Introduction to Python

NaLette Brodnax

Indiana University Bloomington

School of Public & Environmental Affairs

Department of Political Science

www.nalettebrodnax.com

my goals

- Demystify programming
- Introduce common functions
- Write useful programs

what we'll cover in this workshop

1

getting set up

2

programming basics

3

working with files

what is python?

Python is a general purpose programming language.

It is easy to learn, highly readable, powerful and flexible.

when should you use python?

Programming should help you be more **efficient**

Countless applications

- data collection
- data cleaning
- analysis
- visualization
- automation

part 1: getting set up

getting the tools

Interpreter → Output

Text Editor + Interpreter → Output

Command Line + Text Editor + Interpreter → Output

Integrated Development Environment (IDE) → Output

integrated development environment

```
Python 3.4.2
>>>
>>> print(“Hello, world.”)
Hello world.
>>>
```

❶

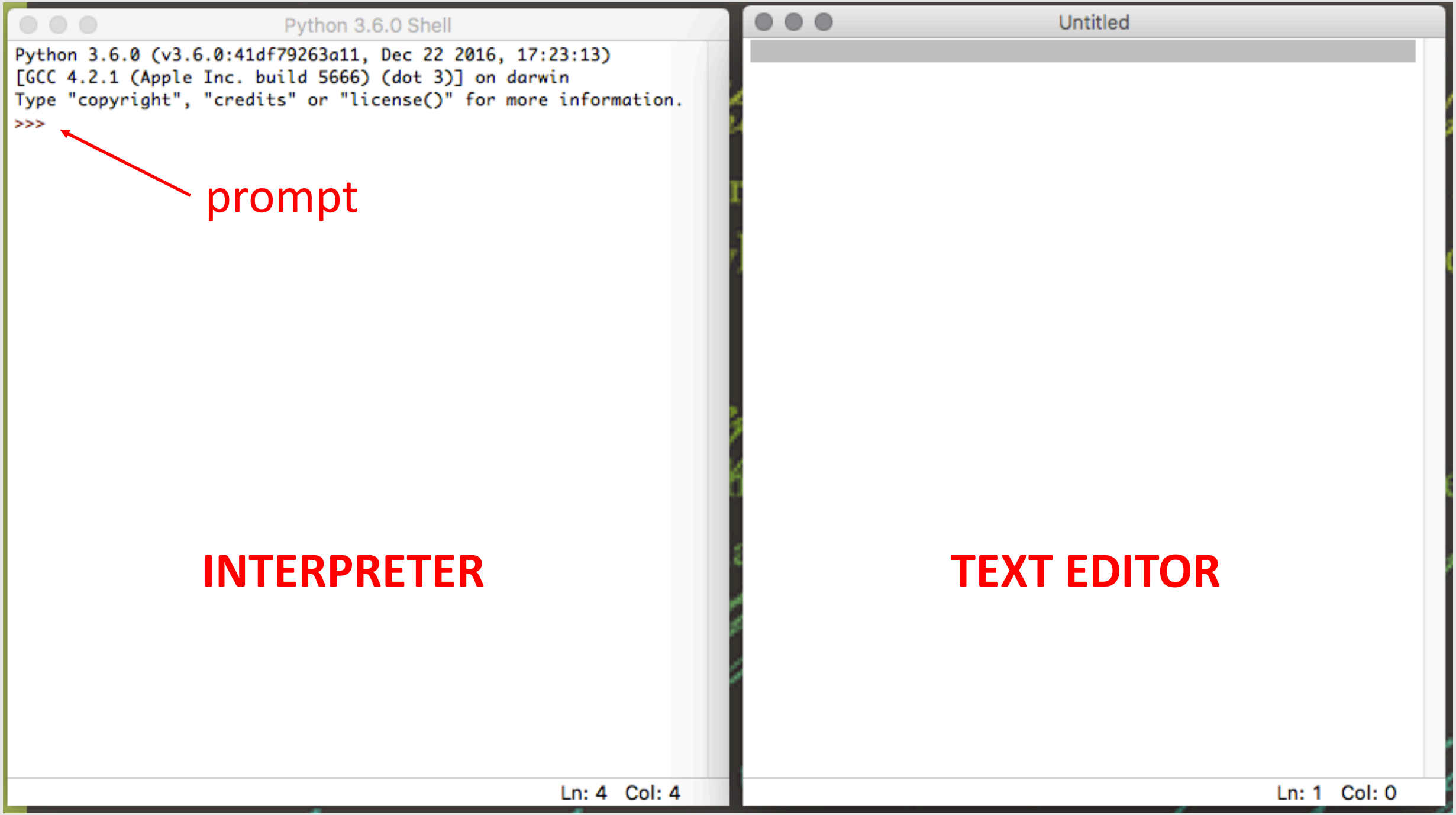
```
#My first script

print(“Hello, world.”)
```

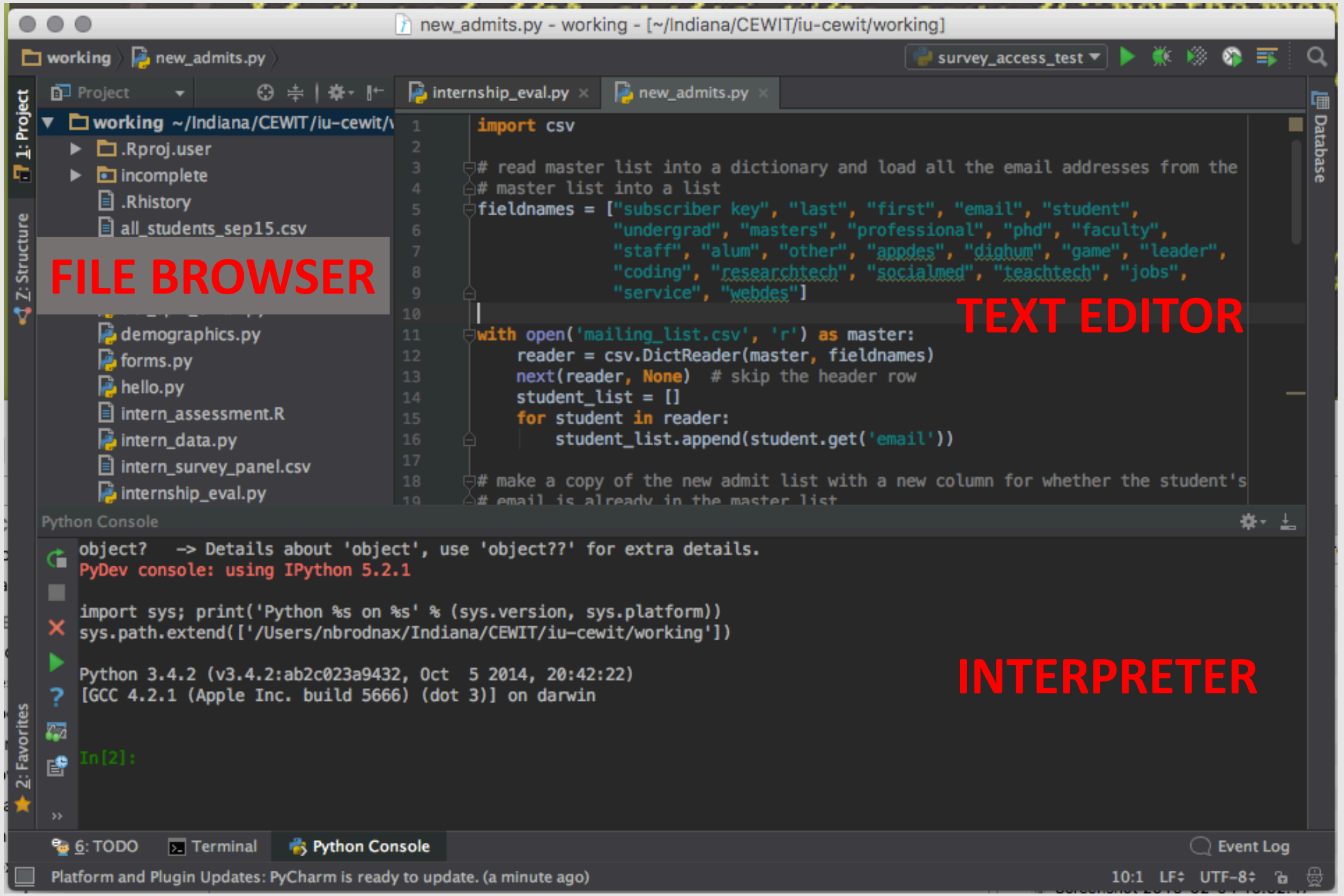
❷

```
Last login: Tue Nov 1 13:05:22
149-160-200-169:~ nbrodnax$
```

