# Reddit API:

<https://github.com/praw-dev/praw>

* Public API, python wrapper called PRAW (python reddit api wrapper)
* Can pull top comments, hot posts, etc
* PRAW aims to be easy to use and internally follows all of [Reddit's API rules](https://github.com/reddit/reddit/wiki/API). (Thank fook)
* With PRAW there's no need to introduce **sleep calls** in your code. Give your client an appropriate user agent and you're set.

Set up:

* PRAW is supported on python 2.7, 3.3, 3.4, 3.5 and 3.6. The recommended way to install PRAW is via [pip](https://pypi.python.org/pypi/pip).
* PRAW Documentation <https://praw.readthedocs.io/en/latest/>

Subreddits:

* [/r/redditdev](https://www.reddit.com/r/redditdev) is the best place on Reddit to ask PRAW related questions. This subreddit is for all Reddit API related discussion so please tag submissions with [PRAW]. Please perform a search on the subreddit first to see if anyone has similar questions.
* r/CryptoCurrency – 870,000 members
* r/Bitcoin – 1,000,000 members

## PRAW Documentation and set up

PRAW Documentation <https://praw.readthedocs.io/en/latest/>

# Authenticating via OAuth

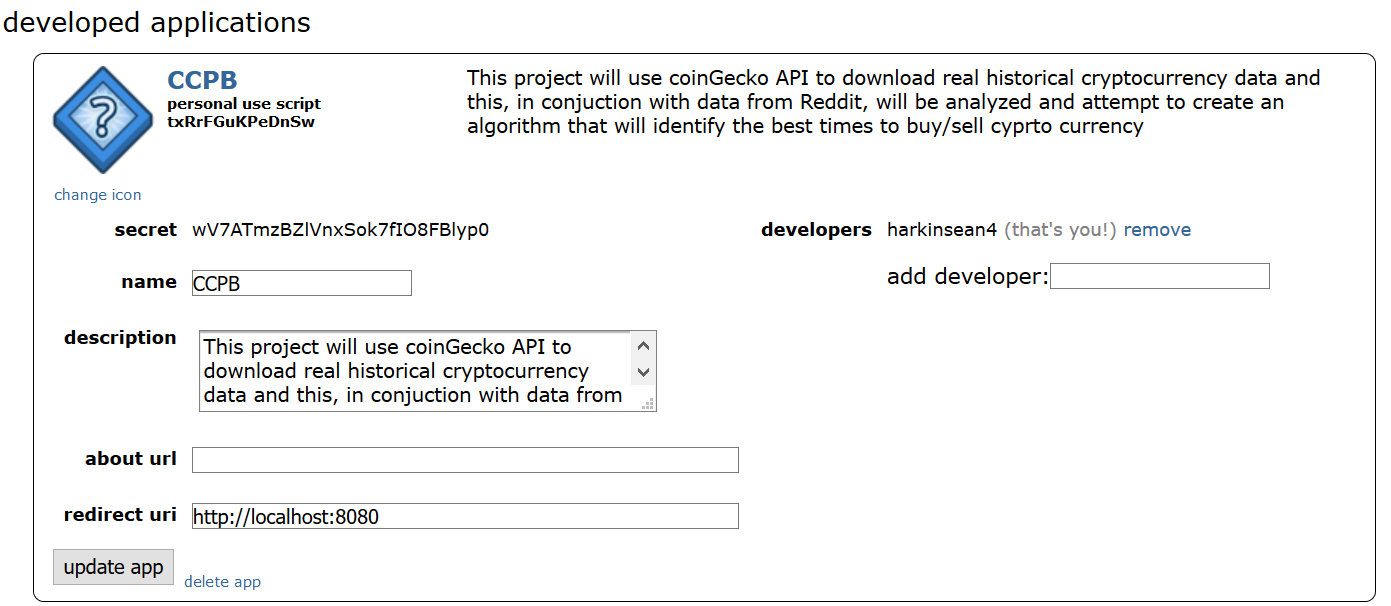
All info is here <https://praw.readthedocs.io/en/latest/getting_started/authentication.html>

PRAW supports the three types of applications that can be registered on Reddit. Those are:

