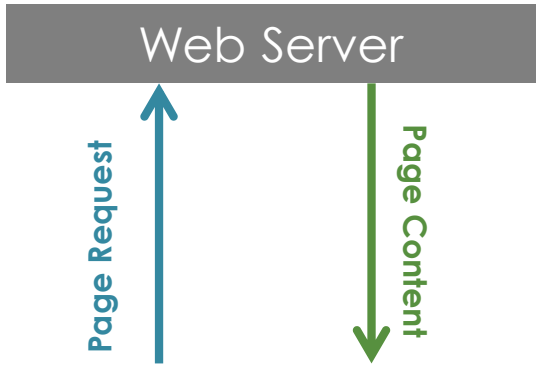

WORKING WITH DATA API
Ratchainant Thammasudjarit, Ph.D.



- APIs are most commonly used to retrieve data
- **Request** is necessary for requesting data via APIs
  - For instance, when you visited some blog post, your web browser made a request to its web server, which responded with the content of this web page
  - API requests work in exactly the same way – you make a request to an API server for data, and it responds to your request

What is API

API is an **A**pplication **P**rogramming **I**nterface

- Option 1

```
pip install requests
```

- Option 2

```
conda install requests
```

- Import

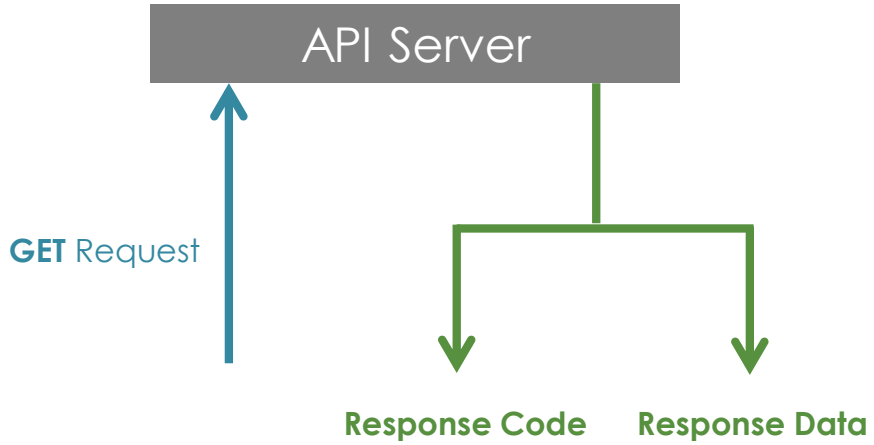
```
import requests
```

Making API Requests

Use the requests module

The requests module requires installation

- The response from the API comes with a **response code** which tells us whether our request was successful



- Response codes are important because they immediately tell us if something went wrong

Your First API Request

There are many different types of requests.

The most used one, a **GET** request, is used to retrieve data

- Let's start from a non-existing URL and observe the returned output

```
response = requests.get("http://api.open-notify.org/this-api-doesnt-exist")
```

- The response object has `status_code` as the attribute to tell the response code

```
print(response.status_code)

404
```

What does it mean?

Your First API Request

Use `requests.get()` to make a GET request

Input argument is the URL to be requested

- Some codes that are relevant to *GET* requests

Status Code	Descriptions
200	Everything went okay, and the result has been returned (if any).
301	The server is redirecting you to a different endpoint. This can happen when a company switches domain names, or an endpoint name is changed.
400	The server thinks you made a bad request. This can happen when you don't send along the right data, among other things.
401	The server thinks you're not authenticated. Many APIs require login credentials, so this happens when you don't send the right credentials to access an API.
403	The resource you're trying to access is forbidden. You don't have the right permissions to see it.
404	The resource you tried to access wasn't found on the server.
503	The server is not ready to handle the request.

Your First API Request

API Status codes indicate information about what happened with a request

- Let's try again with the **Open Notify**

```
response = requests.get("http://api.open-notify.org/astros.json")
print(response.status_code)

200
```

- Let's use the `response.json()` method to see the data we received back from the API

```
print(response.json())
```

```
{'message': 'success', 'people': [{'name': 'Alexey Ovchinin', 'craft': 'ISS'}, {'na
```

Your First API Request

Open Notify is an open source project to provide a simple programming interface for some of NASA's awesome data.

