4-12-2016

Make a Twitter Bot in Python: Iterative Code Examples

Robin Camille Davis  
*CUNY John Jay College*

Mark E. Eaton  
*Kingsborough Community College, CUNY*

This Article is brought to you for free and open access by the Kingsborough Community College at CUNY Academic Works. It has been accepted for inclusion in Publications and Research by an authorized administrator of CUNY Academic Works. For more information, please contact AcademicWorks@cuny.edu.

Recommended Citation
Twitter bots are everywhere on Twitter, making us laugh, annoying us, and occasionally spitting out profound truths. These bots are made by artists and activists, scholars and spammers. And as it turns out, building a Twitter bot is a fun and productive way to introduce yourself to basic programming in Python. We have provided five sample scripts that work with pretty minimal set-up, along with instructions and suggestions for customizing the scripts.

This tutorial is based on the LACUNY Emerging Technologies Committee’s “Build Your Own Twitter Bot” day in December 2015, which was billed as a gentle introduction to programming in Python. Below, we will expand on some of our insights and examples from this workshop.

Set up your bot

If you are creating a bot, it is a good idea — although not strictly necessary — to create a Twitter account dedicated exclusively to your bot. You could run your bot on an existing account, such as your personal Twitter login, although this will depend on what your bot is doing. You probably do not want your followers to see the test tweets you will send out.

Your bot’s account will need to be registered with apps.twitter.com, and you will need to authenticate the account using a mobile phone number. Authenticating allows you to get Twitter API keys for your bot, which effectively serve as a password to the Twitter API. The Twitter API is an “Application Programming Interface,” which allows your application (your bot) to interact programmatically with Twitter and exchange data with Twitter. The API helpfully allows you to build programs that expand upon the existing functionality of Twitter. In the examples below, your bot will use the API to send tweets. When you register your Twitter “app” (your bot), API keys are created. These can be found under the “Keys and Access Tokens” tab of your app’s page.
It is also necessary to generate an Access Token and Access Token Secret by clicking on “Generate My Access Token and Token Secret.” Keep track of these keys and tokens; you will need them for your bot.

After obtaining your Keys and Tokens, download the bot scripts from GitHub. These will work almost immediately, after minimal setup. The setup may be a bit more complicated if you are using a Windows machine; there are some suggestions for Windows users below.

Paste the Keys and Tokens into the credentials.py file, in the spots indicated with the placeholders XXXXXXX (see code below). These credentials are used by each of the sample bots when they communicate with the Twitter API.

```python
# Credentials for your Twitter bot account

# Original script (kept up to date): https://github.com/robincamille/bot-tutorial/blob/master/credentials.py

# 1. Sign into Twitter or create new account
# 2. Make sure your mobile number is listed at twitter.com/settings/devices
# 3. Head to apps.twitter.com and select Keys and Access Tokens

CONSUMER_KEY = 'XXXXXXX'
CONSUMER_SECRET = 'XXXXXXX'
ACCESS_TOKEN = 'XXXXXXX'
ACCESS_SECRET = 'XXXXXXX'
```

Figure 1: Keys and Access Tokens
Set up your environment

To get started, you will need IDLE, a development environment for Python; and Pip, the Python Package Manager (which we will use to install a couple of code libraries). For most Linux and Mac users, these items may already be in place.

Open the command line, also called ‘Terminal’ on a Mac computer. This should appear as a blank window with your username, computer name and a cursor. If you are using Windows and you do not have a command line interface on your machine, you can get one by installing cygwin or git bash. You may also have to modify your PATH variable. These steps are beyond the scope of this tutorial. Alternately, some Python functionality is available at the DOS prompt, as described here.

1. At the command line, check to make sure you have Python installed by typing:

```
python --version
```

This should return something similar to:

```
Python 3.5.1
```

Any Python version beginning with 2.7 or 3 will work for the bots in this tutorial. If you do not have Python installed on your machine, you can download it here:

[python.org](http://python.org)

2. IDLE is a basic Integrated Development Environment (IDE) which is included with any modern version of Python. Its purpose is to make programming and debugging your Python scripts a bit easier by allowing you to edit, run and debug your program in one interface. To make sure it is available on your machine, call it at the command line by typing:

```
idle
```

IDLE should open automatically.

