Principles/Social Media Mining

CIS 600

Week 3: Twitter APIs, Part 1: Search

Edmund Yu, PhD
Associate Teaching Professor
esyu@syr.edu

September 8, 10, 2020

Quiz

- What are the **six** categories of social media, according to Kaplan & Haenlein's paper "Users of the world, unite! The challenges and opportunities of social media"?

Social Media Classified

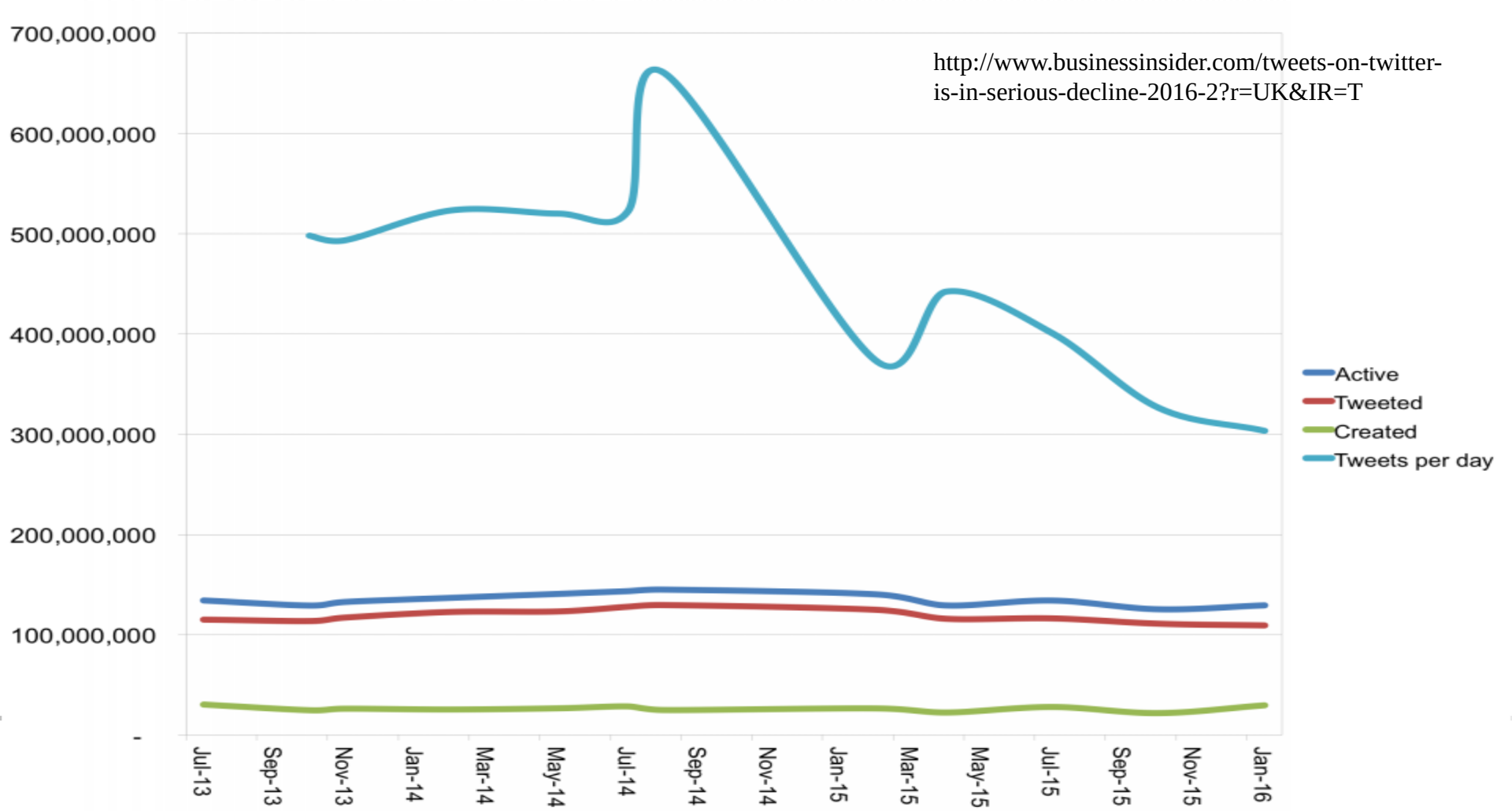
Twitter

Self-presentation/ Self-disclosure	Social presence/ Media richness: Low	Medium	High
High	Blogs	Social networking sites (e.g., Facebook)	Virtual social worlds (e.g., Second Life)
Low	Collaborative projects (e.g., Wikipedia)	Content communities (e.g., YouTube)	Virtual game worlds (e.g., World of Warcraft)

Kaplan, Andreas M., Haenlein, Michael, (2010), Users of the world, unite! The challenges and opportunities of social media, Business Horizons, Vol. 53, Issue 1

Twitter APIs

- Twitter is an information network and communication mechanism that produces more than 300+ million tweets a day



Twitter APIs

- The Twitter platform offers access to that all those tweets (corpus of data) via their APIs

- It's important to note that the Twitter APIs are constantly evolving, and developing on the Twitter platform is not a one-off event.

https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/api-reference/get-search-tweets

Search Tweets

Overview Quick start Guides FAQ API reference

API reference contents

Standard search API

Premium search APIs

Enterprise search APIs

Please note:

We launched a new version of the standard Search Tweets endpoint as part of Twitter API v2: Early Access. If you are currently using any of these endpoints, you can use our migration materials to start working with the newer endpoint.

Standard search API

Returns a collection of relevant Tweets matching a specified query.

Please note that Twitter's search service and, by extension, the Search API is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

Twitter Search API

- The Search API is designed for users to query Twitter content, including:
  - Finding tweets with specific keywords
  - Finding tweets referencing a specific user
  - Finding tweets from a particular user
- This API will also provide you access to data around **Trends**.
- If your application requires repeated Search API polling at high speed, and are hitting rate limits, then you should be working with the Streaming API (See next 2 slides)
- **Documentation**: https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets

https://developer.twitter.com/en/docs/twitter-api/v1/tweets/filter-realtime/overview

Filter realtime Tweets

A related endpoint is available in Labs. If you haven't done so, we invite you to join Labs to provide feedback. For more details, see Filtered stream.

The Twitter API platform offers two options for streaming realtime Tweets. Each option offers a varying number of filters and filtering capabilities - see the below summary for more details:

API	Category	Number of filters	Filtering operators	Rule management
statuses/filter	Standard	400 keywords, 5,000 userids and 25 location boxes	Standard operators	One filter rule on one allowed connection, disconnection required to adjust rule
PowerTrack	Enterprise	Up to 250,000 filters per stream, up to 2,048 characters each	Premium operators	Thousands of rules on a single connection, no disconnection needed to add/remove rules using Rules API

Used to be called the REST API

Follow, search, and get users

Overview API reference

API reference contents

https://developer.twitter.com/en/docs/twitter-api/v1/accounts-and-users/follow-search-get-users/api-reference/get-followers-ids

GET followers/ids

Returns a cursored collection of user IDs for every user following the specified user.

At this time, results are ordered with the most recent following first — however, this ordering is subject to unannounced change and eventual consistency issues. Results are given in groups of 5,000 user IDs and multiple "pages" of results can be navigated through using the `next_cursor` value in subsequent requests. See Using cursors to navigate collections for more information.

Friends and followers	POST friendships	Get user info
• GET followers/ids	• POST friendships/create	• GET users/lookup
• GET followers/list	• POST friendships/destroy	• GET users/search
• GET friends/ids	• POST friendships/update	• GET users/show
• GET friends/list		
• GET friendships/incoming		
• GET friendships/lookup		
• GET friendships/no_retweets/ids		
• GET friendships/outgoing		
• GET friendships/show		

For more details, please see the individual endpoint information within the API reference section.

Terminology

To avoid confusion around the term "friends" and "followers" with respect to the API endpoints, below is a definition of each:

Friends - we refer to "friends" as the Twitter users that a specific user follows (e.g., following). Therefore, the `GET friends/ids` endpoint returns a collection of user IDs that the specified user follows.

Followers - refers to the Twitter users that follow a specific user. Therefore, making a request to the `GET followers/ids` endpoint returns a collection of user IDs for every user following the specified user.

https://developer.twitter.com/en/docs/twitter-api/tools-and-libraries

Documentation

Search the docs

Twitter API

Getting started
Tutorials
Tools and libraries
Migrate
API reference Index

Twitter API v2 Early Access

Fundamentals
Tweets
Users

Tools and libraries

The Twitter teams maintain a set of official libraries and SDKs, listed here.

Can't find what you're looking for? Let us know or vote on existing requests via our feedback platform

To see tools and libraries for the Twitter Ads API or Twitter for Websites, please visit their respective pages:

- Twitter Ads API tools
- Twitter for Websites tools

Official tools and libraries

JavaScript / Node.js	Clients	SDKs / Libraries	Tools

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

Community tools and libraries

These are some of the many community-supported libraries that cover the Twitter API, across several programming languages and platforms.

The libraries listed here should support most features of the Standard API `v1.1`, unless otherwise noted - check with the authors for details, and for support.

If you have built a library that supports Twitter API `v2`, please let us know about it via our community forums, for possible addition to this page. You can also use the forums to let us know about any changes to the listings on this page.

If you're missing a library or tool for your favorite programming language, let us know via the feedback platform, where you can also vote for ideas, or get inspired to build and submit something new.

.NET (ASP, C#, VB)

Clients	SDKs / Libraries	Tools
ASPTwitter by @timacheson	SocialOpinion by @jamie_maguire1 (`Labs`, preliminary `v2` support)	
	CoreTweet	

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

Community tools and libraries

These are some of the many community-supported libraries that cover the Twitter API, across several programming languages and platforms.

The libraries listed here should support most features of the Standard API `v1.1`, unless otherwise noted - check with the authors for details, and for support.

If you have built a library that supports Twitter API `v2`, please let us know about it via our community forums, for possible addition to this page. You can also use the forums to let us know about any changes to the listings on this page.

If you're missing a library or tool for your favorite programming language, let us know via the feedback platform, where you can also vote for ideas, or get inspired to build and submit something new.

.NET (ASP, C#, VB)

Clients	SDKs / Libraries	Tools
ASPTwitter by @timacheson	SocialOpinion by @jamie_maguire1 (`Labs`, preliminary `v2` support)	
	CoreTweet	

Documentation
Search the docs
Twitter API
Getting started
Tutorials
Tools and libraries
Migrate
API reference Index
Twitter API v2
Early Access
Fundamentals
Tweets
Users