❸

1. Python interpreter (required)
2. Text editor (optional)
3. Command line (optional)


Python 3.6.0 Shell
Python 3.6.0 (v3.6.0:41df79263a11, Dec 22 2016, 17:23:13)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "copyright", "credits" or "license()" for more information.
>>>
prompt
INTERPRETER
Ln: 4 Col: 4
Untitled
TEXT EDITOR
Ln: 1 Col: 0


new_admits.py - working - [~/Indiana/CEWIT/iu-cewit/working]
working
new_admits.py
survey_access_test
Project
working ~/Indiana/CEWIT/iu-cewit/
.Rproj.user
incomplete
.Rhistory
all_students_sep15.csv
FILE BROWSER
demographics.py
forms.py
hello.py
intern_assessment.R
intern_data.py
intern_survey_panel.csv
internship_eval.py
internship_eval.py
new_admits.py
import csv
# read master list into a dictionary and load all the email addresses from the
# master list into a list
fieldnames = ["subscriber key", "last", "first", "email", "student",
"undergrad", "masters", "professional", "phd", "faculty",
"staff", "alum", "other", "appdes", "dighum", "game", "leader",
"coding", "researchtech", "socialmed", "teachtech", "jobs",
"service", "webdes"]
TEXT EDITOR
with open('mailing_list.csv', 'r') as master:
reader = csv.DictReader(master, fieldnames)
next(reader, None) # skip the header row
student_list = []
for student in reader:
student_list.append(student.get('email'))
# make a copy of the new admit list with a new column for whether the student's
Python Console
object? -> Details about 'object', use 'object??' for extra details.
PyDev console: using IPython 5.2.1
import sys; print('Python %s on %s' % (sys.version, sys.platform))
sys.path.extend(['/Users/nbrodnax/Indiana/CEWIT/iu-cewit/working'])
Python 3.4.2 (v3.4.2:ab2c023a9432, Oct 5 2014, 20:42:22)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
In[2]:
INTERPRETER
1: Project
7: Structure
2: Favorites
Database
6: TODO
Terminal
Python Console
Event Log
Platform and Plugin Updates: PyCharm is ready to update. (a minute ago)
10:1 LF UTF-8

getting the tools for today

- Use Python on SSRC computer
  - Search for IDLE, a basic Python IDE

- Use Python on your machine
  - Download Python 3.6 from www.python.org
  - Open IDLE from programs/applications

installing the tools on your machine

Download Python 3.6
https://www.python.org/downloads/

→Includes IDLE, an IDE with a text editor and interpreter

→Includes pip, Python's standard package manager

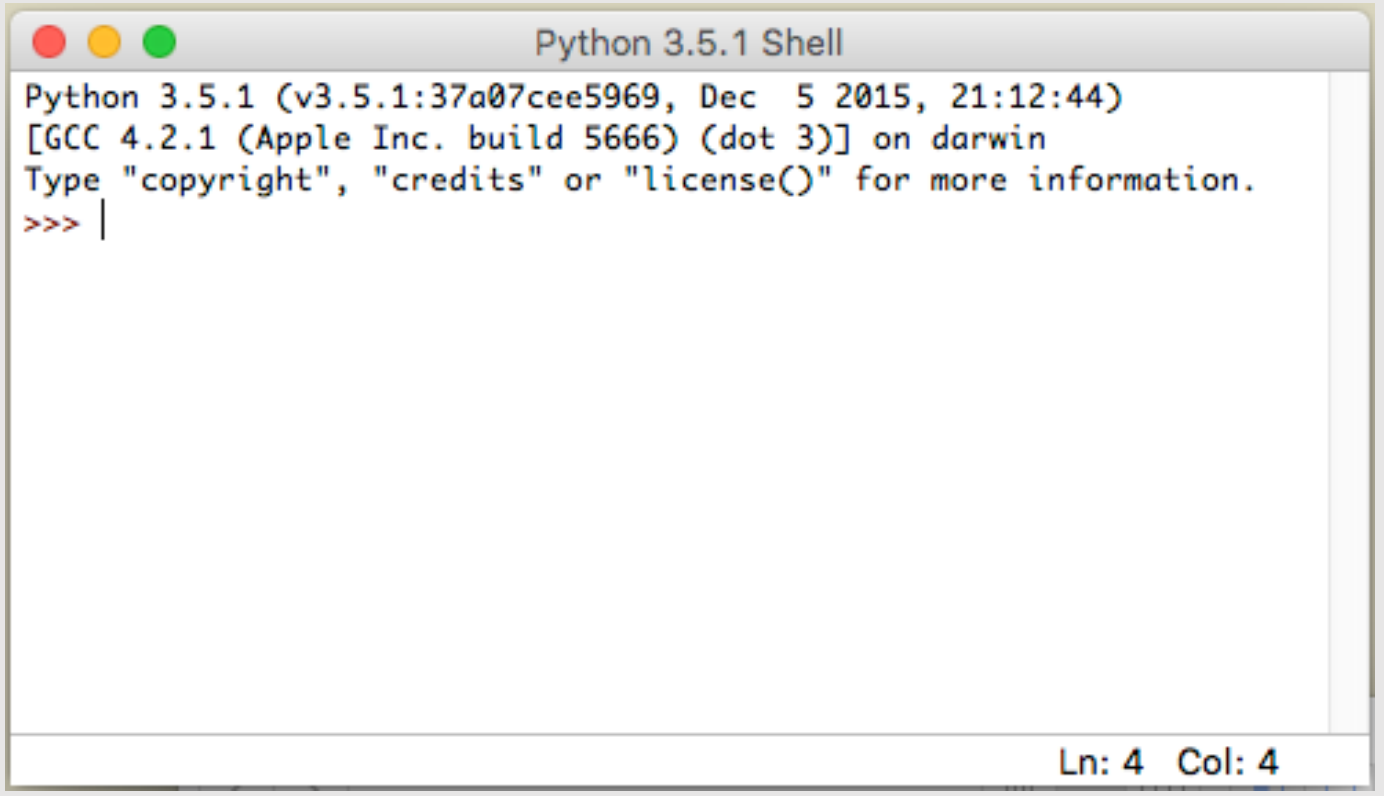
command line interface

Mac OS X/Linux → **Terminal, Bash**
Windows → **CMD, Powershell**

- Interact with computer's operating system
- Manage Python installation
- Access the Python Interpreter
- Execute a program without the Interpreter