* [Web Applications](https://github.com/reddit-archive/reddit/wiki/OAuth2-App-Types#web-app)
* [Installed Applications](https://github.com/reddit-archive/reddit/wiki/OAuth2-App-Types#installed-app)
* [Script Applications](https://github.com/reddit-archive/reddit/wiki/OAuth2-App-Types#script-app)

The app types are explained here: <https://github.com/reddit-archive/reddit/wiki/OAuth2-App-Types#web-app>

Use <http://localhost:8080> for the redirect uri



**Client ID & Client Secret:**

These two values are needed to access Reddit’s API as a **script** application (see [Authenticating via OAuth](https://praw.readthedocs.io/en/latest/getting_started/authentication.html#oauth) for other application types). If you don’t already have a client ID and client secret, follow Reddit’s [First Steps Guide](https://github.com/reddit/reddit/wiki/OAuth2-Quick-Start-Example#first-steps) to create them.

**User Agent:**

A user agent is a unique identifier that helps Reddit determine the source of network requests. To use Reddit’s API, you need a unique and descriptive user agent. The recommended format is

<platform>:<app ID>:<version string> (by /u/<Reddit username>). For example,android:com.example.myredditapp:v1.2.3 (by /u/kemitche). Read more about user-agents at [Reddit’s API wiki page](https://github.com/reddit/reddit/wiki/API).

## REDDIT API Rules

<https://github.com/reddit-archive/reddit/wiki/API>

**sean-script.py**

## Comments

**submission.id :**  c8h7gh

**comment.id :**  eso3w1u

**comment.parent:**  <bound method Comment.parent of Comment(id='eso3w1u')>

**comment.parent() :**  c8h7gh

**comment.parent\_id :**  t3\_c8h7gh

**comment.submission:**  c8h7gh

* A submission is the subreddit
* Comments have their own ID, a top-level comment has the same ID as the submission
* comment.parent() returns either a submission object or a comment object depending if a top-level comment called it or a reply
* The parent\_id returns the prefix in front of the ID, this signifies that the ID it has just pulled must be that of a submission, a parent comment has a prefix of before it

**Typical Comment Attributes**

This table describes attributes that typically belong to objects of this class. Since attributes are dynamically provided (see [Determine Available Attributes of an Object](https://praw.readthedocs.io/en/latest/getting_started/quick_start.html#determine-available-attributes-of-an-object)), there is not a guarantee that these attributes will always be present, nor is this list comprehensive in any way.

[Link to table](https://praw.readthedocs.io/en/latest/code_overview/models/comment.html)

**sean\_script\_application.py**

Comments:

Comment.parent() if a top level comment, this returns the submission id

If it is a reply to a comment, it returns the id of the comment it is replying to

.list()

The .list() is a PRAW functionality, it returns a flattened list of all Comments. This list may contain MoreComments instances if replace\_more() was not called first.

So, the order you get things when you use the comments.list() is all of the top level comments, followed by 2nd level comments, followed by 3rd level, so these still aren't necessarily sorted how you want them, but you have all comments, and can access every comment's id and parent's id.

.replacemore

.replace\_more(limit=0) we set the limit to 0 to remove all [MoreComments](https://praw.readthedocs.io/en/latest/code_overview/models/more.html#praw.models.MoreComments) instances without additional requests

on really long comments, there will be a ‘load more comments’ link which is a ‘more\_comments’ object, so we won’t to replace these ‘more\_comments’ objects

**sean\_script\_application\_live.py**

Part 3 of Video

* We can stream posts in a subreddit
* We can stream comments

reddit.comment(comment.parent()) -finds original submission or parent of comment

## JSON Library

Python has a built-in package called json, which can be used to work with JSON data.

dumps()

If you have a Python object, you can convert it into a JSON string by using the json.dumps() method.

You can convert Python objects of the following types, into JSON strings:

* dict
* list
* tuple
* string
* int
* float
* True
* False
* None

# Parsing Reddit Comments

My next step now is to analyse all the reddit comments. I have two scripts to work with

1. Sean\_script\_application.py
2. Sean\_script\_application\_live.py

Step 1: choose file

1. Sean\_script\_application.py

This script stores the comments in a dictionary and at the same time writes them to a text file.