Python

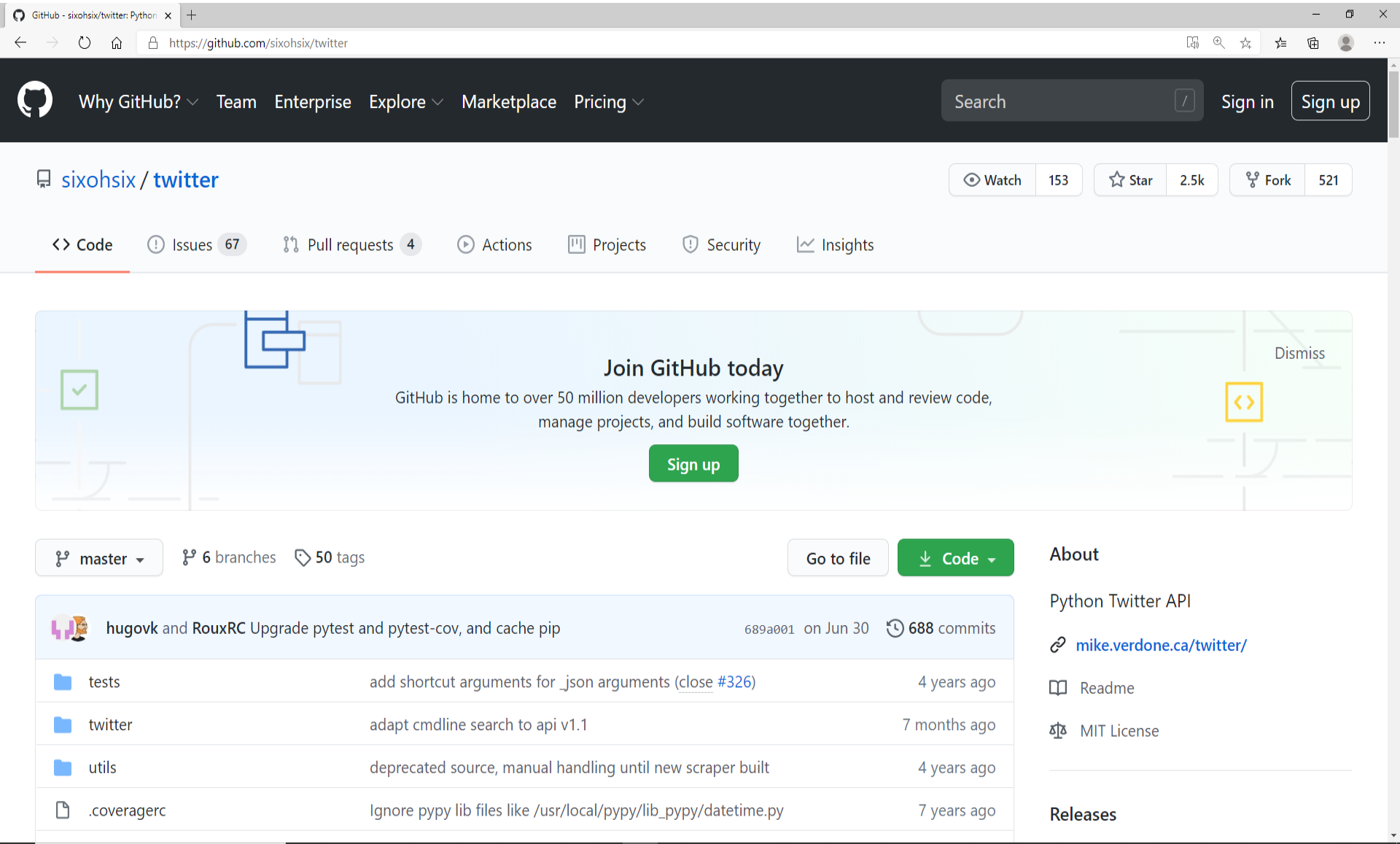
Clients	SDKs / Libraries	Tools
TwitterSearch by @ckoepp	**TwitterAPI** by @geduldig. Includes **Premium v1.1** and **Labs** support.	**twitter** by the Python Twitter Tools developer team. Includes a CLI, IRC tools, and minimalist API library for **v1.1**
tweetget by @gnascimento. A search client library for **Premium v1.1**.	**python-twitter** by the Python-Twitter Developers	
	tweepy by the tweepy Developers	
	twython by @ryanmcgrath and @mikehelmick	

R

Clients	SDKs / Libraries	Tools
	rtweet by @kearnymw. Includes **Premium v1.1** search	**rtweetXtras** Helper functions for rTweet, by @Arf9999

Our Python Library for Twitter API

https://github.com/sixohsix/twitter



Get The Latest Twitter Package

- Install the Python twitter library:

  pip install twitter (or **easy_install twitter**)

- Current version is: twitter 1.18.0

- If you already have an older version of the Python twitter library installed:

  pip install -- upgrade twitter

  easy_install -- upgrade twitter

- Note: easy_install or pip should be run outside Python

twitter 1.18.0

```
pip install twitter
```

Latest version

Released: Oct 20, 2017

An API and command-line toolset for Twitter (twitter.com)

Navigation

Project description

Release history

Download files

Project links

Homepage

Statistics

View statistics for this project via

Project description

Python Twitter Tools

build passing coverage 30%

The Minimalist Twitter API for Python is a Python API for Twitter, everyone's favorite Web 2.0 Facebook-style status updater for people on the go.

Also included is a Twitter command-line tool for getting your friends' tweets and setting your own tweet from the safety and security of your favorite shell and an IRC bot that can announce Twitter updates to an IRC channel.

For more information, after installing the `twitter` package:

- import the `twitter` package and run `help()` on it
- run `twitter -h` for command-line tool help

Administrator: Command Prompt - python

```
C:¥WINDOWS¥system32>python
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import twitter
>>> help(twitter)
Help on package twitter:

NAME
    twitter - The minimalist yet fully featured Twitter API and Python toolset.

DESCRIPTION
    The Twitter and TwitterStream classes are the key to building your own
    Twitter-enabled applications.


    The Twitter class
    -----------------

    The minimalist yet fully featured Twitter API class.

    Get RESTful data by accessing members of this class. The result
    is decoded python objects (lists and dicts).

    The Twitter API is documented at:

      https://dev.twitter.com/overview/documentation

    The list of most accessible functions is listed at:
```

Administrator: Command Prompt - python -mpydoc twitter

```
C:¥WINDOWS¥system32>
C:¥WINDOWS¥system32>
C:¥WINDOWS¥system32>python -mpydoc twitter
Help on package twitter:

NAME
    twitter - The minimalist yet fully featured Twitter API and Python toolset.

DESCRIPTION
    The Twitter and TwitterStream classes are the key to building your own
    Twitter-enabled applications.


    The Twitter class
    -----------------

    The minimalist yet fully featured Twitter API class.

    Get RESTful data by accessing members of this class. The result
    is decoded python objects (lists and dicts).

    The Twitter API is documented at:

      https://dev.twitter.com/overview/documentation

    The list of most accessible functions is listed at:

      https://dev.twitter.com/rest/public
```

Example 1-1. Authorizing an application to access Twitter account data

```
import twitter

# Go to https://developer.twitter.com/en/apps to create an app and get values
# for these credentials, which you'll need to provide in place of these
# empty string values that are defined as placeholders.
# See https://developer.twitter.com/en/docs/authentication/overview
# for more information on Twitter's OAuth implementation.

CONSUMER_KEY = ''
CONSUMER_SECRET = ''
OAUTH_TOKEN = ''
OAUTH_TOKEN_SECRET = ''
```

OAuth

- OAuth stands for "open authorization"
  - It provides a means for users to authorize an application to access their account data through an API without the users needing to hand over sensitive credentials such as a username and password combination.
  - It's a specification that has wide applicability in any context in which users would like to authorize an application to take certain actions on their behalf.
  - Platforms such as Twitter, Facebook, LinkedIn, Google+ (defunct), etc. and the vast utility of 3rd party applications that are developed on these social web platforms have all adopted OAuth as a common means of opening up their platforms.
  - See also: **https://en.wikipedia.org/wiki/OAuth**

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Community portal
Recent changes
Upload file

Tools
What links here

OAuth

From Wikipedia, the free encyclopedia

For MediaWiki's (the software used by Wikipedia) OAuth support, see mw:Help:OAuth

OAuth is an open standard for access delegation, commonly used as a way for Internet users to grant websites or applications access to their information on other websites but without giving them the passwords.[1] This mechanism is used by companies such as Amazon,[2] Google, Facebook, Microsoft and Twitter to permit the users to share information about their accounts with third party applications or websites.

The OAuth logo, designed by American blogger Chris Messina

Generally, OAuth provides clients a "secure delegated access" to server resources on behalf of a resource owner. It specifies a process for resource owners to authorize third-party access to their server resources without sharing their credentials. Designed specifically to work with Hypertext Transfer Protocol (HTTP), OAuth essentially allows access tokens to be issued to third-party clients by an authorization server, with the approval of the resource owner. The third party then uses the access token to access the protected resources hosted by the resource server.[3]

OAuth is a service that is complementary to and distinct from OpenID. OAuth is unrelated to OATH, which is a *reference*

Twitter. It's what's happening / T

https://twitter.com/?logout=1599313169363

Follow your interests.

Hear what people are talking about.

Join the conversation.

Phone, email, or username

edmundyu1001

Password

••••••••••

Log in

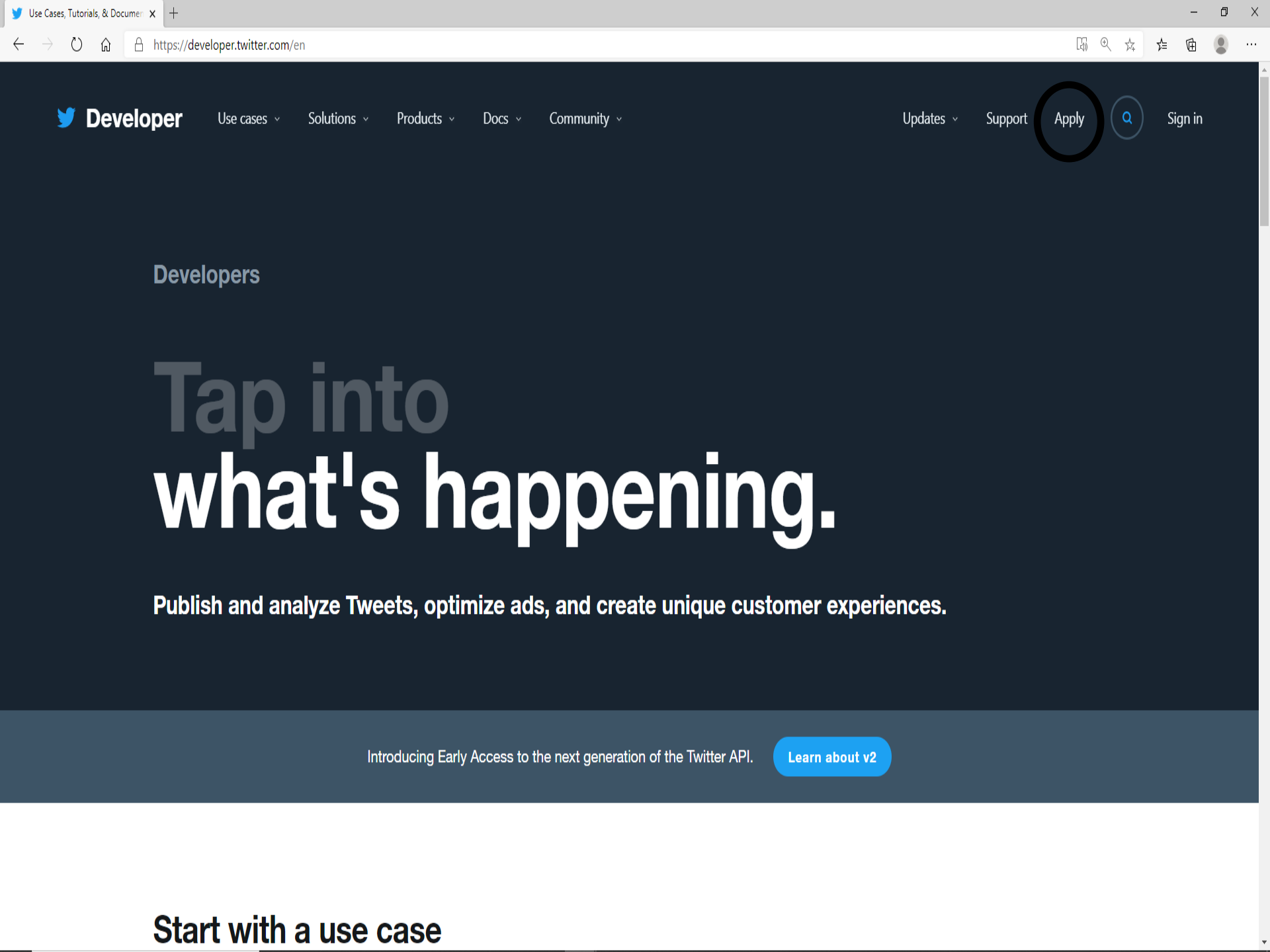
Forgot password?

See what's happening in the world right now

Join Twitter today.