open python's built-in IDE

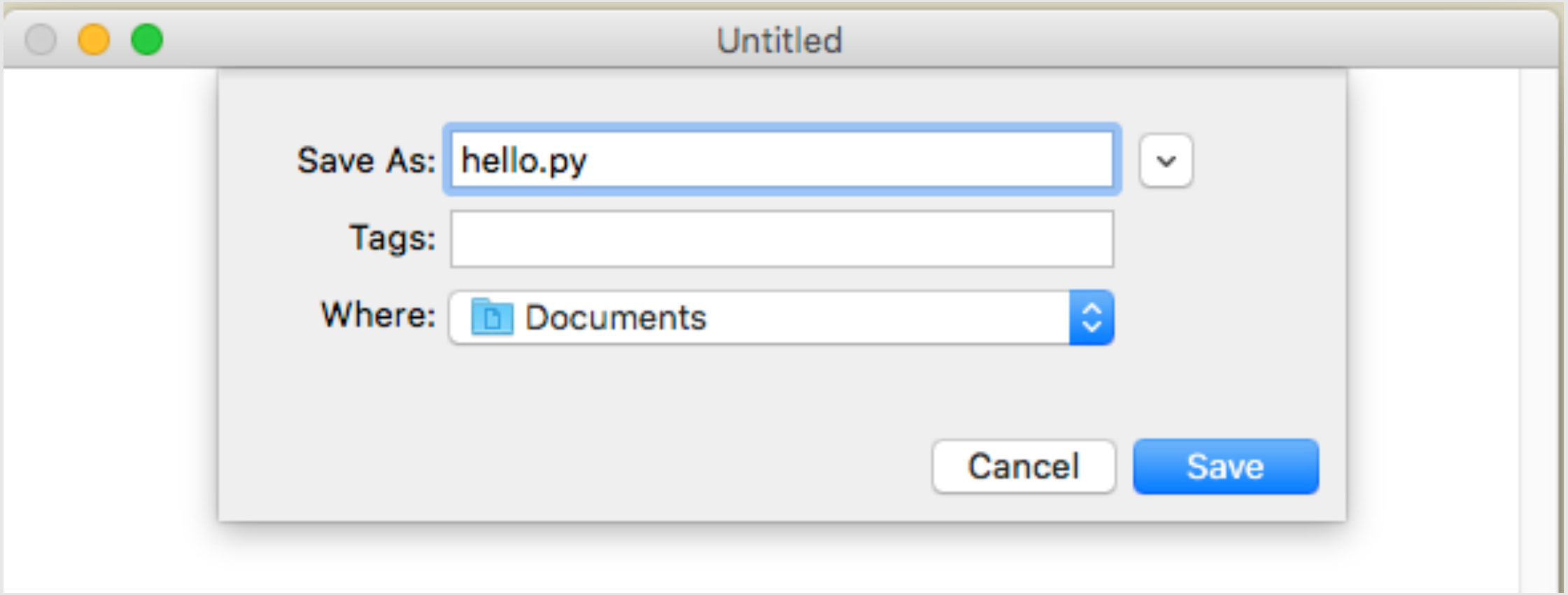
IDE → **IDLE**



Interact with Python

Write programs: **File → New File**

create your first script



create your first script

```
print(“Hello, world.”)
```

Save the program/script.
Run the program in the interpreter: **F5** or **Run → Run Module**

Download all the code for today’s workshop:
www.nalettebrodnax.com/python

part 2: programming basics

files for part 2

www.nalettebrodnax.com/python/

hello.py

a few programming language features

1. **Data types** – categories for storing different kinds of information in memory

2. **Conditionals** – control structures that allow decision making within a program

3. **Loops** – control structures that allow repeated behavior within a program

4. **Functions** – blocks of commands that can be reused

data types: sequences

String—ordered sequence of characters

```
‘happy’
```

List—ordered sequence of items

```
[‘Leia’, ‘Rey’, ‘Maz’]
```

Dictionary—unordered sequence of key-value pairs

```
{‘name’:‘Kylo’, ‘side’:‘dark’}
```

referencing sequences

Reference by index number, starting with zero

```
mystring = ‘happy’
print(mystring[0])
print(mystring[2:4])

mylist = [‘Leia’, ‘Rey’, ‘Maz’]
print(mylist[-1])
```

Reference by key

```
mydict = {‘name’:‘Kylo’, ‘side’:‘dark’}
print(mydict[‘name’])
```

conditionals

4-space **indentation** is very important in Python. In this case, it tells Python what to execute if the condition is true.


```
name = ‘Grace Hopper’

if len(name) < 20:
    print(‘Yes’)
else:
    print(‘No’)
```

operators

Assignment

assignment	=	`movie = "Rogue One"`
add and assign	+=	`i += 1` is the same as `i = i + 1`

String

concatenate	+	`"A" + "B"` produces `"AB"`
repeat	*	`"me"*3` produces `"mememe"`

Comparison

equal	==	not equal	!=
greater than	>	greater than/equal	>=
less than	<	less than/equal	<=

loops

colon

```
name = ‘Grace Hopper’

i = 0
for letter in name:
    if letter in [‘a’,‘e’,‘i’,‘o’,‘u’]:
        i = i + 1
print(name + ‘ has ’ + str(i) + ‘ vowels.’)
```

indentation

convert the integer i to a string in order to concatenate it with other strings

loops

```
name = ‘Grace Hopper’

i = 0
vowel_count = 0
while i < len(name):
    if name[i] in [‘a’,‘e’,‘i’,‘o’,‘u’]:
        vowel_count = vowel_count + 1
    i = i + 1
print(name + ‘ has’ + str(vowel_count) + ‘ vowels.’)
```

functions v. methods

Function-–named block of code that can accept any number of arguments

```
my_string = ‘aBcDe’
print(my_string)
```

Method—a function with a built-in parameter for the object being acted on

```
print(my_string.lower())
```

writing functions

def function_name(argument1, argument2, ...):
    first command
    second command
    return output

```
def say_hello(name_string):
    print('Hello, ' + str(name_string) + '!')
    return None

say_hello('NaLette')
```

part 3: working with files

files for part 3

www.nalettebrodnax.com/python/

workshop.py

kipling_jungle_book.txt

what do we want to do?

1. Ask the user for a text filename: `input()`

   Note: `input()` accepts and returns a string

2. Ask the user for the number of lines to display : `input()`

3. Convert the number of lines to an integer: `int()`

4. Display that number of lines: `print()`

user input

displayed to the user

```
print("What file should I read from?")
filename = input("> ")

print("How many lines should I read?")
lines_to_read = input("> ")
```

prompt

store input in variable

reading files using **open()**

You must close any file that has been opened

```
line_counter = 0

file = open(filename, 'r')
while line_counter < int(lines_to_read):
    print(file.readline())
    line_counter = line_counter + 1
file.close()
```

reading files using **with open()**

```
with open(filename, 'r') as file:
     while line_counter < int(lines_to_read):
          print(file.readline())
          line_counter += 1
print(str(line_counter) + " lines read\n")
```

The **with** clause automatically closes the file

working with CSV files

1. Count the number of words in each line
   Create a new function: `count_words()`

2. Find length of the longest word
   Create a new function: `longest_word_length()`

3. Record the line number, word count, and length of the longest word in a CSV file

writing a function

function name

argument

```
def count_words(mytext):
    """Returns the number of words in a string
    str -> int"""
    words = mytext.split(" ")
    return len(words)
```

return value

another function

```
def longest_word_length(mytext):
"""Returns the average word length in a string
    str -> int"""
    words = mytext.split(" ")
    word_lengths = []
    for word in words:
        word_lengths.append(len(word))
    return max(word_lengths)
```

modules

Import statements allow you to add functions

```
import csv

with open("workshop.csv", 'w') as csvfile, \
     open(filename, 'r') as txtfile:
    writer = csv.writer(csvfile)
```

a “writer” object

writing files

```
with open("workshop.csv", 'w') as csvfile, \
     open(filename, 'r') as txtfile:
    writer = csv.writer(csvfile)
    writer.writerow(["line number", "word count", "longest word"])
    line_counter = 0
    while line_counter < int(lines_to_read):
        line_number = line_counter + 1
        content = txtfile.readline()
        word_count = count_words(content)
        longest_word = longest_word_length(content)
        writer.writerow([line_number, word_count, longest_word])
        line_counter += 1
```

run your script

From IDLE

- **F5** or **Run → Run Module**

From the Command Line

Terminal (Mac OS X) Example

- Use **ls** to see a list of files in the directory
- Use **cd** to move to the directory that contains your script

```
149-160-200-169:~nbrodnax$ ls
149-160-200-169:~nbrodnax$ cd Documents
149-160-200-169:~nbrodnax$ python3 workshop.py
```

Questions?

www.nalettebrodnax.com

email: nbrodnax@indiana.edu

linkedin: nalettebrodnax
github: nmbrodnax
twitter: @nbrodnax

Introduction to Using APIs with Python

NaLette Brodnax

Indiana University Bloomington

School of Public & Environmental Affairs

Department of Political Science

www.nalettebrodnax.com

what is an API?

