

01-Feb-2017

Faculty of Computer Science, Dalhousie University

Shiftkey-Py Week 1

“Ain’t No Party Like a Twee-Py Party”

Location: Shiftkey Labs

Time: 5:30pm-7:00pm

Welcome to ShiftyKey-py tutorial! This is an extra-curricular tutorial on Python provided by ShiftKey Labs and Dalhousie's Master of Electronic Commerce program. The tutorial is designed for highly motivated students who are familiar with coding (but not necessarily Python) and want to learn to learn Python and Django from scratch. It's a highly experimental tutorial, so bear with us as we try to do something new.

The main objective of this tutorial is to go from knowing no Python to knowing how to develop a Django web app. However, before we can build an app, we need to know how to develop in the Python programming language. To learn Python, we will review some things very quickly, and build a simple app using the python Twitter API. If you aren't yet comfortable with Python, I strongly recommend that you review the Google Python class, or Codecademy <https://developers.google.com/edu/python/?csw=1>. The goal of this week's session is to get up and running with the Python and the Twitter API.

Why might you want to learn Python? The Python programming language is an intuitive *scripted* and *interpreted* language used in a very wide variety of cases. Basically, if you aren't a professional software engineer, Python is great to learn because a) it is easy to read b) there are a wide number of uses for this tool c) you get a lot of help with debugging. Python is also the "go to" language of Data Scientists, and after all, we all want to be Data Scientists so that we can make a lot of money: <http://www.kdnuggets.com/2014/04/elusive-data-scientists-driving-high-salaries.html>

What is Django? Django is Python's most popular web development framework, and is among the most popular web development tools. Django has been used to develop sites such as Pinterest and Eventbrite (see more at <https://www.shuup.com/en/blog/25-of-the-most-popular-python-and-djangowebsites/>). Though YouTube and Google do not use Django specifically, they use their own Python-based frameworks. Django follows the Model/View/Controller paradigm, and is thus like other popular web development frameworks. It's a great starting point to modern web applications.

If you are using Windows, the very first thing you will need to do is install and configure Python. You can read about how to do that by clicking on "Python Set Up" in the Google tutorial. Note for the purposes of this tutorial, we will be using Python 2.7. If you are using Linux, you likely already have Python installed. It's just a matter of writing "python" in the terminal to get this running.

STEP 1: LEARN YOUR FIRST PYTHON, QUICKLY

Once you have Python configured, you should take a look at Google's "Introduction" section. In this section, they introduce key concepts in Python. It's interesting that they recommend that you take a MOOC along side the tutorial... we will be running our own "MOOC" instead. One thing that's worth noting if you click on their selected readings. There is a book called "Learn Python the Hard Way". This is a REALLY good resource; I strongly recommend that reading as well.

So, OK. A good way to start with python is using the shell. If you use Windows, the way to access this is through a program called IDLE. Run that program and you will see the command line. As the tutorial suggests, try playing around with it. Try to see what you can do.

First steps: variables and functions. In programming, you can declare a variable to assign a value. Let's start by assigning a number. A typical example of how this is done in Python is:

```
>>> secret = 42
```

Note that you can name variables anything you would like. In this case, I chose the name "secret", reminiscent of Douglas Adams' "the secret to life, the universe and everything." By doing this, I can more easily remember the name of the thing I want to change if my code discovers a different answer to this question. Declaring variables allows me to store values, which can be retrieved, used and altered throughout the process. For instance, if we wanted to add value to secret, we might try the following:

```
>>> secret += 2
```

...and then tell the python interpreter to return the value:

```
>>> secret
44
```

Note how we changed the value of secret. In the second command, we asked the python interpreter to return the value of the variable that we have been storing. A great deal of what programming concerns storing data and manipulating values ... after all, computers are just big math machines!

Numbers aren't the only values we can store. We can try to store words or phrases... called strings. Let's give it a try. Run something like:

```
>>> secret = "The answer to life, the universe and everything."

>>> secret
'The answer to life, the universe and everything.'
```

This will assign a "str" type of value to secret. Strings are not the same as int values ... they cannot have arithmetic performed on them. However, strings are cool because they concern words and characters, and come with a lot of functions of their own. One of Python's primary uses is for string manipulation.

```
>>> secret = secret + 2
Traceback (most recent call last):
  File "<pyshell#8>", line 1, in <module>
    secret = secret + 2
TypeError: cannot concatenate 'str' and 'int' objects
```

Python doesn't like that. However, given that we are programming in Python and not something like JavaScript, we are given a hint as to why it didn't like that. The interpreter tells us that we cannot treat strings like integers. Let's try running a string function instead:

```
>>> len(secret)
47
```

Interesting! The `len()` function is a feature included with Python for telling the length of a phrase. This is a very basic function that simply takes the string and returns the number of characters in the string. Still, this demonstrates some of the unique features of strings in Python, and I am sure you can already see how this might be useful for developing applications such as password validators. Oh yeah, one more thing. Try something like this:

```
>>> professor = "Colin"
>>> print professor
"Colin"
>>> professor += " is awesome"
>>> print professor
"Colin is awesome"
```

Python has very powerful string manipulation features. I think this is really cool... maybe you will too?