Sign up

Log in


Use Cases, Tutorials, & Documer
https://developer.twitter.com/en
Developer
Use cases
Solutions
Products
Docs
Community
Updates
Support
Apply
Sign in
Developers
Tap into
what's happening.
Publish and analyze Tweets, optimize ads, and create unique customer experiences.
Introducing Early Access to the next generation of the Twitter API.
Learn about v2
Start with a use case

CIS.600.M007.FALL20.Prncpls:SocialMedia&DataMining (14237.1211)
Announcements
Information
Content
Discussions
Assignments
Groups
Tools

COURSE MANAGEMENT
Control Panel
Content Collection
Course Tools

Information

Build Content | Assessments | Tools | Partner Content

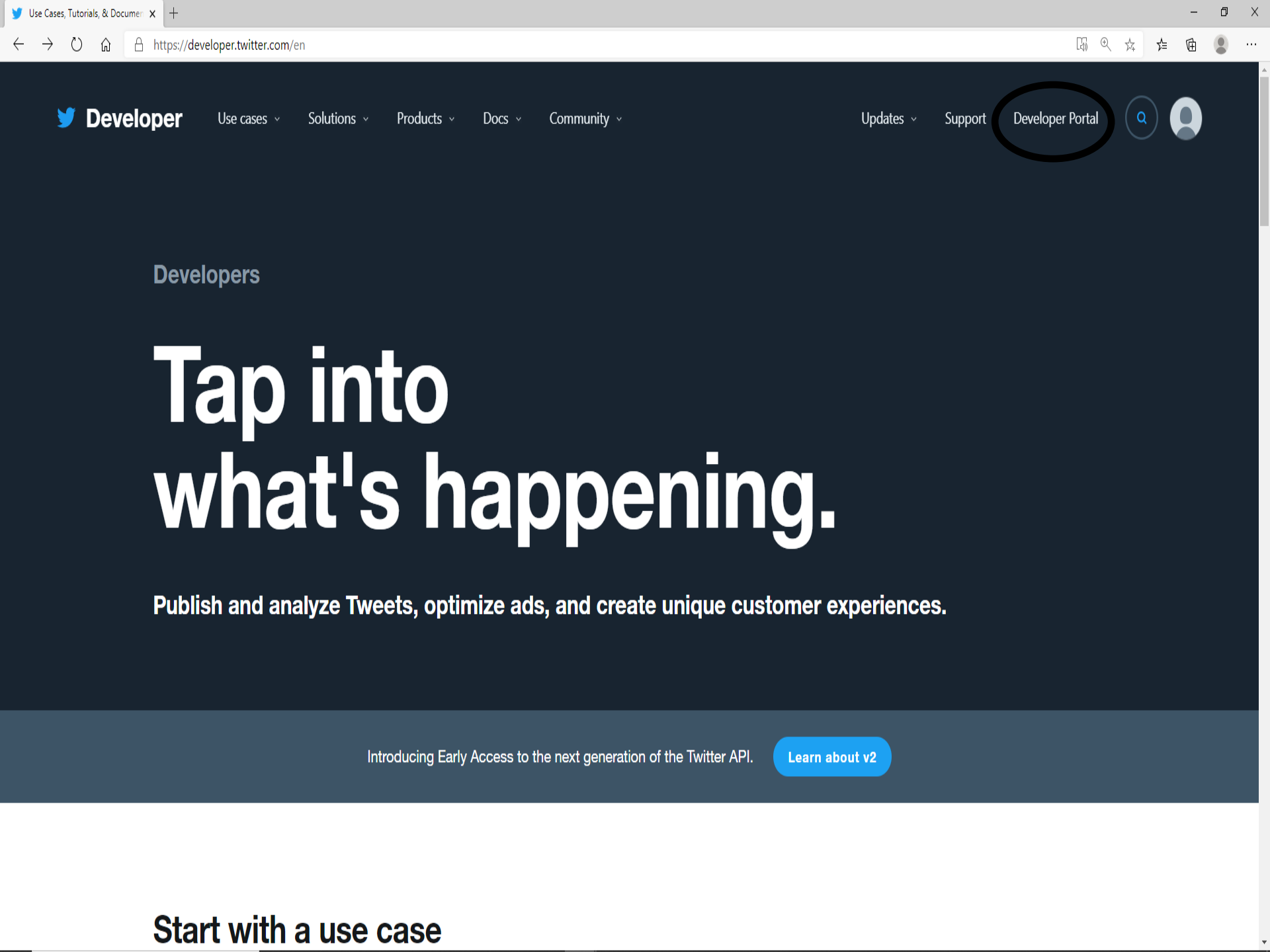
Info needed for applying for a Twitter Develop account

My answers to the following 2 questions that you will be asked:
1. Are you planning to analyze Twitter data?

Please describe how you will analyze Twitter data including any analysis of Tweets or Twitter users.
Answer: This application is used to run programs used in my CIS 400/600/700 classes. Those programs help us learn how to use Twitter APIs, including the Search API, the REST API and the Streaming API, to collect and analyze user information as well as tweets, totally for educational purposes. (Please feel free to adapt.)

2. Will your app use Tweet, Retweet, like, follow, or Direct Message functionality?

Please describe your planned use of these features.
Answer: No. (Add any additional info as you see fit.)


Use Cases, Tutorials, & Documer
https://developer.twitter.com/en
Developer
Use cases
Solutions
Products
Docs
Community
Updates
Support
Developer Portal
Developers
Tap into what's happening.
Publish and analyze Tweets, optimize ads, and create unique customer experiences.
Introducing Early Access to the next generation of the Twitter API.
Learn about v2
Start with a use case

Dashboard — Twitter Developer

https://developer.twitter.com/en/dashboard

Developer | Use cases | Solutions | Products | Docs | Community | Updates | Support | Dashboard | Edmund Yu

- Get started
- Subscriptions
- Apps
- Dev environments
- Billing

It's here.
Early Access to the new Twitter API. Intuitive look. New features. **Start now**

Premium Products

Search Tweets API: 30-Days

Tier: Sandbox | Upgrade

Usage | About

You must first set up a dev environment before accessing an endpoint and viewing usage.

Requests this month	Projected requests	Tweets this month	Projected tweets
0 of 250	0	0 of 25k	0

Requests Usage

10

Tweets Usage

10

PRIVACY | COOKIES | TWITTER TERMS & CONDITIONS | DEVELOPER POLICY & TERMS | © 2020 TWITTER INC. | FOLLOW @TWITTERDEV | SUBSCRIBE TO DEVELOPER NEWS

https://developer.twitter.com/en/apps

Apps

Create an app

	App ID	
CIS700	2251914	Details
CSE400	5715229	Details
CIS700CSE791	6875271	Details
NoName1001	16084184	Details

Understanding apps

What is an app?

Why register an app?

Which products require an API key?

App details

The following app details will be visible to app users and are required to generate the API keys needed to authenticate Twitter developer products.

App name (required)

Maximum characters: **32**

Application description (required)

Share a description of your app. This description will be visible to users so this is a good place to tell them what your app does.

Please be detailed.

App details

Details and URLs

Edit

App icon

App icon is default, click edit to upload.

App Name

NoName1001

Description

This application is used to run programs used in my CIS 400/600/700 classes.

Website URL

https://www.syr.edu

Sign in with Twitter

Disabled

Sign in with Twitter

Disabled

Callback URL

None

Terms of service URL

None

Privacy policy URL

None

Organization name

Syracuse University

Organization website URL

https://www.syr.edu

App usage

This application is used to run programs used in my CIS 400/600/700 classes. Those programs access Twitter APIs, including the Search API, the REST API and the Streaming API, to collect and analyze user information as well as tweets.

Keys and tokens

Keys, secret keys and access tokens management.

Consumer API keys

Regenerate

API key: lTJckzBSeQhE2i3BfLvaomBur

API secret key: HGXVRYPAQcMjCwaj77BdY1pwCQsB5YKK1H7Zhxhj4GyQmR2pg1

Access token & access token secret

Revoke Regenerate

We only show your access token and secret when you first generate it in order to make your account more secure. You can revoke or regenerate them at any time, which will invalidate your existing tokens.

Access token: xxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Last generated: Jan 31, 2020

Access token secret: xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx

Permissions

Edit

Changes to the app permissions will be reflected in access tokens generated after the permissions are saved. You will need to regenerate existing access tokens to alter permissions levels.

Access permission

Read, write, and Direct Messages

Additional permissions

None

Example 1-1. Authorizing an application to access Twitter account data

```
import twitter

# Go to http://dev.twitter.com/apps/new to create an app and get values
# for these credentials, which you'll need to provide in place of these
# empty string values that are defined as placeholders.
# See https://developer.twitter.com/en/docs/basics/authentication/overview/oauth
# for more information on Twitter's OAuth implementation.

CONSUMER_KEY = '**********************'
CONSUMER_SECRET = '**************************'
OAUTH_TOKEN = '**********************'
OAUTH_TOKEN_SECRET = '**********************'
```

Example 1-1. Authorizing an application to access Twitter account data

```
import twitter

CONSUMER_KEY = '**********************'
CONSUMER_SECRET = '************************'
OAUTH_TOKEN = '**************************'
OAUTH_TOKEN_SECRET = '**********************'

auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                           CONSUMER_KEY,  CONSUMER_SECRET)
twitter_api = twitter.Twitter(auth=auth)

# Nothing to see by displaying twitter_api except that it's now a
# defined variable
print(twitter_api)
```

Twitter Cookbook – Chapter 9

Example 9-1. Accessing Twitter's API for development purposed

```
import twitter
def oauth_login():
    CONSUMER_KEY = '***'
    CONSUMER_SECRET = '***'
    OAUTH_TOKEN = '***'
    OAUTH_TOKEN_SECRET = '***'

    auth = twitter.oauth.OAuth(OAUTH_TOKEN, OAUTH_TOKEN_SECRET,
                               CONSUMER_KEY, CONSUMER_SECRET)
    twitter_api = twitter.Twitter(auth=auth)
    return twitter_api

# Sample usage
twitter_api = oauth_login()
```

Chapter 9. Twitter Cookbook

This cookbook is a collection of recipes for mining Twitter data. Each recipe is designed to solve a particular problem and to be as simple and atomic as possible so that multiple recipes can be composed into more complex recipes with minimal effort. Think of each recipe as being a building block that, while useful in its own right, is even more useful in concert with other building blocks that collectively constitute more complex units of analysis. Unlike the previous chapters, which contain a lot more prose than code, this one provides relatively little discussion and lets the code do more of the talking. The thought is that you'll likely be manipulating and composing the code in various ways to achieve your own particular objectives.

While most recipes involve little more than issuing a parameterized API call and post-processing the response into a convenient format, some recipes are even simpler (involving just a few lines of code),

9.1. Accessing Twitter's API for Development Purposes

9.1.1. Problem

You want to mine your own account data or otherwise gain quick and easy API access for development purposes.

9.1.2. Solution

Use the `twitter` package and the OAuth 1.0a credentials provided in the application's settings to gain API access to your own account without any HTTP redirects.

9.1.3. Discussion

Twitter implements OAuth 1.0a, an authorization mechanism that's expressly designed so that users can grant third parties access to their data without having to do the unthinkable—doling out their usernames and passwords. While you can certainly take advantage of Twitter's OAuth implementation for production situations in which you'll need users to authorize your application to access their accounts, you can also use the credentials in your application's settings to gain instant access for development purposes or to mine the data in your own account.

Retrieving Trends

```
# The Yahoo! Where On Earth ID for the entire world is 1.
# See http://bit.ly/2BGWJBU and http://bit.ly/2MsvwCQ

WORLD_WOE_ID = 1
US_WOE_ID = 23424977


# Prefix ID with the underscore for query string parameterization.
# Without the underscore, the twitter package appends the ID value
# to the URL itself as a special case keyword argument.

world_trends = twitter_api.trends.place(_id=WORLD_WOE_ID)
us_trends = twitter_api.trends.place(_id=US_WOE_ID)

print(world_trends)
print()
print(us_trends)
```

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

https://developer.twitter.com/en/docs/twitter-api/v1/trends/trends-for-location/api-reference/get-trends-place

Get trends near a location

Overview API reference

API reference contents

GET trends/place

GET trends/place

Returns the top 50 trending topics for a specific `WOEID`, if trending information is available for it.

The response is an array of `trend` objects that encode the name of the trending topic, the query parameter that can be used to search for the topic on Twitter Search, and the Twitter Search URL.

This information is cached for 5 minutes. Requesting more frequently than that will not return any more data, and will count against rate limit usage.

The `tweet_volume` for the last 24 hours is also returned for many trends if this is available.

Resource URL

Documentation
Search the docs
Twitter API
Getting started
Tutorials
Tools and libraries
Migrate
API reference Index
Twitter API v2
Early Access
Fundamentals
Tweets
Users

Parameters

Name	Required	Description	Default Value	Example
id	required	The Yahoo! Where On Earth ID of the location to return trending information for. Global information is available by using *1* as the *WOEID* .		*1*
exclude	optional	Setting this equal to *hashtags* will remove all hashtags from the trends list.		

Example Request

```
GET https://api.twitter.com/1.1/trends/place.json?id=1
```

Example Response


```
[
 {
   "trends": [
     {
       "name": "#ChainedToTheRhythm",
       "url": "http://twitter.com/search?q=%23ChainedToTheRhythm",
       "promoted_content": null,
       "query": "%23ChainedToTheRhythm",
       "tweet_volume": 48857
     },
```

Verizon completes its $4.48 billion acquisition of Yahoo; Marissa Mayer leaves with $23 million

- Verizon says it has completed its $4.48 billion acquisition of Yahoo.
- Yahoo CEO Marissa Mayer will resign.
- The assets acquired from Yahoo will be combined with AOL brands under a new subsidiary called Oath, headed by former AOL CEO Tim Armstrong.