An application programming interface (api) is a tool that allows computers to exchange information.

uses of APIs

- Social – Twitter, Facebook, etc.
- Internet – bit.ly, domain registration
- Mapping – Google Maps, Bing Maps, etc.
- Search – Google, Yahoo, etc.

APIs make information transferred across the web digestible for a computer.

what we'll cover in this workshop

1	2	3
python review	simple example	complex example

getting the tools for today

- Use Python 3 on SSRC computer
  - Search for IDLE, a basic Python IDE
  - **requests** library already installed

- Use Python on your machine
  - Download Python 3.6 from www.python.org
  - Use IDLE (or IDE of choice)
  - Install the **requests** library using pip (or pip3 on Mac OSX). See docs.python-requests.org

part 1: python review

data types: sequences

String—ordered sequence of characters

```
‘happy’
```

List—ordered sequence of items

```
[‘Leia’, ‘Rey’, ‘Maz’]
```

Dictionary—unordered sequence of key-value pairs

```
{‘name’:‘Kylo’, ‘side’:‘dark’}
```

referencing sequences

Reference by index number, starting with zero

```
mystring = ‘happy’
print(mystring[0])
print(mystring[2:4])

mylist = [‘Leia’, ‘Rey’, ‘Maz’]
print(mylist[-1])
```

Reference by key

```
mydict = {‘name’:‘Kylo’, ‘side’:‘dark’}
print(mydict[‘name’])
```

modules

Import statements allow you to add functions

```
import csv
import json
import requests
```

csv – read and/or write csv files

json – decode or encode data in json (javascript object notation) format

requests – use the HTTP protocol to exchange data over the web

functions v. methods

Function-–named block of code that can accept any number of arguments

```
my_string = ‘aBcDe’
print(my_string)
```

Method—a function with a built-in parameter for the object being acted on

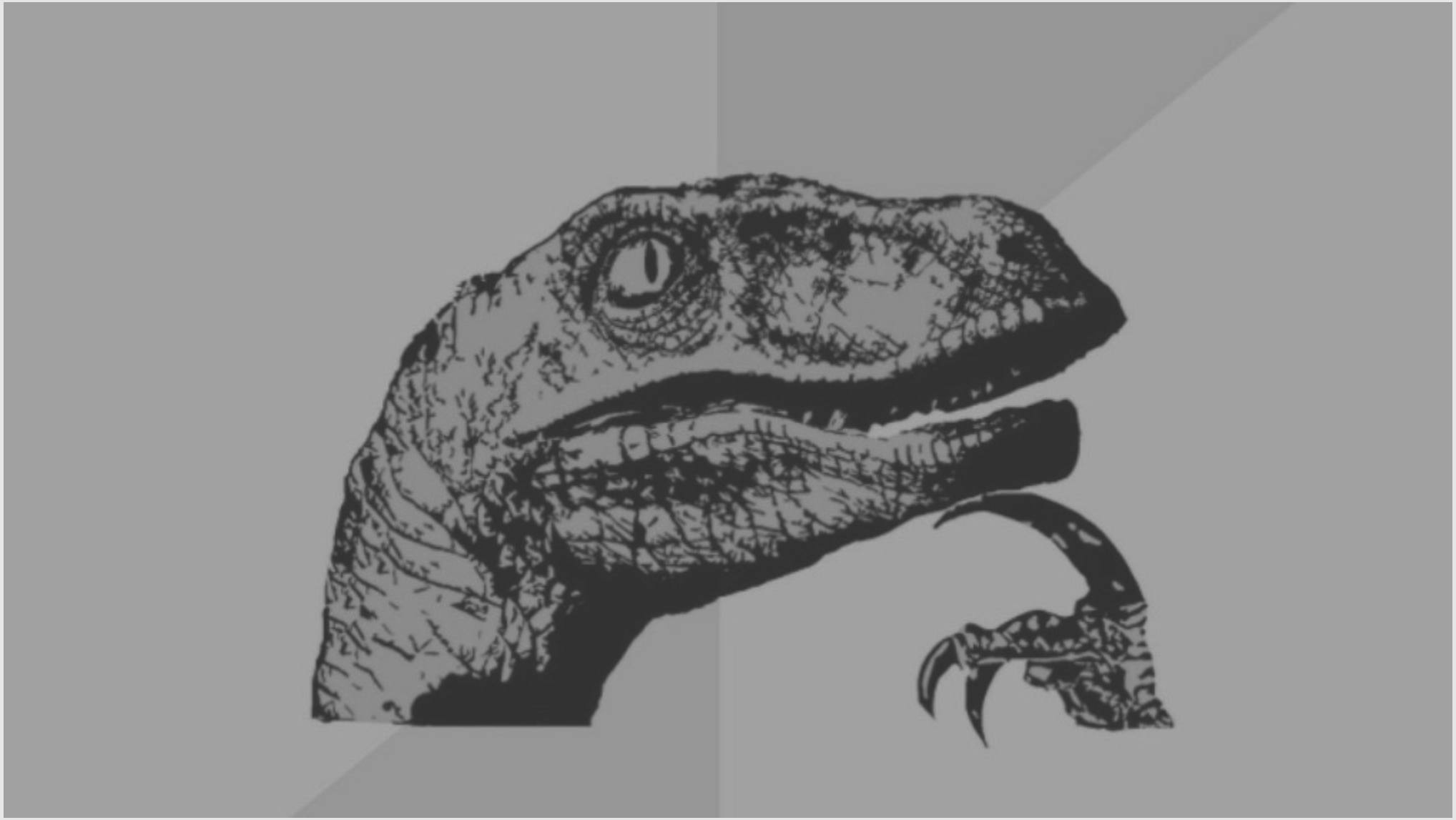
```
print(my_string.lower())
```

part 2: geocode an address

files for part 2

www.nalettebrodnax.com/python/

maps_api.py



Review • **A**ccess • **P**arse • **T**ransform • st**OR**e

RAPTOR

	Web Server	**Web Server + API**
Review	HTML structure (tags, attributes, etc.)	Parameters and structure from documentation
Access	No registration, no authentication	Registration and sometimes authentication
Parse	HTML	JSON or XML
Transform	Nested tables, lists	Nested dictionary
St**OR**e	Text, CSV	Text, CSV

