API Calls in Python

Tutorial 2: Building an NLC Application Part 2

Plan to Build

Part 1:

1. ~~Generate Credentials on Bluemix;~~
2. ~~Collecting a Sample;~~
3. ~~Determine the Base-Classes for Classifier;~~
4. ~~Collect the Initial Training Set;~~
5. ~~Try the web interface for Training;~~
6. ~~Test the Classifier Online;~~

Part 2:

7. Run the API Call in Python;
8. Generate a CF Starter Pack Flask Application;
9. Build Out MVP Flask Application on the Above API Calls;
10. Push to Bluemix Hosting

Deliverables

Here are your deliverables for this tutorial:

Activity: You will learn how to make API calls to your NLC service in Python, how to parse the output and allow for user input.

Submission: Take a screen shot of the following:

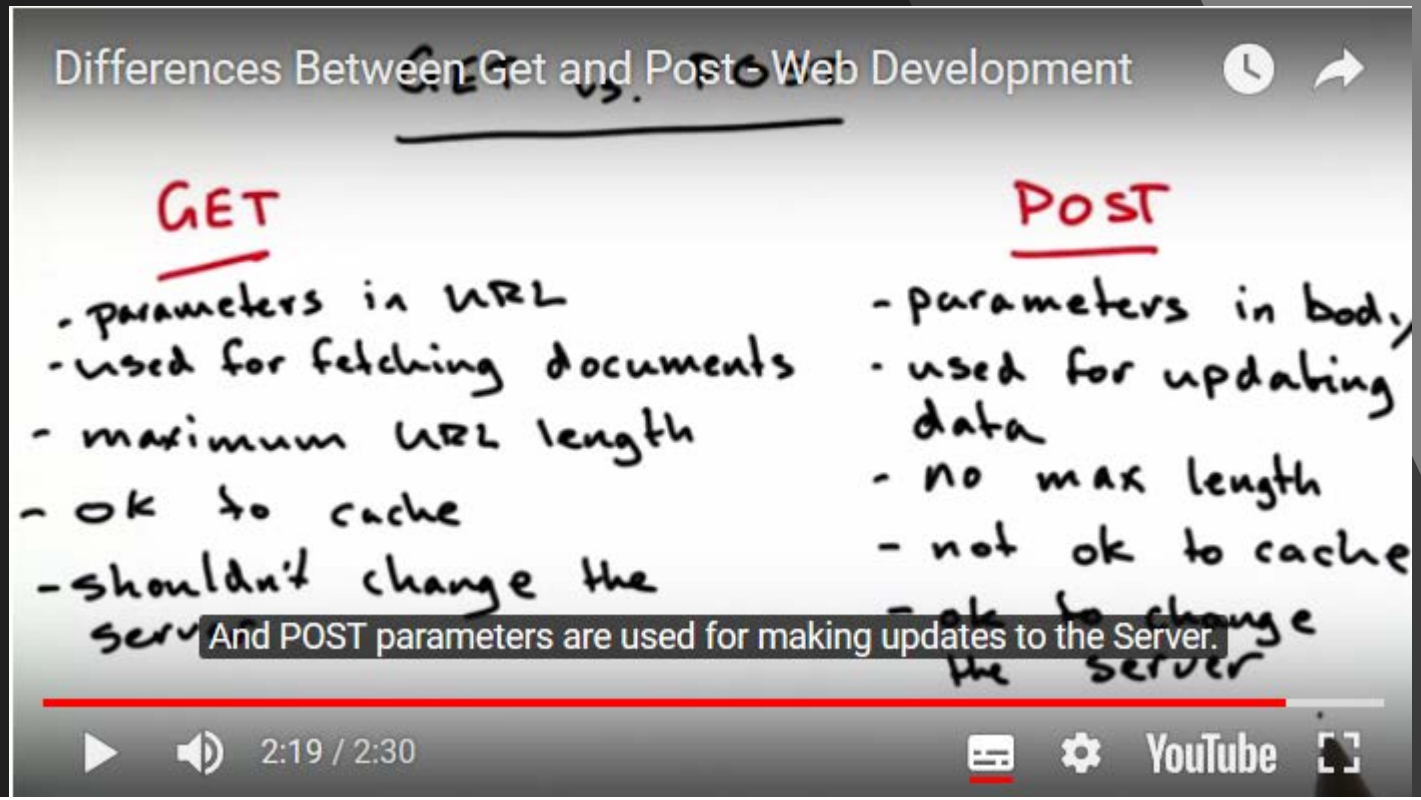
1) The final code you run in your Jupyter Notebook, and the results.

