In [1]:
```
# Dependencies
import requests
import json
```

In [2]:
```
# URL for GET requests to retrieve vehicle data
url = "https://api.spacexdata.com/v2/launchpads"
```

In [3]:
```
# Print the response object to the console
print(requests.get(url))
```

```
<Response [200]>
```

In [4]:
```
# Retrieving data and converting it into JSON
print(requests.get(url).json())
```

```
[{'padid': 6, 'id': 'vafb_slc_4e', 'name': 'VAFB SLC 4E', 'full_name': 'Vandenberg Air Force
Base Space Launch Complex 4E', 'status': 'active', 'location': {'name': 'Vandenberg Air Forc
e Base', 'region': 'California', 'latitude': 34.632093, 'longitude': -120.610829}, 'vehicles
_launched': ['Falcon 9'], 'attempted_launches': 15, 'successful_launches': 15, 'wikipedia':
'https://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_4', 'details': 'SpaceX pr
imary west coast launch pad for polar orbits and sun synchronous orbits, primarily used for
Iridium. Also intended to be capable of launching Falcon Heavy.'}, {'padid': 2, 'id': 'ccafs
_slc_40', 'name': 'CCAFS SLC 40', 'full_name': 'Cape Canaveral Air Force Station Space Launc
h Complex 40', 'status': 'active', 'location': {'name': 'Cape Canaveral', 'region': 'Florid
a', 'latitude': 28.5618571, 'longitude': -80.577366}, 'vehicles_launched': ['Falcon 9'], 'at
tempted_launches': 49, 'successful_launches': 47, 'wikipedia': 'https://en.wikipedia.org/wik
i/Cape_Canaveral_Air_Force_Station_Space_Launch_Complex_40', 'details': 'SpaceX primary Falc
on 9 launch pad, where all east coast Falcon 9s launched prior to the AMOS-6 anomaly. Initia
lly used to launch Titan rockets for Lockheed Martin. Back online since CRS-13 on 2017-12-1
5.'}, {'padid': 8, 'id': 'stls', 'name': 'STLS', 'full_name': 'SpaceX South Texas Launch Sit
e', 'status': 'under construction', 'location': {'name': 'Boca Chica Village', 'region': 'Te
xas', 'latitude': 25.9972641, 'longitude': -97.1560845}, 'vehicles_launched': ['Falcon 9'],
'attempted_launches': 0, 'successful_launches': 0, 'wikipedia': 'https://en.wikipedia.org/wi
ki/SpaceX_South_Texas_Launch_Site', 'details': 'SpaceX new launch site currently under const
ruction to help keep up with the Falcon 9 and Heavy manifests. Expected to be completed in l
ate 2018. Initially will be limited to 12 flights per year, and only GTO launches.'}, {'padi
d': 1, 'id': 'kwajalein_atoll', 'name': 'Kwajalein Atoll', 'full_name': 'Kwajalein Atoll Ome
lek Island', 'status': 'retired', 'location': {'name': 'Omelek Island', 'region': 'Marshall
Islands', 'latitude': 9.0477206, 'longitude': 167.7431292}, 'vehicles_launched': ['Falcon
1'], 'attempted_launches': 5, 'successful_launches': 2, 'wikipedia': 'https://en.wikipedia.o
rg/wiki/Omelek_Island', 'details': 'SpaceX original launch site, where all of the Falcon 1 l
aunches occured. Abandoned as SpaceX decided against upgrading the pad to support Falcon
9.'}, {'padid': 4, 'id': 'ksc_lc_39a', 'name': 'KSC LC 39A', 'full_name': 'Kennedy Space Cen
ter Historic Launch Complex 39A', 'status': 'active', 'location': {'name': 'Cape Canaveral',
'region': 'Florida', 'latitude': 28.6080585, 'longitude': -80.6039558}, 'vehicles_launched':
['Falcon 9', 'Falcon Heavy'], 'attempted_launches': 18, 'successful_launches': 18, 'wikipedi
a': 'https://en.wikipedia.org/wiki/Kennedy_Space_Center_Launch_Complex_39#Launch_Pad_39A',
'details': 'NASA historic launch pad that launched most of the Saturn V and Space Shuttle mi
ssions. Initially for Falcon Heavy launches, it is now launching all of SpaceX east coast mi
ssions due to the damage from the AMOS-6 anomaly. After SLC-40 repairs are complete, it will
be upgraded to support Falcon Heavy, a process which will take about two months. In the futu
re it will launch commercial crew missions and the Interplanetary Transport System.'}, {'pad
id': 5, 'id': 'vafb_slc_3w', 'name': 'VAFB SLC 3W', 'full_name': 'Vandenberg Air Force Base
Space Launch Complex 3W', 'status': 'retired', 'location': {'name': 'Vandenberg Air Force Ba
se', 'region': 'California', 'latitude': 34.6440904, 'longitude': -120.5931438}, 'vehicles_l
aunched': ['Falcon 1'], 'attempted_launches': 0, 'successful_launches': 0, 'wikipedia': 'htt
ps://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_3', 'details': 'SpaceX origin
al west coast launch pad for Falcon 1. Performed a static fire but was never used for a laun
ch and abandoned due to scheduling conflicts.'}]
```

In [5]:
```
# Pretty Print the output of the JSON
response = requests.get(url).json()
print(json.dumps(response, indent=4, sort_keys=True))
```

```
[
    {
        "attempted_launches": 15,
        "details": "SpaceX primary west coast launch pad for polar orbits and sun synchron
ous orbits, primarily used for Iridium. Also intended to be capable of launching Falcon He
avy.",
        "full_name": "Vandenberg Air Force Base Space Launch Complex 4E",
        "id": "vafb_slc_4e",
        "location": {
            "latitude": 34.632093,
            "longitude": -120.610829,
            "name": "Vandenberg Air Force Base",
            "region": "California"
        },
        "name": "VAFB SLC 4E",
        "padid": 6,
        "status": "active",
        "successful_launches": 15,
        "vehicles_launched": [
```

In [ ]:

In [1]:
```
# Dependencies
import requests
import json
```

In [2]:
```
# URL for GET requests to retrieve vehicle data
url = "https://api.spacexdata.com/v2/launchpads"
```

In [3]:
```
# Retrieving data and converting it into JSON
print(requests.get(url).json())
```

```
[{'padid': 6, 'id': 'vafb_slc_4e', 'name': 'VAFB SLC 4E', 'full_name': 'Vandenberg Air Force
Base Space Launch Complex 4E', 'status': 'active', 'location': {'name': 'Vandenberg Air Forc
e Base', 'region': 'California', 'latitude': 34.632093, 'longitude': -120.610829}, 'vehicles
_launched': ['Falcon 9'], 'attempted_launches': 15, 'successful_launches': 15, 'wikipedia':
'https://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_4', 'details': 'SpaceX pr
imary west coast launch pad for polar orbits and sun synchronous orbits, primarily used for
Iridium. Also intended to be capable of launching Falcon Heavy.'}, {'padid': 2, 'id': 'ccafs
_slc_40', 'name': 'CCAFS SLC 40', 'full_name': 'Cape Canaveral Air Force Station Space Launc
h Complex 40', 'status': 'active', 'location': {'name': 'Cape Canaveral', 'region': 'Florid
a', 'latitude': 28.5618571, 'longitude': -80.577366}, 'vehicles_launched': ['Falcon 9'], 'at
tempted_launches': 49, 'successful_launches': 47, 'wikipedia': 'https://en.wikipedia.org/wik
i/Cape_Canaveral_Air_Force_Station_Space_Launch_Complex_40', 'details': 'SpaceX primary Falc
on 9 launch pad, where all east coast Falcon 9s launched prior to the AMOS-6 anomaly. Initia
lly used to launch Titan rockets for Lockheed Martin. Back online since CRS-13 on 2017-12-1
5.'}, {'padid': 8, 'id': 'stls', 'name': 'STLS', 'full_name': 'SpaceX South Texas Launch Sit
e', 'status': 'under construction', 'location': {'name': 'Boca Chica Village', 'region': 'Te
xas', 'latitude': 25.9972641, 'longitude': -97.1560845}, 'vehicles_launched': ['Falcon 9'],
'attempted_launches': 0, 'successful_launches': 0, 'wikipedia': 'https://en.wikipedia.org/wi
ki/SpaceX_South_Texas_Launch_Site', 'details': 'SpaceX new launch site currently under const
ruction to help keep up with the Falcon 9 and Heavy manifests. Expected to be completed in l
ate 2018. Initially will be limited to 12 flights per year, and only GTO launches.'}, {'padi
d': 1, 'id': 'kwajalein_atoll', 'name': 'Kwajalein Atoll', 'full_name': 'Kwajalein Atoll Ome
lek Island', 'status': 'retired', 'location': {'name': 'Omelek Island', 'region': 'Marshall
Islands', 'latitude': 9.0477206, 'longitude': 167.7431292}, 'vehicles_launched': ['Falcon
1'], 'attempted_launches': 5, 'successful_launches': 2, 'wikipedia': 'https://en.wikipedia.o
rg/wiki/Omelek_Island', 'details': 'SpaceX original launch site, where all of the Falcon 1 l
aunches occured. Abandoned as SpaceX decided against upgrading the pad to support Falcon
9.'}, {'padid': 4, 'id': 'ksc_lc_39a', 'name': 'KSC LC 39A', 'full_name': 'Kennedy Space Cen
ter Historic Launch Complex 39A', 'status': 'active', 'location': {'name': 'Cape Canaveral',
'region': 'Florida', 'latitude': 28.6080585, 'longitude': -80.6039558}, 'vehicles_launched':
['Falcon 9', 'Falcon Heavy'], 'attempted_launches': 18, 'successful_launches': 18, 'wikipedi
a': 'https://en.wikipedia.org/wiki/Kennedy_Space_Center_Launch_Complex_39#Launch_Pad_39A',
'details': 'NASA historic launch pad that launched most of the Saturn V and Space Shuttle mi
ssions. Initially for Falcon Heavy launches, it is now launching all of SpaceX east coast mi
ssions due to the damage from the AMOS-6 anomaly. After SLC-40 repairs are complete, it will
be upgraded to support Falcon Heavy, a process which will take about two months. In the futu
re it will launch commercial crew missions and the Interplanetary Transport System.'}, {'pad
id': 5, 'id': 'vafb_slc_3w', 'name': 'VAFB SLC 3W', 'full_name': 'Vandenberg Air Force Base
Space Launch Complex 3W', 'status': 'retired', 'location': {'name': 'Vandenberg Air Force Ba
se', 'region': 'California', 'latitude': 34.6440904, 'longitude': -120.5931438}, 'vehicles_l
aunched': ['Falcon 1'], 'attempted_launches': 0, 'successful_launches': 0, 'wikipedia': 'htt
ps://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_3', 'details': 'SpaceX origin
al west coast launch pad for Falcon 1. Performed a static fire but was never used for a laun
ch and abandoned due to scheduling conflicts.'}]
```

In [4]:
```python
# Pretty print JSON for all launchpads
response = requests.get(url).json()
print(json.dumps(response, indent=4, sort_keys=True))
```

```
        "details": "SpaceX original west coast launch pad for Falcon 1. Performed a static
fire but was never used for a launch and abandoned due to scheduling conflicts.",
        "full_name": "Vandenberg Air Force Base Space Launch Complex 3W",
        "id": "vafb_slc_3w",
        "location": {
            "latitude": 34.6440904,
            "longitude": -120.5931438,
            "name": "Vandenberg Air Force Base",
            "region": "California"
        },
        "name": "VAFB SLC 3W",

        "padid": 5,
        "status": "retired",
        "successful_launches": 0,
        "vehicles_launched": [
            "Falcon 1"
        ],
        "wikipedia": "https://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_3"
    }
]
```

In [5]:
```python
# Pretty print JSON for a specific launchpad
url_vafb_slc_4e = "https://api.spacexdata.com/v2/launchpads/vafb_slc_4e"
```

In [6]:
```python
response = requests.get(url_vafb_slc_4e).json()
print(json.dumps(response, indent=4, sort_keys=True))

#another option is to concatenate or even make it a variable
response = requests.get(url + "/vafb_slc_4e").json()
print(json.dumps(response, indent=4, sort_keys=True))
```

```
{
    "attempted_launches": 15,
    "details": "SpaceX primary west coast launch pad for polar orbits and sun synchronous or
bits, primarily used for Iridium. Also intended to be capable of launching Falcon Heavy.",
    "full_name": "Vandenberg Air Force Base Space Launch Complex 4E",
    "id": "vafb_slc_4e",
    "location": {
        "latitude": 34.632093,
        "longitude": -120.610829,
        "name": "Vandenberg Air Force Base",
        "region": "California"
    },
    "name": "VAFB SLC 4E",
    "padid": 6,
    "status": "active",
    "successful_launches": 15,
    "vehicles_launched": [
        "Falcon 9"
    ],
    "wikipedia": "https://en.wikipedia.org/wiki/Vandenberg_AFB_Space_Launch_Complex_4"
}
```

In [1]:
```python
# Dependencies
import requests
import json
```

In [2]:
```python
# Performing a GET Request and saving the
# API's response within a variable
url = "https://api.spacexdata.com/v2/rockets/falcon9"
response = requests.get(url)
response_json = response.json()
print(json.dumps(response_json, indent=4, sort_keys=True))
```

```
    "first_stage": {
        "burn_time_sec": 162,
        "engines": 9,
        "fuel_amount_tons": 385,
        "reusable": true,
        "thrust_sea_level": {
            "kN": 7607,
            "lbf": 1710000
        },
        "thrust_vacuum": {
            "kN": 8227,
            "lbf": 1849500
        }
    },
    "flickr_images": [
        "https://farm1.staticflickr.com/929/28787338307_3453a11a77_b.jpg",
        "https://farm4.staticflickr.com/3955/32915197674_eee74d81bb_b.jpg",
        "https://farm1.staticflickr.com/293/32312415025_6841e30bf1_b.jpg",
        "https://farm1.staticflickr.com/623/23660653516_5b6cb301d1_b.jpg",
        "https://farm6.staticflickr.com/5518/31579784413_d853331601_b.jpg",
```

In [3]:
```python
# It is possible to grab a specific value
# from within the JSON object
print(response_json["cost_per_launch"])
```

```
50000000
```

In [4]:
```python
# It is also possible to perform some
# analyses on values stored within the JSON object
number_payloads = len(response_json["payload_weights"])
print(f"There are {number_payloads} payloads.")
```

```
There are 3 payloads.
```

In [7]:
```python
# Finally, it is possible to reference the
# values stored within sub-dictionaries and sub-lists
payload_weight = response_json["payload_weights"][0]["kg"]
print(f"The first payload weighed {payload_weight} Kilograms")
```

```
The first payload weighed 22800 Kilograms
```

In [26]:
```
# Dependencies
import requests
import json
```

In [27]:
```
# URL for GET requests to retrieve Star Wars character data
base_url = "https://swapi.co/api/people/"
```

In [28]:
```
# Create a url with a specific character id
character_id = '4'
url = base_url + character_id
print(url)
```

```
https://swapi.co/api/people/4 (https://swapi.co/api/people/4)
```

In [29]:
```
# Perform a get request for this character
response = requests.get(url)
print(response.url)
```

```
https://swapi.co/api/people/4/ (https://swapi.co/api/people/4/)
```