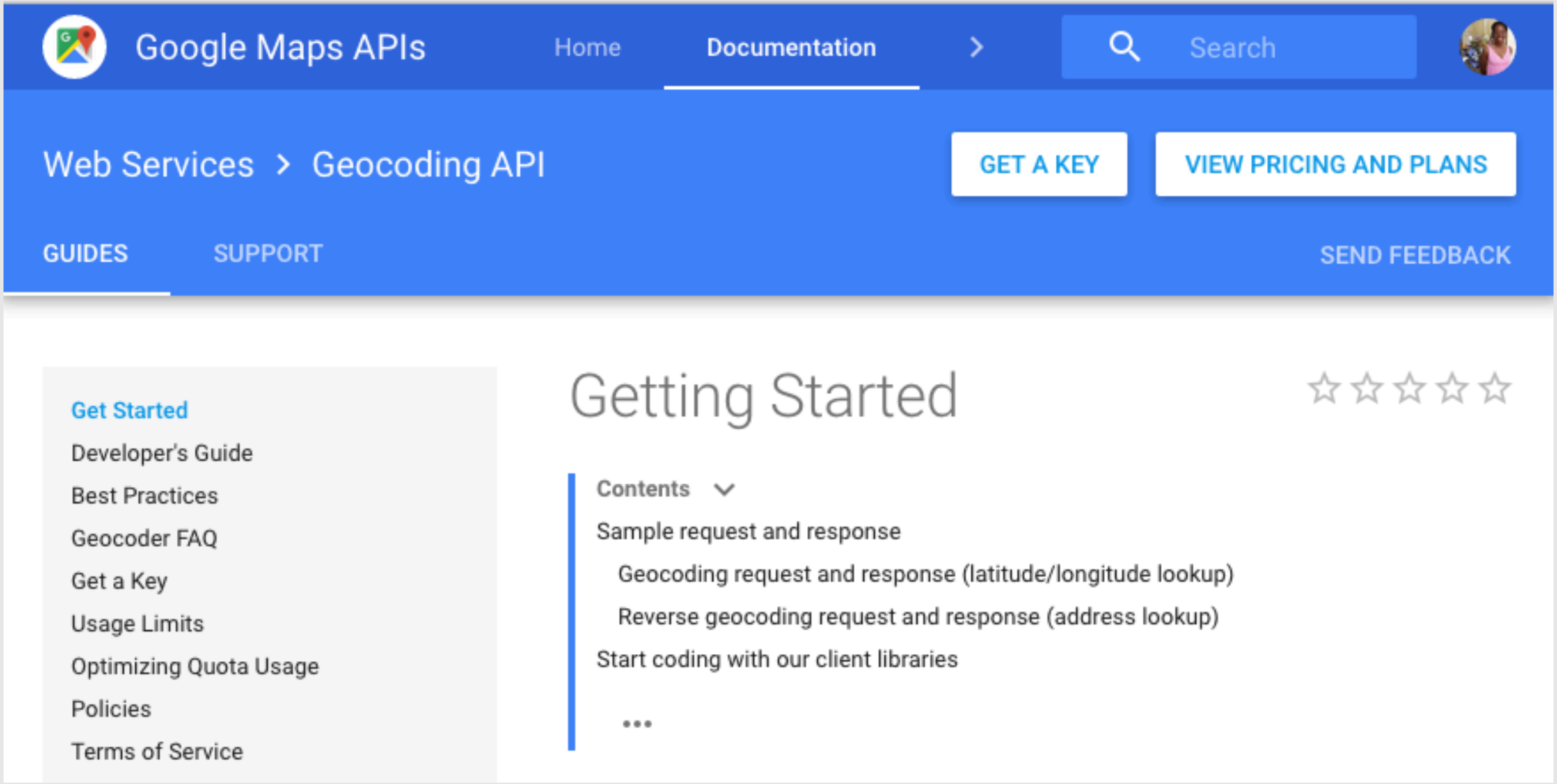
next steps

Google Maps API
Web Services APIs: Google Maps Geocoding API

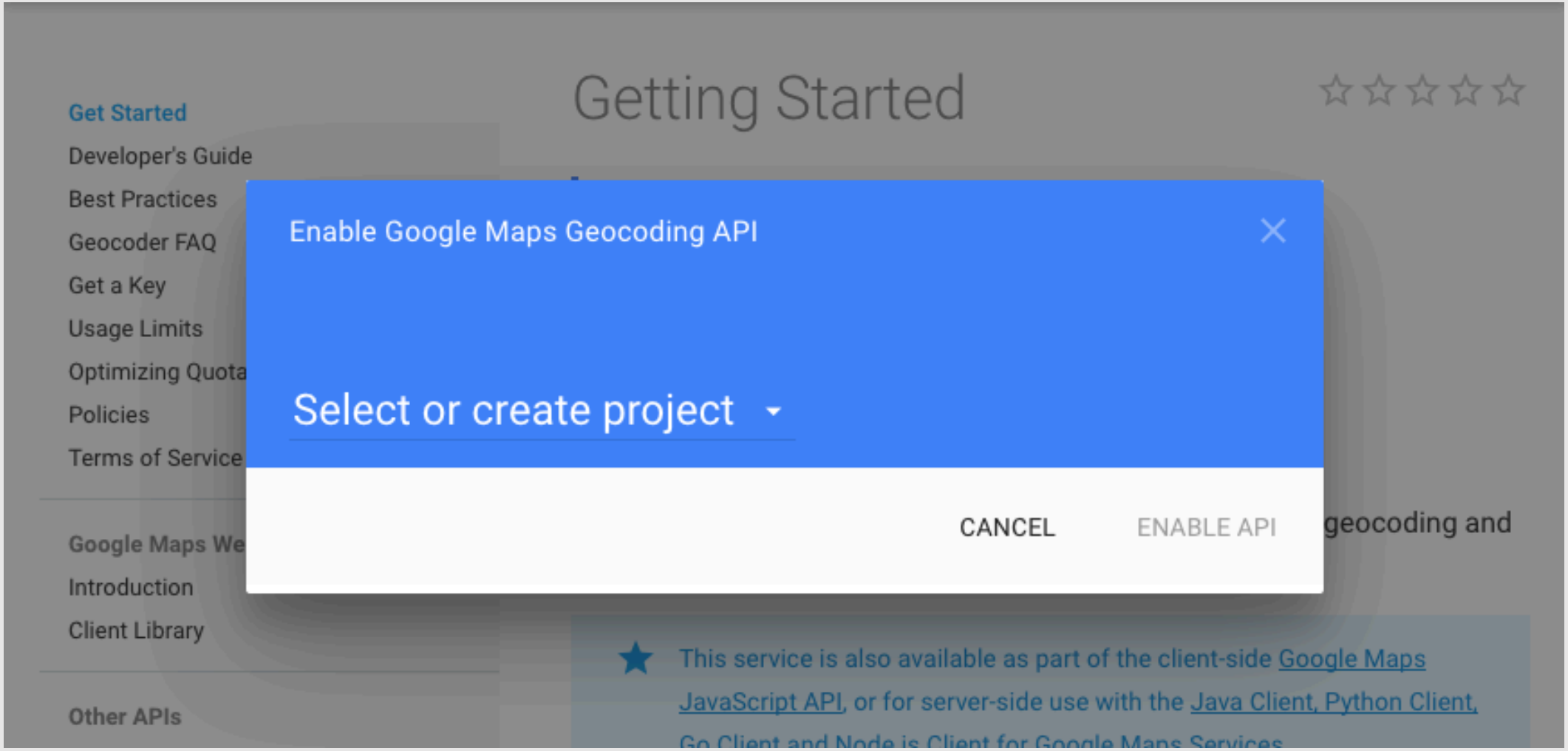
Steps

- Register as a developer
- Create an application
- Create an authentication document
- Use the API - RAPTOR