2) The terminal screen where you have allowed for user input, testing two (2) comments.

Copy and paste your screenshots in a word document.

Understanding the Difference: GET VS POST Calls

This video explains the difference



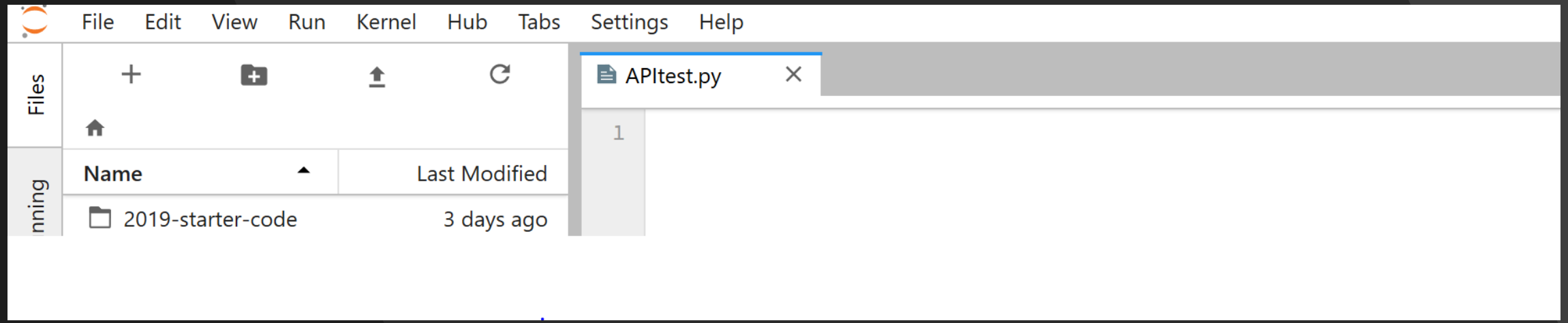
https://www.youtube.com/watch?time_continue=148&v=UObINRj2EGY

Run the API Call in Python

Step 7

Run the API call in Python

- Go to the CAC online platform and in JupyterHub create a new text file.
- Rename your file APItest.py or something similar.



Get Your Service Information

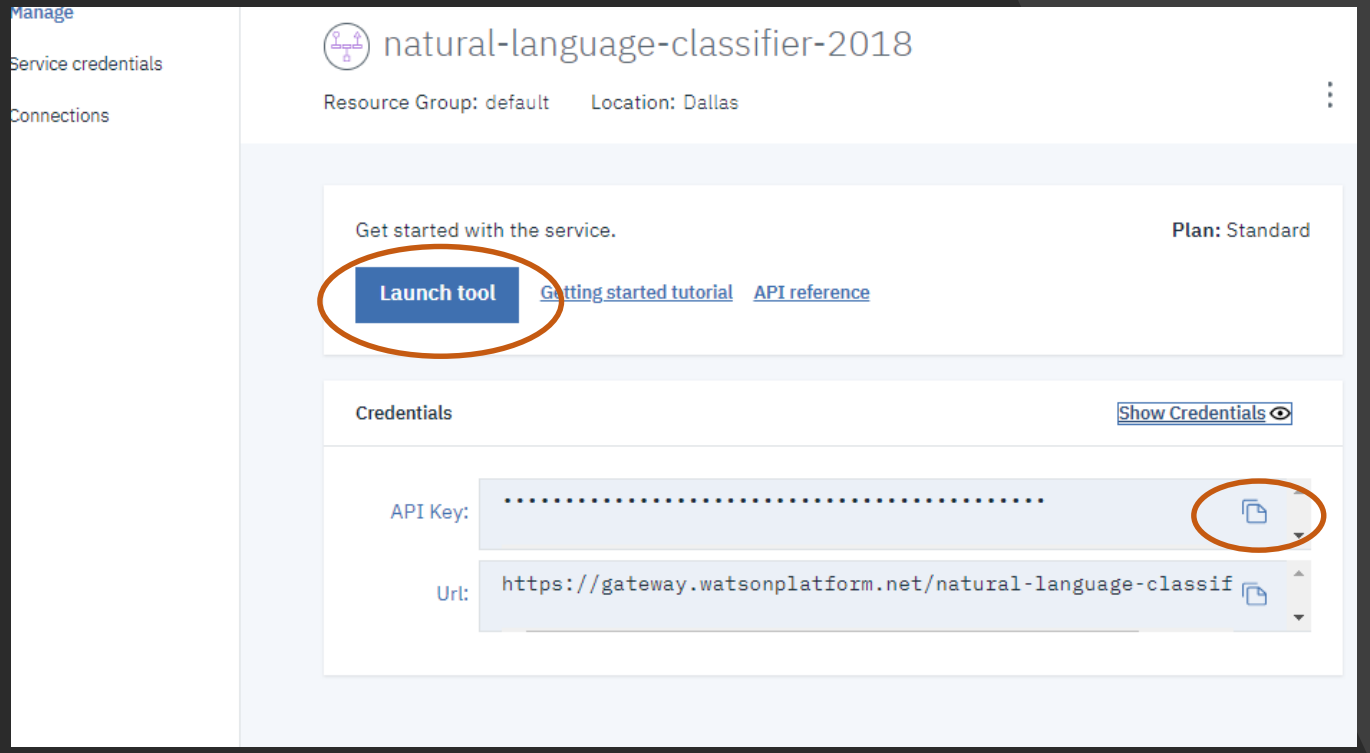
- To call an API in Python, most API providers (in this case IBM Cloud/Watson Developer Services) require some kind of authorization credentials.
- To do a basic API call to your Natural Language Classifier service, you will need to get your API key as well as your unique NLC classifier id.