STEP 2: PYTHON FUNCTIONS

Say we wanted to create our own `len()` function, could we do that in Python? The answer is: "of course!" You probably already know a lot about functions, so I won't review that here. Let's just jump right in and write a function:

```
>>> def findUltimateAnswer():
...     return 42
```

So, this is straight forward. A function is declared using `def _____ () :` and defined in the space below. Note how the `*return*` declares what the function will bring back. One other very important

thing about functions is that you must respect Python's built-in whitespace. If we were instead to write the function as follows (without the tab), we would get an error.

```
>>> def findUltimateAnswer():
...     return 42
      File "<stdin>", line 2
        return 42
```

This is perhaps the most distinctive feature of Python. The language forces you to respect white space, in an effort to make it more readable. OK, let's call the function and keep going.

```
>>> findUltimateAnswer()
42
```

That makes sense. However, this is not a terribly interesting use of functions. As Douglas Adams clearly showed, the ultimate answer isn't useful if we don't know the ultimate question. Let's make a function with some requirements, and maybe throw in a conditional.

```
>>> def findUltimateAnswer(question):
...     if question == "When is lunch?":
...         return "Time is an illusion"
...     else:
...         return 42
```

So what's going on here? Whenever this function is called, it now requires a question. Cleverly, the function now *does* something, namely check to see whether the question is about lunch. The function uses an if/else statement to make this check and to return 42 if needed. So, if we were to try something like:

```
>>> findUltimateAnswer("When is lunch?")
'Time is an illusion'

>>> findUltimateAnswer("Bored yet?")
42
```

So, that's pretty cool. I guess the other thing we should cover before moving on from this section is loops. Like with most object-oriented programming languages, we usually use for or while loops. I will provide two examples below:

```
>>> while x < 10:
...     print "Resistance is Useless"
...     x += 1
...
Resistance is Useless
Resistance is Useless
Resistance is Useless
```

```
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
```

"Resistance is Useless". Wow, those must have been uttered by the most boring beings in the universe, or something. We can write the same loop as a for loop as follows:

```
>>> for x in range(0, 10):
...     print "Resistance is Useless"
...
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
```

Of course, that is not a terribly interesting use of `*for*` in Python, as it utilizes a range to iterate. Maybe we should try something like:

```
>>> for letter in "Resistance is Useless":
...     print "Resistance is Useless"
...
Resistance is Useless
Resistance is Useless
Resistance is Useless
Resistance is Useless
```

[... you get the point]

The for loop is most often used to iterate over a range, string, array or dictionary. For more information, check out the Python docs or the Google tutorial.

STEP 3: PYTHON OBJECTS

Before we get to the fun stuff, we should probably briefly review objects. For those who are used to OOP, this will be straight forward. For everyone else, this will only scratch the surface, and you REALLY NEED to do some extra study. OK, let's try the following:

```

>>> class Chin:
...     kind = 'chinchilla'
...     color = 'grey'
...     fur = 'ZOMG! SO FLUFFY!'
...     def __init__(self, name):
...         self.name = name
...
>>> s = Chin('Stormy')
>>> print str(s.name) + " is Colin's pet " + str(s.kind) + " and she
is " + str(s.fur)
Stormy is Colin's pet chinchilla and she is ZOMG! SO FLUFFY!

```

You see, the consequence of you being here is that you have to listen to me talk or read what I write. Well, sort of. I guess you could always leave. However, assuming you want to learn Django, you have to read code about my pet Chinchilla! If you don't know what a chinchilla is, LOOK IT UP! It's the fluffiest animal you can safely domesticate, and yes, her name is Stormy.

Back to the tutorial. Python will return the string Storm is Colin's pet chinchilla and she is super-fluffy! What did we do here? We used object oriented programming to declare a class (Chin) and to create a Chin object named Stormy. Stormy inherited properties from the Chin class (in this case, kind and color and fur properties) and a name, which we gave it. Objects are used to make complex programs, and most programs you will encounter in life use this paradigm.