- JSON looks like python dictionary
- JSON can be a combination of dictionaries, lists, strings, integers, etc. represented as strings

Working with JSON

JSON (JavaScript Object Notation) is the language of APIs

JSON is the primary format in which data is passed back and forth to APIs, and most API servers will send their responses in JSON format

```python
import json

def jprint(obj):
    # create a formatted string of the Python JSON object
    text = json.dumps(obj, sort_keys=True, indent=4)
    print(text)

jprint(response.json())
```

- The json module has two main functions
  - `json.dumps()` — Takes in a Python object, and converts (dumps) it to a string
  - `json.loads()` — Takes a JSON string, and converts (loads) it to a Python object.


```
{
    "message": "success",
    "number": 6,
    "people": [
        {
            "craft": "ISS",
            "name": "Alexey Ovchinin"
        },
        {
            "craft": "ISS",
            "name": "Nick Hague"
        },
        {
            "craft": "ISS",
            "name": "Christina Koch"
        },
        {
            "craft": "ISS",
            "name": "Alexander Skvortsov"
        },
        {
            "craft": "ISS",
            "name": "Luca Parmitano"
        },
        {
            "craft": "ISS",
            "name": "Andrew Morgan"
        }
    ]
}
```


Working with JSON

The json package is part of the standard python modules

- Let's define parameters and make request

```
parameters = {
    "lat": 40.71,
    "lon": -74
}
```

```
response = requests.get("http://api.open-notify.org/iss-pass.json", params=paramet

jprint(response.json())
```


```
{
   "message": "success",
   "request": {
      "altitude": 100,
      "datetime": 1568062811,
      "latitude": 40.71,
      "longitude": -74.0,
      "passes": 5
   },
   "response": [
      {
         "duration": 395,
         "risetime": 1568082479
      },
      {
         "duration": 640,
         "risetime": 1568088118
      },
      {
         "duration": 614,
         "risetime": 1568093944
      },
      {
         "duration": 555,
         "risetime": 1568099831
      },
      {
         "duration": 595,
         "risetime": 1568105674
      }
   ]
}
```


API Query Parameters

From Open Notify documents, latitude and longitude are API parameters

- Let's extract the pass times from our JSON object

```
pass_times = response.json()['response']
jprint(pass_times)
```


```
[
    {
        "duration": 395,
        "risetime": 1568082479
    },
    {
        "duration": 640,
        "risetime": 1568088118
    },
    {
        "duration": 614,
        "risetime": 1568093944
    },
    {
        "duration": 555,
        "risetime": 1568099831
    },
    {
        "duration": 595,
        "risetime": 1568105674
    }
]
```


Understand the Pass Time

The JSON response matches what the documentation specified

- A dictionary with three keys
- The third key, response, contains a list of pass times
- Each pass time is a dictionary with risetime (pass start time) and duration keys

- Let's extract the risetime

```
risetimes = []

for d in pass_times:
    time = d['risetime']
    risetimes.append(time)

print(risetimes)
```

```
[1568082479, 1568088118, 1568093944, 1568099831, 1568105674]
```

Understand the Pass Time

The API returns a list of upcoming ISS passes for a particular location formatted as JSON

- Use the Python `datetime.fromtimestamp()` method to convert these into easier to understand times

```
from datetime import datetime

times = []

for rt in risetimes:
    time = datetime.fromtimestamp(rt)
    times.append(time)
    print(time)
```

```
2019-09-09 21:27:59
2019-09-09 23:01:58
2019-09-10 00:39:04
2019-09-10 02:17:11
2019-09-10 03:54:34
```