In [30]:
```
# Storing the JSON response within a variable
data = response.json()
# Use json.dumps to print the json
print(json.dumps(data, indent=4, sort_keys=True))
```

```
{
    "birth_year": "41.9BBY",
    "created": "2014-12-10T15:18:20.704000Z",
    "edited": "2014-12-20T21:17:50.313000Z",
    "eye_color": "yellow",
    "films": [
        "https://swapi.co/api/films/2/",
        "https://swapi.co/api/films/6/",
        "https://swapi.co/api/films/3/",
        "https://swapi.co/api/films/1/"
    ],
    "gender": "male",
    "hair_color": "none",
    "height": "202",
    "homeworld": "https://swapi.co/api/planets/1/",
    "mass": "136",
    "name": "Darth Vader",
    "skin_color": "white",
    "species": [
        "https://swapi.co/api/species/1/"
    ],
    "starships": [
        "https://swapi.co/api/starships/13/"
    ],
    "url": "https://swapi.co/api/people/4/",
    "vehicles": []
}
```

In [31]:
```
# Print the name of the character retrieved
character_name = data["name"]
print(character_name)
```
```
Darth Vader
```

In [32]:
```
# Print the number of films that they were in (hint: use len())
film_number = len(data["films"])
print(film_number)
```
```
4
```

In [33]:
```
# Request the starships URI found in the starships property of the
# previously retreived json, then use the response to figure out what this
# character's first starship was
first_ship_url = data["starships"][0]
ship_response = requests.get(first_ship_url).json()
ship_response
```
Out[33]:
```
{'name': 'TIE Advanced x1',
 'model': 'Twin Ion Engine Advanced x1',
 'manufacturer': 'Sienar Fleet Systems',
 'cost_in_credits': 'unknown',
 'length': '9.2',
 'max_atmosphering_speed': '1200',
 'crew': '1',
 'passengers': '0',
 'cargo_capacity': '150',
 'consumables': '5 days',
 'hyperdrive_rating': '1.0',
 'MGLT': '105',
 'starship_class': 'Starfighter',
 'pilots': ['https://swapi.co/api/people/4/'],
 'films': ['https://swapi.co/api/films/1/'],
 'created': '2014-12-12T11:21:32.991000Z',
 'edited': '2014-12-22T17:35:44.549047Z',
 'url': 'https://swapi.co/api/starships/13/'}
```

In [34]:
```
# Print the name of the character's first starship
first_ship = ship_response["name"]
print(f"Their first ship: {first_ship}")
```
```
Their first ship: TIE Advanced x1
```

In [35]:
```
# BONUS
films = []

for film in data['films']:
    cur_film = requests.get(film).json()
    film_title = cur_film["title"]
    films.append(film_title)

print(f"{character_name} was in:")
print(films)
```
```
Darth Vader was in:
['The Empire Strikes Back', 'Revenge of the Sith', 'Return of the Jedi', 'A New Hope']
```

In [2]:
```
# Dependencies
import requests
import json
```

In [3]:
```
# Base URL for GET requests to retrieve number/date facts
url = "http://numbersapi.com/"
```

In [4]:
```
# Ask the user what kind of data they would like to search for
question = ("What type of data would you like to search for? "
            "[Trivia, Math, Date, or Year] ")
kind_of_search = input(question)
```

```
What type of data would you like to search for? [Trivia, Math, Date, or Year] Math
```

In [5]:
```
# If the kind of search is "date" take in two numbers
if(kind_of_search.lower() == "date"):

  # Collect the month to search for
  month = input("What month would you like to search for? ")
  # Collect the day to search for
  day = input("What day would you like to search for? ")

  # Make an API call to the "date" API and convert response object to JSON
  response = requests.get(f"{url}{month}/{day}/{kind_of_search.lower()}?json").json()
  # Print the fact stored within the response
  print(response["text"])

# If the kind of search is anything but "date" then take one number
else:

  # Collect the number to search for
  number = input("What number would you like to search for? ")

  # Make an API call to the API and convert response object to JSON
  response = requests.get(url + number + "/" +  kind_of_search.lower()+ "?json").json()
  # Print the fact stored within the response
  print(response["text"])
```

```
What number would you like to search for? 50
50 is the smallest number that can be written as the sum of of 2 squares in 2 ways.
```

In [ ]:

In [1]:
```
import requests
import json
```

In [2]:
```
# New Dependency! Use this to pretty print the JSON
# https://docs.python.org/3/library/pprint.html
from pprint import pprint
```

In [3]:
```
# Note that the ?t= is a query param for the t-itle of the
# movie we want to search for.
url = "http://www.omdbapi.com/?t="
api_key = "&apikey=5ef8d5e2"
```

In [4]:
```
url + "Aliens" + api_key
```

Out[4]:
```
'http://www.omdbapi.com/?t=Aliens&apikey=5ef8d5e2'
```

In [5]:
```
# Performing a GET request similar to the one we executed
# earlier
response = requests.get(url + "Aliens" + api_key)
print(response.url)
```

```
http://www.omdbapi.com/?t=Aliens&apikey=5ef8d5e2 (http://www.omdbapi.com/?t=Alie
ns&apikey=5ef8d5e2)
```

In [6]:
```python
# Converting the response to JSON, and printing the result.
data = response.json()
pprint(data)
```

```
{'Actors': 'Sigourney Weaver, Carrie Henn, Michael Biehn, Paul Reiser',
 'Awards': 'Won 2 Oscars. Another 18 wins & 23 nominations.',
 'BoxOffice': 'N/A',
 'Country': 'USA, UK',
 'DVD': '01 Jun 1999',
 'Director': 'James Cameron',
 'Genre': 'Action, Adventure, Sci-Fi, Thriller',
 'Language': 'English',
 'Metascore': '84',
 'Plot': 'Ellen Ripley is rescued by a deep salvage team after being in '
         'hypersleep for 57 years. The moon that the Nostromo visited has been '
         'colonized, but contact is lost. This time, colonial marines have '
         'impressive firepower, but will that be enough?',
 'Poster': 'https://m.media-amazon.com/images/M/MV5BZGU2OGY5ZTYtMWNhYy00NjZiLWI0
NjUtZmNhY2JhNDRmODU3XkEyXkFqcGdeQXVyNzkwMjQ5NzM@._V1_SX300.jpg',
 'Production': '20th Century Fox',
 'Rated': 'R',
 'Ratings': [{'Source': 'Internet Movie Database', 'Value': '8.4/10'},
             {'Source': 'Rotten Tomatoes', 'Value': '99%'},
             {'Source': 'Metacritic', 'Value': '84/100'}],
 'Released': '18 Jul 1986',
 'Response': 'True',
 'Runtime': '137 min',
 'Title': 'Aliens',
 'Type': 'movie',
 'Website': 'N/A',
 'Writer': 'James Cameron (story by), David Giler (story by), Walter Hill '
           "(story by), Dan O'Bannon (based on characters created by), Ronald "
           'Shusett (based on characters created by), James Cameron '
           '(screenplay by)',
 'Year': '1986',
 'imdbID': 'tt0090605',
 'imdbRating': '8.4',
 'imdbVotes': '616,155'}
```

In [7]:
```python
# Print a few keys from the response JSON.
print(f"Movie was directed by {data['Director']}.")
print(f"Movie was released in {data['Country']}.")
```

```
Movie was directed by James Cameron.
Movie was released in USA, UK.
```

# OMDb API

* **Instructions:**

  * Read the OMDb documentation, and make a few API calls to
    get some information about your favorite movie: <http://www.omdbapi.com/>

- - -

## Copyright

In [1]:
```
# Dependencies
import requests
from config import api_key

url = f"http://www.omdbapi.com/?apikey={api_key}&t="
```

In [2]:
```
# Who was the director of the movie Aliens?
movie = requests.get(url + "Aliens").json()
print(f'The director of Aliens was {movie["Director"]}.')
```

```
The director of Aliens was James Cameron.
```

In [3]:
```
# What was the movie Gladiator rated?
movie = requests.get(url + "Gladiator").json()
print(f'The rating of Gladiator was {movie["Rated"]}.')
```

```
The rating of Gladiator was R.
```

In [4]:
```
# What year was 50 First Dates released?
movie = requests.get(url + "50 First Dates").json()
print(f'The movie 50 First Dates was released in {movie["Year"]}.')
```

```
The movie 50 First Dates was released in 2004.
```

In [5]:
```
# Who wrote Moana?
movie = requests.get(url + "Moana").json()
print(f'Moana was written by {movie["Writer"]}.')
```

```
Moana was written by Jared Bush (screenplay by), Ron Clements (story by), John Musker (story b
y), Chris Williams (story by), Don Hall (story by), Pamela Ribon (story by), Aaron Kandell (sto
ry by), Jordan Kandell (story by).
```

In [6]:
```
# What was the plot of the movie Sing?
movie = requests.get(url + "Sing").json()
print(f'The plot of Sing was: {movie["Plot"]}')
```

```
The plot of Sing was: In a city of humanoid animals, a hustling theater impresario's attempt to
save his theater with a singing competition becomes grander than he anticipates even as its fin
alists' find that their lives will never be the same.
```

In [1]:
```
# Dependencies
import random
import json
import requests
```

In [2]:
```
# Let's get the JSON for 100 posts sequentially.
url = "http://jsonplaceholder.typicode.com/posts/"
```

In [3]:
```
# Create an empty list to store the responses
response_json = []
```

In [4]:
```
# Create random indices representing
# a user's choice of posts
indices = random.sample(list(range(1, 100)), 10)
indices
```

Out[4]:
```
[97, 71, 83, 56, 2, 53, 67, 79, 35, 55]
```

In [5]:
```
# Make a request for each of the indices
for x in range(len(indices)):
    print(f"Making request number: {x} for ID: {indices[x]}")

    # Get one of the posts
    post_response = requests.get(url + str(indices[x]))

    # Save post's JSON
    response_json.append(post_response.json())
```

```
Making request number: 0 for ID: 97
Making request number: 1 for ID: 71
Making request number: 2 for ID: 83
Making request number: 3 for ID: 56
Making request number: 4 for ID: 2
Making request number: 5 for ID: 53
Making request number: 6 for ID: 67
Making request number: 7 for ID: 79
Making request number: 8 for ID: 35
Making request number: 9 for ID: 55
```

In [6]:
```
# Now we have 10 post objects,
# which we got by making 100 requests to the API.
print(f"We have {len(response_json)} posts!")
```

```
We have 10 posts!
```

In [7]:
```
response_json
```

Out[7]:
```
[{'body': 'eum non blanditiis soluta porro quibusdam voluptas\nvel voluptatem qui plac
eat dolores qui velit aut\nvel inventore aut cumque culpa explicabo aliquid at\nperspi
ciatis est et voluptatem dignissimos dolor itaque sit nam',
  'id': 97,
  'title': 'quas fugiat ut perspiciatis vero provident',
  'userId': 10},
 {'body': 'occaecati a doloribus\niste saepe consectetur placeat eum voluptate dolorem
et\nqui quo quia voluptas\nrerum ut id enim velit est perferendis',
  'id': 71,
  'title': 'et iusto veniam et illum aut fuga',
  'userId': 8},
 {'body': 'est molestiae facilis quis tempora numquam nihil qui\nvoluptate sapiente co
nsequatur est qui\nnecessitatibus autem aut ipsa aperiam modi dolore numquam\nreprehen
derit eius rem quibusdam',
  'id': 83,
  'title': 'odit et voluptates doloribus alias odio et',
  'userId': 9},
 {'body': 'aut est omnis dolores\nneque rerum quod ea rerum velit pariatur beatae exce
pturi\net provident voluptas corrupti\ncorporis harum reprehenderit dolores eligendi',
  'id': 56,
  'title': 'qui et at rerum necessitatibus',
  'userId': 6},
 {'body': 'est rerum tempore vitae\nsequi sint nihil reprehenderit dolor beatae ea dol
ores neque\nfugiat blanditiis voluptate porro vel nihil molestiae ut reiciendis\nqui a
periam non debitis possimus qui neque nisi nulla',
  'id': 2,
  'title': 'qui est esse',
  'userId': 1},
 {'body': 'minima harum praesentium eum rerum illo dolore\nquasi exercitationem rerum
nam\nporro quis neque quo\nconsequatur minus dolor quidem veritatis sunt non explicabo
similique',
  'id': 53,
  'title': 'ut quo aut ducimus alias',
  'userId': 6},
 {'body': 'reprehenderit id nostrum\nvoluptas doloremque pariatur sint et accusantium
quia quod aspernatur\net fugiat amet\nnon sapiente et consequatur necessitatibus moles
tiae',
  'id': 67,
  'title': 'aliquid eos sed fuga est maxime repellendus',
  'userId': 7},
 {'body': 'libero accusantium et et facere incidunt sit dolorem\nnon excepturi qui qui
a sed laudantium\nquisquam molestiae ducimus est\nofficiis esse molestiae iste et quo
s',
  'id': 79,
  'title': 'pariatur consequatur quia magnam autem omnis non amet',
  'userId': 8},
 {'body': 'nisi error delectus possimus ut eligendi vitae\nplaceat eos harum cupiditat
e facilis reprehenderit voluptatem beatae\nmodi ducimus quo illum voluptas eligendi\ne
t nobis quia fugit',
  'id': 35,
  'title': 'id nihil consequatur molestias animi provident',
  'userId': 4},
 {'body': 'debitis excepturi ea perferendis harum libero optio\neos accusamus cum fuga
ut sapiente repudiandae\net ut incidunt omnis molestiae\nnihil ut eum odit',
  'id': 55,
  'title': 'sit vel voluptatem et non libero',
  'userId': 6}]
```

In [1]:
```
# Dependencies
import requests

url = "http://www.omdbapi.com/?apikey=trilogy&t="

movies = ["Aliens", "Sing", "Moana"]
```

In [2]:
```
responses=[]
```

In [5]:
```
for movie in movies:
    movie_data=requests.get(url+movie).json()
    responses.append(movie_data)
    print(f'The director of {movie} is {movie_data["Director"]}')
```

```
The director of Aliens is James Cameron
The director of Sing is Garth Jennings, Christophe Lourdelet(co-director)
The director of Moana is Ron Clements, John Musker, Don Hall(co-director), Chris Williams
(co-director)
```

In [ ]:

In [1]:
```
# Dependencies
import requests
from pprint import pprint
from config import api_key

url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?"
api_key
```

Out[1]:
```
'WMqKOZaxVzKqIS7SnKCvAZuxXBE37aAr'
```

In [2]:
```
# Search for articles that mention granola
query = "granola"
```

In [3]:
```
# Build query URL
query_url = url + "api-key=" + api_key + "&q=" + query
query_url
```

Out[3]:
```
'https://api.nytimes.com/svc/search/v2/articlesearch.json?api-key=WMqKOZaxVzKqIS7SnKCvAZux
XBE37aAr&q=granola'
```

In [4]:
```
# Request articles
articles = requests.get(query_url).json()
articles
```

Out[4]:

```
{'status': 'OK',
 'copyright': 'Copyright (c) 2020 The New York Times Company. All Rights Reserved.',
 'response': {'docs': [{'abstract': '',
    'web_url': 'https://cooking.nytimes.com/recipes/1014414-granola',
    'snippet': '',
    'lead_paragraph': 'This somewhat clumpy granola isn’t too sweet, so you can feel a
little virtuous when you snack on it. It is best to eat it soon after baking, while the
clumps still hold; they will eventually fall apart once you put the granola in a jar, b
ut there are enough right after you bake it to qualify it as snacking food in addition
to breakfast. Don’t stir it when you bake it and when it is done let it cool completely
on the baking sheet; that is the trick to clumping. The technique of leaving a “donut h
ole” in the middle when you spread the granola on the baking sheet comes from www.brigh
teyedbaker.com. It helps the granola bake evenly. ',
    'multimedia': [{'rank': 0,
      'subtype': 'watch308',
      'caption': None,
      'credit': None,
      'type': 'image',
      'url': 'images/2012/12/17/health/19recipehealth/19recipehealth-watch308-v2.jpg',
```