We can also manipulate objects. Try adding the speed property to Stormy:

```

>>> s.speed = "unbearable!"
>>> print s.speed
unbearable!

```

You will see that the Stormy's speed is unbearable. This is how we append properties to objects. We can also add properties to whole classes, and they are inherited:

```

>>> Chin.jump = "Abnormally high!"
>>> s.jump
'Abnormally high!'

```

You will see that chinchillas jump abnormally high. Finally, know that Arrays and Dictionaries are things in Python. Typically you simply declare an array and will its existence. Investigate the following.

```

>>> bowl = []
>>> for x in range(0, 10):
...     bowl.append("oats!")
...
>>> print bowl

```

```
['oats!', 'oats!', 'oats!', 'oats!', 'oats!', 'oats!', 'oats!',  
'oats!', 'oats!', 'oats!']
```

Now there is a bowl full of oats, which chinchillas seem to love a little too much. Note that each item in the array is comma separated. Dictionaries work much the same way but must be more rigorously defined due to the key/value relationship. Try this:

```
>>> chinDict = {'Habitat':'Antacama Desert',  
'Conservation':'Endangered', 'Health':'Fragile'}  
>>> chinDict['Health']  
'Fragile'
```

I think that wraps up the crash Python tutorial. Let's move on to some useful code and start installing some software.

STEP 4: INSTALLING AND CONFIGURING GIT

You should consider installing and configuring git so that you can easily access the course materials. Git is a version control software that allows people to share and control code easily. Github is site where cloud storage meets social networking-- it's where developers can brag about their cred and stuff. You can learn about how to install and configure git and github at <https://help.github.com/categories/bootcamp/> Note that the repo we will be using for these tutorials is at <https://github.com/cdconrad/shiftkey-py> If you have installed git and haven't yet retrieved the repo for this course, please do so now.

```
git clone https://github.com/cdconrad/shiftkey-py
```

For MS Windows users, you will have to install the git program to run this command. If you don't have time to do this, you can download the folder directly from Github by visiting the link above and clicking "Clone or Download". In this folder, you will find five week repositories. This tutorial, and the relevant materials, are contained in the week1 folder. Each subsequent week contains relevant files. In this week's folder, you should be able to find the twitter-profiler.py file. Open this file in your editor.

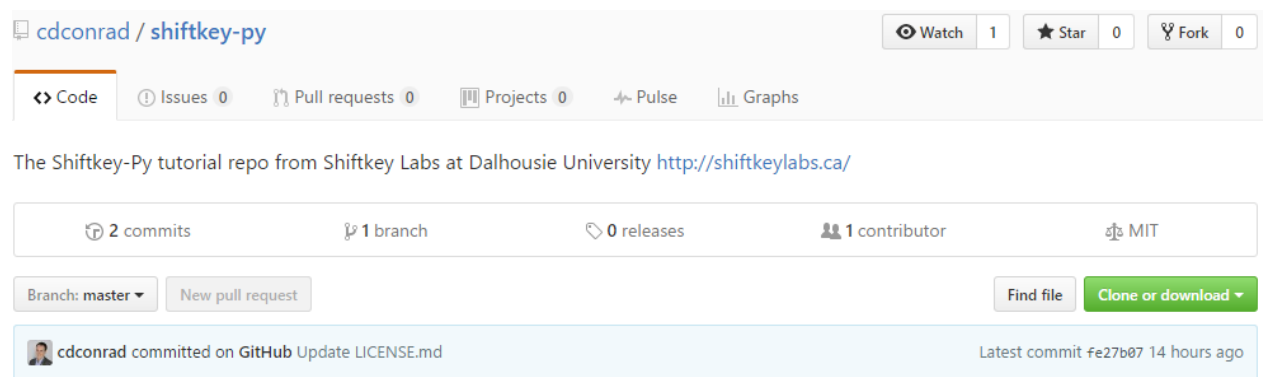


Figure 1 - Click on the green button to download if you don't have time to configure git