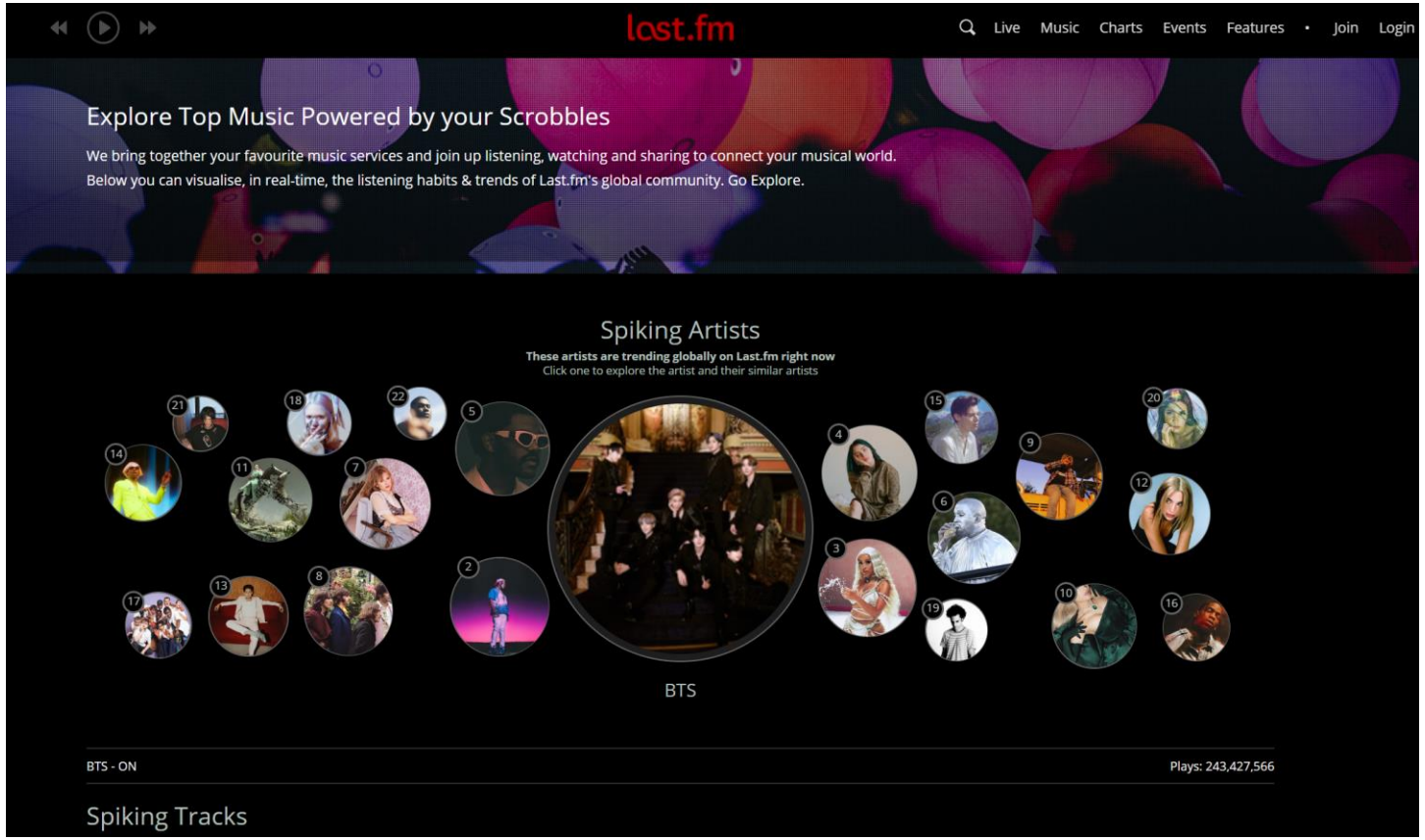
Understand the Pass Time

The API returns a list of upcoming ISS passes for a particular location formatted as JSON

- New things to learn
  - How to authenticate yourself with an API key
  - How to use rate limiting and other techniques to work within the guidelines of an API
  - How to use pagination to work with large responses