NOTE: when you create a service (e.g., NLC) within Bluemix, you will get credentials specific to that instance. Sometimes this is in the form of a username and password; other times it is an API key.

Get Your Service Information (2)

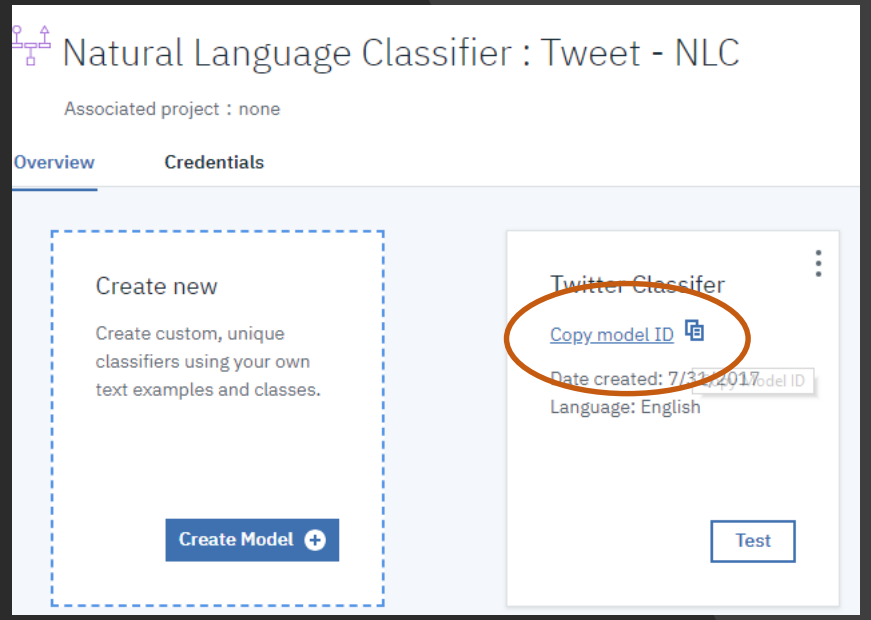
- Login to the IBM cloud console (https://console.bluemix.net/dashboard/apps/)
- Under services, click on the Natural Language Classifier created previously in the tutorial.

  - Click the button to the right of your API key.
  - Paste the value into the Python file you just created.
  - Next, click the 'Launch Tool' button to get your classifier ID.