Arjun Kharpal | @ArjunKharpal
Published 8:00 AM ET Tue, 13 June 2017 | Updated 11:36 AM ET Tue, 13 June 2017

Yahoo! Where On Earth ID(WOEID)

- A way to represent a place in a permanent and language-independent way in Twitter Trends API
- Yahoo!'s **Where On Earth ID**s, or **WOEID**s, are 32-bit identifiers that are “unique and non-repetitive”
  - If a location is assigned a WOEID, the WOEID assigned is never changed, and that particular WOEID is never given to another location.
- In addition, a WOEID has certain properties such as an implied hierarchy (e.g. a particular city is in a particular state), and a taxonomy of categories (e.g. a “country”, “county” or “town").

Yahoo! Where On Earth ID

1 ("Terra")

- 23424775 ("Canada" - Country)
- 23424803 ("Ireland" - Country)
- 23424975 ("United Kingdom" - Country)
  - 24554868 ("England" - State)
    - 23416974 ("Greater London" - County)
      - 44418 ("London" - Town)
- 23424900 ("Mexico" - Country)
- 23424768 ("Brazil" - Country)
  - 2344868 ("Sao Paulo" - State)
    - 12582314 ("São Paulo" - County)
      - 455827 ("Sao Paulo" - Town)
- 23424977 ("United States" - Country)
  - 2347572 ("Illinois" - State)
    - 12588093 ("Cook" - County)
      - 2379574 ("Chicago" - Town)

WOEID

From Wikipedia, the free encyclopedia

A **WOEID** (Where On Earth IDentifier) is a unique 32-bit reference identifier, originally defined by GeoPlanet and now assigned by Yahoo!, that identifies any feature on Earth.[1][2] In 2009, Yahoo! released GeoPlanet's WOEID data to the public,[3] with the last release on 1 June 2012, after which Yahoo! decided to cease making the data downloadable until they "determine a better way to surface the data as a part of the service".[4]

WOEIDs are used by a number of other projects, including Flickr[5] OpenStreetMap,[6], Twitter[7], and zWeather.[8]

References [edit]

1. ^ "Yahoo! WOEID Lookup". Zourbuth Project.
2. ^ "WOEID Lookup". Find your WOEID.
3. ^ "Free geo data solutions compared: GeoNames.org vs. Yahoo! GeoPlanet". Cosmo Code. 24 January 2010.
4. ^ "Yahoo! GeoPlanet™ Data". Yahoo!. Archived from the original on 25 June 2014.
5. ^ "Flickr Services: Flickr API: flickr.places.getInfo". *www.flickr.com*. Retrieved 2016-05-01.
6. ^ "Key:woeid - OpenStreetMap Wiki". *wiki.openstreetmap.org*. Retrieved 2016-05-01.
7. ^ "Twitter Developer Documentation". *dev.twitter.com*. Retrieved 2017-06-24.
8. ^ "zWeatherFeed - Weather Feed Using WOEID Locations". Zazar.

Categories: Geographic object identifiers | Geocodes

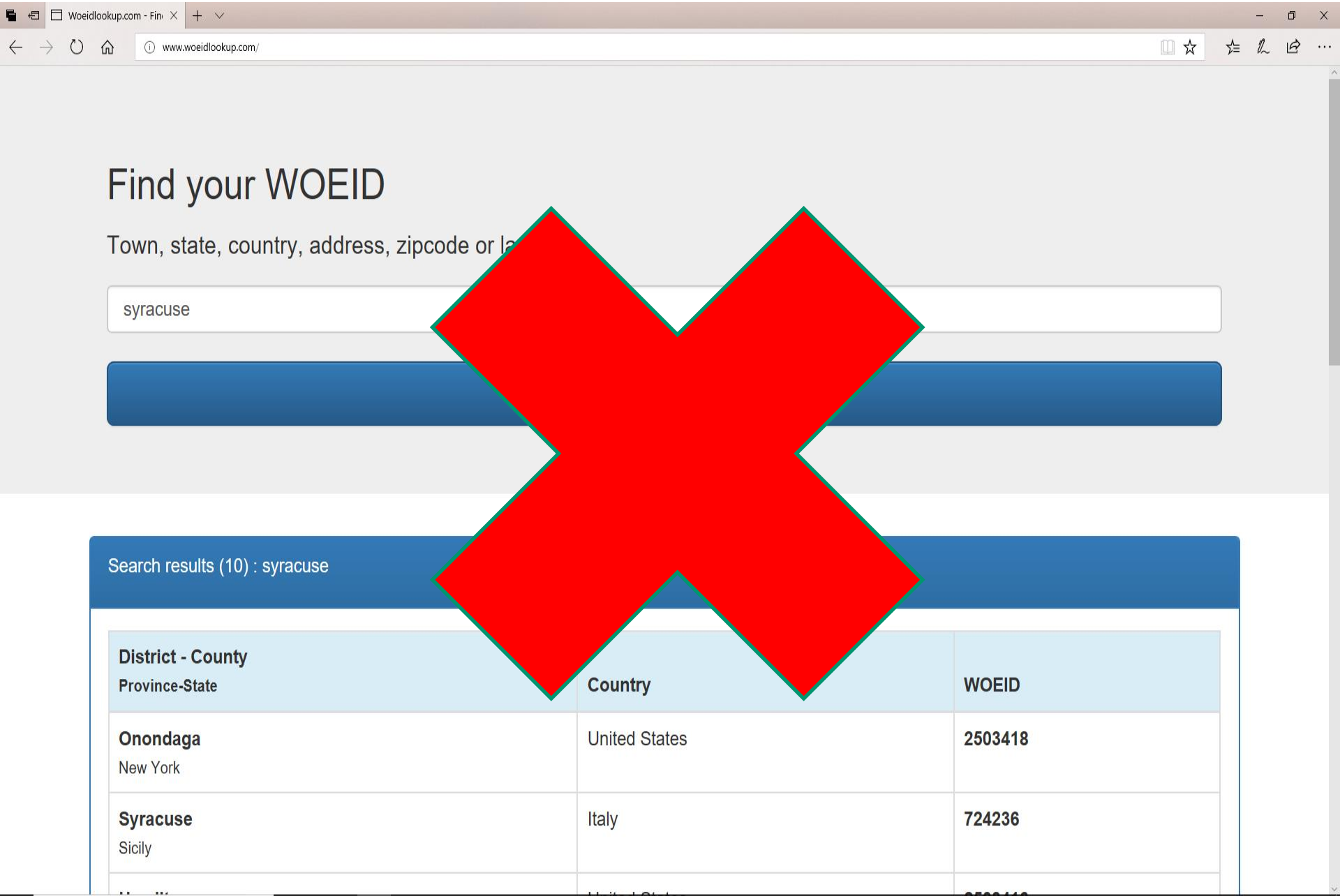
Wikipedia
The Free Encyclopedia

Main page
Contents
Featured content
Current events
Random article
Donate to Wikipedia
Wikipedia store

Interaction
Help
About Wikipedia
Community portal
Recent changes
Contact page

Tools
What links here
Related changes
Upload file
Special pages
Permanent link

Where On Earth Can I Find My ID?



Yahoo WOEID Lookup

woeid.rosselliot.co.nz/lookup/syracuse

http://woeid.rosselliot.co.nz/

woeid lookup

where on earth identifiers

Hire Me!

I'm Ross. I developed this site.

Contact Ross

WOEID Lookup makes it really easy to find [illegible] on Ea[illegible] address or zip code. It's also good for loo[illegible] discover that there are twenty-eight places nam[illegible] Weird, but interesting.

Search Place

Town, state, country, address, zipcode

syracuse

Look[illegible]

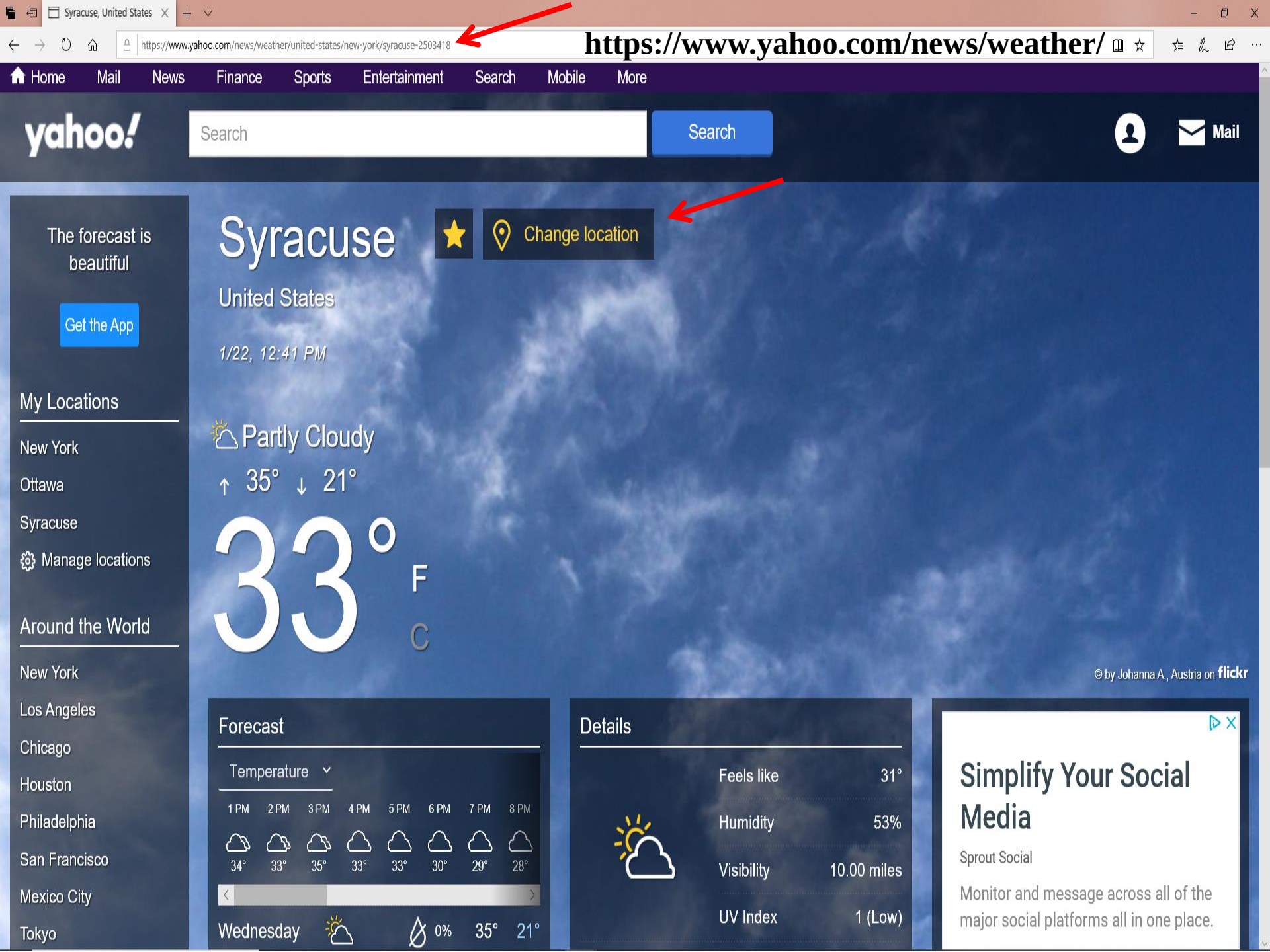
Place Info

Syracuse

Syracuse, New York
United States

Date / Time

2019-01-26 / 10:29
or
26 Jan 2019 / 10:29 am


Syracuse, United States
https://www.yahoo.com/news/weather/united-states/new-york/syracuse-2503418
https://www.yahoo.com/news/weather/
Home Mail News Finance Sports Entertainment Search Mobile More
yahoo!
Search
Search
Mail
The forecast is beautiful
Get the App
My Locations
New York
Ottawa
Syracuse
Manage locations
Around the World
New York
Los Angeles
Chicago
Houston
Philadelphia
San Francisco
Mexico City
Tokyo
Syracuse
Change location
United States
1/22, 12:41 PM
Partly Cloudy
↑ 35° ↓ 21°
33° F C
© by Johanna A., Austria on flickr
Forecast
Temperature
1 PM 2 PM 3 PM 4 PM 5 PM 6 PM 7 PM 8 PM
34° 33° 35° 33° 33° 30° 29° 28°
Wednesday 0% 35° 21°
Details
Feels like 31°
Humidity 53%
Visibility 10.00 miles
UV Index 1 (Low)
Simplify Your Social Media
Sprout Social
Monitor and message across all of the major social platforms all in one place.