Which is better to read from? Dictionary of text file

Dictionary is faster thing in python

Step 2:

I took the Sean\_script\_application.py and made into the Reddit\_Comments.py, the changes I made were:

* Take out the ‘on the fly’ writing to a text file
* Added a list to store the ordered\_reddit\_comments for each thread submission
* Define a function to return ordered\_reddit\_comments to a main application

Step 3:

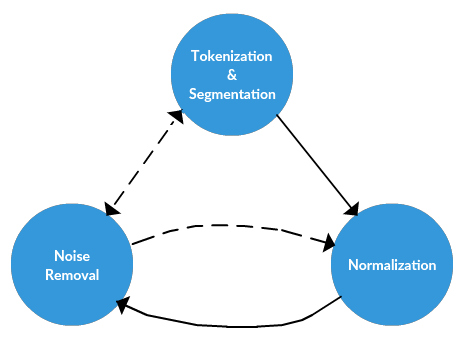
Now that I have the comments returning to the main application. I need to figure a way to parse through all these comments and extract useful information

I need to look into **text pre-processing**

## Text Pre-Processing

Data preprocessing consists of a number of steps, any number of which may or not apply to a given task, but generally fall under the broad categories of

[source](https://www.kdnuggets.com/2017/12/general-approach-preprocessing-text-data.html)

* Tokenization
* Normalization,
* Substitution

### Tokenization:

Tokenization is a step which splits longer strings of text into smaller pieces, or tokens. Larger chunks of text can be tokenized into sentences, sentences can be tokenized into words, etc. Further processing is generally performed after a piece of text has been appropriately tokenized.

Consideration: when we segment text chunks into sentences, should we preserve sentence-ending delimiters? Are we interested in remembering where sentences ended?

### Normalisation:

Before further processing, text needs to be normalized. Normalization generally refers to a series of related tasks meant to put all text on a level playing field: converting all text to the same case (upper or lower), removing punctuation, converting numbers to their word equivalents, and so on. Normalization puts all words on equal footing, and allows processing to proceed uniformly.

Normalizing text can mean performing a number of tasks, but for our framework we will approach normalization in 3 distinct steps:

(1) Miscellaneous

(2) *stemming*

(3) *lemmatization*

1. ***Miscellaneous***

There are, however, numerous other steps that can be taken to help put all text on equal footing, many of which involve the comparatively simple ideas of substitution or removal. They are, however, no less important to the overall process. These include:

* set all characters to lowercase
* remove numbers (or convert numbers to textual representations)
* remove punctuation (generally part of tokenization, but still worth keeping in mind at this stage, even as confirmation)
* strip white space (also generally part of tokenization)
* remove default stop words (general English stop words)

Stop words are those words which are filtered out before further processing of text, since these words contribute little to overall meaning, given that they are generally the most common words in a language. For instance, "the," "and," and "a," while all required words in a particular passage, don't generally contribute greatly to one's understanding of content. As a simple example, the following panagram is just as legible if the stop words are removed:

~~The~~ quick brown fox jumps over ~~the~~ lazy dog.

1. ***Stemming***

Stemming is the process of eliminating affixes (suffixed, prefixes, infixes, circumfixes) from a word in order to obtain a word stem.

running → run

1. ***Lemmatization***

Lemmatization is related to stemming, differing in that lemmatization is able to capture canonical forms based on a word's lemma.

For example, stemming the word "better" would fail to return its citation form (another word for lemma); however, lemmatization would result in the following:

better → good

It should be easy to see why the implementation of a stemmer would be the less difficult feat of the two.

**Noise Removal**

Noise removal continues the substitution tasks of the framework.

Keep in mind again that we are not dealing with a linear process, the steps of which must exclusively be applied in a specified order. Noise removal, therefore, can occur before or after the previously-outlined sections, or at some point between).