3. Check in the command line to make sure you have Pip, the Python Package Manager by typing:

```
pip --version
```

This should return something like this:

```
pip 8.0.2
```

Pip is needed to install some of the packages that are used in the subsequent scripts. Pip should already be installed if you have Python 2.7.9+ or Python 3.4+.

4. Once Pip is installed, you can use it on the command line to install two libraries that are necessary for this tutorial:

```
pip install tweepy
```

```
pip install setuptools
```

Tweepy lets you use the Twitter API through Python. It depends on the setuptools library to work properly.

The urllib2 and json libraries should be included by default, but to check for them, type the following:

```
pip install urllib2
```

```
pip install json
```

If you receive an error about permissions when installing these libraries, put sudo in front of pip (e.g., sudo pip install tweepy). This will work if you have administrator privileges on your machine; sudo bypasses permissions checks, so use it carefully.
Now you are ready to begin building bots! Each bot below demonstrates different functionality of Python and of the Twitter API. They are all working bots. We have presented them in order of complexity, from the simplest to the most complicated. You can run these bots as they are, or modify them to your liking!

A basic bot

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

def main():
  # This bot tweets three times, waiting 15 seconds between tweets.

  # Housekeeping: do not edit
  import tweepy, time

  auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
  auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)

  api = tweepy.API(auth)

  tweetlist = ['Test tweet one!', 'Test tweet two!', 'Test tweet three!']

  for line in tweetlist:
      api.update_status(line)
      print(line)
      print('...')
      time.sleep(15) # Sleep for 15 seconds
  print('All done!')

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

The code above is presented for illustration only. Use the scripts you downloaded and edit them on your computer.

Open `mybot.py` in IDLE with this command at the command line:

```
idle mybot.py
```

Select **Run > Run Module** from the menu bar to run the bot. The other Python scripts that follow can be opened and run in the same way.

The bot will send out three tweets, at regular intervals. These tweets are pre-defined “strings.” Strings in Python are enclosed in single or double quotes. In this case, the strings are specified as a “list”— designated by square brackets in Python — and this list is assigned to the variable `tweetlist`:

```python
tweetlist = ['Test tweet one!', 'Test tweet two!', 'Test tweet three!']
```

Modifying the strings will alter what the bot tweets. The list of strings could, of course, be expanded, modified, or shortened. However, running this script repeatedly without changing it will present error messages, because the Twitter API will not allow re-posting of identical tweets. Try changing the strings to tweet whatever you like!
If you receive confusing error messages that we do not mention here, copy and paste them into a Google search box. The chances are high that someone has explained the problem on Stack Overflow or in a similar forum.

### Tweeting lines from a text file

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Original script (kept up to date): https://github.com/robincamille/bot-tutorial/blob/master/mybot2.py

# Twitter Bot Starter Kit: Bot 2

# This bot tweets a text file line by line, waiting a
given period of time between tweets.

# Download a Project Gutenberg "Plain Text UTF-8" file,
# open it in Notepad, remove junk at beginning,
# and replace all double-linebreaks with single linebreaks.

# Housekeeping: do not edit
import tweepy, time
from credentials import *
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)
api = tweepy.API(auth)

# What the bot will tweet
filename = open('twain.txt','r')
tweettext = filename.readlines()
filename.close()

for line in tweettext[0:5]: # Will only write first 5 lines
    api.update_status(line)
    print line
    time.sleep(15) # Sleep for 15 seconds

print "All done!"

# To quit early: CTRL+C and wait a few seconds
```

The code above is presented for illustration only. Use the scripts you downloaded and edit them on your computer.

*Code for mybot2.py* (Davis and Eaton 2016)

The second bot, `mybot2.py`, expands on the first bot by drawing its tweets line by line from a text file. In this case, a novel by Mark Twain is loaded into a variable called `filename` using this code (line 24):

```python
filename = open('twain.txt','r')
```

`twain.txt` is the text file, and `r` specifies that this file is being read. This example helpfully shows how Python can utilize data from outside sources, in this case a text file. This bot could be easily modified by choosing another text file, for example a book or poem from Project Gutenberg or some other source. Pre-processing the text by removing double line-breaks and header information is necessary for best results.

This bot iterates through the lines of the chosen text using a `for` loop. The `for` loop is initiated with this line:

```python
for line in tweettext[0:5]:
```
The indented lines that follow are executed with each iteration of the loop. Indentation and whitespace are very important in Python; this for loop will only work as expected by maintaining this indentation.

`tweettext` is the variable containing the text. The text is stored as a list of sentences, similar to the first bot.

`tweettext[0:5]` means that the looping will begin at the beginning of the text, and end at the fifth line. In Python this is called a “slice.” You can read up on slicing here (the bit on slicing is about halfway down).

In an interesting use of this bot, Jill Cirasella (CUNY Graduate Center), modified a spreadsheet of recent institutional repository uploads so her bot could tweet titles and links:

```
@GCDailyDiss
Integration or Interrogation? Franco-Maghrebi Rap and Hip-Hop Culture in Marseille:
http://academicworks.cuny.edu/ge_etds/195
(link to tweet)
```

### Using JSON for madlibs

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Original script (kept up to date): https://github.com/robincamille/bot-tutorial/blob/master/mashup_madlib.py

# Mad Lib Maker!

# This script will generate mad-libs based off of a William Carlos
# Williams poem, 'The Red Wheelbarrow.' Each poem will then be
# tweeted by your bot account.

# Housekeeping: do not edit
import json, io, tweepy, time, urllib2
from random import randint
from credentials import *
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)
api = tweepy.API(auth)

# Housekeeping: opening JSON files
# Find more item lists at https://github.com/dariusk/corpora/tree/master/data
# ***Click "Raw" button, copy URL***
list1file = urllib2.urlopen('https://raw.githubusercontent.com/dariusk/corpora/master/data/objects/objects.json')
list1read = list1file.read()
list2file = urllib2.urlopen('https://raw.githubusercontent.com/dariusk/corpora/master/data/foods/menuItems.json')
list2read = list2file.read()
list3file = urllib2.urlopen('https://raw.githubusercontent.com/dariusk/corpora/master/data/humans/occupations.json')
list3read = list3file.read()

# Create Python-readable lists of items in JSON files
list1 = json.loads(list1read)['objects'] # Change 'objects' to the title of the list
list2 = json.loads(list2read)['menuItems']
list3 = json.loads(list3read)['occupations']

# Repeatable poem-generator
poemlist = []
counter = 0

while counter < 2: # Change 2 to however many poems you want to produce
```
The code above is presented for illustration only. Use the scripts you downloaded and edit them on your computer.

Code for mashup_madlib.py (Davis and Eaton 2016)

The third bot, mashup_madlib.py, produces a madlib of a William Carlos Williams poem, drawing randomly from lists of words by topic from JSON corpora compiled by Darius Kazemi, a prolific bot-maker. This bot also draws upon other interesting aspects of Python, such as random number generation, while loops, and inserting variables into strings of text.

Variations on this bot arguably produce some of the most amusing tweets of the bots we have tried so far. Try rewriting the poem or switching to different wordlists from the JSON corpora. For instance, you could use this list of Greek monsters instead of the list of objects for list1:

```
list1file = urllib2.urlopen('https://raw.githubusercontent.com/dariusk/corpora/master/data/mythology/greek_monsters.json')
```

If you pick a list from Kazemi’s JSON corpora, you will need to use the raw JSON for that corpus. To find the raw JSON, click the “Raw” button on the Github page for that corpus. Use the URL for the raw page. This URL will begin with http://raw… Your bot script will be able to read this data.

One more thing: you must specify the name of the list as it appears in the raw JSON file. For example, the name of the list ('greek_monsters') is near the top of the JSON file:

```
{
  "description":"Monsters from Greek myth",
  "greek_monsters": [
    "Arachne",
  ]
```

In your bot code, you will refer to the name of the list like this:

```
list1 = json.loads(list1read)['greek_monsters']
```
You may encounter problems when the JSON list you want to use is deeply nested, like in this list. Although we will not dive into the `json` library in this tutorial, it is worth exploring further if you want to learn more about nested lists.