Display Results as Pretty-Printed JSON

```
import json


print(json.dumps(world_trends, indent=1))
print()
print(json.dumps(us_trends, indent=1))
```

The JSON Module (Review)

- JSON (JavaScript Object Notation, http://json.org) is a subset of JavaScript syntax used as a lightweight data interchange format:

dump(obj, **fp**, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, indent=None, sort_keys=False, separators=None, encoding='utf-8', default=None, **kw)

dumps(obj, skipkeys=False, ensure_ascii=True, check_circular=True, allow_nan=True, cls=None, **indent=None, sort_keys=False**, separators=None, encoding='utf-8', default=None, ****kw)**

load(fp, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

loads(s, encoding=None, cls=None, object_hook=None, parse_float=None, parse_int=None, parse_constant=None, object_pairs_hook=None, **kw)

The JSON Module

```
Administrator: Command Prompt - python
C:¥Mining-the-Social-Web-3rd-Edition-master¥notebooks>
C:¥Mining-the-Social-Web-3rd-Edition-master¥notebooks>python
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import json
>>> help(json)
Help on package json:

NAME
    json

DESCRIPTION
    JSON (JavaScript Object Notation) <http://json.org> is a subset of
    JavaScript syntax (ECMA-262 3rd edition) used as a lightweight data
    interchange format.

    :mod:`json` exposes an API familiar to users of the standard library
    :mod:`marshal` and :mod:`pickle` modules.  It is derived from a
    version of the externally maintained simplejson library.

    Encoding basic Python object hierarchies::

        >>> import json
        >>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
        '["foo", {"bar": ["baz", null, 1.0, 2]}]'
        >>> print(json.dumps("¥"foo¥bar"))
        "¥"foo¥bar"
        >>> print(json.dumps('¥u1234'))
        "¥u1234"
        >>> print(json.dumps('¥¥'))
        "¥¥"
        >>> print(json.dumps({"c": 0, "b": 0, "a": 0}, sort_keys=True))
        {"a": 0, "b": 0, "c": 0}
        >>> from io import StringIO
        >>> io = StringIO()
        >>> json.dump(['streaming API'], io)
        >>> io.getvalue()
```

json — JSON encoder and decoder

Source code: Lib/json/__init__.py

JSON (JavaScript Object Notation), specified by **RFC 7159** (which obsoletes **RFC 4627**) and by ECMA-404, is a lightweight data interchange format inspired by JavaScript object literal syntax (although it is not a strict subset of JavaScript [1]).

json exposes an API familiar to users of the standard library marshal and pickle modules.

Encoding basic Python object hierarchies:

```
>>> import json
>>> json.dumps(['foo', {'bar': ('baz', None, 1.0, 2)}])
'["foo", {"bar": ["baz", null, 1.0, 2]}]'
>>> print(json.dumps("\"foo\bar"))
"\"foo\bar"
>>> print(json.dumps('\u1234'))
"\u1234"
>>> print(json.dumps('\\'))
"\\"
>>> print(json.dumps({"c": 0, "b": 0, "a": 0}, sort_keys=True))
{"a": 0, "b": 0, "c": 0}
>>> from io import StringIO
>>> io = StringIO()
>>> json.dump(['streaming API'], io)
>>> io.getvalue()
'["streaming API"]'
```

How are trends determined?

Trends are determined by an algorithm and, by default, are tailored for you based on who you follow, your interests, and your location. This algorithm identifies topics that are popular now, rather than topics that have been popular for a while or on a daily basis, to help you discover the hottest emerging topics of discussion on Twitter.

You can choose to see trends that are not tailored for you by selecting a specific trends location on twitter.com, iOS, or Android (instructions below). Location trends identify popular topics among people in a specific geographic location.

Note: The number of Tweets that are related to the trends is just one of the factors the algorithm looks at when ranking and determining trends. Algorithmically, trends and hashtags are grouped together if they are related to the same topic. For instance, #MondayMotivation and #MotivationMonday may both be represented by #MondayMotivation.

What does the # sign mean?

You may notice that some trends have # sign before the word or phrase. This is called a hashtag and is included specifically in Tweets to mark them as relating to a topic, so that people can

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

https://developer.twitter.com/en/docs/twitter-api/v1/trends/trends-for-location/api-reference/get-trends-place

Get trends near a location

Overview API reference

API reference contents

GET trends/place

GET trends/place

Returns the top 50 trending topics for a specific `WOEID`, if trending information is available for it.

The response is an array of `trend` objects that encode the name of the trending topic, the query parameter that can be used to search for the topic on Twitter Search, and the Twitter Search URL.

This information is cached for 5 minutes. Requesting more frequently than that will not return any more data, and will count against rate limit usage.

The `tweet_volume` for the last 24 hours is also returned for many trends if this is available.

Resource URL

Display Names of the Trends Only

```
# An example for printing just the names of the trends

for trends in [i['trends'] for i in us_trends]:
    for names in [i['name'] for i in trends]:
        print(names)
```

Display Available Trends

```
print(json.dumps(twitter_api.trends.available(), indent=1))
```

Or

```
for i in twitter_api.trends.available():
    print(i['name'])
```

https://developer.twitter.com/en/docs/twitter-api/v1/trends/locations-with-trending-topics/api-reference/get-trends-available

GET trends/available

GET trends/available

Returns the locations that Twitter has trending topic information for.

The response is an array of "locations" that encode the location's `WOEID` and some other human-readable information such as a canonical name and country the location belongs in.

A `WOEID` is a Yahoo! Where On Earth ID.

Resource URL

`https://api.twitter.com/1.1/trends/available.json`

Resource Information

Response formats	JSON
Requires authentication?	Yes

Check If a Location Is Supported

```
location = 'Syraucse’

if location in [i['name'] for i in twitter_api.trends.available()]:
    print('Location: ' + location + ' is supported')
else:
    print('Location: ' + location + ' is NOT supported')
```

GET trends/closest

Returns the locations that Twitter has trending topic information for, closest to a specified location.

The response is an array of "locations" that encode the location's `WOEID` and some other human-readable information such as a canonical name and country the location belongs in.

A `WOEID` is a Yahoo! Where On Earth ID.

Resource URL

`https://api.twitter.com/1.1/trends/closest.json`

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	75

Display Trends for Closest Location

```
# Syracuse: 43.035198, -76.139297
print json.dumps(twitter_api.trends.closest(lat=43, long=-76),
    indent=1)
 [
  {
   "name": "Ottawa",
   "countryCode": "CA",
   "url": "http://where.yahooapis.com/v1/place/3369",
   "woeid": 3369,
   "parentid": 23424775,
   "placeType": {
    "code": 7,
    "name": "Town"
   },
   "country": "Canada"
  }
 ]
```

Get Latitude and Longitude

http://www.latlong.net/

To make a search, use the name of a place, city, state, or address, or click the location on the map to **find lat long coordinates**.

Place Name

Syracuse

Find

Add the country code for better results. Ex: London, UK

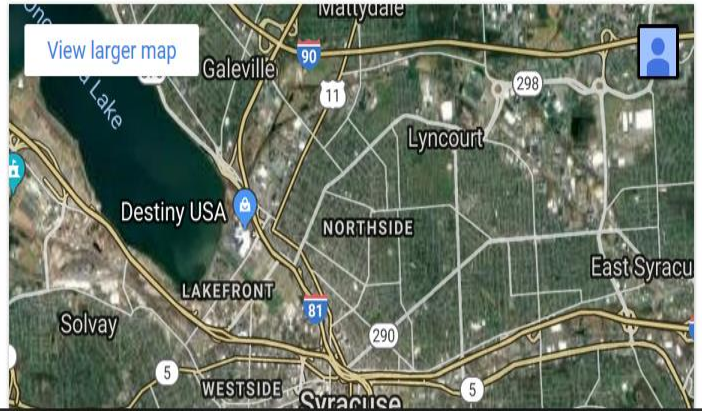
Latitude

43.048122

Longitude

-76.147423

Facebook Google+ Twitter



Trends - Twitter Cookbook

```
import json
import twitter
def twitter_trends(twitter_api, woe_id):
    # Prefix ID with the underscore for query string parameterization.
    # Without the underscore, the twitter package appends the ID value
    # to the URL itself as a special-case keyword argument.
    return twitter_api.trends.place(_id=woe_id)
# Sample usage
twitter_api = oauth_login()
# See https://developer.twitter.com/en/docs/trends/trends-for-location/api-reference/get-trends-place
# and http://www.woeidlookup.com to look up different Yahoo! Where On Earth IDs

WORLD_WOE_ID = 1
world_trends = twitter_trends(twitter_api, WORLD_WOE_ID)
print(json.dumps(world_trends, indent=1))

US_WOE_ID = 23424977
us_trends = twitter_trends(twitter_api, US_WOE_ID)
print(json.dumps(us_trends, indent=1))
```

9.3. Discovering the Trending Topics

9.3.1. Problem

You want to know what is trending on Twitter for a particular geographic area such as the United States, another country or group of countries, or possibly even the entire world.

9.3.2. Solution

Twitter's Trends API enables you to get the trending topics for geographic areas that are designated by a Where On Earth (WOE) ID, as defined and maintained by Yahoo!.

9.3.3. Discussion

A *place* is an essential concept in Twitter's development platform, and trending topics are accordingly constrained by geography to provide the best API possible for querying for trending topics (as shown in Example 9-3). Like all other APIs, it returns the trending topics as JSON data, which can be converted to standard Python objects and then manipulated with list comprehensions or similar techniques. This means it's fairly easy to explore the API responses. Try experimenting with a variety of WOE IDs to compare and contrast the trends from various geographic regions. For example, compare and

Computing the Intersection of Two Sets

```
world_trends_set = set([trend['name']
                        for trend in world_trends[0]['trends']])

us_trends_set = set([trend['name']
                     for trend in us_trends[0]['trends']])

common_trends = world_trends_set.intersection(us_trends_set)
#common_trends = world_trends_set & us_trends_set  # same
print(common_trends)

for t in common_trends: print(t)
```

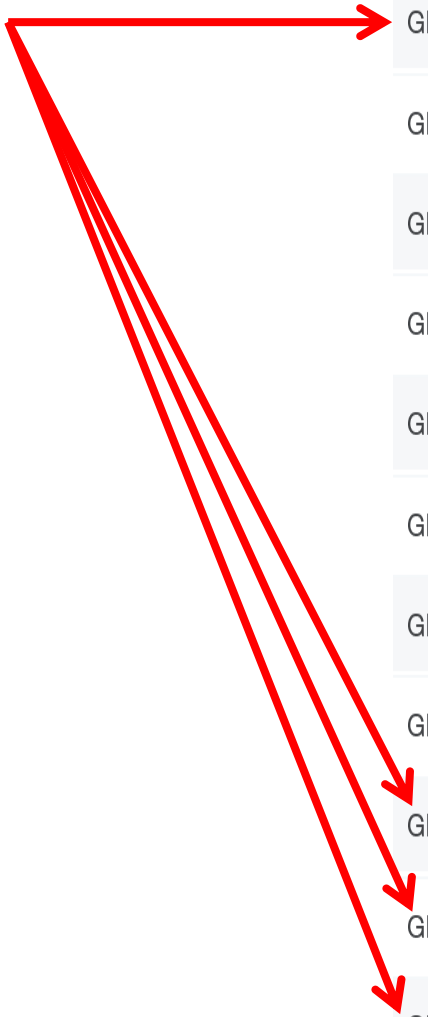
Sets (Review)

- A set is an unordered collection with no duplicate elements. Basic uses include membership testing and eliminating duplicate entries. Set objects also support mathematical operations like union, intersection, difference, and symmetric difference.
- Curly braces or the set() function can be used to create sets. Note: to create an empty set you have to use set(), not {}; the latter creates an empty dictionary, a data structure that we will discuss next.

```
>>> basket = {'apple', 'orange', 'apple', 'pear', 'orange', 'banana'}
>>> print(basket) # show that duplicates have been removed
{'orange', 'banana', 'pear', 'apple'}
>>> 'orange' in basket # fast membership testing
True
>>> 'crabgrass' in basket
False
```

Sets (Review)

```
>>> # Demonstrate set operations on unique letters from two words
>>> a = set('abracadabra')
>>> b = set('alacazam')
>>> a # unique letters in a
{'a', 'r', 'b', 'c', 'd'}
>>> a - b # letters in a but not in b – DIFFERENCE
{'r', 'd', 'b'}
>>> a | b # letters in a or b or both – OR/UNION
{'a', 'c', 'r', 'd', 'b', 'm', 'z', 'l'}
>>> a & b # letters in both a and b – AND/INTERSECTION
{'a', 'c'}
>>> a ^ b # letters in a or b but not both – Exclusive OR
{'r', 'd', 'b', 'm', 'z', 'l'}
```



GET search/tweets	search	180	450
GET statuses/lookup	statuses	900	300
GET statuses/mentions_timeline	statuses	75	0
GET statuses/retweeters/ids	statuses	75	300
GET statuses/retweets_of_me	statuses	75	0
GET statuses/retweets/:id	statuses	75	300
GET statuses/show/:id	statuses	900	900
GET statuses/user_timeline	statuses	900	1500
GET trends/available	trends	75	75
GET trends/closest	trends	75	75
GET trends/place	trends	75	75
GET users/lookup	users	900	300

Searching for Tweets – Part 1

```
# Set this variable to a trending topic, or anything else for that matter.
q = ‘Syracuse University' # max 500 characters
count = 1 # max = 100, default = 15

#See https://developer.twitter.com/en/docs/tweets/search/api-
    reference/get-search-tweets

search_results = twitter_api.search.tweets(q=q, count=count)
statuses = search_results['statuses’] # tweets are here
# You can access each tweet in this ways
for tweet in [status['text'] for status in statuses]:
    print tweet
```

https://developer.twitter.com/en/docs/twitter-api/v1/tweets/search/overview

Search Tweets

Please note:

We launched a new version of the standard Search Tweets endpoint as part of Twitter API v2: Early Access. If you are currently using any of these endpoints, you can use our migration materials to start working with the newer endpoint.