In [5]:
```
# The "response" property in articles contains the actual articles
# list comprehension.
articles_list = [article for article in articles["response"]["docs"]]
pprint(articles_list)
```

```
[{'_id': 'nyt://recipe/29cf217b-d8d8-5f77-8559-5a02eb1b0b34',
  'abstract': '',
  'byline': {'organization': None,
             'original': 'Martha Rose Shulman',
             'person': [{'firstname': 'Martha',
                         'lastname': 'Shulman',
                         'middlename': 'Rose',
                         'organization': '',
                         'qualifier': None,
                         'rank': 1,
                         'role': 'reported',
                         'title': None}]},
  'document_type': 'recipe',
  'headline': {'content_kicker': None,
               'kicker': None,
               'main': 'Granola',
               'name': 'Granola',
               'print_headline': None,
               'seo': None,
```

In [6]:
```
# Print the web_url of each stored article
print("Your Reading List")
for article in articles_list:
    print(article["web_url"])
```

```
Your Reading List
https://cooking.nytimes.com/recipes/1014414-granola (https://cooking.nytimes.com/recipes/1
014414-granola)
https://www.nytimes.com/2012/03/25/magazine/who-made-that-granola.html (https://www.nytime
s.com/2012/03/25/magazine/who-made-that-granola.html)
https://cooking.nytimes.com/recipes/1014040-granola (https://cooking.nytimes.com/recipes/1
014040-granola)
https://www.nytimes.com/2016/05/11/dining/granola-recipe-clusters-video.html (https://www.
nytimes.com/2016/05/11/dining/granola-recipe-clusters-video.html)
https://cooking.nytimes.com/recipes/1822-granola-muffins (https://cooking.nytimes.com/reci
pes/1822-granola-muffins)
https://cooking.nytimes.com/recipes/1019514-granola-bites (https://cooking.nytimes.com/rec
ipes/1019514-granola-bites)
https://www.nytimes.com/1999/01/24/nyregion/they-stand-by-their-granola.html (https://www.
nytimes.com/1999/01/24/nyregion/they-stand-by-their-granola.html)
https://www.nytimes.com/2000/11/22/living/granola.html (https://www.nytimes.com/2000/11/2
2/living/granola.html)
https://cooking.nytimes.com/recipes/1012921-granola-muffins (https://cooking.nytimes.com/r
ecipes/1012921-granola-muffins)
https://www.nytimes.com/video/dining/1194817105861/making-granola.html (https://www.nytime
s.com/video/dining/1194817105861/making-granola.html)
```

In [7]:
```
# Dependencies
import requests
from config import api_key
import time

url = "https://api.nytimes.com/svc/search/v2/articlesearch.json?"

# Store a search term
query = "obama"

# Search for articles published between a begin and end date
begin_date = "20160101"
end_date = "20160130"

query_url = f"{url}api-key={api_key}&q={query}&begin_date={begin_date}&end_date={end_date}"
```

In [8]:
```
# Retrieve articles
articles = requests.get(query_url).json()
articles_list = articles["response"]["docs"]

for article in articles_list:
    print(f'A snippet from the article: {article["snippet"]}')
    print('---------------------------')
```

```
A snippet from the article: I will not campaign for, vote for or support any candidate, even in
my own party, who does not support common-sense gun reform.
---------------------------
A snippet from the article: In presidential elections, opposites attract.
---------------------------
A snippet from the article: Todos –gobierno, sector privado y los ciudadanos– debemos hacer un
frente común para acabar con esta violencia sin sentido.
---------------------------
A snippet from the article: Dire warnings from Republicans about the effect of President Obam
a’s policies on employment have simply not come true.
---------------------------
A snippet from the article: “Let me tell you, there are three things that are certain in life:
death, taxes, and Michelle is not running for president.” — President Obama
---------------------------
A snippet from the article: Highlights from President Obama’s remarks at a town-hall style meet
ing in Baton Rouge, La, on Thursday.
---------------------------
A snippet from the article: For many young people, the first president they consciously registe
red was a black man.
---------------------------
A snippet from the article: Readers mostly praise the president’s speech as hopeful and inclusi
ve.
---------------------------
A snippet from the article: Readers discuss the Supreme Court’s decision to hear a case challen
ging the president’s executive actions.
---------------------------
A snippet from the article: President Obama, speaking at the North American International Auto
Show in Detroit, expressed sympathy and concern for residents of Flint, Mich., where there has
been lead contamination in the drinking water.
---------------------------
```

In [9]:

```
# BONUS: How would we get 30 results?
# HINT: Look up the page query param

# Empty list for articles
articles_list = []

# loop through pages 0-2
for page in range(0, 3):
    query_url = f"{url}api-key={api_key}&q={query}&begin_date={begin_date}&end_date={end_date}"
    # create query with page number
    query_url = f"{query_url}&page={str(page)}"
    articles = requests.get(query_url).json()

    # Add a one second interval between queries to stay within API query limits
    time.sleep(1)
    # loop through the response and append each article to the list
    for article in articles["response"]["docs"]:
        articles_list.append(article)
```

In [11]:
```
for article in articles_list:
    print(article['snippet'])
    print('--------------------------')
```

```
I will not campaign for, vote for or support any candidate, even in my own party, who does not
support common-sense gun reform.
--------------------------
In presidential elections, opposites attract.
--------------------------
Todos –gobierno, sector privado y los ciudadanos– debemos hacer un frente común para acabar con
esta violencia sin sentido.
--------------------------
Dire warnings from Republicans about the effect of President Obama’s policies on employment hav
e simply not come true.
--------------------------
“Let me tell you, there are three things that are certain in life: death, taxes, and Michelle i
s not running for president.” — President Obama
--------------------------
Highlights from President Obama’s remarks at a town-hall style meeting in Baton Rouge, La, on T
hursday.
--------------------------
For many young people, the first president they consciously registered was a black man.
--------------------------
Readers mostly praise the president’s speech as hopeful and inclusive.
--------------------------
Readers discuss the Supreme Court’s decision to hear a case challenging the president’s executi
ve actions.
--------------------------
President Obama, speaking at the North American International Auto Show in Detroit, expressed s
ympathy and concern for residents of Flint, Mich., where there has been lead contamination in t
he drinking water.
--------------------------
President Obama expressed relief and happiness that several Americans who had been detained in
Iran were returning home after the completion of a nuclear agreement with Tehran.
--------------------------
Just hours before his final State of the Union address, President Obama used Facebook Live to t
alk to users of the social network about the speech.
--------------------------
To function properly, the clemency process needs to be removed from the grasp of the Justice De
partment.
--------------------------
President Barack Obama returned to Washington after spending two weeks in Hawaii.
--------------------------
President Obama signed a presidential memorandum creating a White House task force on cancer.
--------------------------
The practice can lead to “devastating, lasting psychological consequences,” he said in an opini
on article in The Washington Post.
--------------------------
President Obama hosted the emir of Qatar, Sheikh Tamim bin Hamad al-Thani, on Tuesday, and both
said they were committed to defeating the Islamic State and other terrorist organizations.
--------------------------
The president seeks $4 billion to help states expand in an area he views as critical to young p
eople’s success in a changing job market.
--------------------------
On policy, the president has been remarkable, but on changing hearts and minds, he falls short.
--------------------------
Mr. Obama delivered his final State of the Union address on Tuesday.
--------------------------
Instead of listing initiatives that are likely to fail, the president will use his last State o
f the Union address to focus on American potential and the need to take on long-term challenge
s, aides said.
--------------------------
The president has said he will seek more encounters in his last year with people who disagree w
ith him, though recent events have drawn supportive crowds.
--------------------------
The president delivered his final State of the Union address on Tuesday and discussed the threa
t posed by the Islamic State.
--------------------------
```

```
The president delivered his final State of the Union address on Tuesday and discussed American
innovation.
---------------------------
I was live-tweeting President Obama’s State of the Union Address. For those of you not on Twitt
er, here’s what I had to say.
---------------------------
The president delivered his final State of the Union address on Tuesday and discussed economic
growth.
---------------------------
In his final State of the Union message, the president urged Americans to confront the challeng
es of the future by rejecting fear and embracing change.
---------------------------
Readers respond to an Op-Ed essay by the president calling for the country to join together to
fight the gun crisis.
---------------------------
The administration will overhaul its response to online propaganda from the Islamic State after
acknowledging it had largely failed to counter extremists on social media.
---------------------------
President Obama is the first president to publish visitor logs of the journalists who attend of
f-the-record discussions meant to promote context and candor.
---------------------------
```

```
In [ ]:
```

In [1]:
```
# Dependencies
import json
import os

# Load JSON
filepath = os.path.join("..", "Resources", "youtube_response.json")
with open(filepath) as jsonfile:
    json_data = json.load(jsonfile)
```

In [8]:
```
#preview data
print(json.dumps(json_data, indent=4, sort_keys=True))
```

```
{
    "apiVersion": "2.0",
    "data": {
        "items": [
            {
                "accessControl": {
                    "comment": "allowed",
                    "commentVote": "allowed",
                    "embed": "allowed",
                    "list": "allowed",
                    "rate": "allowed",
                    "syndicate": "allowed",
                    "videoRespond": "moderated"
                },
                "aspectRatio": "widescreen",
                "category": "News",
                "commentCount": 22,
                "content": {
                    "1": "rtsp://v5.cache3.c.youtube.com/CiILENy.../0/0/0/video.3gp",
                    "5": "http://www.youtube.com/v/hYB0mn5zh2c?f...",
                    "6": "rtsp://v1.cache1.c.youtube.com/CiILENy.../0/0/0/video.3gp"
                },
                "description": "Google Maps API Introduction ...",
                "duration": 2840,
                "favoriteCount": 201,
                "id": "hYB0mn5zh2c",
                "player": {
                    "default": "http://www.youtube.com/watch?vu003dhYB0mn5zh2c"
                },
                "rating": 4.63,
                "ratingCount": 68,
                "status": {
                    "reason": "limitedSyndication",
                    "value": "restricted"
                },
                "tags": [
                    "GDD07",
                    "GDD07US",
                    "Maps"
                ],
                "thumbnail": {
                    "default": "http://i.ytimg.com/vi/hYB0mn5zh2c/default.jpg",
                    "hqDefault": "http://i.ytimg.com/vi/hYB0mn5zh2c/hqdefault.jpg"
                },
                "title": "Google Developers Day US - Maps API Introduction",
                "updated": "2010-01-07T13:26:50.000Z",
                "uploaded": "2007-06-05T22:07:03.000Z",
                "uploader": "GoogleDeveloperDay",
                "viewCount": 220101
            }
        ],
        "itemsPerPage": 1,
        "startIndex": 1,
        "totalItems": 800,
        "updated": "2010-01-07T19:58:42.949Z"
    }
}
```

In [13]:
```
items=json_data['data']['items'][0]
items['title']
print('Title:',items['title'])
```

```
Title: Google Developers Day US - Maps API Introduction
```

In [14]:
```
items['thumbnail']
print('Title:',items['thumbnail'])
```

```
Title: {'default': 'http://i.ytimg.com/vi/hYB0mn5zh2c/default.jpg', 'hqDefault': 'http://i.ytim
g.com/vi/hYB0mn5zh2c/hqdefault.jpg'}
```

In [ ]:
```

```

In [7]:
```python
# Dependencies
import json
import requests
from pprint import pprint
```

In [8]:
```python
# Specify the URL
url="http://nyt-mongo-scraper.herokuapp.com/api/headlines"
# Make request and store response
response=requests.get(url)
```

In [9]:
```python
# JSON-ify response
response_json=response.json()
```

In [12]:
```python
# Print first and last articles
pprint(response_json)
```

```
[{'__v': 0,
  '_id': '5e1a0eb759cff600159aee17',
  'date': '2020-01-11T18:06:47.966Z',
  'headline': 'Current Job: Award-Winning Chef. Education: University of IHOP.',
  'saved': False,
  'summary': 'American chain restaurants don’t reap much critical praise, but '
             'many high-end chefs say they got a priceless, practical '
             'education there.',
  'url': 'https://www.nytimes.com/2020/01/07/dining/chef-chain-restaurant.html'},
 {'__v': 0,
  '_id': '5e1a0eb759cff600159aee16',
  'date': '2020-01-11T18:06:47.965Z',
  'headline': '‘Techlash’ Hits College Campuses',
  'saved': False,
  'summary': 'Facebook, Google and other major tech firms were every student’s '
             'dream workplaces. Until they weren’t.',
  'url': 'https://www.nytimes.com/2020/01/11/style/college-tech-recruiting.html'},
 {'__v': 0,
  '_id': '5e1a0eb759cff600159aee15',
```

In [14]:
```python
pprint(response_json[0])
```

```
{'__v': 0,
 '_id': '5e1a0eb759cff600159aee17',
 'date': '2020-01-11T18:06:47.966Z',
 'headline': 'Current Job: Award-Winning Chef. Education: University of IHOP.',
 'saved': False,
 'summary': 'American chain restaurants don’t reap much critical praise, but '
            'many high-end chefs say they got a priceless, practical education '
            'there.',
 'url': 'https://www.nytimes.com/2020/01/07/dining/chef-chain-restaurant.html'}
```

In [15]:
```python
#print the last article
#-1 gives you the last object (counting from the bottom)
pprint(response_json[-1])
```

```
{'__v': 0,
 '_id': '5e1a0eb759cff600159aee0f',
 'date': '2020-01-11T18:06:47.963Z',
 'headline': 'At 16, She’s a Pioneer in the Fight to Cure Sickle Cell Disease',
 'saved': False,
 'summary': 'Helen Obando is the youngest person ever to get a gene therapy '
            'that scientists hope will cure the disease, which afflicts '
            '100,000 Americans.',
 'url': 'https://www.nytimes.com/2020/01/11/health/sickle-cell-disease-cure.html'}
```

In [ ]:

```
#Print the number of responses received.
count_articles=len(response_json)
count_articles
```

In [1]:
```
# Dependencies
import json
import requests
from config import api_key
```

In [4]:
```
# Save config information
url = "http://api.openweathermap.org/data/2.5/weather?"
city = "London"

# Build query URL
query_url= url+"appid="+api_key+"&q="+city
query_url
```

Out[4]:
```
'http://api.openweathermap.org/data/2.5/weather?appid=92ffc2db7b186a083bb128cf65dd2c57&q=London'
```

In [5]:
```
# Get weather data
weather_response=requests.get(query_url)
weather_json=weather_response.json()

# Get the temperature from the response
print(f'the weather API responded with: {weather_json}.')
```

```
the weather API responded with: {'coord': {'lon': -0.13, 'lat': 51.51}, 'weather': [{'id': 804,
'main': 'Clouds', 'description': 'overcast clouds', 'icon': '04n'}], 'base': 'stations', 'main':
{'temp': 282.97, 'feels_like': 278.02, 'temp_min': 282.04, 'temp_max': 284.15, 'pressure': 1022,
'humidity': 76}, 'visibility': 10000, 'wind': {'speed': 5.7, 'deg': 230}, 'clouds': {'all': 90},
'dt': 1578768282, 'sys': {'type': 1, 'id': 1414, 'country': 'GB', 'sunrise': 1578729764, 'sunse
t': 1578759192}, 'timezone': 0, 'id': 2643743, 'name': 'London', 'cod': 200}.
```