One participant in our workshop, Leslie Ward (Queensborough Community College), chose dinosaur species and -ism words to fill in the blanks of the poem:

```
@lesliepythonbot
so much depends / upon / a Diamantinasaurus / glazed with / eclecticism / beside the / Eucercosaurus
(link to tweet)
```

A bot that responds to Twitter activity

```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Original script (kept up to date): https://github.com/robincamille/bot-tutorial/blob/master/respondingbot.py

# Twitter Bot Starter Kit: Responding Bot

# This bot listens to the account @ocertat, and when that account
# tweets, it responds with a line of Twain

# Download a Project Gutenberg "Plain Text UTF-8" file,
# open it in Notepad, remove junk at beginning,
# and replace all double-linebreaks with single linebreaks.

# A bot that responds to Twitter activity

import tweepy
import time
from credentials import *
from random import randint
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)
api = tweepy.API(auth)

# initially, the script will assume that the last tweet was a null value
lasttweet = None

# What the bot will tweet
filename = open('twain.txt', 'r')
tweettext = filename.readlines()
filename.close()

# a function that picks a random line
def linenum():
    return randint(0, len(tweettext))

# this is the function that does most of the work of the bot
def runTime():
    global lasttweet

    # gets the most recent tweet by @ocertat and prints its id
    mostrecenttweet = api.user_timeline('ocertat')[0]
    print(mostrecenttweet.id)

    # compares the two tweets, and tweets a line of Twain
    if mostrecenttweet != lasttweet:
        line = tweettext[linenum()]
        api.update_status(line, mostrecenttweet.id)

    lasttweet = mostrecenttweet

# run the bot non-stop
while True:
    runTime()
```

```
While all of the previous bots use Tweepy’s `update_status` method to post tweets, `respondingbot.py` uses the `user_timeline` method to bring some different functionality to our use of the Twitter API. `Respondingbot.py` bot will listen to a particular Twitter account, and when it detects activity, it will immediately tweet a random line of text, again from Mark Twain.

To understand the various methods available in the Tweepy library, look at the Tweepy documentation pages. Documentation is a helpful, although sometimes frustrating, way to discover the many things that a library can do for our code. Often, documentation is most helpful when looked at alongside working examples of the code in use.

To customize this bot, pick another Twitter user to “listen” to (line 44):

    mostrecenttweet = api.user_timeline('ocertat')[0]

When that user tweets, your bot will tweet the line:

    line = tweettext[linenum()]

Currently, the variable, line, is set to be a random line from a text file, by default `twain.txt` (defined in line 27). You can choose a different text file.

To customize your bot to tweet directly to the person it is “listening” to, you can add “@accountname” to the line, like so:

    line = "@lacunybot " + tweettext[linenum()]

Note that the bot is not responding to a particular tweet or thread, but is simply @-ing at the account as soon as the other account tweets anything. (Use this bot responsibly!)

Robin, one of the workshop leaders, modified `respondingbot.py` to respond to another participant’s tweets with lines from a compliment corpus:

    @lacunybot
    @[participant] You’re even better than a unicorn, because you’re real.
    (link to tweet)

## Using Markov chains

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

# Usage: python markovmaker.py
# Original script (kept up to date): https://github.com/robincamille/bot-tutorial/blob/master/mashup_markov/markovmaker.py

# Markov-chain text maker
```
# This script takes a .txt file and makes a mashup of it
# using a Markov chain: linking word phrases together
# from different spots in the text.

# For instance, if the text contained two lines,
# "she has a dog" and "my dog has a tail,"
# this might generate "my dog has a dog" and "she has a tail."

# Housekeeping
import markovgen, re, string

# Choose original file, new filename
original = open('twain.txt')
outfile = open('twain_markov.txt','w')

# Repeatable Markov'd text generator
newtext = []
mk = markovgen.Markov(original)

counter = 0
while counter < 10:  # Change 10 to however many lines you want to generate
    line = '
' + mk.generate_markov_text()

    # Remove punctuation
    exclude = [',','.','(',')',';']
    line = ''.join(ch for ch in line if ch not in exclude)

    # Make line lowercase, add period at end
    line = line.lower() + '.'

    print line
    newtext.append(line)
    counter = counter + 1

for aline in newtext:
    outfile.write(aline)  # Makes text file line by line

outfile.close()
original.close()

# Next steps if you want to tweet these lines:
# Move the newly-made file into the mybot folder
# Open mybot2.py
# Insert new filename

# Original MarkovGen library from https://github.com/mattspitz/markovgen - modified by RobinCamille to spit out smaller chunks

The code above is presented for illustration only. Use the scripts you downloaded and edit them on your computer.