Standard

Standard Search API

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

Search Tweets

Overview Quick start Guides FAQ API reference

API reference contents

Standard search API

Premium search APIs

Enterprise search APIs

Please note:

We launched a new version of the standard Search Tweets endpoint as part of Twitter API v2: Early Access. If you are currently using any of these endpoints, you can use our migration materials to start working with the newer endpoint.

Standard search API

(app auth)

Parameters

Name	Required	Description	Default Value	Example
q	required	A UTF-8, URL-encoded search query of 500 characters maximum, including operators. Queries may additionally be limited by complexity.		`@noradio`
geocode	optional	Returns tweets by users located within a given radius of the given latitude/longitude. The location is preferentially taking from the Geotagging API, but will fall back to their Twitter profile. The parameter value is specified by " `latitude,longitude,radius` ", where radius units must be specified as either " `mi` " (miles) or " `km` " (kilometers). Note that you cannot use the near operator via the API to geocode arbitrary locations; however you can use this `geocode` parameter to search near geocodes directly. A maximum of 1,000 distinct "sub-regions" will be considered when using the radius modifier.		`37.781157 -122.398720 1mi`
lang	optional	Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort.		`eu`
locale	optional	Specify the language of the query you are sending (only `ja` is currently effective). This is intended for language-specific consumers and the default should		`ja`

		considered when using the radius modifier.		
lang	optional	Restricts tweets to the given language, given by an ISO 639-1 code. Language detection is best-effort.		`eu`
locale	optional	Specify the language of the query you are sending (only `ja` is currently effective). This is intended for language-specific consumers and the default should work in the majority of cases.		`ja`
result_type	optional	Optional. Specifies what type of search results you would prefer to receive. The current default is "mixed." Valid values include: * `mixed` : Include both popular and real time results in the response. * `recent` : return only the most recent results in the response * `popular` : return only the most popular results in the response.		`mixed` `recent` `popular`
count	optional	The number of tweets to return per page, up to a maximum of 100. Defaults to 15. This was formerly the "rpp" parameter in the old Search API.		`100`
until	optional	Returns tweets created before the given date. Date		`2015-`

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

Example Response


```
{
 "search_metadata": {
  "completed_in": 0.024,
  "count": 1,
  "max_id": 1171104422874816512,
  "max_id_str": "1171104422874816512",
  "next_results": "?max_id=1171104422874816511&q=Syracuse%20University&count=1&include_entities=1",
  "query": "Syracuse+University",
  "refresh_url": "?since_id=1171104422874816512&q=Syracuse%20University&include_entities=1",
  "since_id": 0,
  "since_id_str": "0"
 },
 "statuses": [
  {
   "contributors": null,
   "coordinates": {
    "coordinates": [
     -76.13435517,
     43.04154344
    ],
    "type": "Point"
   },
   "created_at": "Mon Sep 09 16:53:47 +0000 2019",
   "entities": {
    "hashtags": [],
    "symbols": [],
    "urls": [
```


Ln: 157 Col: 4

Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

Example Response


```
},
"statuses": [
 {
  "contributors": null,
  "coordinates": {
   "coordinates": [
    -76.13435517,
    43.04154344
   ],
   "type": "Point"
  },
  "created_at": "Mon Sep 09 16:53:47 +0000 2019",
  "entities": {
   "hashtags": [],
   "symbols": [],
   "urls": [
    {
     "display_url": "swarmapp.com/c/ilo0ATENv0z",
     "expanded_url": "https://www.swarmapp.com/c/ilo0ATENv0z",
     "indices": [
      74,
      97
     ],
     "url": "https://t.co/vyeJIlE6zp"
    }
   ],
   "user_mentions": []
  }
```


Ln: 157 Col: 4

"truncated": false,
"user": {
"contributors_enabled": false,
"created_at": "Mon May 16 14:23:30 +0000 2011",
"default_profile": true,
"default_profile_image": false,
"description": "",
"entities": {
"description": {
"urls": []
}
},
"favourites_count": 6154,
"follow_request_sent": false,
"followers_count": 79,
"following": false,
"friends_count": 261,
"geo_enabled": true,
"has_extended_profile": true,
"id": 299679012,
"id_str": "299679012",
"is_translation_enabled": false,
"is_translator": false,
"lang": null,
"listed_count": 4,
"location": "Syracuse, NY",
"name": "Peter LaPage",

Example Response

Anatomy of a Tweet


```
"statuses": [
  {…
    "created_at": …,
    "id": …,
    "text": …,
    …
    "user": {
      "followers_count": …,
      "friends_count": …, …
      "id": …, …
       "name": …, …
       "screen_name": …., ...
        …
    }…
  }…
```

Data dictionary

Overview

Overview contents

Tweet Object

Tweets are the basic atomic building block of all things Twitter. Tweets are also known as “status updates.” The Tweet object has a long list of ‘root-level’ attributes, including fundamental attributes such as `id`, `created_at`, and `text`. Tweet objects are also the ‘parent’ object to several child objects. Tweet child objects include `user`, `entities`, and `extended_entities`. Tweets that are geo-tagged will have a `place` child object.

When the following Tweet is rendered in JSON:

Searching for Tweets – Part 2

```
# Iterate through 5 more batches of results by following the cursor
for _ in range(5):
    print('Length of statuses', len(statuses))
    try:
        next_results = search_results['search_metadata']['next_results']
    except KeyError as e: # No more results when next_results doesn't exist
        break
    # Create a dictionary from next_results, which has the following form:
    # ?max_id=313519052523986943&q=NCAA&count=100&include_entities=1
    kwargs = dict([ kv.split('=') for kv in unquote(next_results[1:]).split("&") ])

    search_results = twitter_api.search.tweets(**kwargs)
    statuses += search_results['statuses']

# Show one sample search result by slicing the list...
# print(json.dumps(statuses[0], indent=1))
```

Example Response


```
{
 "search_metadata": {
  "completed_in": 0.024,
  "count": 1,
  "max_id": 1171104422874816512,
  "max_id_str": "1171104422874816512",
  "next_results": "?max_id=1171104422874816511&q=Syracuse%20University&count=1&include_entities=1",
  "query": "Syracuse+University",
  "refresh_url": "?since_id=1171104422874816512&q=Syracuse%20University&include_entities=1",
  "since_id": 0,
  "since_id_str": "0"
 },
 "statuses": [
  {
   "contributors": null,
   "coordinates": {
    "coordinates": [
     -76.13435517,
     43.04154344
    ],
    "type": "Point"
   },
   "created_at": "Mon Sep 09 16:53:47 +0000 2019",
   "entities": {
    "hashtags": [],
    "symbols": [],
    "urls": [
```

Dictionaries (Review)

- To check whether a single key is in the dictionary, use the **in** keyword.

  >>> ‘guido’ **in** tel

  True

- The **dict()** constructor builds dictionaries directly from lists of key-value pairs stored as tuples

  >>> dict([(‘sape’, 4139), (‘guido’, 4127), (‘jack’, 4098)])

  {‘sape’: 4139, ‘jack’: 4098, ‘guido’: 4127}

Defining Functions (Review)

3. Arbitrary Argument Lists: the least frequently used option is to specify that a function can be called with an arbitrary number of arguments. These arguments will be wrapped up in a **tuple:**

```
>>> def f(*args): print args
>>> f(1, 2)
(1, 2)
```

Similarly for keyword arguments, which will be in a **dictionary**:

```
>>> def f(**args): print args
>>> f(a=1, b=2)
{'a': 1, 'b': 2}
```

String to List to String (Review)

- Join() turns a list of strings into one string:
  <separator_string>.join(<some_list>)

  >>> “;”.join([“abc”, “def”, “ghi”])
  “abc;def;ghi”

- Split turns one string into a list of strings.
  <some_string>.split(<separator_string>)

  >>> “abc;def;ghi”.split(“;”)
  [“abc”, “def”, “ghi”]

- Note the inversion in the syntax

Standard search API

Returns a collection of relevant Tweets matching a specified query.

Please note that Twitter's search service and, by extension, the Search API is not meant to be an exhaustive source of Tweets. Not all Tweets will be indexed or made available via the search interface.

To learn how to use Twitter Search effectively, please see the Standard search operators page for a list of available filter operators. Also, see the Working with Timelines page to learn best practices for navigating results by `since_id` and `max_id`.

Resource URL

`https://api.twitter.com/1.1/search/tweets.json`

Resource Information

Response formats	JSON
Requires authentication?	Yes
Rate limited?	Yes
Requests / 15-min window (user auth)	180
Requests / 15-min window	450

standard search:

Operator	Finds Tweets...
watching now	containing both “watching” and “now”. This is the default operator.
“happy hour”	containing the exact phrase “happy hour”.
love OR hate	containing either “love” or “hate” (or both).
beer -root	containing “beer” but not “root”.
#haiku	containing the hashtag “haiku”.
from:interior	sent from Twitter account “interior”.
list:NASA/astronauts-in-space-now	sent from a Twitter account in the NASA list astronauts-in-space-now
to:NASA	a Tweet authored in reply to Twitter account “NASA”.
@NASA	mentioning Twitter account “NASA”.
politics filter:safe	containing “politics” with Tweets marked as potentially sensitive removed.
puppy filter:media	containing “puppy” and an image or video.
puppy -filter:retweets	containing “puppy”, filtering out retweets
puppy filter:native_video	containing “puppy” and an uploaded video, Amplify video, Periscope, or Vine.
puppy filter:periscope	containing “puppy” and a Periscope video URL.

9.4. Searching for Tweets

9.4.1. Problem

You want to search Twitter for tweets using specific keywords and query constraints.

9.4.2. Solution

Use the Search API to perform a custom query.

9.4.3. Discussion

Example 9-4 illustrates how to use the Search API to perform a custom query against the entire Twitterverse. Similar to the way that search engines work, Twitter's Search API returns results in batches, and you can configure the number of results per batch to a maximum value of 200 by using the `count` keyword parameter. It is possible that more than 200 results (or the maximum value that you specify for `count`) may be available for any given query, and in the parlance of Twitter's API, you'll need to use a *cursor* to navigate to the next batch of results.

Cursors are a new enhancement to Twitter's v1.1 API and provide a more robust scheme than the pagination paradigm offered by the v1.0 API, which involved specifying a page number and a results-per-page constraint. The essence of the cursor paradigm is that it

Search - Twitter Cookbook

```
def twitter_search(twitter_api, q, max_results=200, **kw):
    # See https://developer.twitter.com/en/docs/tweets/search/api-reference/get-search-tweets
    # and https://developer.twitter.com/en/docs/tweets/search/guides/standard-operators
    # for details on advanced search criteria that may be useful for
    # keyword arguments
    # See https://dev.twitter.com/docs/api/1.1/get/search/tweets
    search_results = twitter_api.search.tweets(q=q, count=100, **kw)

    statuses = search_results['statuses']

    # Iterate through batches of results by following the cursor until we
    # reach the desired number of results, keeping in mind that OAuth users
    # can "only" make 180 search queries per 15-minute interval. See
    # https://developer.twitter.com/en/docs/basics/rate-limits
    # for details. A reasonable number of results is ~1000, although
    # that number of results may not exist for all queries.
```

Search - Twitter Cookbook

```
# Enforce a reasonable limit
    max_results = min(1000, max_results)
    for _ in range(10): # 10*100 = 1000
        try:
            next_results = search_results['search_metadata']['next_results']
        except KeyError as e: # No more results when next_results doesn't exist
            break
        # Create a dictionary from next_results, which has the following form:
        # ?max_id=313519052523986943&q=NCAA&include_entities=1
        kwargs = dict([ kv.split('=')
                        for kv in next_results[1:].split("&") ])
        search_results = twitter_api.search.tweets(**kwargs)
        statuses += search_results['statuses']

        if len(statuses) > max_results:
            break

    return statuses
```

Search - Twitter Cookbook

```
# Sample usage

twitter_api = oauth_login()

q = "CrossFit"
results = twitter_search(twitter_api, q, max_results=10)

# Show one sample search result by slicing the list...
print(json.dumps(results[0], indent=1))
```

Searching for Tweets – Part 3

```
for i in range(10):
    print()
    print(statuses[i]['text'])
    print('Favorites: ', statuses[i]['favorite_count'])
    print('Retweets: ', statuses[i]['retweet_count'])
```

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

GET endpoints

The standard API rate limits described in this table refer to GET (read) endpoints. Note that endpoints not listed in the chart default to 15 requests per allotted user. All request windows are 15 minutes in length. These rate limits apply to the standard API endpoints only, does not apply to premium APIs.