In [ ]:
```
```

In [2]:
```
# Dependencies
import requests
from config import api_key
import json
```

In [3]:
```
# Build query URL and request your results in Celsius
url = "http://api.openweathermap.org/data/2.5/weather?"
city = "Burundi"
query_url= url+"appid="+api_key+"&q="+city+"&units=metric"
query_url
```

Out[3]:
```
'http://api.openweathermap.org/data/2.5/weather?appid=92ffc2db7b186a083bb128cf65dd2c57&q=Bu
rundi&units=metric'
```

In [7]:
```
# Get weather data
weather_response=requests.get(query_url)
weather_json=weather_response.json()
```

In [13]:
```
# Get temperature from JSON response
#print(f'The weather API responded with:{weather_json}.') or:
weather_json
```

Out[13]:

```
{'coord': {'lon': 30, 'lat': -3.5},
 'weather': [{'id': 501,
   'main': 'Rain',
   'description': 'moderate rain',
   'icon': '10n'}],
 'base': 'model',
 'main': {'temp': 63.12,
  'feels_like': 64.36,
  'temp_min': 63.12,
  'temp_max': 63.12,
  'pressure': 1015,
  'humidity': 90,
  'sea_level': 1015,
  'grnd_level': 836},
 'wind': {'speed': 3.71, 'deg': 198},
 'rain': {'3h': 5.19},
 'clouds': {'all': 84},
 'dt': 1578770190,
 'sys': {'country': 'BI', 'sunrise': 1578715092, 'sunset': 1578759398},
 'timezone': 7200,
 'id': 433561,
 'name': 'Burundi',
 'cod': 200}
```

In [12]:
```
# Report temperature
temp=weather_json['main']['temp']
print(f'The temperature is {temp} Celcius.')
```

```
The temperature is 63.12C.
```

In [14]:
```
#temp in imperial
query_url= url+"appid="+api_key+"&q="+city+"&units=imperial"
weather_response=requests.get(query_url)
weather_json=weather_response.json()
temp=weather_json['main']['temp']
print(f'The temperature is {temp} Farenheit.')
```

```
The temperature is 63.12 Farenheit.
```

In [1]:
```
# Dependencies
import requests
from config import api_key

# Save config information.
url = "http://api.openweathermap.org/data/2.5/weather?"
city = "Bujumbura"
units = "metric"
```

In [2]:
```
# Build query URL and request your results in Celsius
query_url = f"{url}appid={api_key}&q={city}&units={units}"

# Get weather data
weather_response = requests.get(query_url)
weather_json = weather_response.json()
```

In [3]:
```
# Get temperature from JSON response
temperature = weather_json["main"]["temp"]
```

In [4]:
```
# Report temperature
print(f"The temperature in Bujumbura is {temperature} C.")
```

```
The temperature in Bujumbura is 16.81 C.
```

In [5]:
```
# BONUS

# use list of units
units = ["metric", "imperial"]

# set up list to hold two different temperatures
temperatures = []

# loop throught the list of units and append them to temperatures list
for unit in units:
    # Build query URL based on current element in units
    query_url = url + "appid=" + api_key + "&q=" + city + "&units=" + unit

    # Get weather data
    weather_response = requests.get(query_url)
    weather_json = weather_response.json()

    # Get temperature from JSON response
    temperature = weather_json["main"]["temp"]

    temperatures.append(temperature)

# Report temperatures by accessing each element in the list
print(
    f"The temperature in Bujumbura is {temperatures[0]}C or {temperatures[1]}F.")
```

```
The temperature in Bujumbura is 16.81C or 62.27F.
```

In [1]:
```
# Dependencies
import csv
import matplotlib.pyplot as plt
import requests
import pandas as pd
from config import api_key
```

In [2]:
```
# Save config information.
url = "http://api.openweathermap.org/data/2.5/weather?"
units = "metric"

# Build partial query URL
query_url = f"{url}appid={api_key}&units={units}&q="
```

In [3]:
```
cities = ["Paris", "London", "Oslo", "Beijing"]

# set up lists to hold reponse info
lat = []
temp = []

# Loop through the list of cities and perform a request for data on each
for city in cities:
    response = requests.get(query_url + city).json()
    lat.append(response['coord']['lat'])
    temp.append(response['main']['temp'])

print(f"The latitude information received is: {lat}")
print(f"The temperature information received is: {temp}")
```

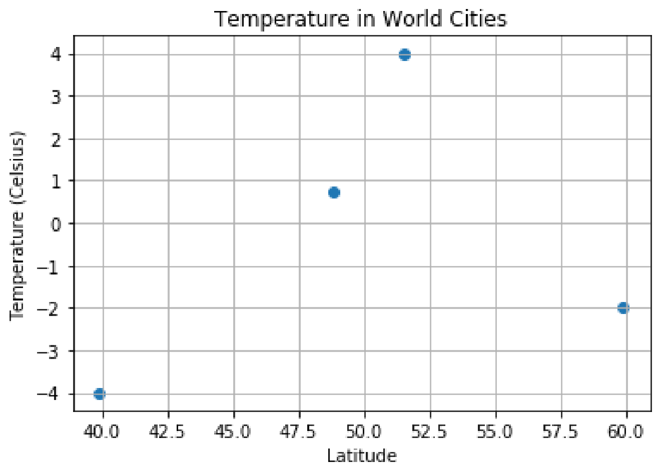
```
The latitude information received is: [48.86, 51.51, 59.91, 39.91]
The temperature information received is: [0.75, 4, -2, -4]
```

In [4]:
```
# create a data frame from cities, lat, and temp
weather_dict = {
    "city": cities,
    "lat": lat,
    "temp": temp
}
weather_data = pd.DataFrame(weather_dict)
weather_data.head()
```

Out[4]:

	city	lat	temp
0	Paris	48.86	0.75
1	London	51.51	4.00
2	Oslo	59.91	-2.00
3	Beijing	39.91	-4.00

In [5]:
```
# Build a scatter plot for each data type
plt.scatter(weather_data["lat"], weather_data["temp"], marker="o")

# Incorporate the other graph properties
plt.title("Temperature in World Cities")
plt.ylabel("Temperature (Celsius)")
plt.xlabel("Latitude")
plt.grid(True)

# Save the figure
plt.savefig("TemperatureInWorldCities.png")

# Show plot
plt.show()
```



In [1]:
```
#Dependencies
import requests
import json
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
```

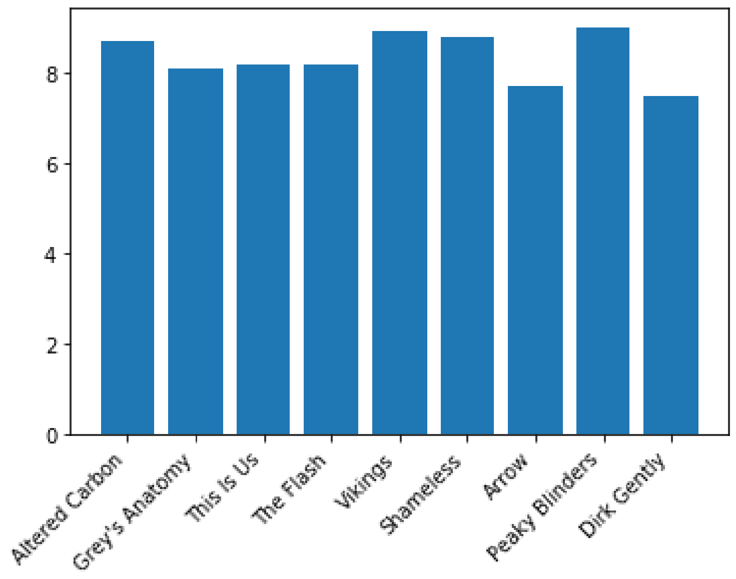
In [2]:
```
# list of tv show titles to query
tv_shows = ["Altered Carbon", "Grey's Anatomy", "This is Us", "The Flash",
            "Vikings", "Shameless", "Arrow", "Peaky Blinders", "Dirk Gently"]

# tv maze show search base url
base_url = "http://api.tvmaze.com/search/shows?q="

# set up lists to hold response data for name and rating
titles = []
ratings = []
networks = []

# loop through tv show titles, make requests and parse
for show in tv_shows:
    target_url = base_url + show
    response = requests.get(target_url).json()
    titles.append(response[0]['show']['name'])
    ratings.append(response[0]['show']['rating']['average'])
```

In [3]:
```
# create dataframe
shows_df = pd.DataFrame({
    "title": titles,
    "rating": ratings
})

shows_df
```

Out[3]:

	rating	title
0	8.7	Altered Carbon
1	8.1	Grey's Anatomy
2	8.2	This Is Us
3	8.2	The Flash
4	8.9	Vikings
5	8.8	Shameless
6	7.7	Arrow
7	9.0	Peaky Blinders
8	7.5	Dirk Gently

In [4]:
```
# create a list of numbers for x values
tick_locations = np.arange(len(shows_df))

# create bar chart and set the values of xticks
plt.bar(tick_locations, shows_df['rating'], align="center")
plt.xticks(tick_locations, shows_df['title'], rotation=45, ha="right")

plt.savefig("tv_show_ratings.png")
plt.show()
```



In [1]:
```
# Dependencies
import csv
import matplotlib.pyplot as plt
import requests
from scipy import stats
import pandas as pd
from config import api_key
```

In [2]:
```
# Save config information.
url = "http://api.openweathermap.org/data/2.5/weather?"
units = "metric"

# Build partial query URL
query_url = f"{url}appid={api_key}&units={units}&q="
```

In [3]:
```
cities = ["Paris", "London", "Oslo", "Beijing", "Mumbai", "Manila", "New York", "Seattle", "Dallas", "Taiwan"]

# set up lists to hold reponse info
lat = []
temp = []

# Loop through the list of cities and perform a request for data on each
for city in cities:
    response = requests.get(query_url + city).json()
    lat.append(response['coord']['lat'])
    temp.append(response['main']['temp'])

print(f"The latitude information received is: {lat}")
print(f"The temperature information received is: {temp}")
```

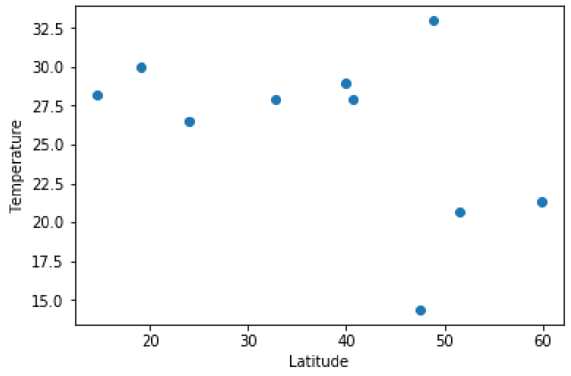
```
The latitude information received is: [48.86, 51.51, 59.91, 39.91, 19.01, 14.59, 40.73, 47.6, 32.78, 24]
The temperature information received is: [32.97, 20.65, 21.33, 28.94, 30, 28.21, 27.96, 14.38, 27.95, 26.52]
```

In [4]:
```
# create a data frame from cities, lat, and temp
weather_dict = {
    "city": cities,
    "lat": lat,
    "temp": temp
}
weather_data = pd.DataFrame(weather_dict)
weather_data
```

Out[4]:

	city	lat	temp
0	Paris	48.86	32.97
1	London	51.51	20.65
2	Oslo	59.91	21.33
3	Beijing	39.91	28.94
4	Mumbai	19.01	30.00
5	Manila	14.59	28.21
6	New York	40.73	27.96
7	Seattle	47.60	14.38
8	Dallas	32.78	27.95
9	Taiwan	24.00	26.52

In [5]:
```
# Create a Scatter Plot for temperature vs latitude
x_values = weather_data['lat']
y_values = weather_data['temp']
plt.scatter(x_values,y_values)
plt.xlabel('Latitude')
plt.ylabel('Temperature')
plt.show()
```



In [6]:
```
# Perform a linear regression on temperature vs. latitude
(slope, intercept, rvalue, pvalue, stderr) = stats.linregress(x_values, y_values)

# Get regression values
regress_values = x_values * slope + intercept
print(regress_values)
```

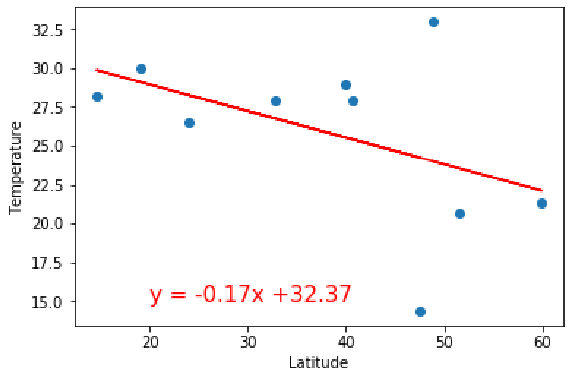
```
0    24.014549
1    23.561259
2    22.124414
3    25.545473
4    29.120480
5    29.876534
6    25.405210
7    24.230076
8    26.765081
9    28.266925
Name: lat, dtype: float64
```

In [7]:
```
# Create line equation string
line_eq = "y = " + str(round(slope,2)) + "x +" + str(round(intercept,2))
print(line_eq)
```

```
y = -0.17x +32.37
```

In [8]:
```python
# Create Plot
plt.scatter(x_values,y_values)
plt.plot(x_values,regress_values,"r-")

# Label plot and annotate the line equation
plt.xlabel('Latitude')
plt.ylabel('Temperature')
plt.annotate(line_eq,(20,15),fontsize=15,color="red")

# Print r square value
print(f"The r-squared is: {rvalue}")

# Show plot
plt.show()
```

```
The r-squared is: -0.46618011693090955
```



In [11]:
```python
# Calculate the temperature for Florence at 34.8
florence_lat = 34.8
florence_predicted_temp = round(slope * florence_lat + intercept,2)

print(f"The Predicted temperature for Florence will be {florence_predicted_temp}.")
```

```
The Predicted temperature for Florence will be 26.42.
```

In [12]:
```python
# Use API to determine actual temperature
response = requests.get(query_url + "Florence").json()
florence_actual_temp = response['main']['temp']

print(f"The actual temperature of Florence is {florence_actual_temp}")
```

```
The actual temperature of Florence is 27.27
```

In [ ]:

In [1]:
```
students = {
    # Name  : Age
    "James": 27,
    "Sarah": 19,
    "Jocelyn": 28
}

print(students["Jezebel"])

print("This line will never print.")
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-1-4692324b5d88> in <module>()
      6 }
      7
----> 8 print(students["Jezebel"])
      9
     10 print("This line will never print.")

KeyError: 'Jezebel'
```

In [1]:
```
students = {
    # Name  : Age
    "James": 27,
    "Sarah": 19,
    "Jocelyn": 28
}

print(students["James"])

print("This line will never print.")
```

```
27
This line will never print.
```

In [ ]:

In [1]:

```
students = {
    # Name  : Age
    "James": 27,
    "Sarah": 19,
    "Jocelyn": 28
}

# Try to access key that doesn't exist
try:
    students["Jezebel"]
except KeyError:
    print("Oops, that key doesn't exist.")

# "Catching" the error lets the rest of our code execute
print("...But the program doesn't die early!")
#KeyError catches a specific error
```

```
Oops, that key doesn't exist.
...But the program doesn't die early!
```