Code for markovmaker.py (Davis and Eaton 2016)
Lastly, markovmaker.py makes use of Markov chains to generate semi-intelligible text from a source text, like a book from Project Gutenberg. The script selects bits of text and links them to other bits using a common word. For instance, the source text *My dog has a tail. Our neighbor has a PhD.* might generate *My dog has a PhD.*, linking the phrases on the common word *has.* Generating text with Markov chains often results in gibberish, and it can sometimes be funny or profound.

To make your own gibberish using markovmaker.py, choose a source text file (line 21) and name your output file whatever you want (line 22), for example:

```python
original = open('twain.txt')
outfile = open('twain_markov.txt','w')
```

In this example, the source `twain.txt` file must be in the same directory as the python script itself. Once you run the script, it will print the gibberish lines on the screen and also save them in a .txt file called `twain_markov.txt`. The gibberish will look something like: “didn’t lose the chance. you see, i was all the time and whole. / his fellows into her presence. king uriens of the pieties enjoined the.”

So you have made a text file containing gibberish. How will we get your bot to tweet it? We can reuse mybot2.py, which you will recall tweets lines from a source text. You can use the text file you just created as the source file. Do not forget to include the directory it is in:

```python
filename = open('mashup_markov/twain_markov.txt','r')
```

Now your bot is tweeting Twain-flavored gibberish. Try it with another plain-text file from Project Gutenberg. Or, for an Uncanny Valley experience, use a plain-text file of your own writing. You do not have to work directly with the complex supplemental script that powers Markov chain generation (we simply imported it with the line `import markovgen`), but you could peek under the hood of `markovgen.py` to see what more complicated Python looks like. (Original code for Markov chaining came from Shabda Raaj.)

### Next steps

So you have gotten the hang of using the Twitter API to run bots… What next? You could now write another bot from scratch. Does your institution or pet project have data that could be turned into an engaging series of tweets? Is there a problem you could solve with bots, like online harassment? Do you have a research interest that needs more public attention? Go forth and bot!

---

### Insights from running a bot workshop to teach Python

We used these scripts to introduce a room full of librarians to programming in Python. The librarians attending, most of whom had programmed previously in other languages, got a chance to immerse themselves in some sample Python scripts. The goal was to get all attendees up and running with bots within half an hour.

In our experience, many code workshops get off to a rough start when attendees are unable to get the right dependencies and packages installed on their laptops. We held our workshop in a lab, and we pre-installed all the necessary libraries and IDLE on the desktop computers. As a result, very little setup time was needed during the workshop, and participants were able to progress quickly because their coding environment was stable and consistent. The only downside is that participants had to transfer their scripts off the lab computers if they wanted to keep working on their bots.

During the workshop, as we progressed through each bot, we worked on our code individually and in small ad hoc groups. Impromptu modifications of the code did not always work as expected, so we helped each other debug code and showed off interesting customizations. As workshop leaders, we did not explain every part of the code, since part of the fun of learning programming is breaking and fixing code yourself to see what each part does.

Progressing through these examples worked well for the audience of tech-savvy librarians who attended our workshop. As an added bonus, some of the nonsense that their bots generated was amusing or surprising. Some participants later said they wanted to keep their bot running as an ongoing project, which was wonderful to hear, since the goal of the workshop was to have...
everyone making and deploying bots. Overall, our workshop was an entertaining and productive way to introduce Python.

References:

https://github.com/robincamille/bot-tutorial/

Robin Camille Davis is the Emerging Technologies & Distance Services Librarian at John Jay College of Criminal Justice (CUNY), where she leads digital projects and pursues the future of libraries.

Mark Eaton is a Reader Services Librarian and Assistant Professor at Kingsborough Community College (CUNY). He is responsible for social media at the Kingsborough Library, and works broadly on technology projects that support his colleagues and students.