Endpoint	Requests / window per user	Requests / window per app
GET account/verify_credentials	75	0
GET application/rate_limit_status	180	180
GET favorites/list	75	75
GET followers/ids	15	15
GET followers/list	15	15
GET friends/ids	15	15
GET friends/list	15	15
GET friendships/show	180	15
GET geo/id/:place_id	75	0
GET help/configuration	15	15
GET help/languages	15	15
GET help/privacy	15	15

Documentation

Search the docs

Twitter API

Getting started

Tutorials

Tools and libraries

Migrate

API reference Index

Twitter API v2 Early Access

Fundamentals

Tweets

Users

GET lists/subscribers/show	15	15
GET lists/subscriptions	15	15
GET search/tweets	180	450
GET statuses/lookup	900	300
GET statuses/mentions_timeline	75	0
GET statuses/retweeters/ids	75	300
GET statuses/retweets_of_me	75	0
GET statuses/retweets/:id	75	300
GET statuses/show/:id	900	900
GET statuses/user_timeline	900	1500
GET trends/available	75	75
GET trends/closest	75	75
GET trends/place	75	75
GET users/lookup	900	300
GET users/search	900	0
GET users/show	900	900
GET users/suggestions	15	15

GET search/tweets	search	180	450
GET statuses/lookup	statuses	900	300
GET statuses/mentions_timeline	statuses	75	0
GET statuses/retweeters/ids	statuses	75	300
GET statuses/retweets_of_me	statuses	75	0
GET statuses/retweets/:id	statuses	75	300
GET statuses/show/:id	statuses	900	900
GET statuses/user_timeline	statuses	900	1500
GET trends/available	trends	75	75
GET trends/closest	trends	75	75
GET trends/place	trends	75	75
GET users/lookup	users	900	300

Extracting Entities From Tweets (Example 1-6)

```
status_texts = [ status['text']
                 for status in statuses ]
screen_names = [ user_mention['screen_name']
                 for status in statuses
                     for user_mention in status['entities']['user_mentions'] ]
hashtags = [ hashtag['text']
                 for status in statuses
                     for hashtag in status['entities']['hashtags'] ]
# Compute a collection of all words from all tweets
words = [ w for t in status_texts for w in t.split() ]

# Explore the first 5 items for each...
print(json.dumps(status_texts[0:5], indent=1))
print(json.dumps(screen_names[0:5], indent=1))
print(json.dumps(hashtags[0:5], indent=1))
print(json.dumps(words[0:5], indent=1))
```

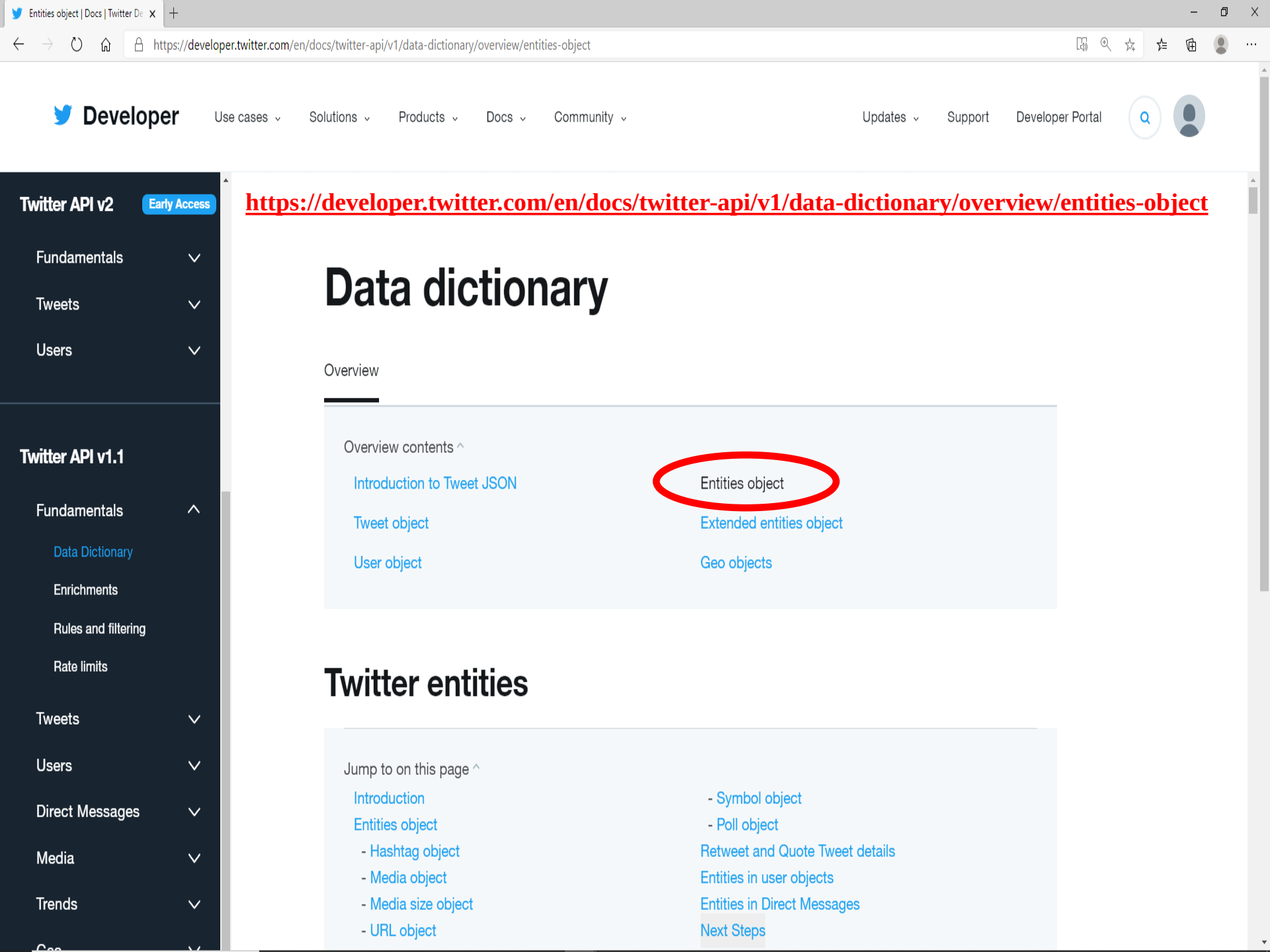
Python 3.6.0 Shell

File Edit Shell Debug Options Window Help

Example Response


```
},
"statuses": [
 {
  "contributors": null,
  "coordinates": {
   "coordinates": [
    -76.13435517,
    43.04154344
   ],
   "type": "Point"
  },
  "created_at": "Mon Sep 09 16:53:47 +0000 2019",
  "entities": {
   "hashtags": [],
   "symbols": [],
   "urls": [
    {
     "display_url": "swarmapp.com/c/ilo0ATENv0z",
     "expanded_url": "https://www.swarmapp.com/c/ilo0ATENv0z",
     "indices": [
      74,
      97
     ],
     "url": "https://t.co/vyeJIlE6zp"
    }
   ],
   "user_mentions": []
  }
```


Ln: 157 Col: 4


Entities object | Docs | Twitter De
https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/entities-object
Developer
Use cases
Solutions
Products
Docs
Community
Updates
Support
Developer Portal
Twitter API v2
Early Access
Fundamentals
Tweets
Users
Twitter API v1.1
Fundamentals
Data Dictionary
Enrichments
Rules and filtering
Rate limits
Tweets
Users
Direct Messages
Media
Trends
https://developer.twitter.com/en/docs/twitter-api/v1/data-dictionary/overview/entities-object
Data dictionary
Overview
Overview contents
Introduction to Tweet JSON
Tweet object
User object
Entities object
Extended entities object
Geo objects
Twitter entities
Jump to on this page
Introduction
Entities object
- Hashtag object
- Media object
- Media size object
- URL object
- Symbol object
- Poll object
Retweet and Quote Tweet details
Entities in user objects
Entities in Direct Messages
Next Steps

Introduction

Entities provide metadata and additional contextual information about content posted on Twitter. The `entities` section provides arrays of common things included in Tweets: hashtags, user mentions, links, stock tickers (symbols), Twitter polls, and attached media. These arrays are convenient for developers when ingesting Tweets, since Twitter has essentially pre-processed, or pre-parsed, the text body. Instead of needing to explicitly search and find these entities in the Tweet body, your parser can go straight to this JSON section and there they are.

Beyond providing parsing conveniences, the `entities` section also provides useful 'value-add' metadata. For example, if you are using the Enhanced URLs enrichment, URL metadata include fully-expanded URLs, as well as associated website titles and descriptions. Another example is when there are user mentions, the entities metadata include the numeric user ID, which are useful when making requests to many Twitter APIs.

Every Tweet JSON payload includes an `entities` section, with the minimum set of `hashtags`, `urls`, `user_mentions`, and `symbols` attributes, even if none of those entities are part of the Tweet message. For example, if you examine the JSON for a Tweet with a body of "Hello World!" and no attached media, the Tweet's JSON will include the following content with entity arrays containing zero items:

```
"entities" : {
  "hashtags" : [
  ] ,
  "urls" : [
  ] ,
  "user_mentions" : [
  ] ,
```

9.10. Extracting Tweet Entities

9.10.1. Problem

You want to extract entities such as *@username* mentions, #hashtags, and URLs from tweets for analysis.

9.10.2. Solution

Extract the tweet entities from the `entities` field of tweets.

9.10.3. Discussion

Twitter's API now provides tweet entities as a standard field for most of its API responses, where applicable. The `entities` field, illustrated in Example 9-10, includes user mentions, hashtags, references to URLs, media objects (such as images and videos), and financial *symbols* such as stock tickers. At the current time, not all fields may apply for all situations. For example, the `media` field will appear and be populated in a tweet only if a user embeds the media using a Twitter client that specifically uses a particular API for embedding the content; simply copying/pasting a link to a YouTube video won't necessarily populate this field.

Extracting Entities - Twitter Cookbook

```
def extract_tweet_entities(statuses):
    # See https://developer.twitter.com/en/docs/tweets/data-dictionary/overview/entities-object
    # for more details on tweet entities

    if len(statuses) == 0:
        return [], [], [], [], []

    screen_names = [ user_mention['screen_name']
                     for status in statuses
                       for user_mention in status['entities']['user_mentions'] ]

    hashtags = [ hashtag['text']
                 for status in statuses
                   for hashtag in status['entities']['hashtags'] ]
```

Extracting Entities - Twitter Cookbook

```
urls = [ url['expanded_url']
            for status in statuses
              for url in status['entities']['urls'] ]
# In some circumstances (such as search results), the media entity
# may not appear
medias = []
symbols = []
for status in statuses:
    if 'media' in status['entities']:
        for media in status['entities']['media']:
            medias.append(media['url'])
    if 'symbol' in status['entities']:
        for symbol in status['entities']['symbol']:
            symbols.append(symbol)

return screen_names, hashtags, urls, medias, symbols
```

Extracting Entities - Twitter Cookbook

```
# Sample usage

q = ‘COVID-19 vaccine'

statuses = twitter_search(twitter_api, q)

screen_names, hashtags, urls, media, symbols = extract_tweet_entities(statuses)

# Explore the first five items for each...

print(json.dumps(screen_names[0:5], indent=1))
print(json.dumps(hashtags[0:5], indent=1))
print(json.dumps(urls[0:5], indent=1))
print(json.dumps(media[0:5], indent=1))
print(json.dumps(symbols[0:5], indent=1))
```

Twitter Cookbook Example 18

Example 18. Extracting tweet entities from arbitrary text

```
# pip install twitter_text or twitter-text
import twitter_text

# Sample usage

txt = "RT @SocialWebMining Mining 1M+ Tweets About #Syria http://wp.me/p3QiJd-1I"

ex = twitter_text.Extractor(txt)

print("Screen Names:", ex.extract_mentioned_screen_names_with_indices())
print("URLs:", ex.extract_urls_with_indices())
print("Hashtags:", ex.extract_hashtags_with_indices())
```

twitter-text 3.0

```
pip install twitter-text
```

Latest version

Released: Oct 23, 2017

A library for auto-converting URLs, mentions, hashtags, lists, etc. in Twitter text. Also does tweet validation and search term highlighting. Fork of twitter-text-py, that supports python 3. Originally by David Ryan, Py3 port by Glyph.