In [2]:

```
students = {
    # Name  : Age
    "James": 27,
    "Sarah": 19,
    "Jocelyn": 28
}

# Try to access key that doesn't exist
#try:
    students["Jezebel"]
#except KeyError:
#    print("Oops, that key doesn't exist.")

# "Catching" the error lets the rest of our code execute
print("...But the program doesn't die early!")
```

```
  File "<ipython-input-2-6a5389f4f9e6>", line 10
    students["Jezebel"]
    ^
IndentationError: unexpected indent
```

In [ ]:

In [7]:
```
# Your assignment is to get the last line to print without changing any
# of the code below. Instead, wrap each line that throws an error in a
# try/except block.
print("Infinity looks like + " + str(10 / 0) + ".")

print("I think her name was + " + name + "?")

print("Your name is a nonsense number. Look: " + int("Gabriel"))

print("You made it through the gauntlet--the message survived!")
```

```
---------------------------------------------------------------------------
ZeroDivisionError                         Traceback (most recent call last)
<ipython-input-7-8ee39ec469f1> in <module>
      2 # of the code below. Instead, wrap each line that throws an error i
n a
      3 # try/except block.
----> 4 print("Infinity looks like + " + str(10 / 0) + ".")
      5 
      6 print("I think her name was + " + name + "?")

ZeroDivisionError: division by zero
```

In [8]:
```
try:
    print("Infinity looks like + " + str(10 / 0) + ".")
except ZeroDivisionError:
    print("Don't divide by zero")
```

```
Don't divide by zero
```

In [9]:
```
print("I think her name was + " + name + "?")
```

```
---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-9-fc4e8e0642ed> in <module>
----> 1 print("I think her name was + " + name + "?")

NameError: name 'name' is not defined
```

In [10]:
```
try:
    print("I think her name was + " + name + "?")
except NameError:
    print("Define a name")
```

```
Define a name
```

In [11]:
```python
print("Your name is a nonsense number. Look: " + int("Gabriel"))
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-11-3dc196b6b737> in <module>
----> 1 print("Your name is a nonsense number. Look: " + int("Gabriel"))

ValueError: invalid literal for int() with base 10: 'Gabriel'
```

In [12]:
```python
try:
    print("Your name is a nonsense number. Look: " + int("Gabriel"))
except ValueError:
    print("you have entered an undefined value")
```

```
you have entered an undefined value
```

In [13]:
```python
print("You made it through the gauntlet--the message survived!")
```

```
You made it through the gauntlet--the message survived!
```

In [16]:
```python
try:
    print("Infinity looks like + " + str(10 / 0) + ".")
except ZeroDivisionError as err:
    print("Don't divide by zero: Error:",err)

try:
    print("I think her name was + " + name + "?")
except NameError:
    print("Define a name")

try:
    print("Your name is a nonsense number. Look: " + int("Gabriel"))
except ValueError:
    print("you have entered an undefined value")

print("You made it through the gauntlet--the message survived!")
```

```
Don't divide by zero: Error: division by zero
Define a name
you have entered an undefined value
You made it through the gauntlet--the message survived!
```

In [ ]:
```python

```

In [1]:
```
import json
import requests
import pandas as pd
```

In [2]:
```
# List of character
search_characters = ['R2-D2', 'Darth Vader', 'Godzilla', 'Luke Skywalker', 'Frodo', \
              'Boba Fett', 'Iron Man', 'Jon Snow', 'Han Solo']

# Set url for API
url = 'https://swapi.co/api/people/?search='

# Set empty lists to hold characters height and mass
height = []
mass = []
starwars_characters = []

# Loop through each character
for character in search_characters:

    # Create search query, make request and store in json
    query = url + character
    response = requests.get(query)
    response_json = response.json()

    # Try to grab the height and mass of characters if they are available in the Star Wars API
    try:
        height.append(response_json['results'][0]['height'])
        mass.append(response_json['results'][0]['mass'])
        starwars_characters.append(character)
        print(f"{character} found! Appending stats")

    # Handle exceptions for a character that is not available in the Star Wars API
    except:
        # Append null values
        print("Character not found")
        pass
```

```
R2-D2 found! Appending stats
Darth Vader found! Appending stats
Character not found
Luke Skywalker found! Appending stats
Character not found
Boba Fett found! Appending stats
Character not found
Character not found
Han Solo found! Appending stats
```

In [3]:
```
# Create DataFrame
character_height = pd.DataFrame({
    'character': starwars_characters,
    'height': height,
    'mass': mass
})
character_height
```

Out[3]:

	character	height	mass
0	R2-D2	96	32
1	Darth Vader	202	136
2	Luke Skywalker	172	77
3	Boba Fett	183	78.2
4	Han Solo	180	80

In [1]:

```
# Dependencies
import requests

url = "http://api.worldbank.org/v2/"
format = "json"

# Get country information in JSON format
countries_response = requests.get(f"{url}countries?format={format}").json()

# First element is general information, second is countries themselves
countries = countries_response[1]
```

In [2]:

```python
# Report the names
for country in countries:
    print(country["name"])
```

```
Aruba
Afghanistan
Africa
Angola
Albania
Andorra
Andean Region
Arab World
United Arab Emirates
Argentina
Armenia
American Samoa
Antigua and Barbuda
Australia
Austria
Azerbaijan
Burundi
East Asia & Pacific (IBRD-only countries)
Europe & Central Asia (IBRD-only countries)
Belgium
Benin
Burkina Faso
Bangladesh
Bulgaria
IBRD countries classified as high income
Bahrain
Bahamas, The
Bosnia and Herzegovina
Latin America & the Caribbean (IBRD-only countries)
Belarus
Belize
Middle East & North Africa (IBRD-only countries)
Bermuda
Bolivia
Brazil
Barbados
Brunei Darussalam
Sub-Saharan Africa (IBRD-only countries)
Bhutan
Botswana
Sub-Saharan Africa (IFC classification)
Central African Republic
Canada
East Asia and the Pacific (IFC classification)
Central Europe and the Baltics
Europe and Central Asia (IFC classification)
Switzerland
Channel Islands
Chile
China
```

In [1]:
```
# Dependencies
import requests

url = "http://api.worldbank.org/v2/"
```

In [5]:
```
# Get the list of lending types the world bank has
lending_response=requests.get(f'{url}lendingTypes?format=json').json()
#lending_response
lending_types=[lending_type['id'] for lending_type in lending_response[1]]
lending_types
```

Out[5]:
```
['IBD', 'IDB', 'IDX', 'LNX']
```

In [12]:
```
# Next, determine how many countries fall into each lending type.
# Hint: Look at the first element of the response array.
country_count_by_type={}
for lending_type in lending_types:
    query=f"{url}countries?lendingType={lending_type}&format=json"
    #print(query)
    response=requests.get(query).json()
    country_count_by_type[lending_type]=response[0]['total']
```

In [13]:
```
# Print the number of countries of each lending type
for key,value in country_count_by_type.items():
    print(f'The number of countries with lending type {key} is {value}')
```

```
The number of countries with lending type IBD is 68
The number of countries with lending type IDB is 17
The number of countries with lending type IDX is 59
The number of countries with lending type LNX is 74
```

In [ ]:

In [1]:

```
# Dependencies
import requests
import json

# Google developer API key
from config import gkey

# Target city
target_city = "Boise, Idaho"

# Build the endpoint URL
target_url = ('https://maps.googleapis.com/maps/api/geocode/json?'
    'address={0}&key={1}').format(target_city, gkey)
```

In [2]:

```
# Run a request to endpoint and convert result to json
geo_data = requests.get(target_url).json()

# Print the json
print(geo_data)
```

```
{'results': [{'address_components': [{'long_name': 'Boise', 'short_name': 'Boise', 'ty
pes': ['locality', 'political']}, {'long_name': 'Ada County', 'short_name': 'Ada Count
y', 'types': ['administrative_area_level_2', 'political']}, {'long_name': 'Idaho', 'sh
ort_name': 'ID', 'types': ['administrative_area_level_1', 'political']}, {'long_name':
'United States', 'short_name': 'US', 'types': ['country', 'political']}], 'formatted_a
ddress': 'Boise, ID, USA', 'geometry': {'bounds': {'northeast': {'lat': 43.6898951, 'l
ng': -116.1019091}, 'southwest': {'lat': 43.511717, 'lng': -116.3658869}}, 'location':
{'lat': 43.6150186, 'lng': -116.2023137}, 'location_type': 'APPROXIMATE', 'viewport':
{'northeast': {'lat': 43.6898951, 'lng': -116.1019091}, 'southwest': {'lat': 43.51171
7, 'lng': -116.3658869}}}, 'place_id': 'ChIJnbRH6XLxrlQRm51nNpuYW5o', 'types': ['local
ity', 'political']}], 'status': 'OK'}
```

In [3]:
```
# Print the json (pretty printed)
print(json.dumps(geo_data, indent=4, sort_keys=True))
```

```
{
    "results": [
        {
            "address_components": [
                {
                    "long_name": "Boise",
                    "short_name": "Boise",
                    "types": [
                        "locality",
                        "political"
                    ]
                },
                {
                    "long_name": "Ada County",
                    "short_name": "Ada County",
                    "types": [
                        "administrative_area_level_2",
                        "political"
                    ]
                },
                {
                    "long_name": "Idaho",
                    "short_name": "ID",
                    "types": [
                        "administrative_area_level_1",
                        "political"
                    ]
                },
                {
                    "long_name": "United States",
                    "short_name": "US",
                    "types": [
                        "country",
                        "political"
                    ]
                }
            ],
            "formatted_address": "Boise, ID, USA",
            "geometry": {
                "bounds": {
                    "northeast": {
                        "lat": 43.6898951,
                        "lng": -116.1019091
                    },
                    "southwest": {
                        "lat": 43.511717,
                        "lng": -116.3658869
                    }
                },
                "location": {
                    "lat": 43.6150186,
                    "lng": -116.2023137
                },
                "location_type": "APPROXIMATE",
                "viewport": {
                    "northeast": {
                        "lat": 43.6898951,
                        "lng": -116.1019091
                    },
                    "southwest": {
```

```
                    "lat": 43.511717,
                    "lng": -116.3658869
                }
            }
        },
        "place_id": "ChIJnbRH6XLxrlQRm51nNpuYW5o",
        "types": [
            "locality",
            "political"
        ]
    }
],
"status": "OK"
}
```

In [4]:
```
# Extract latitude and longitude
lat = geo_data["results"][0]["geometry"]["location"]["lat"]
lng = geo_data["results"][0]["geometry"]["location"]["lng"]

# Print the latitude and longitude
print('''
    City: {0}
    Latitude: {1}
    Longitude: {2}
    '''.format(target_city, lat, lng))
```

```
    City: Boise, Idaho
    Latitude: 43.6150186
    Longitude: -116.2023137
```

In [ ]:

In [1]:
```python
# Dependencies
import requests
import json

# Google developer API key
from config import gkey
```

In [2]:
```python
# geocoordinates
target_coordinates = "43.6187102, -116.2146068"
target_search = "Chinese"
target_radius = 8000
target_type = "restaurant"

# set up a parameters dictionary
params = {
    "location": target_coordinates,
    "keyword": target_search,
    "radius": target_radius,
    "type": target_type,
    "key": gkey
}

# base url
base_url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"

# run a request using our params dictionary
response = requests.get(base_url, params=params)
```

In [3]:
```python
# print the response url, avoid doing for public github repos in order to avoid exposing key
print(response.url)
```

```
https://maps.googleapis.com/maps/api/place/nearbysearch/json?location=43.6187102%2C+-116.214
6068&keyword=Chinese&radius=8000&type=restaurant&key=YOUR+KEY+HERE%21 (https://maps.googleap
is.com/maps/api/place/nearbysearch/json?location=43.6187102%2C+-116.2146068&keyword=Chinese&
radius=8000&type=restaurant&key=YOUR+KEY+HERE%21)
```

In [4]:
```python
# convert response to json
places_data = response.json()

# Print the json (pretty printed)
print(json.dumps(places_data, indent=4, sort_keys=True))
```

```
{
    "html_attributions": [],
    "next_page_token": "CrQCLgEAAGp0bMqa7dnn1e3ZlwJhHasJUf0_LQ0wikvTQu3IYaVkVFiur16EOpWyif
fJ83GWPdFBCTa25YnEhXRUQq3uqB1cg9GtyIkEZyaTudjfZhdbEExJDK6HUFidqRcHf00RgZznsItxTDq343R5Lm40
V_RoTp5imJ82pXzs0Vh_7bwAr28xNAl25kdW1iCtOdn1tHQFxqoFiOVysjJ1Bgmn9fAuTrr3d2BGQVRvz-Yez3arTv
z3UOltkMydr7t4KlzA8OWBtKHW-Rxefb04TvD9s_3QpR-Jb-Bp-fu_87oMGgJGexcjvr4jk4v5ucHEY8ZpKWajhFUo
mhEm86iYlwLp7zJ1hTS_Yc5wN69hZxSQho8V1D7El2WMzWwh8yGcCq3glNNfKcna6G-os6ShysAvuYESEPON0nsRd0
MvY3KDjJ9sd90aFJhRXYzc2l4ZHWyKDyiUvChU13nt",
    "results": [
        {
            "geometry": {
                "location": {
                    "lat": 43.6201152,
                    "lng": -116.312547
                },
                "viewport": {
                    "northeast": {
                        "lat": 43.62114217989272,
                        "lng": -116.3112022701073
```

In [5]:
```
# Print the name and address of the first restaurant that appears
print(places_data["results"][0]["name"])
print(places_data["results"][0]["vicinity"])
```

```
China Grand Buffet
10498 W Fairview Ave, Boise
```

In [ ]:

In [1]:
```
# Create code to answer each of the following questions.
# Hint: You will need multiple target URLs and multiple API requests.

# Dependencies
import requests
import json

# Retrieve Google API key from config.py
from config import gkey
```

In [3]:
```
# 1. What are the geocoordinates (latitude/longitude) of Seattle, Washington?
target='Seattle, Washington'
params={'address':target, 'key':gkey}
base_url='https://maps.googleapis.com/maps/api/geocode/json'
response=requests.get(base_url,params)
seattle_geo=response.json()
lat=seattle_geo['results'][0]['geometry']['location']['lat']
lng=seattle_geo['results'][0]['geometry']['location']['lng']
print(f'{target};{lat}{lng}')
```

```
Seattle, Washington;47.6062095-122.3320708
```

In [7]:
```
# 2. What are the geocoordinates (latitude/longitude) of The White House?
target='White House'
params={'address':target, 'key':gkey}
base_url='https://maps.googleapis.com/maps/api/geocode/json'
response=requests.get(base_url,params)
seattle_geo=response.json()
lat=seattle_geo['results'][0]['geometry']['location']['lat']
lng=seattle_geo['results'][0]['geometry']['location']['lng']
print(f'{target};{lat}{lng}')
```

```
White House;38.8976763-77.0365298
```