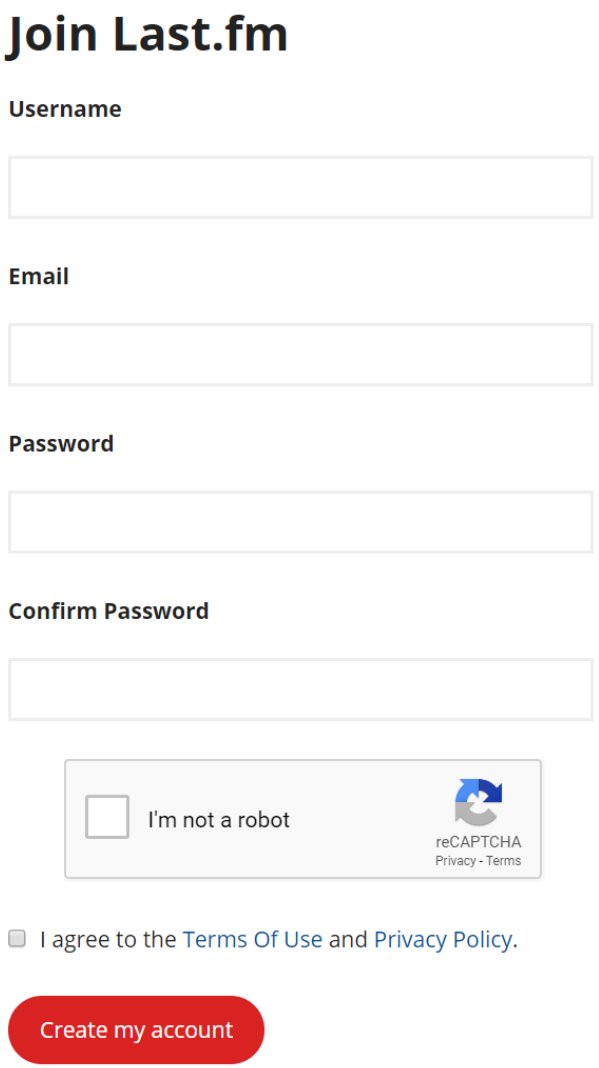
Getting Music Data with the Last.fm API using Python