Step 1:

I had to install and use the contractions library, I loop through every line of text in the text file one at a time and pass it through the in-built contractions function called and append each new line to a new text file

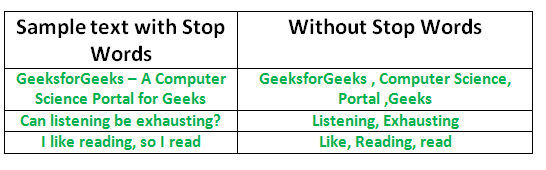
Step 2:

I want to tokenise every word in the text file. As I loop through every line of text to fix contractions, I also pass it through which returns a list of words based on the line of text passed thorugh.

Step 3:

I want to now normalize every list of words. As we loop through every line of text and tokenise it, we also normalize it, getting rid of ascii characters and making everything lowercase etc.

Stop-words: [link](https://www.geeksforgeeks.org/removing-stop-words-nltk-python/) A stop word is a commonly used word (such as “the”, “a”, “an”, “in”)



The lit is stored in

Step 4: [Stemming and Lemmatization](https://www.datacamp.com/community/tutorials/stemming-lemmatization-python)

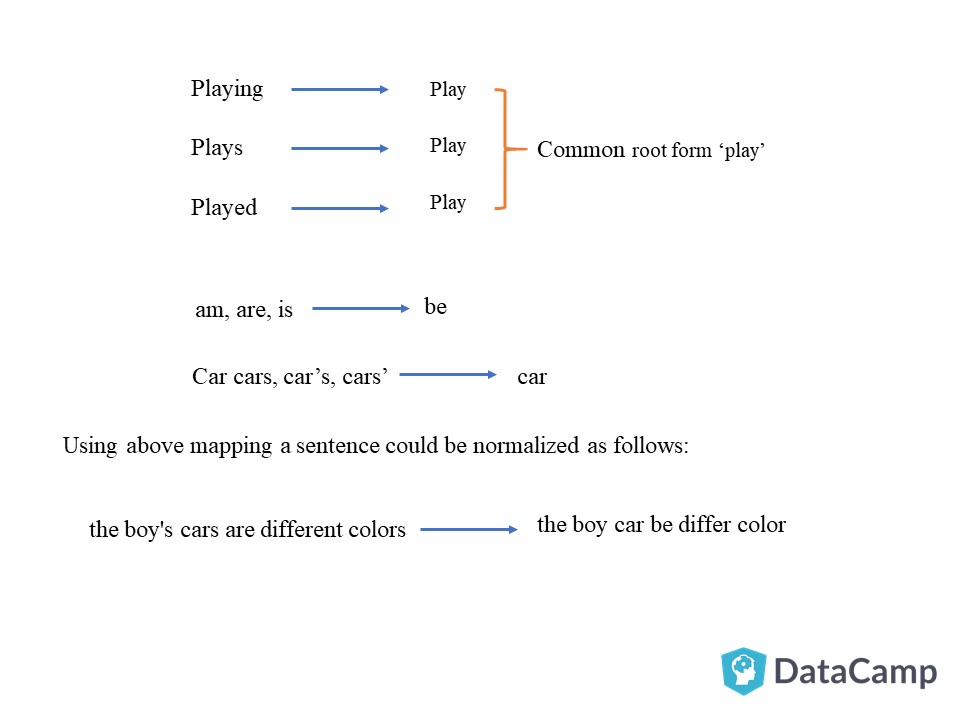
Stemming and Lemmatization both come under **Text Normalization.** Stemming and Lemmatization helps us to achieve the root forms of an inflected word.

**What’s the difference?**

Stemming and Lemming differ in their flavor. *Stemming* usually refers to a crude heuristic process that chops off the ends of words in the hope of achieving this goal correctly most of the time, and often includes the removal of derivational affixes. *Lemmatization* usually refers to doing things properly with the use of a vocabulary and morphological analysis of words, normally aiming to remove inflectional endings only and to return the base or dictionary form of a word, which is known as the *lemma* .

Languages we speak and write are made up of several words often derived from one another. When a language contains words that are derived from another word as their use in the speech changes is called **Inflected Language**.

"In grammar, inflection is the modification of a word to express different grammatical categories such as tense, case, voice, aspect, person, number, gender, and mood. An inflection expresses one or more grammatical categories with a **prefix, suffix or infix**, or another internal modification such as a vowel change”



**Stemming**

"Stemming is the process of reducing inflection in words to their root forms such as mapping a group of words to the same stem even if the stem itself is not a valid word in the Language."