Navigation

Project description

Release history

Download files

Project description

UNKNOWN

Lexical Analysis

- One of the compelling reasons for mining Twitter data is to try to answer the question of what people are talking about right now.
  - One simple technique you could apply to answer that is conducting **frequency analysis -** counting things.
  - In fact, virtually all (statistical) analysis starts out with the simple exercise of counting things
  - Python has a **collections** module that provides a **Counter** that makes computing a frequency distribution (i.e. counting things) rather trivial.
  - Example 1-7 demonstrates how to use a Counter to compute frequency distributions as ranked lists of terms.

Lexical Analysis

```
# Example 1-7. Creating a basic frequency distribution from the
    words in tweets

from collections import Counter

for item in [words, screen_names, hashtags]:
    c = Counter(item)
    print(c.most_common()[:10]) # top 10
    print()
```

Administrator: Command Prompt - python

```
(c) 2018 Microsoft Corporation. All rights reserved.

C:¥WINDOWS¥system32>python
Python 3.6.0 (v3.6.0:41df79263a11, Dec 23 2016, 07:18:10) [MSC v.1900 32 bit (Intel)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import collections
>>> help(collections)
Help on package collections:

NAME
    collections

DESCRIPTION
    This module implements specialized container datatypes providing
    alternatives to Python's general purpose built-in containers, dict,
    list, set, and tuple.

    * namedtuple   factory function for creating tuple subclasses with named fields
    * deque        list-like container with fast appends and pops on either end
    * ChainMap     dict-like class for creating a single view of multiple mappings
    * Counter      dict subclass for counting hashable objects
    * OrderedDict  dict subclass that remembers the order entries were added
    * defaultdict  dict subclass that calls a factory function to supply missing values
    * UserDict     wrapper around dictionary objects for easier dict subclassing
    * UserList     wrapper around list objects for easier list subclassing
    * UserString   wrapper around string objects for easier string subclassing

PACKAGE CONTENTS
    abc

SUBMODULES
    _collections_abc

CLASSES
    builtins.dict(builtins.object)
        Counter
        OrderedDict
```

Administrator: Command Prompt - python

```
>>> help(collections.Counter)
Help on class Counter in module collections:

class Counter(builtins.dict)
 |  Dict subclass for counting hashable items.  Sometimes called a bag
 |  or multiset.  Elements are stored as dictionary keys and their counts
 |  are stored as dictionary values.
 |
 |  >>> c = Counter('abcdeabcdabcaba')  # count elements from a string
 |
 |  >>> c.most_common(3)                # three most common elements
 |  [('a', 5), ('b', 4), ('c', 3)]
 |  >>> sorted(c)                       # list all unique elements
 |  ['a', 'b', 'c', 'd', 'e']
 |  >>> ''.join(sorted(c.elements()))   # list elements with repetitions
 |  'aaaaabbbbcccdde'
 |  >>> sum(c.values())                 # total of all counts
 |  15
 |
 |  >>> c['a']                          # count of letter 'a'
 |  5
 |  >>> for elem in 'shazam':           # update counts from an iterable
 |  ...     c[elem] += 1                # by adding 1 to each element's count
 |  >>> c['a']                          # now there are seven 'a'
 |  7
 |  >>> del c['b']                      # remove all 'b'
 |  >>> c['b']                          # now there are zero 'b'
 |  0
 |
 |  >>> d = Counter('simsalabim')       # make another counter
 |  >>> c.update(d)                     # add in the second counter
 |  >>> c['a']                          # now there are nine 'a'
 |  9
 |
 |  >>> c.clear()                       # empty the counter
 |  >>> c
 |  Counter()
```

```
elements(self)
    Iterator over elements repeating each as many times as its count.

    >>> c = Counter('ABCABC')
    >>> sorted(c.elements())
    ['A', 'A', 'B', 'B', 'C', 'C']

    # Knuth's example for prime factors of 1836:  2**2 * 3**3 * 17**1
    >>> prime_factors = Counter({2: 2, 3: 3, 17: 1})
    >>> product = 1
    >>> for factor in prime_factors.elements():     # loop over factors
    ...     product *= factor                       # and multiply them
    >>> product
    1836

    Note, if an element's count has been set to zero or is a negative
    number, elements() will ignore it.

most_common(self, n=None)
    List the n most common elements and their counts from the most
    common to the least.  If n is None, then list all element counts.

    >>> Counter('abcdeabcdabcaba').most_common(3)
    [('a', 5), ('b', 4), ('c', 3)]

subtract(*args, **kwds)
    Like dict.update() but subtracts counts instead of replacing them.
    Counts can be reduced below zero.  Both the inputs and outputs are
    allowed to contain zero and negative counts.

    Source can be an iterable, a dictionary, or another Counter instance.

    >>> c = Counter('which')
    >>> c.subtract('witch')             # subtract elements from another iterable
    >>> c.subtract(Counter('watch'))    # subtract elements from another counter
    >>> c['h']                          # 2 in which, minus 1 in witch, minus 1 in watch
```

Prettytable

```
# Example 8. Using prettytable to display tuples in a nice tabular format
# Remember to "pip install prettytable"
# It provides a convenient way for fixed width format

from prettytable import PrettyTable

for label, data in (('Word', words),
                    ('Screen Name', screen_names),
                    ('Hashtag', hashtags)):
    pt = PrettyTable(field_names=[label, 'Count'])
    c = Counter(data)
    [ pt.add_row(kv) for kv in c.most_common()[:10] ]
    pt.align[label], pt.align['Count'] = 'l', 'r' # Set column alignment
    print(pt)
```

Lexical Diversity

- A slightly more advanced measurement that involves calculating simple frequencies and can be applied to unstructured text is a metric called lexical diversity

$$lexical\ diversity = \frac{number\ of\ unique\ tokens\ in\ the\ text}{total\ number\ of\ tokens\ in\ the\ text}$$

- Lexical diversity is a concept that is interesting with respect to interpersonal communications because it provides a quantitative measure for the diversity of an individual's or group's vocabulary.

- For tweets, lexical diversity can provide a simple answer to a number of questions such as:

  How broad/narrow is the subject matter that an individual or group discusses?

Lexical Diversity

```
# Example 9. Calculating lexical diversity for tweets
# A function for computing lexical diversity
def lexical_diversity(tokens):
    return len(set(tokens))/len(tokens)

# A function for computing the average number of words per tweet
def average_words(statuses):
    total_words = sum([ len(s.split()) for s in statuses ])
    return total_words/len(statuses)

print(lexical_diversity(words))
print(lexical_diversity(screen_names))
print(lexical_diversity(hashtags))
print(average_words(status_texts))
```

Lexical Diversity – A Comparison

- **Newspaper Corpus Analysis**

Number of words	**15,453,587**
Number of unique words	**269,176**
Lexical diversity	**0.0174**
Number of sentences	**742,311**
Average number of words/sentence	**21**

Retweets

- An interesting exercise at this point would be to further analyze the data to determine if there was a particular tweet that was highly retweeted.
- The approach we'll take to find the most popular retweets is to simply iterate over each status update and store out the retweet count, originator of the retweet, and text of the retweet if the status update is a retweet.
- **Example 1-10** demonstrates how to capture these values with a list comprehension and sort by the retweet count to display the top few results.

Finding Retweets, Part 1

```
retweets = [
        # Store out a tuple of these three values ...
        (status['retweet_count'],
         status['retweeted_status']['user']['screen_name'],
         status['retweeted_status']['id'],
         status['text'])

        # ... for each status ...
        for status in statuses

        # ... so long as the status meets this condition.
            if 'retweeted_status' in status.keys()
       ]
```

Finding Retweets, Part 2

```
# Slice off the first 5 from the sorted results and display each item in the tuple


pt = PrettyTable(field_names=['Count', 'Screen Name', 'Tweet ID', 'Text'])
[ pt.add_row(row) for row in sorted(retweets, reverse=True)[:5] ]
pt.max_width['Text'] = 50
pt.align= 'l'
print(pt)
```

9.21. Harvesting a User's Tweets

9.21.1. Problem

You'd like to harvest all of a user's most recent tweets for analysis.

9.21.2. Solution

Use the `GET statuses/user_timeline` API endpoint to retrieve as many as 3,200 of the most recent tweets from a user, preferably with the added help of a robust API wrapper such as `make_twitter_request` (as introduced in Section 9.16 on page 377) since this series of requests may exceed rate limits or encounter HTTP errors along the way.

9.21.3. Discussion

Timelines are a fundamental concept in the Twitter developer ecosystem, and Twitter provides a convenient API endpoint for the purpose of harvesting tweets by user through the concept of a "user timeline." Harvesting a user's tweets, as demonstrated in Example 9-21, is a meaningful starting point for analysis since a tweet is the most fundamental primitive in the ecosystem. A large collection of tweets by a particular user provides an incredible amount of insight into what the person talks (and thus cares) about. With an archive of several hundred tweets for a particular user, you can conduct dozens of experiments, often with little additional API access. Storing the tweets in a particular collection of a document-oriented database such as MongoDB is a natural way to store and access the data during experimentation. For longer-term Twitter users, performing a time series analysis of how interests or sentiments have changed over time might be a worthwhile exercise.

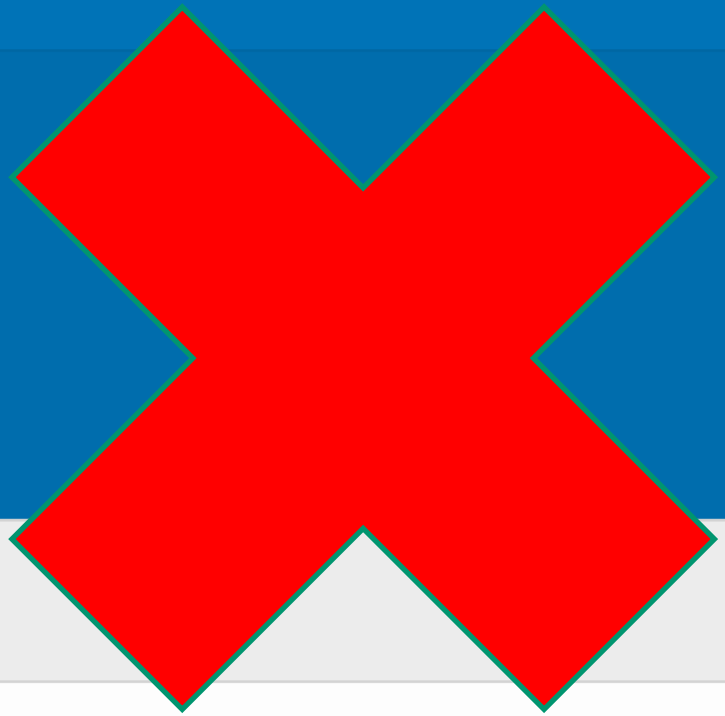
Harvesting a User‘s Tweets

```
# Create your own TwitterCookbook.py module by copying the functions you
# want to use to that module

from TwitterCookbook import oauth_login, make_twitter_request, harvest_user_timeline
import json
twitter_api = oauth_login()
tweets = harvest_user_timeline(twitter_api, screen_name="katyperry", max_results=3200)


for t in tweets:
    try:
        print(t['text'])
        print("Timestamp: ", t['created_at'])
    except:
        pass
```

GetOldTweets3 0.0.11



```
pip install GetOldTweets3
```

Latest version

Released: Nov 27, 2019

Get old tweets from Twitter

Navigation

- Project description
- Release history
- Download files

Project description

GetOldTweets3

A Python 3 library and a corresponding command line utility for accessing old tweets.

python 3.x | build passing | pypi v0.0.11 | downloads 27k

Get Old Tweets

```
import GetOldTweets3 as got

tweetCriteria = got.man                teria                ("barackobama").setMaxTweets(200)
for tweet in got.manager.              etCriteria):
    print(tweet.text)

tweetCriteria = got.manager            ySearch('europe refugees').\
                        setSi                  15-09-30").setMaxTweets(10)
for tweet in got.manage                ger.ge          tCriteria):
    print(tweet.text)
```