Python and APIs

Joey Wunderlich

November 2016

1 Basic Python (3.5) Cheatsheet

In addition to basic python features, we'll be using the 3rd party requests library available at http://docs.python-requests.org/en/master/

1.1 Data Structures

1.1.1 Dictionaries (java equivalent: Maps)

Initialization:

```
dict1 = {"a" : 5, "b" : 10, "c" : 15}
## Can also use
## dict2 = dict(a = 5, b = 10, c = 15)
## but that breaks easily; what if you want the key "a=b"?
```

Put:

```
dict1["d"] = 20
## Contents of dict1 : {'a': 5, 'd': 20, 'c': 15, 'b': 10}
```

Get:

```
print(dict1["d"])
## Prints the letter d to the console.
```

keySet:

```
print(dict1.keys())
## prints a list of the keys:
## dict_keys(['a', 'd', 'c', 'b'])
```

1.1.2 Lists (lists)

Initialization:

```
list1 = ["hah", "funny", "ponies"]
```

add:

```
list1.insert(0, "first!!")
## resulting list = ['first!!', 'hah', 'funny', 'ponies']
## also available: append(value) to add to end of list.
```

get:

```
list1[0] ## gets value at index 0; "first!!"
list1[-1]
## gets first value from the end (i.e. last value); 'ponies'
list1[1:3] ## Gets elements between 1(inclusive) and 3(exclusive);
## ['hah', 'funny']. To get all elements, just use list1[:].
```

1.1.3 Other common data structures && Miscellany

Because we only have time for a basic overview, we're missing out on a few of the other structures. Some to look up if you're interested: sets and tuples (effectively, immutable arrays)

1.2 Features

1.2.1 Consistency

Want the size of a data structure 'data'? It should always be

```
len(data)
```

Python thrives on consistency; veteran programmers in Python insist on the use of 'pythonic' code; that is, code that follows the idioms created for python. Visiting https://www.python.org/dev/peps/pep-0020/ will help you get an idea of what they go for.

Helpful in this are 'magic methods' - methods that overwrite behavior. Google is helpful in this regard; some common ones are __repr__ for the string representation of a value (as a programmer would want it), __len__ for overriding the len(...) function, __init__ for constructors and many more (http://www.python-course.eu/python3_magic_methods.php has a nice, if non-exhaustive list to start with)

1.2.2 Handy behavior

Do you want to switch two variables? In python, it's as simple as

```
a = 7
b = 42
a, b = b, a
## b == 7, a == 42
```

Want to pass around functions like nobodies business? Because they're variables in python, you can do stuff like

```
def addOne(num):
    return num + 1

def doTwice(func, num):
    return func(func(num))

addOne(5) ## returns 6
doTwice(addOne, 5) ## returns 7

funcMap = {"add one" : addOne, "do twice" : doTwice}
funcMap["add one"](5) ## returns 6
```

Python is white space based; as you can see above, no curly braces to surround methods, just tab in and stay there until you're done.

Last thing for now (it's also only 'pretty much' right, but for our purposes is good enough): Most implementations of python are closer to being *interpreted*, rather than *compiled*; that means that we can run the code we write line by line – even in the terminal – without having to compile and run an entire program at once. (Just google 'is python interpreted or compiled' to see endless responses on this.)
One more resource if you want to read a bit more about python:
http://docs.python-guide.org/en/latest/writing/style/

2 APIs using Python!

We'll be using the politifact and giphy APIs to start off; more can be found at http://www.programmableweb.com/apis/directory

2.1 Getting started with the politifact API

Documentation available at http://static.politifact.com/api/doc.html

```
# import the two libraries we'll be using to make HTTP requests
import json, requests

# Define a base url to start from for all HTTP requests
base_url = "http://www.politifact.com/api/"
# Extended URL for the portion that we'll be calling from.
extended_url = base_url + "statements/truth-o-meter/json/"
# List of parameters to be used in our request
input_params = dict(
    n = 5
)

# Make the HTTP request using requests library
response = requests.get(url=extended_url, params=input_params)
## Here, response.text will contain the text obtained from the request

# Use JSON library to make that a little more usable format.
data = json.loads(response.text)
```

So what do we have now? Let's dig into the data and check. (>>used to identify response)

```
print(data)
# There's a lot of stuff their, but it looks like it's some sort of list?
>> Blob
print(data[0]) # Okay, that looks like a map. Let's call get the keys
>> Smaller blob
print(data[0].keys) #
>> dict_keys(['ruling_date', 'ruling_headline', 'target', 'statement', ...])
# Okay, so it's a list of maps. Let's get the first headline then.
print(data[0]["ruling_headline"])
>> 'a headline' (it won't be up to date anyways)
## Hey, we got something usable!
```

2.2 Using giphy to get some gifs

Documentation available at
http://api.giphy.com/

```
import json, requests

base_url = "http://api.giphy.com/"
extended_url = base_url + "v1/gifs/search"

## Public api ‘beta‘ key from their github;
## many websites will use private keys to authenticate requests
public_api_key = "dc6zaTOxFJmzC"

# parameters; more this time. We need to pass in the API key,
# a query q to search for, and optionally a limit to the number of
# responses and a rating limit
my_params = dict(
            api_key = public_api_key,
            q = "cute cats",
            limit = 5,
            rating = "pg-13"
)
# The built up HTTP request will look like the following:
# http://api.giphy.com/v1/gifs/search?api_key
# =dc6zaTOxFJmzC&q=cute+cats&limit=5&rating=pg-13

response = requests.get(url=extended_url, params=my_params)
data = json.loads(response.text)
```

Now, let’s look at it!

```
data[0] # Key error: must not be a list this time?
# Looking at it in firefox, we can see that it actually
# is a map of lists of maps this time; we want the key "data", and then
# the first element of that
>> Traceback (most recent call last):
    File "<stdin>", line 1, in <module>
    KeyError: 0

data["data"][0]
# Okay, got a lot of stuff here... looks like we want the bitly_url!

data["data"][0]["bitly_url"]
>> ’http://gph.is/1XYsr1n’ # SUCCESS!
```