**Stem (root) is the part of the word to which you add inflectional (changing/deriving) affixes such as (-ed,-ize, -s,-de,mis).** So stemming a word or sentence may result in words that are not actual words. Stems are created by removing the suffixes or prefixes used with a word.

This tutorial will see different stemmers available in different languages in Python nltk. For the English language, you can choose between **PorterStammer** or **LancasterStammer**, PorterStemmer being the oldest one originally developed in 1979. LancasterStemmer was developed in 1990 and uses a more aggressive approach than Porter Stemming Algorithm.

The **LancasterStemmer (Paice-Husk stemmer)** is an iterative algorithm with rules saved externally. One table containing about 120 rules indexed by the last letter of a suffix.

On each iteration, it tries to find an applicable rule by the last character of the word. Each rule specifies either a deletion or replacement of an ending. If there is no such rule, it terminates. It also terminates if a word starts with a vowel and there are only two letters left or if a word starts with a consonant and there are only three characters left. Otherwise, the rule is applied, and the process repeats.

**Lemming**

A lemma is the word you find in the dictionary.

In computational linguistics, a **stem** is the part of the word that never changes even when different forms of the word are used. A lemma is the base form of the verb. For example, from "produced", the lemma is "produce", but the stem is "produc-".

[source](https://simple.wikipedia.org/wiki/Lemma_(linguistics))

Rather than using a stemmer, you can use a *lemmatizer* , a tool from Natural Language Processing which does full morphological analysis to accurately identify the lemma for each word.

Step 4: Speech tagging

For this we nltk.corpus

If nouns aren’t capatilised NLTK speech tagging might not recognize it

POS tag list:

CC coordinating conjunction

CD cardinal digit

DT determiner

EX existential there (like: "there is" ... think of it like "there exists")

FW foreign word

IN preposition/subordinating conjunction

JJ adjective 'big'

JJR adjective, comparative 'bigger'

JJS adjective, superlative 'biggest'

LS list marker 1)

MD modal could, will

NN noun, singular 'desk'

NNS noun plural 'desks'

NNP proper noun, singular 'Harrison'

NNPS proper noun, plural 'Americans'

PDT predeterminer 'all the kids'

POS possessive ending parent\'s

PRP personal pronoun I, he, she

PRP$ possessive pronoun my, his, hers

RB adverb very, silently,

RBR adverb, comparative better

RBS adverb, superlative best

RP particle give up

TO to go 'to' the store.

UH interjection errrrrrrrm

VB verb, base form take

VBD verb, past tense took

VBG verb, gerund/present participle taking

VBN verb, past participle taken

VBP verb, sing. present, non-3d take

VBZ verb, 3rd person sing. present takes

WDT wh-determiner which

WP wh-pronoun who, what

WP$ possessive wh-pronoun whose

WRB wh-abverb where, when

Step 4: Chunking

Chunking should help us figure out the meaning of a sentence

The noun (named entity) is usually the subject of the sentence

Then we have to find words that ‘modify’ that named entity (i.e. which noun is the adjectives talking about)

People chunk into ‘noun phrases’, this a noun with a load of modifiers around that noun pretty much

Regex

[Regex tut](https://pythonprogramming.net/regular-expressions-regex-tutorial-python-3/)

In Python 3, the module to use regular expressions is re, and it must be imported to use regular expressions. Re is a part of the standard library, meaning you will not need to do any downloading and installing to use it, it is already there.

We use **regex modifiers** with chunking

+ = match 1 or more

? = match 0 or 1 repetitions.

\* = match 0 or MORE repetitions

. = Any character except a new line

Step 5: Chinking

We ‘chink’ something from a chunk i.e. so chinking is the removal of

We can chunk too much so we can specify what **not to chunk** i.e. “except for”

chunkGram = r"""Chunk: {<.\*>+} }<VB.?|IN|DT|TO>+{"""

Above we say: chunk together for any amount of characters once or more times . So it will basically chunk everything. Then we chink out any VERB, IN, DT TO zero or one times

Step 6: Named Entity Recognition