This part is different from the previous API



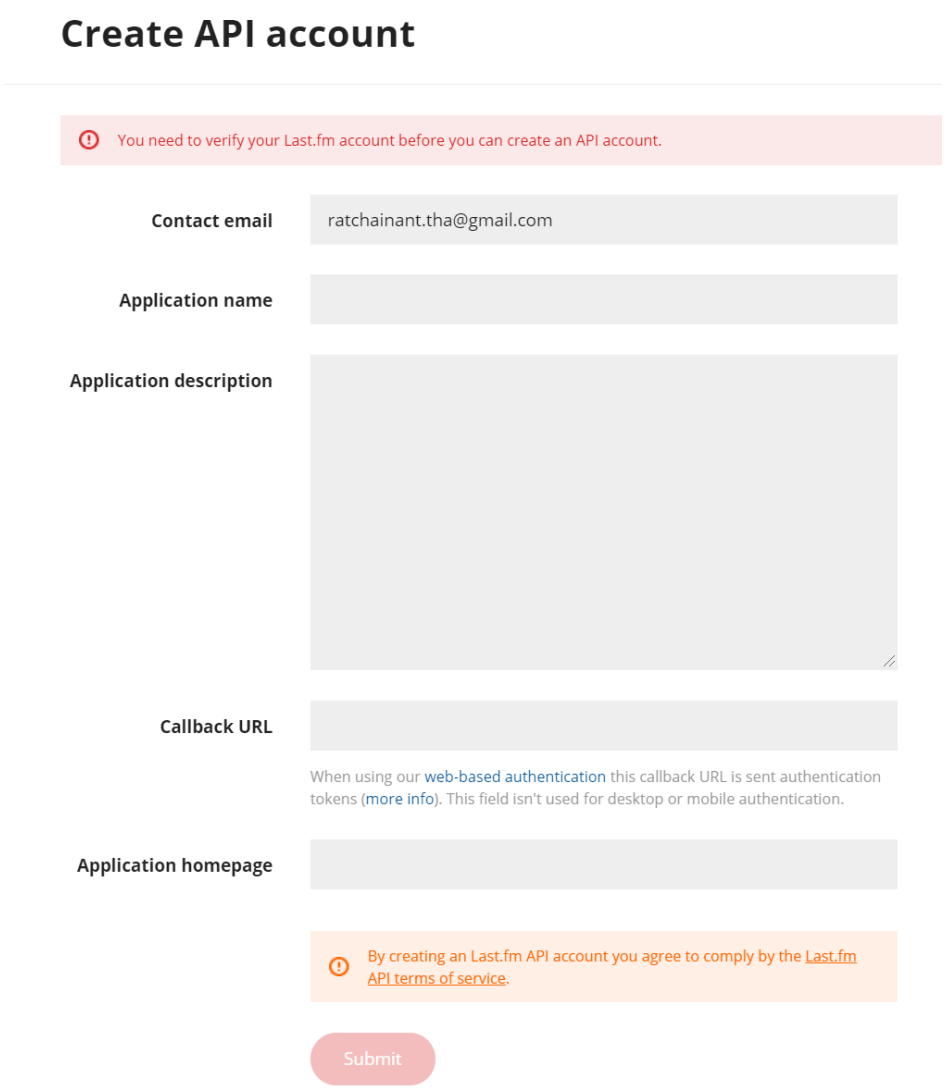
The last.fm

Explore Top Music Powered by your Scrobbles



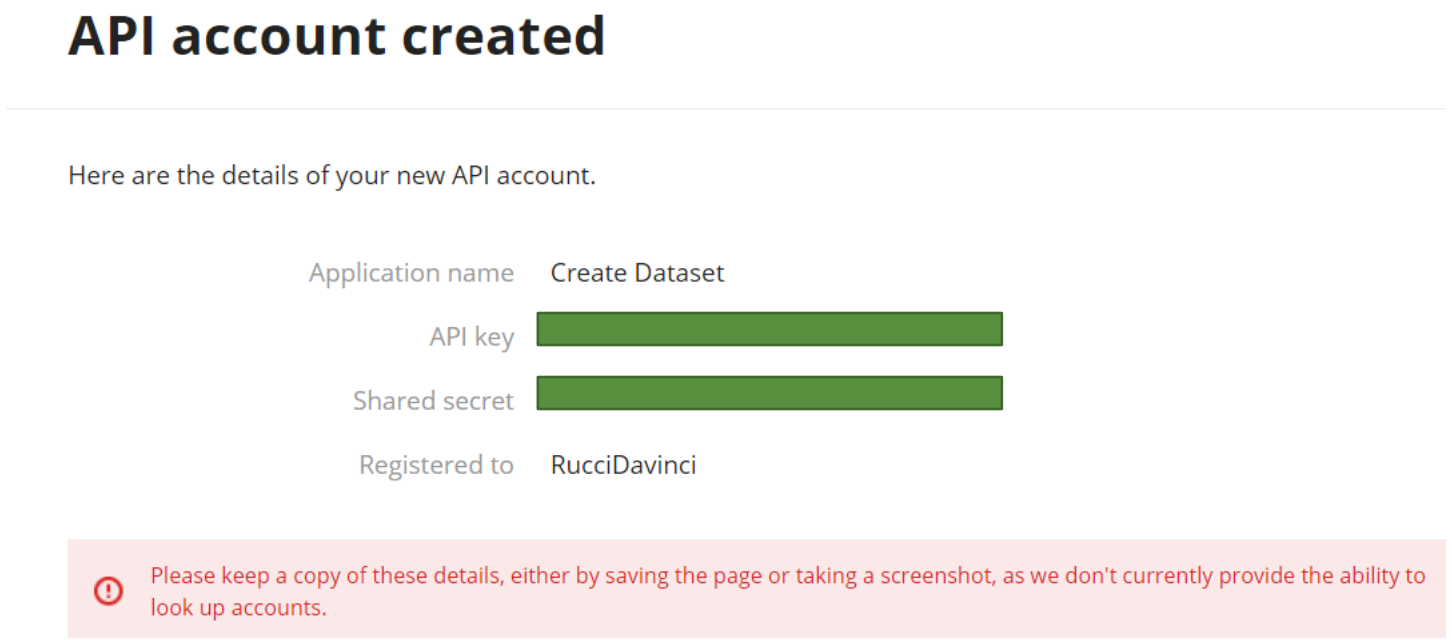
Authenticating with API Keys

To get an API key for Last.fm, start by creating an account. After you create your account, you should be taken to this form:



Authenticating with API Keys

To get an API key for Last.fm, start by creating an account. After you create your account, you should be taken to this form:



Authenticating with API Keys

To get an API key for Last.fm, start by creating an account. After you create your account, you should be taken to this form:

- User-agent header identifies ourselves when we make a request

```
API_KEY = 'c6138fe14a18acc5da07bee1c229f74b'
USER_AGENT = 'MyDemo'
```

```
import requests

headers = {
    'user-agent': USER_AGENT
}

payload = {
    'api_key': API_KEY,
    'method': 'chart.gettopartists',
    'format': 'json'
}

r = requests.get('http://ws.audioscrobbler.com/2.0/',
                 headers=headers, params=payload)
r.status_code
```

Making our first API request

User-Agent Header is required

```
{
    "artists": {
        "@attr": {
            "page": "1",
            "perPage": "50",
            "total": "2901036",
            "totalPages": "58021"
        },
        "artist": [
            {
                "image": [
                    {
                        "#text": "https://lastfm-img2.akamaized.net/i/u/34s/2a9
                        "size": "small"
                    },
                    {
                        "#text": "https://lastfm-img2.akamaized.net/i/u/64s/2a9
                        "size": "medium"
                    },
                    {
                        "#text": "https://lastfm-img2.akamaized.net/i/u/174s/2a
                        "size": "large"
                    },
                    {
                        "#text": "https://lastfm-img2.akamaized.net/i/u/300x300
                        "size": "extralarge"
                    },
                    {
                        "#text": "https://lastfm-img2.akamaized.net/i/u/300x300
                        "size": "mega"
                    }
                ],
                "listeners": "1957174",
                "mbid": "b7539c32-53e7-4908-bda3-81449c367da6",
```


Making our first API request

Let's see the returned data

- Let's look at the '@attr' (attributes) key by itself:

```
jprint(r.json()['artists']['@attr'])
```


```
{
    "page": "1",
    "perPage": "50",
    "total": "2901036",
    "totalPages": "58021"
}
```


Making our first API request

The structure of the JSON response is a dictionary with a single artists key, containing:

- an @attr key containing a number of attributes about the response
- an artist key containing a list of artist objects

- Results can be controlled by the pagination technique using two optional parameters
  - limit: The number of results to fetch per page (defaults to 50).
  - page: Which page of the results we want to fetch

- Example

```
# initialize list for results
results = []

# set initial page and a high total number
page = 1
total_pages = 99999


while page > total_pages:
    # simplified request code for this example
    r = request.get("endpoint_url", params={"page": page})

    # append results to list
    results.append(r.json())

    # increment page
    page += 1
```

Working with Paginated Data

There are almost three million total artists in the results of this API endpoint, and we're being showing the first 50 artists in a single 'page'. This technique of spreading the results over multiple pages is called **pagination**

- The easiest way to perform rate limiting is to use Python `time.sleep()` function.

```
import time

print("one")
time.sleep(0.25)
print("two")
```

Working with Paginated Data

Rate limiting is the way to avoid getting banned by using code to limit the number of times per second

```python
import time
from IPython.core.display import clear_output

responses = []

page = 1
total_pages = 99999 # this is just a dummy number so the loop starts

while page < = total_pages:
    payload = {
        'method': 'chart.gettopartists',
        'limit': 500,
        'page': page
    }

    # print some output so we can see the status
    print("Requesting page {}/{}".format(page, total_pages))
    # clear the output to make things neater
    clear_output(wait = True)

    # make the API call
    response = lastfm_get(payload)

    # if we get an error, print the response and halt the loop
    if response.status_code != 200:
        print(response.text)
        break

    # extract pagination info
    page = int(response.json()['artists']['@attr']['page'])
    total_pages = int(response.json()['artists']['@attr']['totalPages'])

    # append response
    responses.append(response)

    # if it's not a cached result, sleep
    if not getattr(response, 'from_cache', False):
        time.sleep(0.25)

    # increment the page number
    page += 1</code>
```

```
Requesting page 5803/5803
```

```python
def lastfm_get(payload):
    # define headers and URL
    headers = {'user-agent': USER_AGENT}
    url = 'http://ws.audioscrobbler.com/2.0/'

    # Add API key and format to the payload
    payload['api_key'] = API_KEY
    payload['format'] = 'json'

    response = requests.get(url, headers=headers, params=payload)
    return response
```