Get Your Service Information (3)

- Click the Copy model ID button under your model's name.
- Once you have your information, paste it into the Python file you created earlier.
- We are now going to create variables in our code to store this information.
- Variable names don't matter, but this information must be passed to the API as a 'string' datatype. Your code should look something like the image below:




```
workspace_ID = '0e6935x475-nlc-1067'
api_Key = 'HOees-copZVpTK8zXb-Lt9xu4DNmA5CZNANq-MEPXhPu'
```

Setting Up Our API Call

- Now that we have our API information, we need to use ‘import’ to add the packages necessary to call APIs.
- Add the following:
  - from Watson_developer_cloud import NaturalLanguageClassifierV1

This package allows us to provide the arguments required for the NLC API call.

```
#imports the Natural Language Classifier Functionality
from watson_developer_cloud import NaturalLanguageClassifierV1

workspace_ID = '0e6935x475-nlc-1067' #Your Workspace ID goes here
api_Key = 'HOees-copZVpTK8zXb-Lt9xu4DNmA5CZNANq-MEPXhPu' #Your iam_apikey goes here
```

The Natural Language Object

- Next, let's create an natural language object which will function as our main classifier.
- Again, we are going to store this information in a variable. We use the function 'NaturalLanguageClassifierV1' from the Watson_Developer_Cloud package to create this object.
- We provide this function with our API key information using the argument iam_apikey.

```
#imports the Natural Language Classifier Functionality
from watson_developer_cloud import NaturalLanguageClassifierV1

workspace_ID = '0e6935x475-nlc-1067' #Your Workspace ID goes here
api_Key = 'HOees-copZVpTK8zXb-Lt9xu4DNmA5CZNANq-MEPXhPu' #Your iam_apikey goes here

#Create a NaturalLanguageClassiferV1 object
natural_language_classifier = NaturalLanguageClassifierV1(
    iam_apikey= api_Key)
```

Quick aside: the text in red proceeded by a # is one method of commenting out code in python. This code will not be processed but may help others understand your code better.

Generating a Response

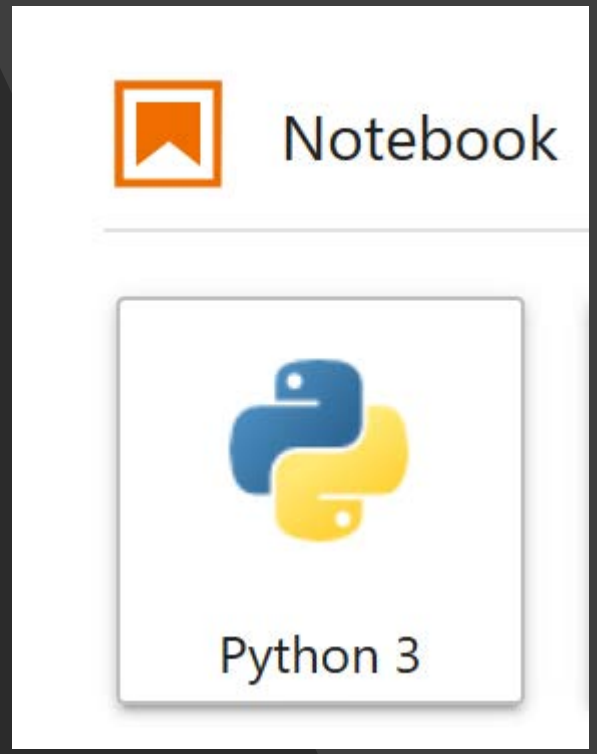
- The code we have will allow us to call our Natural Language Classifier service. Now, we want to sort the results of that API call and display them back to the user. We sort the returned object in the variable ‘response’ and then print that value to the console.