STEP 5: PYTHON PIP AND THE TWEETPY PACKAGE

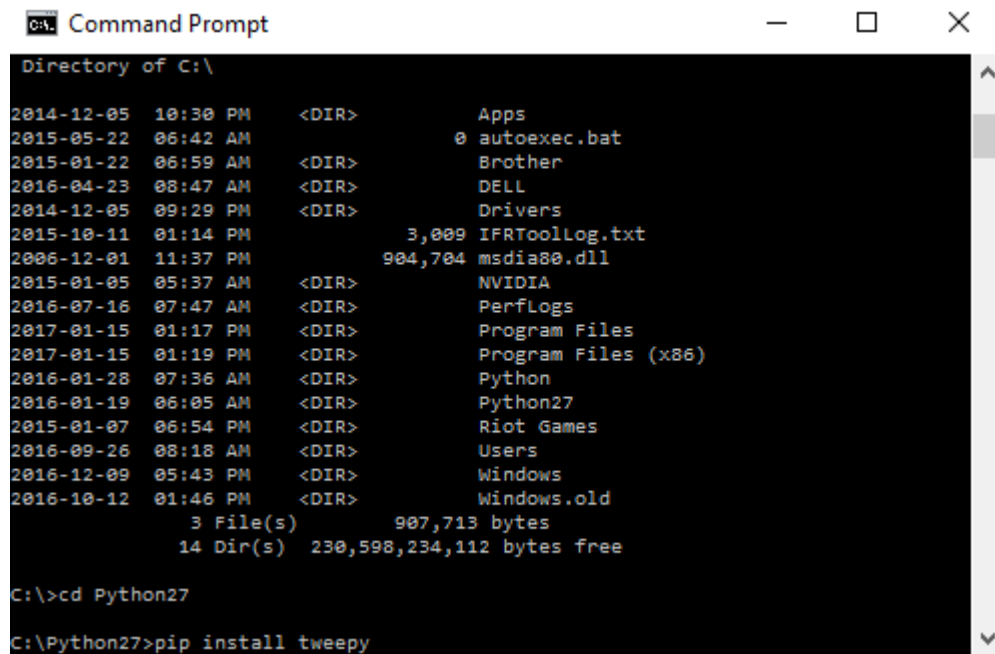
Let's get started on something more substantial. One of the best features of python is that it has a package system which can greatly extend the basic features of the programming language. some python extraordinarily powerful, and tools such as Natural Language Toolkit (NLTK), Python Data Analysis Library (PANDAS), and SciPy are among the most commonly employed by data scientists. To help manage these packages, modern python versions are shipped with a package manager called pip. In this lab, we will use a package called Tweepy, a unique tool for managing the Twitter API.

Installing packages using pip is easy. If you are on linux, it's as simple as ...

```
~$ sudo pip install tweepy
```

If you are on windows, it's the same thing, only you must be in your root python directory. Windows users should open the cmd and find your way to the python C:\Python27 folder. From there you can just write ...

```
~$ pip install tweepy
```



```
C:\> Command Prompt
Directory of C:\
2014-12-05 10:30 PM <DIR> Apps
2015-05-22 06:42 AM 0 autoexec.bat
2015-01-22 06:59 AM <DIR> Brother
2016-04-23 08:47 AM <DIR> DELL
2014-12-05 09:29 PM <DIR> Drivers
2015-10-11 01:14 PM 3,009 IFRToolLog.txt
2006-12-01 11:37 PM 904,704 msdia80.dll
2015-01-05 05:37 AM <DIR> NVIDIA
2016-07-16 07:47 AM <DIR> PerfLogs
2017-01-15 01:17 PM <DIR> Program Files
2017-01-15 01:19 PM <DIR> Program Files (x86)
2016-01-28 07:36 AM <DIR> Python
2016-01-19 06:05 AM <DIR> Python27
2015-01-07 06:54 PM <DIR> Riot Games
2016-09-26 08:18 AM <DIR> Users
2016-12-09 05:43 PM <DIR> Windows
2016-10-12 01:46 PM <DIR> Windows.old
3 File(s) 907,713 bytes
14 Dir(s) 230,598,234,112 bytes free

C:\>cd Python27
C:\Python27>pip install tweepy
```

Figure 2 - Install Tweepy by going into the relevant python folder

Either way, pip will search for the tweepy package and install it. That simple. With Tweepy in hand, we can investigate the twitter-profiler.py script. For more information on the Tweepy package, visit <https://github.com/tweepy/tweepy>

STEP 6: THE TWITTER-PROFILER SCRIPT

Assuming you have the src folder, open the twitter-profiler.py script in your editor. You will notice that it has a few a specific structure. Let's go through it together.

```
import tweepy, time
```

In python, you can import packages you would like to use using the import keyword. We are going to use tweepy and time. Unlike Tweepy, time is a python package that is shipped with all versions of python, designed for handling data and time structures. Tweepy needs time to work properly.

```
consumer_key = ""
consumer_secret = ""
access_key = ""
access_secret = ""
```

The consumer_key and other variables are keys that are used by the Twitter API in order to run. We will need our own Twitter keys to make this work. Step 7 covers this in depth--let's just keep working our way through the file first.

```
def get_profile(screen_name):
    auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
    auth.set_access_token(access_key, access_secret)
    api = tweepy.API(auth)
    try:
        user_profile = api.get_user(screen_name)
    except:
        user_profile = "broken"

    return user_profile
```