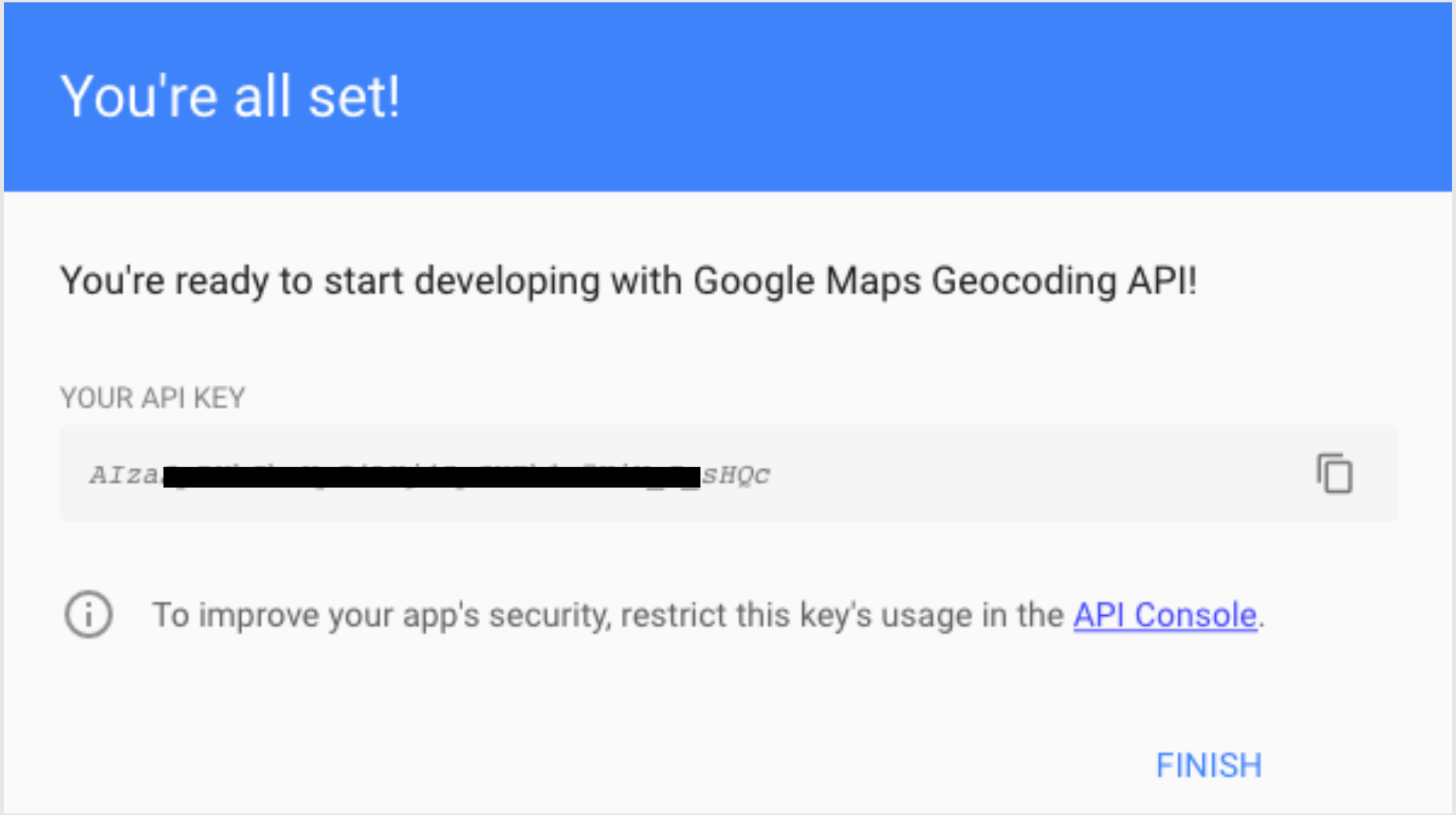
https://developers.google.com/maps



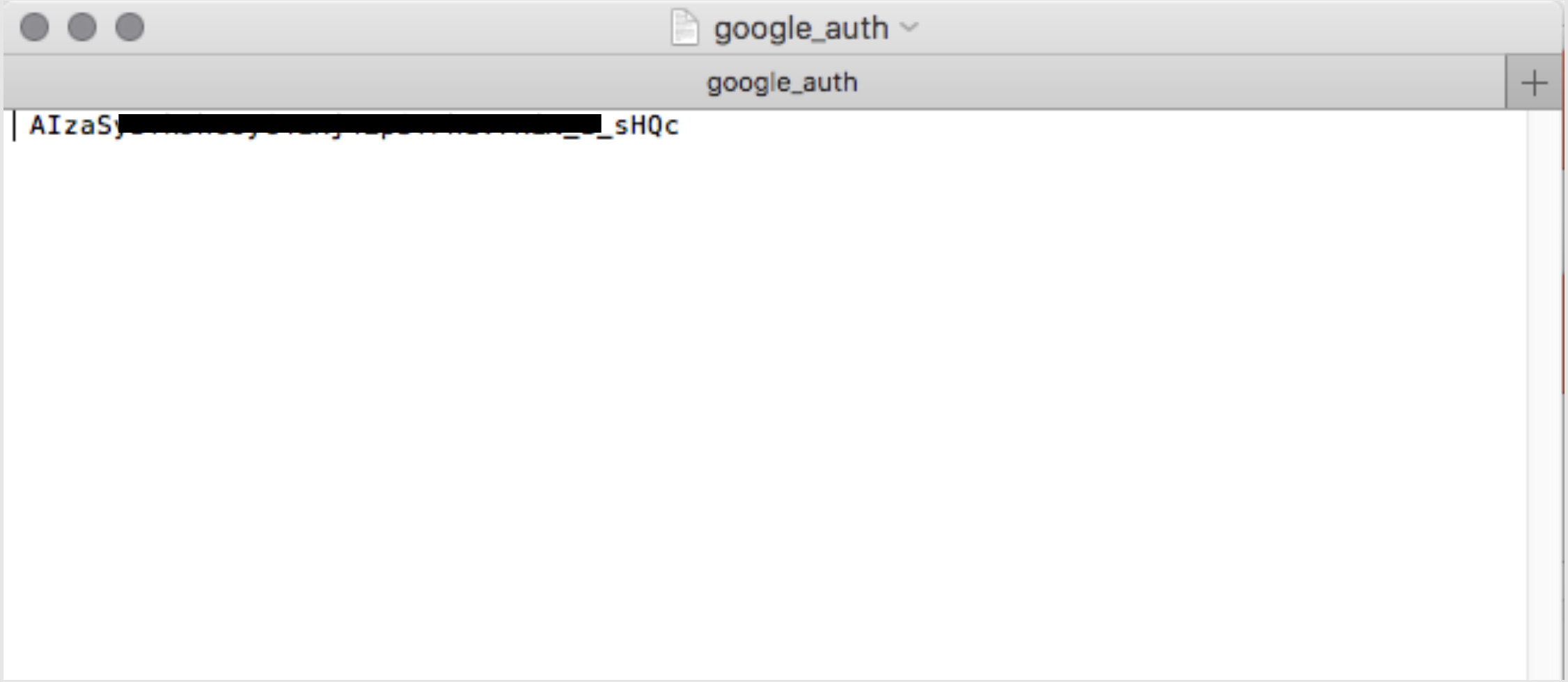
your application

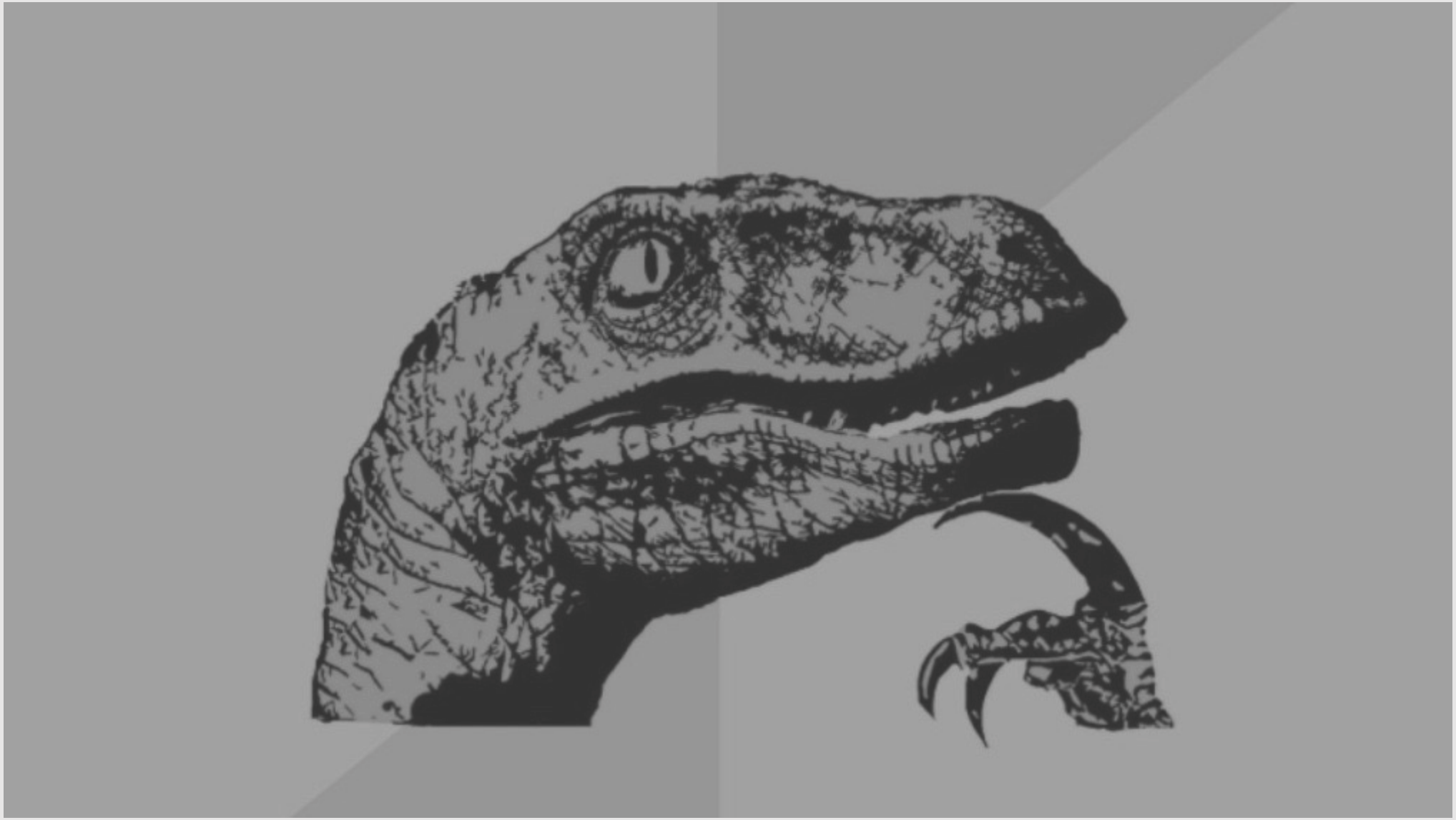


Save API key to a text file



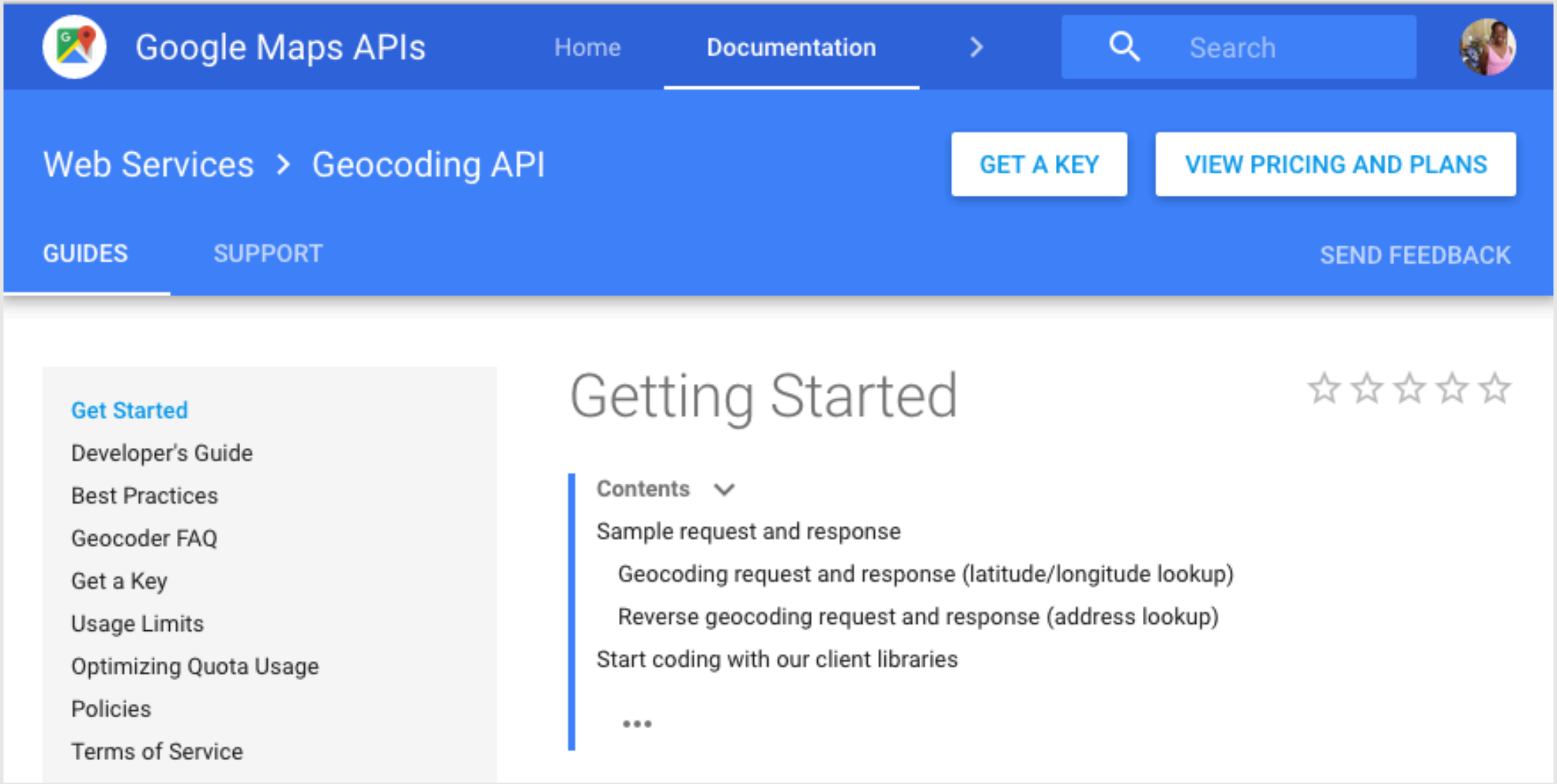
Save API key to a text file





Review • **A**ccess • **P**arse • **T**ransform • st**OR**e

review


Google Maps APIs
Home
Documentation
Search
Web Services > Geocoding API
GET A KEY
VIEW PRICING AND PLANS
GUIDES
SUPPORT
SEND FEEDBACK
Get Started
Developer's Guide
Best Practices
Geocoder FAQ
Get a Key
Usage Limits
Optimizing Quota Usage
Policies
Terms of Service
Getting Started
Contents
Sample request and response
Geocoding request and response (latitude/longitude lookup)
Reverse geocoding request and response (address lookup)
Start coding with our client libraries

review

format

```
host = 'https://maps.googleapis.com/maps/api/geocode/json'
```

web server

access

```
import csv
import json
import requests

local_file = 'google_auth.txt'
with open(local_file) as txtfile:
    my_key = txtfile.read()

print("API Key: " + my_key)
```

string containing the name of the text file with your api key

access

the address to search

```
address = "1350 E 10th St, Bloomington, IN 47405"

url = host + "?address=" + address + "&key=" + my_key

response = requests.request('GET', url)
print(response)
```

check the response code (see https://httpstatuses.com/)

our GET request

```
https://maps.googleapis.com/maps/api/geocode/json?
address=1350+E+10th+St,+Bloomington,+IN+47405&key=
<my_api_key>
```

you can enter your GET request string into a browser

parse

convert JSON data to a Python object

```
geo = response.json()
```

transform

"pretty print" to see the nested structure

```
print(json.dumps(geo, indent = 4, sort_keys = True))
```

transform

save only what you need

```
print(type(geo)) # geo is a dictionary

print(geo.keys()) # check the dictionary keys

results = geo['results'][0]

print(results.keys())
```

store

```
longitude = results['geometry']['location']['lng']

latitude = results['geometry']['location']['lat']
```

run your script

From IDLE

- **F5** or **Run → Run Module**

From the Command Line

Terminal (Mac OS X) Example

- Use **ls** to see a list of files in the directory
- Use **cd** to move to the directory that contains your script

```
149-160-200-169:~nbrodnax$ ls
149-160-200-169:~nbrodnax$ cd Documents
149-160-200-169:~nbrodnax$ python3 maps_api.py
```

let’s take a 5-minute break!