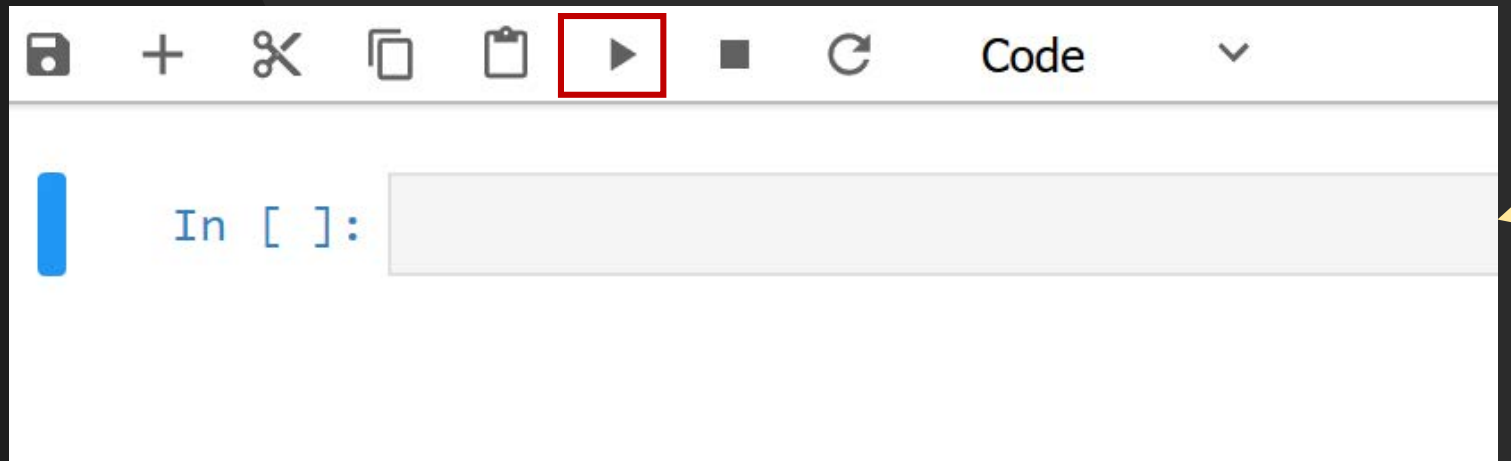
```
#imports the Natural Language Classifier Functionality
from watson_developer_cloud import NaturalLanguageClassifierV1

workspace_ID = '0e6935x475-nlc-1067' #Your Workspace ID goes here
api_Key = 'HOees-copZVpTK8zXb-Lt9xu4DNmA5CZNANq-MEPXhPu' #Your iam_apikey goes here

#Create a NaturalLanguageClassiferV1 object
natural_language_classifier = NaturalLanguageClassifierV1(
    iam_apikey= api_Key)

#Creates a varibale to house returned object
response = natural_language_classifier.classify(workspace_ID, 'Hello World!')

#Prints out the whole returned object
print(response)
```

Notice how we use the classify method (what parameters we pass to it)

Running the Code

- To run the code, we have a couple of options. For now the simplest option is to open a new Jupyter Notebook and copy the code into a cell.
- To do so, open the Launcher (File->New Launcher). Then click on Python 3 under Notebooks.
- Copy your code to the grey cell beside In[1]. Then click the run button (triangle).





```
{
    "result": {
        "classifier_id": "cdf8a7x488-nlc-172",
        "url": "https://gateway.watsonplatform.net/natural-language-classifier/api/v1/classifiers/cdf8a7x488-nlc-172",
        "text": "hello world",
        "top_class": "Satisfied",
        "classes": [
            {
                "class_name": "Satisfied",
                "confidence": 0.4035857010188965
            },
            {
                "class_name": "Overjoyed Praise",
                "confidence": 0.2589732562925655
            },
            {
                "class_name": "gibberish",
                "confidence": 0.16617820528689795
            },
            {
                "class_name": "Ambivalent",
                "confidence": 0.07088575897327204
            },
            {
                "class_name": "complaint re: Product",
                "confidence": 0.0431473147460417
            },
            {
                "class_name": "Complaint re: Product",
                "confidence": 0.035014807218758964
            },
            {
                "class_name": "Complaint re: Process",
                "confidence": 0.022214956463567446
            }
        ]
    },
    "headers": {
        "_store": {
            "x-backside-transport": [
                "X-Backside-Transport",
                "OK OK"
            ],
            "content-type": [
                "Content-Type",
                "application/json"
            ],
            "x-xss-protection": [
                "X-XSS-Protection",
                "1"
```


Understanding the Response

Once we have printed out the value contained within the response variable, we can see that it's a dictionary of dictionaries and lists collection type.

The inner 'headers' dictionary contains information about the call and the inner 'result' dictionary contains information about the actual API results from the inputted text.