Working with Paginated Data

Let's try

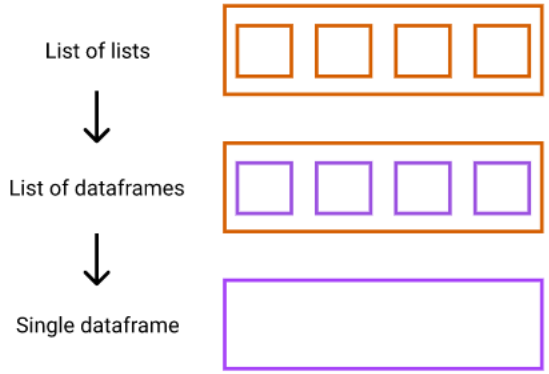
```
import pandas as pd

r0 = responses[0]
r0_json = r0.json()
r0_artists = r0_json['artists']['artist']
r0_df = pd.DataFrame(r0_artists)
r0_df.head()
```

Processing Data

Let's use pandas to look at the data from the first response in our responses list

- Responses are a list of lists
- Transform it to a list of dataframe then
- Use the `pandas.concat()` function to turn the list of dataframes into a single dataframe




```
frames = [pd.DataFrame(r.json()['artists']['artist']) for r in responses]
artists = pd.concat(frames)
artists.head()
```

Processing Data

Combining each response

```
r = lastfm_get({
    'method': 'artist.getTopTags',
    'artist':  'Lana Del Rey'
})

jprint(r.json())
```


```
{
    "toptags": {
        "@attr": {
            "artist": "Lana Del Rey"
        },
        "tag": [
            {
                "count": 100,
                "name": "female vocalists",
                "url": "https://www.last.fm/tag/female+vocalists"
            },
            {
                "count": 93,
                "name": "indie",
                "url": "https://www.last.fm/tag/indie"
            },
            {
                "count": 88,
                "name": "indie pop",
                "url": "https://www.last.fm/tag/indie+pop"
            },
            {
                "count": 80,
                "name": "pop",
                "url": "https://www.last.fm/tag/pop"
            },
            {
                "count": 67,
                "name": "alternative",
                "url": "https://www.last.fm/tag/alternative"
            },
            {
                "count": 14,
```


Augmenting the Data Using a Second Last.fm API Endpoint

In order to make our data more interesting, let's use another last.fm API endpoint to add some extra data about each artist

- Let's use list comprehension to create a list of the top three tag names

```
tags = [t['name'] for t in r.json()['toptags']['tag'][:3]]
tags
```

```
['female vocalists', 'indie', 'indie pop']
```

- use the `str.join()` method to turn the list into a string

```
', '.join(tags)
```

```
'female vocalists, indie, indie pop'
```

Augmenting the Data Using a Second Last.fm API Endpoint

We're really only interested in the tag names, and then only the most popular tags

```python
def lookup_tags(artist):
    response = lastfm_get({
        'method': 'artist.getTopTags',
        'artist':  artist
    })

    # if there's an error, just return nothing
    if response.status_code != 200:
        return None

    # extract the top three tags and turn them into a string
    tags = [t['name'] for t in response.json()['toptags']['tag'][:3]]
    tags_str = ', '.join(tags)

    # rate limiting
    if not getattr(response, 'from_cache', False):
        time.sleep(0.25)
    return tags_str
```

```python
lookup_tags("Billie Eilish")
```

```
'pop, indie pop, indie'
```

```python
from tqdm import tqdm
tqdm.pandas()

artists['tags'] = artists['name'].progress_apply(lookup_tags)
```

Augmenting the Data Using a Second Last.fm API Endpoint

Let's create a function that uses this logic in the previous slide to return a string of the most popular tag for any artist, which we'll use later to apply to every row in our dataframe

```
artists[["playcount", "listeners"]] = artists[["playcount", "listeners"]].astype(int)
```

```
artists = artists.sort_values("listeners", ascending=False)
artists.head(10)
```

```
artists.to_csv('artists.csv', index=False)
```

Finalizing and Exporting the Data

Work on data type before export

Summary

We have learned …

- What is an API
- Types of requests and response codes
- How to make a get request
- How to make a request with parameters
- How to display and extract JSON data from an API
- How to authenticate with an API using an API key
- How to use pagination to collect larger responses from an API endpoint
- How to use rate-limiting to stay within the guidelines of an API