OK, there's a bit going on in this part of the code. This is a function designed to take one argument and return a user_profile object. For those familiar with programming, that much is straight forward. The important bits come up in the function itself. The magic is in the first three lines in the function. This is essentially Tweepy at work. The auth variable contains the OAuth process for logging into the Twitter API using your Twitter account keys. The auth.set_access_token handles the other part of the Twitter OAuth process, which gets you into the API. Finally, the api variable saves the api session for use in the python script. *It's not essential that you understand this process*. What matters is that they are lines of code that activate Tweepy and let you in to Twitter's API *securely*. It works because OAuth works? If you want to learn more about OAuth, check out the Wikipedia page.

```
try:
    user_profile = api.get_user(screen_name)
```

```
except:
    user_profile = "broken"
```

What's going on here? The `api.get_user(screen_name)` will take the screen name defined in the function call and save the Twitter API object (and the data!) to the `user_profile`. We wrapped the api call in try: so that the script doesn't crash if our api keys are wrong or the call is broken. In the case that it is broken, the except condition comes into effect... which sets `user_profile` to the string "broken".

```
s = get_profile("")
print s
```

This is the call of the function and some print statements. In order to call the function, we just need to make sure that there is a value between the quotes. Pretty cool yeah? Let's try using it. Change the call to a twitter profile you know. Let's try my profile.

```
s = get_profile("cd_conrad")
```

Save the script and run it in your terminal.

```
~$ python twitter-profiler.py
broken
```

The script did not work because we have not yet configured the twitter api keys. Let's do that.

STEP 7: GET YOUR TWITTER KEYS AND RUN THE SCRIPT

This next step will require you to have a Twitter account. If you do not have one, please visit twitter.com and sign up. You do not need to use your real name, but I encourage you to use your proper email address. It will also ask you for your phone number. It may need this for twitter developer to authenticate that you are a real developer and not a spambot. NOTE: If you absolutely refuse to give Twitter your cellular number, please come see me.

When you have a twitter account, visit <https://apps.twitter.com> and start your first app! Give it a cool name. Once your app is configured, you can access the app page. Near the top of the page, you will find "Keys and Access Tokens". Go there to retrieve the values you should add to the twitter-profiler. Add the keys to the key variables described in Step 6. Once you have added the keys to the twitter-profiler.py application, run it again!

```
~$ python twitter-profiler.py
User(follow_request_sent=False, has_extended_profile=True,
profile_use_background_image=False, _json={u'follow_request_sent':
False, u'has_extended_profile': True,
u'profile_use_background_image': False, u'profile_text_color':
u'333333',
```

```

u'default_profile_image': False, u'suspended': False, u'id':
494233327, u'profile_background_image_url_https':
u'https://abs.twimg.com/images/themes/theme15/bg.png', u'verified':
False, u'translator_type': u'none', u'profile_location':
{'u'full_name': u'Halifax, Nova Scotia', u'url':
u'https://api.twitter.com/1.1/geo/id/5d058f2e9fe1516c.json',
u'country': u'',
u'place_type': u'unknown', u'bounding_box': None,
u'contained_within': [], u'country_code': u'', u'attributes': {}},
u'id':

[...]

```

What a mess! Don't panic, you haven't done anything wrong. This is simply the Twitter API data. When you print Tweepy's Twitter API object, you will receive a lot of JSON, which is the format that the API uses dump data. Try modifying the script to only show some relevant features. Modify the end of the twitter-profiler.py file to something like:

```

s = get_profile("cd_conrad")
print "Name: " + s.name
print "Location: " + s.location
print "Description: " + s.description

```

Now save and run the script. You should get the following result:

```

Name: Colin Conrad
Location: Halifax, Nova Scotia
Description: PhD Student @RoweBusiness and @dalfcs who writes about
Information Systems, Machine Learning and Human Cognition.

```

There you have it! You have just completed a useful python script for querying the Twitter API, and have even taken the first steps toward completing the Django app! In later weeks, we will use this script with Django to show off the framework's Python muscle. For now, just take a breath and revel in your success. See you next week!

WEEK 1 CHALLENGE

If you are ambitious, or are in CSCI 2113, I have a challenge for you. You will find a week1-challenge.py script in this folder. Open it, and take a look. It's basically the same thing we just did, but with a twist. I want you to modify this script to retrieve the user's most recent TWEETS instead of their location. You will need to read the Twitter API docs to do this. CSCI 2113 students should submit their results on SVN. Good luck!