Classifying Multiple Comments

- Now that we know how to call our NLC API in code, we can use the classify_collection method of the natural_language_classifier object to classify multiple blocks of text.
- Through supplying a list of dictionaries to the method, Watson will classify these two comments. Resulting output is to the side.

```
#imports the Natural Language Classifier Functionality
from watson_developer_cloud import NaturalLanguageClassifierV1

workspace_ID = '0e6935x475-nlc-1067' #Your Workspace ID goes here
api_Key = 'HOees-copZVpTK8zXb-Lt9xu4DNmA5CZNANq-MEPXhPu' #Your iam_apikey goes here

#Create a NaturalLanguageClassiferV1 object
natural_language_classifier = NaturalLanguageClassifierV1(
    iam_apikey= api_Key)

#Creates a varibale to house returned object
response = natural_language_classifier.classify(workspace_ID, 'Hello World!')

#Prints out the whole returned object
print(response)

#Calls the classify_collection function of the NLC object
collection_response = natural_language_classifier.classify_collection(workspace_ID,
                                                                      [ { 'text' : 'This is Great!' },
                                                                       { 'text' : 'This is Horrible!'}])
#Prints out the collection response object
print(collection_response)
```


```
{
    "result": {
        "url": "https://gateway.watsonplatform.net/natural-lan
        "collection": [
            {
                "text": "This is Great!",
                "top_class": "Ecstatic",
                "classes": [
                    {
                        "class_name": "Ecstatic",
                        "confidence": 0.7870649902901918
                    },
                    {
                        "class_name": "Neutral",
                        "confidence": 0.1324929100593808
                    },
                    {
                        "class_name": "Content",
                        "confidence": 0.03788870466465225
                    },
                    {
                        "class_name": "Distraught",
                        "confidence": 0.023598077932089595
                    },
                    {
                        "class_name": "Unhappy",
                        "confidence": 0.01895531705368557
                    }
                ]
            },
            {
                "text": "This is Horrible!",
                "top_class": "Distraught",
                "classes": [
                    {
                        "class_name": "Distraught",
                        "confidence": 0.7962976711220592
                    },
                    {
                        "class_name": "Ecstatic",
                        "confidence": 0.0796321377090793
                    },
                    {
                        "class_name": "Content",
                        "confidence": 0.055691218263098974
                    },
                    {
                        "class_name": "Neutral",
                        "confidence": 0.04078495904503557
                    },
                    {
                        "class_name": "Unhappy",
                        "confidence": 0.027594013860727025
                    }
                ]
            }
        ],
        "classifier_id": "0e6935x475-nlc-1067"
```

Prettier Printing through Parsing

To make the results look nicer when printed, we can begin to parse the classes object in Python. Before we can do this, with need to take our 'result' dictionary out of the returned data structure. Instead of just printing it out, we will loop through the `new_response["classes"]` object and print out both the `class_name` and the `confidence` properties. This code replaces just the `print` line from the code you created earlier.

```
#Parses the Detailed Response Object into a dictionary object
new_response = response.result

#if there is a dictionary key classes in the new_response
if "classes" in new_response.keys():
    #for each loop: looks through the inner dictionary classes and does this for each object within
    for predicted_class in new_response["classes"]:
        #prints out the name of the class and the predicted confidence
        print (predicted_class['class_name'], " - ", predicted_class['confidence'])

#Prints out the whole returned object
print(new_response)
```

1) Remember, Python is sensitive to indentation!

2) The prettier results look like:

```
Overjoyed/Praise  -  0.985229022309242
Satisfied  -  0.005331573808178 91
Ambivalent  -  0.003191352278379764
Complaint re: Product  -  0.0026575768017597307
Complaint re: Process  -  0.002329283615585114
gibberish  -  0.0012611911868545312
```

Allow for User Input

Instead of hard coding the text string that we wish to classify, we can very easily allow for user input to define it. The Python `input()` command allows the user to enter some text, and stores it in the Python variable. If we store this in the `comment_text` variable, and loop over it until the user enters a blank string (`""`), then we can produce a command line application that can classify all of the text that a user may have! This code replaces everything from the initial `comment_text` definition to the `print` line of code.

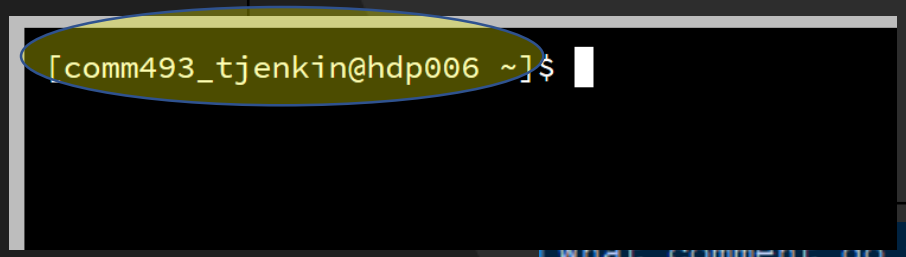
```
#Get user input from the console
comment_text = str(input("What comment do you want to analyze: "))

#while that inpt is not empty
while comment_text != "":

#Create a classifer instance with our classifer and the supplied text
  response = natural_language_classifier.classify(workspace_ID, comment_text)

#Parses the Detailed Response Object
  new_response = response.result

  #print out the results
  if "classes" in new_response.keys():
    for predicted_class in new_response["classes"]:
      print (predicted_class['class_name'], " - ", predicted_class['confidence'])
    comment_text = str(input("What comment do you want to analyze: "))
```

Results from basic API call

Let's test your code again by typing in the following on the command line in the terminal. To do so, open the Launcher (File->New Launcher). Then click on Terminal. Look at the directory to check it is where you saved your Python file. If not, use the change directory command (cd) to change to that directory. The example shows I'm in my root/home directory. Now type in the following (substitute your filename) and click enter:

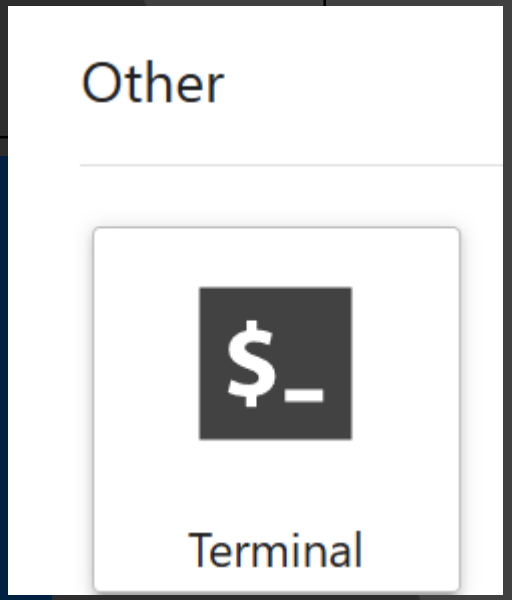
python3 APItest.py

Use the filename you saved your python file as. And be sure to include the 3 after python!



You will see the following results:

```
What comment do you want to analyze: good coffee!
Content  -  0.5359590174503551
Ecstatic  -  0.24282831656405088
Neutral  -  0.11001884361576716
Unhappy  -  0.07837435858049446
Distraught  -  0.032819463789332 39
What comment do you want to analyze: horrible coffee!
Distraught  -  0.6504567899041804
Ecstatic  -  0.12481714411352755
Unhappy  -  0.11020636231209854
Neutral  -  0.08598880778578301
Content  -  0.028530895884410646
What comment do you want to analyze:
```



Older Service API Calls

- A Watson API call used to require both a username and a password from the user to call an API. Now, the iam_apikey allows us to call APIs with less coding and credentials. However, you can still call APIs this way. Code to do so is presented below:
- Older instances of certain services (including NLC) will need to be called this way if an iam_apikey has not been created for the service.

```
#imports in required packages
from watson_developer_cloud import NaturalLanguageClassifierV1

#creates a classifier object using username and password as credentials
natural_language_classifier = NaturalLanguageClassifierV1(
        username='0314deb0-f43c-4909-a4be-ba32c4ad5950',
        password='bq0ESYXBVJ6N')

#calls the .classify function of the classifer object and sets it to a variable
response = natural_language_classifier.classify('359f3fx202-nlc-251945', 'What an Awesome Phone!')

#prints out the whole dictionary object
print(response)

#prints out the sub-dictionary at "classes" which is just the class information
print(response.result["classes"])
```

Next, let’s move to the Classify Application Tutorial

Building applications which use APIs