In [9]:
```
# 3. Find the name and address of a bike store in Seattle, Washington.
#    Hint: See https://developers.google.com/places/web-service/supported_types
target_type='bicycle_store'
seattle_coords='47.6062095 -122.3320708'
radius=8000
params={
    'location':seattle_coords,
    'types':target_type,
    'radius':radius,
    'key':gkey
}

base_url='https://maps.googleapis.com/maps/api/place/nearbysearch/json'
response=requests.get(base_url,params)
seattle_bikes=response.json()

print(seattle_bikes['results'][0]['name'])
print(seattle_bikes['results'][0]['vicinity'])
```

```
REI
222 Yale Avenue North, Seattle
```

In [12]:
```
# 4. Find a balloon store near the White House.
target_type='baloon_store'
seattle_coords='38.8976763 -77.0365298'
radius=8000
params={
    'location':seattle_coords,
    'types':target_type,
    'radius':radius,
    'key':gkey
}

base_url='https://maps.googleapis.com/maps/api/place/nearbysearch/json'
response=requests.get(base_url,params)
seattle_bikes=response.json()

print(seattle_bikes['results'][0]['name'])
print(seattle_bikes['results'][0]['vicinity'])
```

```
Washington
Washington
```

In [11]:
```
# 5. Find the nearest dentist to your house.
#    Hint: Use Google Maps to find your latitude and Google Places to find the
#    dentist. You will also need the rankby property.
target='300 Finch, Lake Forest, CA'
params={'address':target, 'key':gkey}
base_url='https://maps.googleapis.com/maps/api/geocode/json'
response=requests.get(base_url,params)
seattle_geo=response.json()
lat=seattle_geo['results'][0]['geometry']['location']['lat']
lng=seattle_geo['results'][0]['geometry']['location']['lng']
print(f'{target};{lat}{lng}')
```

```
300 Finch, Lake Forest, CA;33.6752577-117.6777081
```

In [ ]:
```
# 6. Bonus: Find the names and addresses of the top five restaurants in your home city.
#    Hint: Read about "Text Search Results"
# (https://developers.google.com/places/web-service/search#TextSearchRequests)
```

In [1]:
```
# Dependencies
# Dependencies
import pandas as pd
import numpy as np
import requests
import json

# Google API Key
from config import gkey
```

In [2]:
```
types_df = pd.read_csv("../Resources/ethnic_restr.csv")
types_df.head()
```

Out[2]:

	ethnicity
0	chinese
1	cuban
2	czech
3	french
4	german

In [3]:
```
# set up additional columns to hold information
types_df['name'] = ""
types_df['address'] = ""
types_df['price_level'] = ""
types_df['rating'] = ""

types_df.head()
```

Out[3]:

	ethnicity	name	address	price_level	rating
0	chinese				
1	cuban				
2	czech				
3	french				
4	german				

In [4]:
```python
# find the closest restaurant of each type to coordinates

base_url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"
params = {
    "location": "39.952583,-75.16522",  # philadelphia coords,
    "rankby": "distance",
    "type": "restaurant",
    "key": gkey,
}
# use iterrows to iterate through pandas dataframe
for index, row in types_df.iterrows():

    # get restaurant type from df
    restr_type = row['ethnicity']

    # add keyword to params dict
    params['keyword'] = restr_type

    # assemble url and make API request
    print(f"Retrieving Results for Index {index}: {restr_type}.")
    response = requests.get(base_url, params=params).json()

    # extract results
    results = response['results']

    try:
        print(f"Closest {restr_type} restaurant is {results[0]['name']}.")

        types_df.loc[index, 'name'] = results[0]['name']
        types_df.loc[index, 'address'] = results[0]['vicinity']
        types_df.loc[index, 'price_level'] = results[0]['price_level']
        types_df.loc[index, 'rating'] = results[0]['rating']

    except (KeyError, IndexError):
        print("Missing field/result... skipping.")

    print("------------")
```

```
Retrieving Results for Index 0: chinese.
Closest chinese restaurant is Master Wok.
------------
Retrieving Results for Index 1: cuban.
Closest cuban restaurant is Alma de Cuba.
------------
Retrieving Results for Index 2: czech.
Closest czech restaurant is Penn's Corner Restaurant.
------------
Retrieving Results for Index 3: french.
Closest french restaurant is Amuse.
------------
Retrieving Results for Index 4: german.
Closest german restaurant is Brü Craft & Wurst.
------------
Retrieving Results for Index 5: greek.
Closest greek restaurant is Zaffron Mediterranean Grill.
------------
Retrieving Results for Index 6: haitian.
Closest haitian restaurant is Caribbean Delight.
------------
Retrieving Results for Index 7: hungarian.
Closest hungarian restaurant is Passero's Coffee Roasters.
Missing field/result... skipping.
------------
Retrieving Results for Index 8: indian.
Closest indian restaurant is Amma's South Indian Cuisine.
------------
```

```
Retrieving Results for Index 9: indonesian.
Closest indonesian restaurant is Mai Sushi.
Missing field/result... skipping.
------------
Retrieving Results for Index 10: irish.
Closest irish restaurant is Tir na nOg Irish Bar & Grill.
------------
Retrieving Results for Index 11: israeli.
Closest israeli restaurant is Goldie.
------------
Retrieving Results for Index 12: italian.
Closest italian restaurant is Davio's Northern Italian Steakhouse.
------------
Retrieving Results for Index 13: japanese.
Closest japanese restaurant is Nom Nom Bowl.
Missing field/result... skipping.
------------
Retrieving Results for Index 14: jewish.
Closest jewish restaurant is Abe Fisher.
------------
Retrieving Results for Index 15: korean.
Closest korean restaurant is GIWA Fresh Korean Kitchen.
------------
Retrieving Results for Index 16: lebanese.
Closest lebanese restaurant is NAYA.
Missing field/result... skipping.
------------
Retrieving Results for Index 17: mexican.
Closest mexican restaurant is Mission Taqueria.
------------
Retrieving Results for Index 18: new american.
Closest new american restaurant is Square 1682.
------------
Retrieving Results for Index 19: pakistani.
Closest pakistani restaurant is Cafe Spice Express.
Missing field/result... skipping.
------------
Retrieving Results for Index 20: polish.
Closest polish restaurant is Franks A-lot.
Missing field/result... skipping.
------------
Retrieving Results for Index 21: russian.
Closest russian restaurant is Brü Craft & Wurst.
------------
Retrieving Results for Index 22: scandinavian.
Closest scandinavian restaurant is Bar Hygge.
------------
Retrieving Results for Index 23: scottish.
Closest scottish restaurant is Amuse.
------------
Retrieving Results for Index 24: soul food.
Closest soul food restaurant is Keven Parker's Soul Food Cafe.
Missing field/result... skipping.
------------
Retrieving Results for Index 25: spanish.
Closest spanish restaurant is Sabroso+Sorbo.
Missing field/result... skipping.
------------
Retrieving Results for Index 26: thai.
Closest thai restaurant is Chatayee Thai.
Missing field/result... skipping.
------------
Retrieving Results for Index 27: turkish.
Closest turkish restaurant is The Original Turkey.
------------
Retrieving Results for Index 28: ukrainian.
Closest ukrainian restaurant is Abe Fisher.
```

```
------------
Retrieving Results for Index 29: vietnamese.
Closest vietnamese restaurant is Pho Street.
Missing field/result... skipping.
------------
```

In [5]:
```
types_df
```

Out[5]:

	ethnicity	name	address	price_level	rating
0	chinese	Chinese Fast Wok	1500 John F Kennedy Blvd, Philadelphia	2	1.3
1	cuban	Alma de Cuba - Cuban Restaurant	1623 Walnut St, Philadelphia	3	4.4
2	czech	SUBWAY®Restaurants	1515 Market St, Philadelphia	1	3.7
3	french	Amuse	1421 Arch St, Philadelphia	4	4.5
4	german	Brü Craft & Wurst	1316 Chestnut St, Philadelphia	4	4
5	greek	Noon Mediterranean	1601 Market St, Philadelphia		3.9
6	haitian	Chez Rosaire Haitian & West Indian Food	121 W Tabor Rd, Philadelphia		4.3
7	hungarian	Wursthaus Schmitz	51 N 12th St, Philadelphia		3.3
8	indian	Cafe Spice Express	1625 Chestnut St # F5, Philadelphia		4.5
9	indonesian	Penang Restaurant	117 N 10th St, Philadelphia	2	4.3
10	irish	Tir na nOg Irish Bar & Grill	1600 Arch St, Philadelphia	2	4.1
11	israeli	Goldie	1526 Sansom St, Philadelphia	1	4.7
12	italian	Davio's Northern Italian Steakhouse	111 S 17th St, Philadelphia	3	4.5
13	japanese	Double Knot	120 S 13th St, Philadelphia	2	4.7
14	jewish	Abe Fisher	1623 Sansom St, Philadelphia	3	4.8
15	korean	Giwa - Fresh Korean Kitchen	1722 Sansom St, Philadelphia	2	4.5
16	lebanese	Boutros Greek & Middle Eastern	200 S Broad St, Philadelphia	1	
17	mexican	Chipotle Mexican Grill	1625 Chestnut St, Philadelphia	1	3.2
18	new american	Square 1682	121 S 17th St, Philadelphia	3	3.8
19	pakistani	Nanee's Kitchen	2954, 51 N 12th St, Philadelphia		3.3
20	polish	Franks A-lot	51 N 12th St, Philadelphia		2.4
21	russian	Rachael's Nosheri	120 S 19th St, Philadelphia	1	4
22	scandinavian	Rooster Soup Co.	1526 Sansom St, Philadelphia	1	4.5
23	scottish				
24	soul food	Keven Parker's Soul Food Cafe	N 12th St & Arch St, Philadelphia		3.9
25	spanish	Jamonera	105 S 13th St, Philadelphia	2	4.3
26	thai	Xiandu Thai Fusion Cuisine	1119 Walnut St, Philadelphia	2	4.6
27	turkish	The Original Turkey	45 N 12th St, Philadelphia	1	4.1
28	ukrainian	McDonald's	1401 Arch St, Philadelphia	1	3.1
29	vietnamese	Vietnam Express	106 S 20th St, Philadelphia		4.7

In [1]:
```
# Dependencies
import pandas as pd
import numpy as np
import requests
import json

# Google API Key
from config import gkey
```

In [2]:
```
# Import cities file as DataFrame
cities_pd = pd.read_csv("../Resources/cities.csv")
cities_pd.head()
```

Out[2]:

	City	State
0	New York City	New York
1	Los Angeles	California
2	Chicago	Illinois
3	Houston	Texas
4	Philadelphia	Pennsylvania

In [3]:
```
# Add columns for lat, lng, airport name, airport address, airport rating
# Note that we used "" to specify initial entry.
cities_pd["Lat"] = ""
cities_pd["Lng"] = ""
cities_pd["Airport Name"] = ""
cities_pd["Airport Address"] = ""
cities_pd["Airport Rating"] = ""
cities_pd.head()
```

Out[3]:

	City	State	Lat	Lng	Airport Name	Airport Address	Airport Rating
0	New York City	New York					
1	Los Angeles	California					
2	Chicago	Illinois					
3	Houston	Texas					
4	Philadelphia	Pennsylvania					

In [4]:

```python
# create a params dict that will be updated with new city each iteration
params = {"key": gkey}

# Loop through the cities_pd and run a lat/long search for each city
for index, row in cities_pd.iterrows():
    base_url = "https://maps.googleapis.com/maps/api/geocode/json"

    city = row['City']
    state = row['State']

    # update address key value
    params['address'] = f"{city},{state}"

    # make request
    cities_lat_lng = requests.get(base_url, params=params)

    # print the cities_lat_lng url, avoid doing for public github repos in order to avoid exposing key
    # print(cities_lat_lng.url)

    # convert to json
    cities_lat_lng = cities_lat_lng.json()

    cities_pd.loc[index, "Lat"] = cities_lat_lng["results"][0]["geometry"]["location"]["lat"]
    cities_pd.loc[index, "Lng"] = cities_lat_lng["results"][0]["geometry"]["location"]["lng"]

# Visualize to confirm lat lng appear
cities_pd.head()
```

Out[4]:

	City	State	Lat	Lng	Airport Name	Airport Address	Airport Rating
0	New York City	New York	40.7128	-74.006			
1	Los Angeles	California	34.0522	-118.244			
2	Chicago	Illinois	41.8781	-87.6298			
3	Houston	Texas	29.7604	-95.3698			
4	Philadelphia	Pennsylvania	39.9526	-75.1652			

In [5]:

```
# params dictionary to update each iteration
params = {
    "radius": 50000,
    "types": "airport",
    "keyword": "international airport",
    "key": gkey
}

# Use the lat/lng we recovered to identify airports
for index, row in cities_pd.iterrows():
    # get lat, lng from df
    lat = row["Lat"]
    lng = row["Lng"]

    # change location each iteration while leaving original params in place
    params["location"] = f"{lat},{lng}"

    # Use the search term: "International Airport" and our lat/lng
    base_url = "https://maps.googleapis.com/maps/api/place/nearbysearch/json"

    # make request and print url
    name_address = requests.get(base_url, params=params)

#     print the name_address url, avoid doing for public github repos in order to avoid exposing key
#     print(name_address.url)

    # convert to json
    name_address = name_address.json()
    # print(json.dumps(name_address, indent=4, sort_keys=True))

    # Since some data may be missing we incorporate a try-except to skip any that are missing a data point.
    try:
        cities_pd.loc[index, "Airport Name"] = name_address["results"][0]["name"]
        cities_pd.loc[index, "Airport Address"] = name_address["results"][0]["vicinity"]
        cities_pd.loc[index, "Airport Rating"] = name_address["results"][0]["rating"]
    except (KeyError, IndexError):
        print("Missing field/result... skipping.")
```

```
Missing field/result... skipping.
Missing field/result... skipping.
Missing field/result... skipping.
```

In [6]:

```
# Save Data to csv
cities_pd.to_csv("Airport_Output.csv")

# Visualize to confirm airport data appears
cities_pd.head(10)
```

Out[6]:

	City	State	Lat	Lng	Airport Name	Airport Address	Airport Rating
0	New York City	New York	40.7128	-74.006	Newark Liberty International Airport	3 Brewster Rd, Newark	3.2
1	Los Angeles	California	34.0522	-118.244	Los Angeles International Airport	1 World Way, Los Angeles	3.6
2	Chicago	Illinois	41.8781	-87.6298	O'Hare International Airport	10000 W O'Hare Ave, Chicago	3.6
3	Houston	Texas	29.7604	-95.3698	George Bush Intercontinental Airport	2800 N Terminal Rd, Houston	3.7
4	Philadelphia	Pennsylvania	39.9526	-75.1652	Philadelphia International Airport	8000 Essington Ave, Philadelphia	3.3
5	Phoenix	Arizona	33.4484	-112.074	Phoenix Sky Harbor International Airport	3400 E Sky Harbor Blvd, Phoenix	3.9
6	San Antonio	Texas	29.4241	-98.4936	San Antonio International Airport	9800 Airport Blvd, San Antonio	4
7	San Diego	California	32.7157	-117.161	San Diego International Airport	3225 N Harbor Dr, San Diego	3.9
8	Dallas	Texas	32.7767	-96.797	Dallas/Fort Worth International Airport	2400 Aviation Dr, DFW Airport	3.8
9	San Jose	California	37.3382	-121.886	San Francisco International Airport	San Francisco	4.1

In [ ]:

If gmaps doesn’t zoom in

On git bash

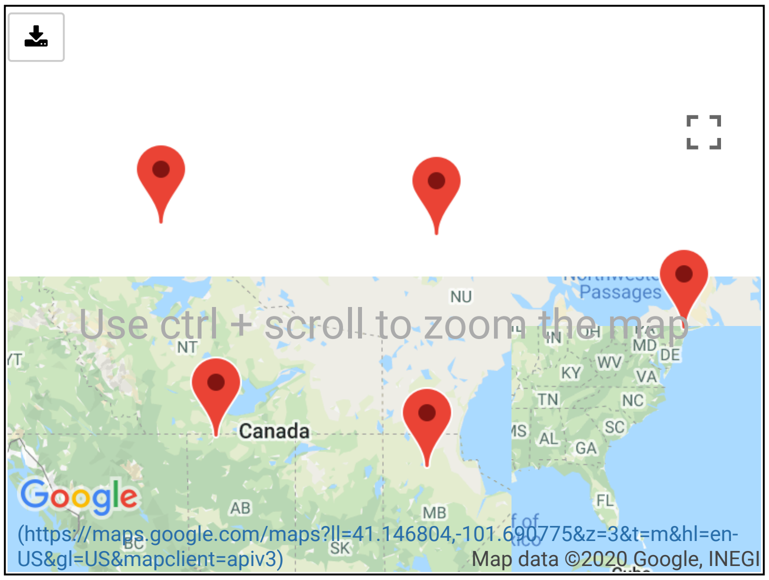
```
# enable jupyter extensions
jupyter nbextension enable --py --sys-prefix widgetsnbextension

# install gmaps
pip install gmaps

# enable gmaps
jupyter nbextension enable --py --sys-prefix gmaps
```

In [1]:
```python
import gmaps

# Google developer API key
from config import gkey

# Access maps with unique API key
gmaps.configure(api_key=gkey)
```

In [2]:
```python
# Create a list containing coordinates
coordinates = [
    (40.71, -74.00),
    (30.26, -97.74),
    (46.87, -96.78),
    (47.60, -122.33),
    (32.71, -117.16)
]
```

In [3]:
```python
# Customize the size of the figure
figure_layout = {
    'width': '400px',
    'height': '300px',
    'border': '1px solid black',
    'padding': '1px',
    'margin': '0 auto 0 auto'
}
fig = gmaps.figure(layout=figure_layout)
```

In [4]:
```python
# Assign the marker layer to a variable
markers = gmaps.marker_layer(coordinates)
# Add the layer to the map
fig.add_layer(markers)
fig
```


Use ctrl + scroll to zoom the map
Canada
NU
NT
AB
SK
MB
BC
KY
WV
VA
MD
DE
TN
NC
SC
GA
AL
FL
Passages
Google
(https://maps.google.com/maps?ll=41.146804,-101.690775&z=3&t=m&hl=en-US&gl=US&mapclient=apiv3)
Map data ©2020 Google, INEGI

Map
Satellite
Canada
United States
Mexico
North Pacific Ocean
North Atlantic Ocean
Venezuela
Google
Map data ©2018
Terms of Use

In [5]:
```
#!pip install ipywidgets
```

In [ ]:
```
```

In [1]:
```python
import gmaps
import pandas as pd

# Configure gmaps
from config import gkey

gmaps.configure(api_key=gkey)
```

In [4]:
```python
# Create aiport dataframe
airport_df=pd.read_csv('../Resources/Airport_Output.csv')
airport_df.dropna()
airport_df.head()
```

Out[4]:

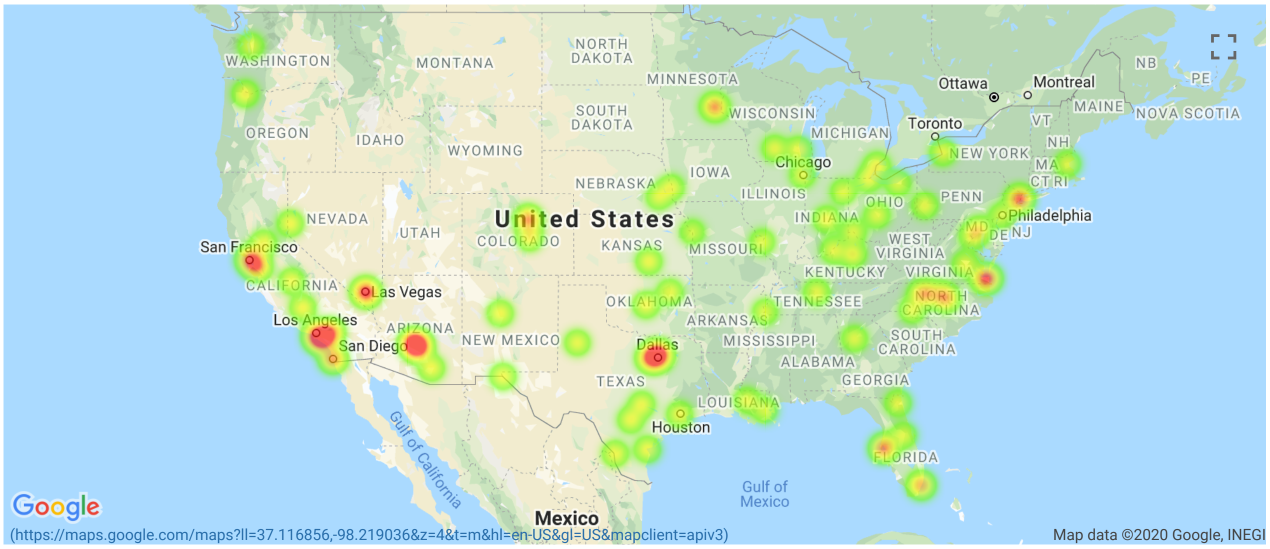
	Unnamed: 0	City	State	Lat	Lng	Airport Name	Airport Address	Airport Rating
0	0	New York City	New York	40.712775	-74.005973	Newark Liberty International Airport	3 Brewster Rd, Newark	3.2
1	1	Los Angeles	California	34.052234	-118.243685	Los Angeles International Airport	1 World Way, Los Angeles	3.5
2	2	Chicago	Illinois	41.878114	-87.629798	O'Hare International Airport	10000 W O'Hare Ave, Chicago	3.6
3	3	Houston	Texas	29.760427	-95.369803	William P. Hobby Airport	7800 Airport Blvd, Houston	4.0
4	4	Philadelphia	Pennsylvania	39.952584	-75.165222	Philadelphia International Airport	8000 Essington Ave, Philadelphia	3.3

In [6]:
```python
# Store latitude and longitude in locations
locations=airport_df[['Lat','Lng']]

# Filla NaN values and convert to float
rating=airport_df['Airport Rating'].astype(float)
```

In [7]:
```python
type(locations)
```

Out[7]:
```
pandas.core.frame.DataFrame
```

In [9]:
```python
# Plot Heatmap
fig=gmaps.figure()
#create the layer
heat_layer=gmaps.heatmap_layer(locations,weights=rating,dissipating=False,max_intensity=10,point_radius=1)
#pass the layer
fig.add_layer(heat_layer)
fig
```



In [ ]:

In [1]:
```
#!pip install Census
```

In [2]:
```
# Dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
from census import Census

# Census API Key
from config import api_key
c = Census(api_key, year=2013)
```

In [3]:
```
# Run Census Search to retrieve data on all zip codes (2013 ACS5 Census)
# See: https://github.com/CommerceDataService/census-wrapper for library documentation
# See: https://gist.github.com/afhaque/60558290d6efd892351c4b64e5c01e9b for labels
census_data = c.acs5.get(("NAME", "B19013_001E", "B01003_001E", "B01002_001E",
                          "B19301_001E",
                          "B17001_002E"), {'for': 'zip code tabulation area:*'})

# Convert to DataFrame
census_pd = pd.DataFrame(census_data)

# Column Reordering
census_pd = census_pd.rename(columns={"B01003_001E": "Population",
                                      "B01002_001E": "Median Age",
                                      "B19013_001E": "Household Income",
                                      "B19301_001E": "Per Capita Income",
                                      "B17001_002E": "Poverty Count",
                                      "NAME": "Name", "zip code tabulation area": "Zipcode"})

# Add in Poverty Rate (Poverty Count / Population)
census_pd["Poverty Rate"] = 100 * \
    census_pd["Poverty Count"].astype(
        int) / census_pd["Population"].astype(int)

# Final DataFrame
census_pd = census_pd[["Zipcode", "Population", "Median Age", "Household Income",
                       "Per Capita Income", "Poverty Count", "Poverty Rate"]]

# Visualize
print(len(census_pd))
census_pd.head()
```

```
33120
```

Out[3]:

	Zipcode	Population	Median Age	Household Income	Per Capita Income	Poverty Count	Poverty Rate
0	08518	5217.0	41.5	74286.0	33963.0	170.0	3.258578
1	08520	27468.0	37.4	90293.0	37175.0	1834.0	6.676860
2	08525	4782.0	47.1	118656.0	59848.0	43.0	0.899205
3	08527	54867.0	42.2	88588.0	37021.0	2191.0	3.993293
4	08528	245.0	48.5	58676.0	49117.0	0.0	0.000000

In [4]:
```
# Save as a csv
# Note to avoid any issues later, use encoding="utf-8"
census_pd.to_csv("census_data.csv", encoding="utf-8", index=False)
```

In [ ]:

In [1]:

```
# Dependencies
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import requests
from census import Census
import gmaps

# Census & gmaps API Keys
from config import (api_key, gkey)
c = Census(api_key, year=2013)

# Configure gmaps
gmaps.configure(api_key=gkey)
```

In [2]:

```
# Run Census Search to retrieve data on all states
# Note the addition of "B23025_005E" for unemployment count
census_data = c.acs5.get(("NAME", "B19013_001E", "B01003_001E", "B01002_001E",
                          "B19301_001E",
                          "B17001_002E",
                          "B23025_005E"), {'for': 'state:*'})

# Convert to DataFrame
census_pd = pd.DataFrame(census_data)

# Column Reordering
census_pd = census_pd.rename(columns={"B01003_001E": "Population",
                                      "B01002_001E": "Median Age",
                                      "B19013_001E": "Household Income",
                                      "B19301_001E": "Per Capita Income",
                                      "B17001_002E": "Poverty Count",
                                      "B23025_005E": "Unemployment Count",
                                      "NAME": "Name", "state": "State"})

# Add in Poverty Rate (Poverty Count / Population)
census_pd["Poverty Rate"] = 100 * \
    census_pd["Poverty Count"].astype(
        int) / census_pd["Population"].astype(int)

# Add in Employment Rate (Employment Count / Population)
census_pd["Unemployment Rate"] = 100 * \
    census_pd["Unemployment Count"].astype(
        int) / census_pd["Population"].astype(int)

# Final DataFrame
census_pd = census_pd[["State", "Name", "Population", "Median Age", "Household Income",
                       "Per Capita Income", "Poverty Count", "Poverty Rate", "Unemployment Rate"]]

census_pd.head()
```

Out[2]:

	State	Name	Population	Median Age	Household Income	Per Capita Income	Poverty Count	Poverty Rate	Unemployment Rate
0	01	Alabama	4799277.0	38.1	43253.0	23680.0	870631.0	18.140878	5.040968
1	02	Alaska	720316.0	33.6	70760.0	32651.0	69514.0	9.650487	4.572854
2	04	Arizona	6479703.0	36.3	49774.0	25358.0	1131901.0	17.468409	4.882323
3	05	Arkansas	2933369.0	37.5	40768.0	22170.0	547328.0	18.658682	4.132961
4	06	California	37659181.0	35.4	61094.0	29527.0	5885417.0	15.628107	5.758662

In [3]:

```
# Save as a csv
# Note to avoid any issues later, use encoding="utf-8"
census_pd.to_csv("census_data_states.csv", encoding="utf-8", index=False)
```

In [4]:
```python
# Read in the csv containing state centroid coordinates
centroids = pd.read_csv("../Resources/state_centroids.csv")
centroids.head()
```

Out[4]:

	State	Latitude	Longitude
0	Alabama	32.7794	-86.8287
1	Alaska	64.0685	-152.2782
2	Arizona	34.2744	-111.6602
3	Arkansas	34.8938	-92.4426
4	California	37.1841	-119.4696

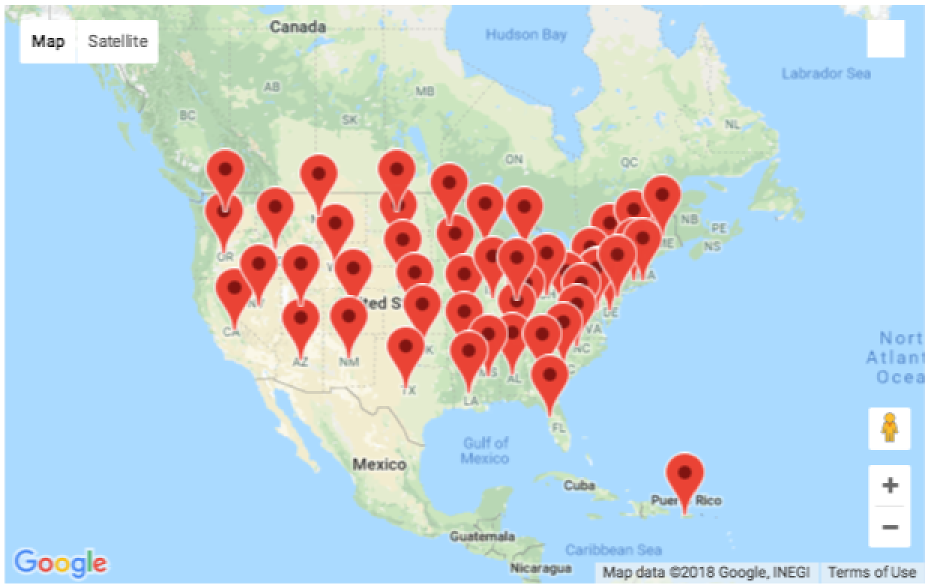
In [5]:
```python
# Merge the datasets using the sate columns
census_data = pd.merge(census_pd, centroids, how="left", left_on="Name", right_on="State")

# Save the updated dataframe as a csv
census_data.to_csv("../Resources/state_census_data.csv", encoding="utf-8", index=False)
census_data.head()
```

Out[5]:

	State_x	Name	Population	Median Age	Household Income	Per Capita Income	Poverty Count	Poverty Rate	Unemployment Rate	State_y	Latitude	Longitude
0	01	Alabama	4799277.0	38.1	43253.0	23680.0	870631.0	18.140878	5.040968	Alabama	32.7794	-86.8287
1	02	Alaska	720316.0	33.6	70760.0	32651.0	69514.0	9.650487	4.572854	Alaska	64.0685	-152.2782
2	04	Arizona	6479703.0	36.3	49774.0	25358.0	1131901.0	17.468409	4.882323	Arizona	34.2744	-111.6602
3	05	Arkansas	2933369.0	37.5	40768.0	22170.0	547328.0	18.658682	4.132961	Arkansas	34.8938	-92.4426
4	06	California	37659181.0	35.4	61094.0	29527.0	5885417.0	15.628107	5.758662	California	37.1841	-119.4696

In [7]:
```python
# Convert poverty rate as a list
# Convert bank rate to list
poverty_rate = census_data["Poverty Rate"].tolist()
```

In [ ]:
```python
# Create a map using state centroid coordinates to set markers
marker_locations = census_data[['Latitude', 'Longitude']]

# Create a marker_layer using the poverty list to fill the info box
fig = gmaps.figure()
markers = gmaps.marker_layer(marker_locations,
    info_box_content=[f"Poverty Rate: {rate}" for rate in poverty_rate])
fig.add_layer(markers)
fig
```



Banking and Poverty

The below script explores the relationship between states with high poverty rates and bank counts per state.