part 3: geocode a list of addresses

files for part 3

www.nalettebrodnax.com/python/

maps_api_loop.py

charter_schools.csv

review – access

```
import csv
import json
import requests

host = 'https://maps.googleapis.com/maps/api/geocode/json'

local_file = 'google_auth.txt'
with open(local_file) as txtfile:
    my_key = txtfile.read()

print("API Key: " + my_key)
```

automate: access – parse – transform

```
def get_coordinates(address, api_key):
    """Returns a list with latitude and longitude for a given address
    str, str -> list"""

    url = host + "?address=" + address + "&key=" + api_key
    response = requests.request('GET', url)

    geo = response.json()

    results = geo['results'][0]
    return [results['geometry']['location']['lat'],\
    results['geometry']['location']['lng']]
```

store

	A	B	C	D
1	NAME	ADDRESS	CITY STATE ZIP	ZIP
2	Academy of Communications and Technology (ACT) Charter School	4319 W. Washington Blvd.	Chicago, IL 60624	60624
3	Chicago International Charter School - Bucktown Campus	2235 N. Hamilton	Chicago, IL 60647	60647
4	Chicago International Charter School - Longwood Campus	1309 W. 95th St.	Chicago, IL 60643	60643

[{'NAME': 'Academy of Communications and Technology (ACT) Charter School, 'ADDRESS': '4319 W. Washington Blvd.', 'CITY STATE ZIP': 'Chicago, IL 60624}, {'NAME': 'Chicago International Charter School – Bucktown Campus, 'ADDRESS': '2235 N. Hamilton', 'CITY STATE ZIP': 'Chicago, IL 60647}]

store

```
csvfile = open("charter_schools.csv", 'r')
header = csvfile.readline()
csvfile.close()

# print(header)
header = header.strip('\n')

fieldnames = header.split(',')
```

store

```
with open("charter_schools.csv", "r") as infile, \
     open("charter_schools_geo.csv", "w") as outfile:

     reader = csv.DictReader(infile, fieldnames)
     next(reader, None)

     fieldnames.append('LATITUDE')
     fieldnames.append('LONGITUDE')
     writer = csv.DictWriter(outfile, fieldnames)
     writer.writeheader()
```

store

```
with open("charter_schools.csv", "r") as infile, \
     .
     .
     .
     for school in reader:
          school_address = school['ADDRESS'] + ", " +
                           school['CITY STATE ZIP']
          location = get_coordinates(school_address, my_key)
          school['LATITUDE'] = location[0]
          school['LONGITUDE'] = location[1]
          writer.writerow(school)
```

run your script

From IDLE

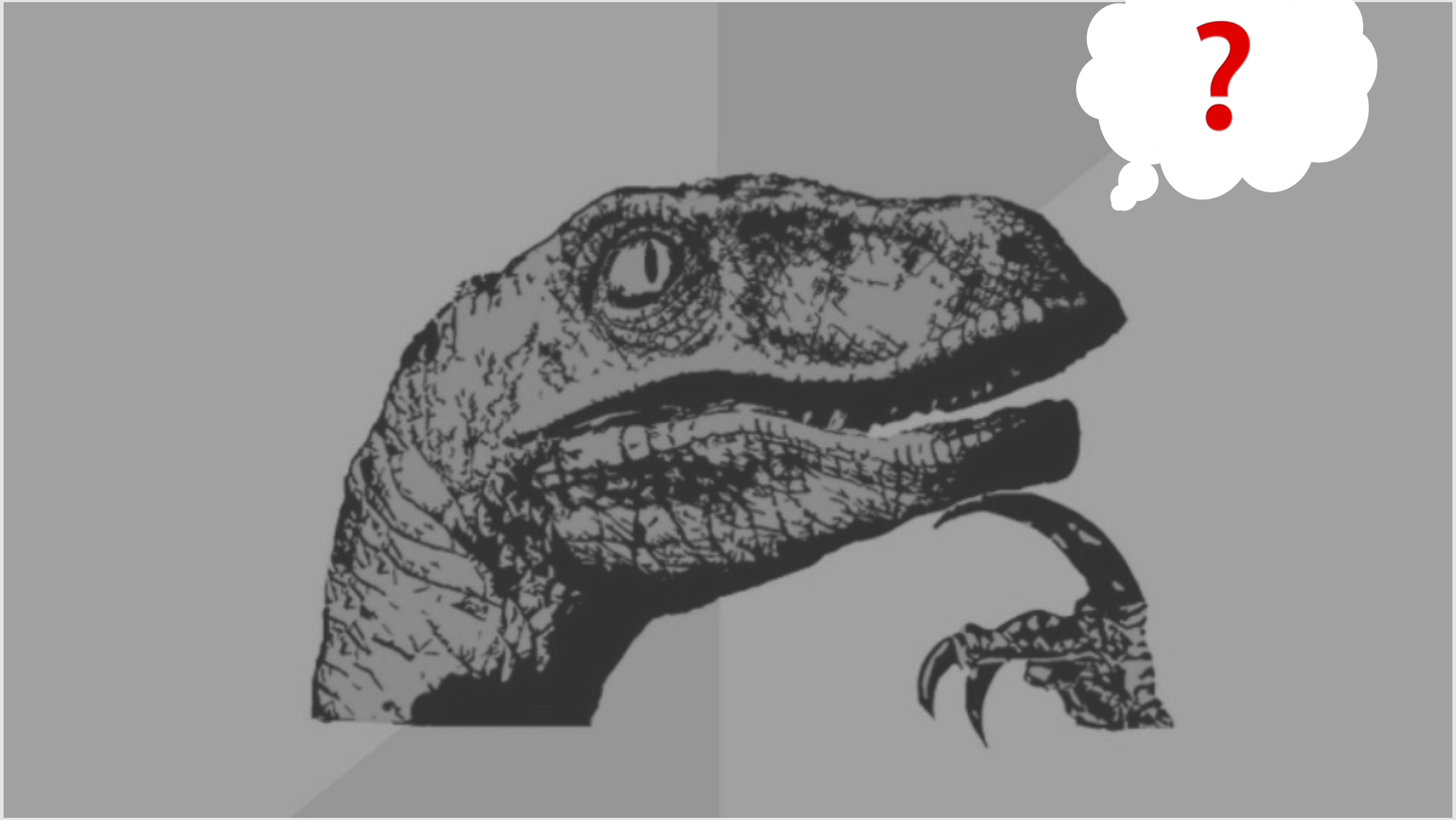
- **F5** or **Run → Run Module**

From the Command Line

Terminal (Mac OS X) Example

- Use **ls** to see a list of files in the directory
- Use **cd** to move to the directory that contains your script

```
149-160-200-169:~nbrodnax$ ls
149-160-200-169:~nbrodnax$ cd Documents
149-160-200-169:~nbrodnax$ python3 maps_api_loop.py
```



Review • **A**ccess • **P**arse • **T**ransform • st**OR**e

Thank You!

www.nalettebrodnax.com

email: nbrodnax@indiana.edu

linkedin: nalettebrodnax
github: nmbrodnax
twitter: @nbrodnax