only need to import that and add a single line of code.

Add the following line at the top of your .py file

```
import urllib
```

And now modify the `send_message` function to read as follows:

Enjoy this post?

```
def send_message(text, chat_id):
    text = urllib.parse.quote_plus(text)
    url = URL + "sendMessage?text={}&chat_id={}".format(text, chat_id)
    get_url(url)
```

Restart the Bot once more, and send it some messages that were problematic before, such as:

```
+
Hello+
Hello&test
```

Now it should be able to reply to all of these messages (and pretty much anything else you throw at it, including emoji) flawlessly.

End of Part 1

That brings us to the end of the first part of this tutorial. We built a simple Echo Bot using the Telegram Bot API from scratch and implemented some more advanced features such as keeping track of which messages we'd already processed, using Long Polling, and correctly encoding our messages for URLs. In the next part, we'll add a database and turn our Bot into something more useful—a To Do List.

The final code listing for the Echo Bot presented here can be found at https://github.com/sixhobbits/python-telegram-tutorial.

Part 2 of this tutorial can be found here.

Part 3 of this tutorial was a live demo of how to deploy the bot to a VPS.

Enjoy this post?

Python Telegram Chatbot

Enjoy this post? Give **Gareth Dwyer** a like if it's helpful.

188 133 SHARE

Gareth Dwyer

Pythoneer and Writer

Hey, I'm Gareth. I love teaching and most things Python. I have general experience in problem solving, building scalable solutions, and can provide specific or general advice.

FOLLOW

133 Replies

Leave a reply

Mor Yosef 13 days ago

Enjoy this post?

Reply

Onyeibo Oku 24 days ago

I will buy a book from you any day sir. This is illuminating

Reply

Gareth Dwyer 24 days ago

You can check out 'Flask by Example' on PacktPub that I wrote, but it's a bit old now and uses Python 2, so maybe look for more recent books rather :)

Glad you liked it though!

1 Reply

hanafee jamaluudin 2 months ago

great tutorial Gareth Dwyer

Reply

Gareth Dwyer 24 days ago

Thanks you! I'm glad you liked it

Reply

Show more replies

Ketan Bhatt

Build Systems with Speed and Confidence by Closing the Loop First!

I re-learnt something recently: the importance of closing the loop on a system you are trying to build, as quickly as possible, and then adding the juicy bits later (Thank you Kesha for helping me with the concept 😊).

A completely finished "loop" is when you can provide the required input to your

Enjoy this post?

READ MORE

Enjoy this post?