In this script, we retrieved and plotted data from the 2013 US Census and Google Places API to show the relationship between various socioeconomic parameters and bank count across 700 randomly selected zip codes. We used Pandas, Numpy, Matplotlib, Requests, Census API, and Google API to accomplish our task.

In [1]:
```
# !jupyter nbextension enable --py --sys-prefix widgetsnbextension
# !pip install gmaps
# !jupyter nbextension enable --py --sys-prefix gmaps
# !pip install us
```

In [2]:
```
# Dependencies
from census import Census
from config import (census_key, gkey)
import gmaps
import numpy as np
import pandas as pd
import requests
import time
from us import states
from scipy.stats import linregress
from matplotlib import pyplot as plt


# Census API Key
c = Census(census_key, year=2013)
```

Data Retrieval

In [3]:
```
# Run Census Search to retrieve data on all zip codes (2013 ACS5 Census)
# See: https://github.com/CommerceDataService/census-wrapper for library documentation
# See: https://gist.github.com/afhaque/60558290d6efd892351c4b64e5c01e9b for labels
census_data = c.acs5.get(("B01003_001E", "B17001_002E"), {'for': 'zip code tabulation area:*'})

# Convert to DataFrame
census_pd = pd.DataFrame(census_data)

# Column Reordering
census_pd = census_pd.rename(columns={"B01003_001E": "Population",
                                      "B17001_002E": "Poverty Count",
                                      "zip code tabulation area": "Zipcode"})

# Add in Poverty Rate (Employment Count / Population)
census_pd["Poverty Rate"] = 100 * \
    census_pd["Poverty Count"].astype(
        int) / census_pd["Population"].astype(int)

# Final DataFrame
census_pd = census_pd[["Zipcode", "Population", "Poverty Rate"]]

# Visualize
print(len(census_pd))
census_pd.head()
```

```
33120
```

Out[3]:

	Zipcode	Population	Poverty Rate
0	01832	22121.0	10.903666
1	01833	8295.0	2.302592
2	01834	6675.0	2.187266
3	01835	13527.0	8.523693
4	01840	4547.0	39.542556

Combine Data

In [4]:
```
# Import the original data we analyzed earlier. Use dtype="object" to match other
census_data_original = pd.read_csv(
    "../Resources/zip_bank_data.csv", dtype="object", encoding="utf-8")

# Visualize
census_data_original.head()
```

Out[4]:

	Zipcode	Address	Median Age	Household Income	Per Capita Income	Lat	Lng	Bank Count
0	624	Hastings, MI 49058, USA	40.8	46777	22137	42.6306916	-85.2929384	9
1	692	Ball, LA 71405, USA	35.8	55242	23941	31.4061799	-92.396174	12
2	730	Great Mills, MD 20634, USA	31.9	79944	35961	38.2201614	-76.4967919	9
3	757	Williamsport, TN 38487, USA	41.6	38125	18884	35.7310368	-87.2419299	0
4	957	Marion, ND 58466, USA	44.5	69844	36981	46.5594224	-98.3481542	1

In [5]:
```
# Merge the two data sets along zip code
census_data_complete = pd.merge(
    census_data_original, census_pd, how="left", on=["Zipcode", "Zipcode"])

# Remove rows missing data
census_data_complete = census_data_complete.dropna()

# Visualize
census_data_complete.head()
```

Out[5]:

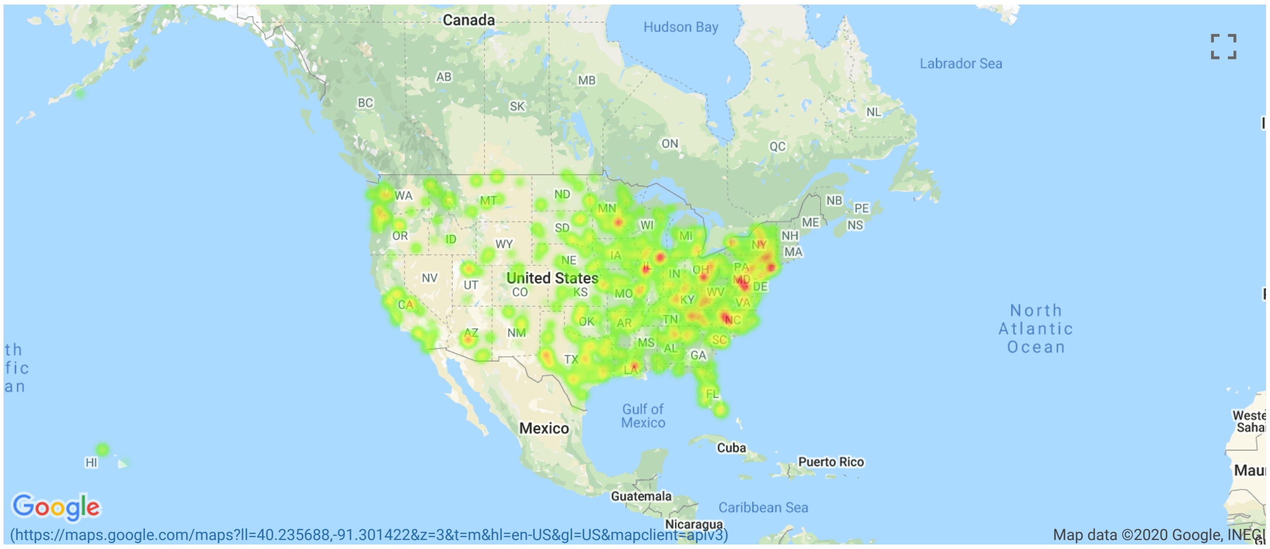
	Zipcode	Address	Median Age	Household Income	Per Capita Income	Lat	Lng	Bank Count	Population	Poverty Rate
63	10007	Flatonia, TX 78941, USA	45.7	40304	23115	29.7574106	-97.1574784	0	6525.0	2.206897
64	10303	Nevada, IA 50201, USA	40.4	56619	28908	42.065743	-93.4599326	7	24537.0	21.828259
65	10309	Lukachukai, AZ 86507, USA	24.1	22009	8346	36.4106866	-109.2593642	0	32646.0	6.766526
66	10553	Lone Pine, CA 93545, USA	40.6	32473	18444	36.5131184	-118.0888578	0	9895.0	15.260232
67	10803	Niagara, WI 54151, USA	45.7	45813	23500	45.715354	-87.9804239	2	12439.0	3.006673

Heatmap of Poverty Rate

In [6]:
```
# Configure gmaps with API key
gmaps.configure(api_key=gkey)
```

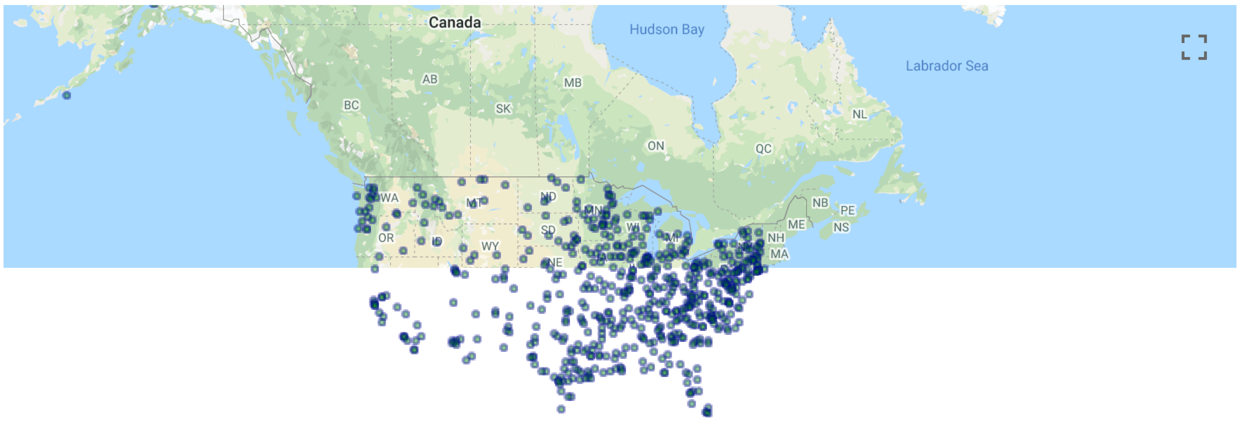
In [7]:
```
# Store 'Lat' and 'Lng' into  locations
locations = census_data_complete[["Lat", "Lng"]].astype(float)

# Convert Poverty Rate to float and store
# HINT: be sure to handle NaN values
census_data_complete = census_data_complete.dropna()
poverty_rate = census_data_complete["Poverty Rate"].astype(float)
```

In [8]:
```
# Create a poverty Heatmap layer
fig = gmaps.figure()

heat_layer = gmaps.heatmap_layer(locations, weights=poverty_rate,
                                 dissipating=False, max_intensity=100,
                                 point_radius = 1)

fig.add_layer(heat_layer)

fig
```

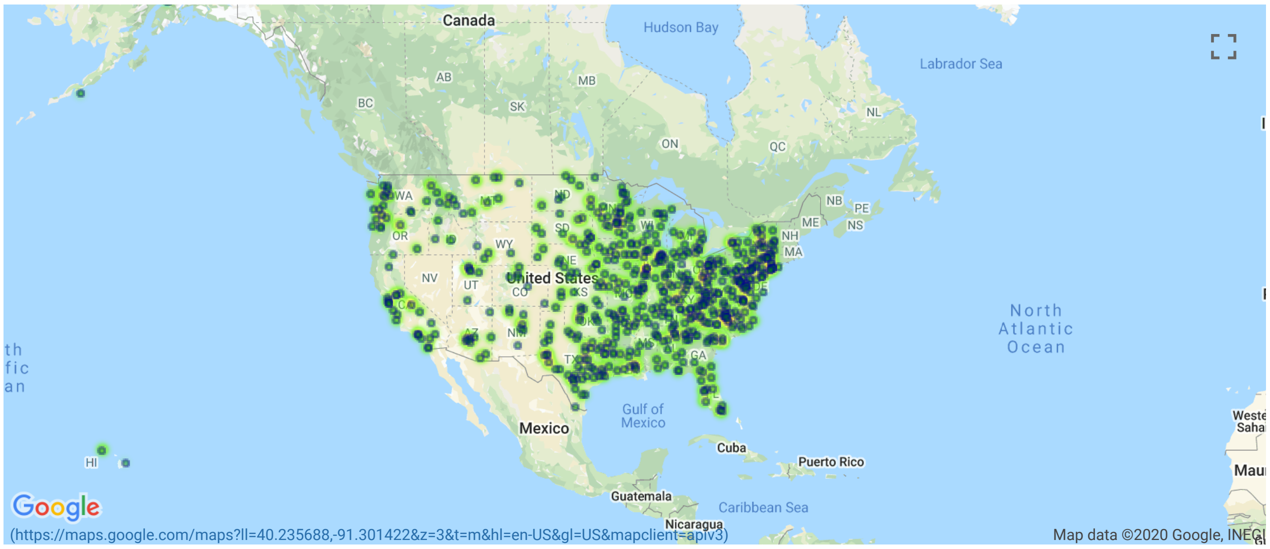


In [9]:
```
# Convert bank rate to list
bank_rate = census_data_complete["Bank Count"].tolist()
```

In [10]:
```
# Create bank symbol layer
bank_layer = gmaps.symbol_layer(
    locations, fill_color='rgba(0, 150, 0, 0.4)',
    stroke_color='rgba(0, 0, 150, 0.4)', scale=2,
    info_box_content=[f"Bank amount: {bank}" for bank in bank_rate]
)


fig = gmaps.figure()
fig.add_layer(bank_layer)

fig
```



In [11]:
```
# Create a combined map
fig = gmaps.figure()

fig.add_layer(heat_layer)
fig.add_layer(bank_layer)

fig
```



Statistical Analysis

Summary Statistics

In [12]:
```
# Mean, median, mode for Poverty Rate
poverty_mean = round(census_data_complete['Poverty Rate'].astype('float').mean(), 2)
poverty_median = round(census_data_complete['Poverty Rate'].astype('float').median(), 2)
poverty_mode = round(census_data_complete['Poverty Rate'].astype('float').mode(), 2)

print(f"Poverty Rate Mean: {poverty_mean}")
print(f"Poverty Rate Median {poverty_median}")
print(f"Poverty Rate mode {poverty_mode}")
```

```
Poverty Rate Mean: 14.62
Poverty Rate Median 12.49
Poverty Rate mode 0    0.0
dtype: float64
```

In [13]:
```
# Mean, median, mode for Bank Count
bank_mean = round(census_data_complete['Bank Count'].astype('float').mean(), 2)
bank_median = round(census_data_complete['Bank Count'].astype('float').median(), 2)
bank_mode = round(census_data_complete['Bank Count'].astype('float').mode(), 2)

print(f"Bank Count Mean: {bank_mean}")
print(f"Bank Count Median {bank_median}")
print(f"Bank Count mode {bank_mode}")
```

```
Bank Count Mean: 21.11
Bank Count Median 2.0
Bank Count mode 0    0.0
dtype: float64
```

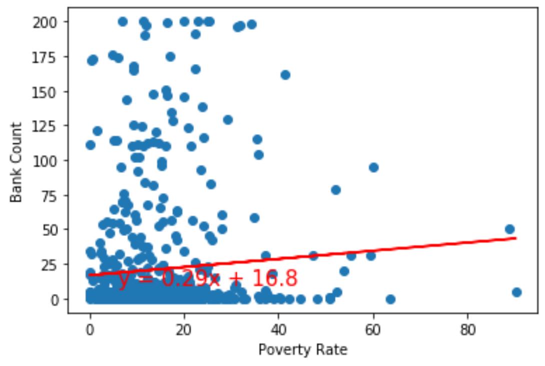
In [14]:
```
# Mean, median, mode for Population
population_mean = round(census_data_complete['Population'].astype('float').mean(), 2)
population_median = round(census_data_complete['Population'].astype('float').median(), 2)
population_mode = round(census_data_complete['Population'].astype('float').mode(), 2)

print(f"Population Mean: {population_mean}")
print(f"Population Median {population_median}")
print(f"Population mode {population_mode}")
```

```
Population Mean: 10115.99
Population Median 2971.0
Population mode 0    337.0
dtype: float64
```

Linear Regression

In [15]:
```
## Convert to floats and store Poverty Rate and Bank Count as x and y values
x_values = census_data_complete['Poverty Rate'].astype('float')
y_values = census_data_complete['Bank Count'].astype('float')

# Run linear regression
(slope, intercept, rvalue, pvalue, stderr) = linregress(x_values, y_values)
regress_values = x_values * slope + intercept
line_eq = "y = " + str(round(slope,2)) + "x + " + str(round(intercept,2))

# Plot scatter plot
plt.scatter(x_values,y_values)

# Plot regression line
plt.plot(x_values,regress_values,"r-")
plt.annotate(line_eq,(6,10),fontsize=15,color="red")

# Label plot
plt.xlabel('Poverty Rate')
plt.ylabel('Bank Count')

# Print r square value
print(f"R squard: {rvalue}")

# Show plot
plt.show()
```

```
R squard: 0.07390527851338498
```



Analysis

- There is a very weak correlation between poverty rates and bank counts. Keep in mind that linear regression will not consider other factors such as population or size of the city.

In [ ]: