Youtube Scraping API

The Silly Coder

Mar 13, 2022

CONTENTS:

1 Features **3**

2 Todo **5**

2.1 Installation . . . 5
 2.1.1 PyPI . . . 5
 2.1.2 Source . . . 6
2.2 Quickstart . . . 6
 2.2.1 Initialize . . . 6
 2.2.2 Search . . . 6
 2.2.3 Video . . . 7
  2.2.3.1 Retrieve metadata . . . 7
  2.2.3.2 Get video captions . . . 7
  2.2.3.3 Download video . . . 7
 2.2.4 Caption . . . 8
  2.2.4.1 XML . . . 9
  2.2.4.2 Dictionary . . . 9
  2.2.4.3 Text . . . 9
 2.2.5 Playlist . . . 9
 2.2.6 Channel . . . 10
2.3 API References . . . 10
 2.3.1 YoutubeAPI . . . 10
 2.3.2 Filter . . . 10
 2.3.3 Video . . . 10
 2.3.4 Caption . . . 10
 2.3.5 Playlist . . . 10
 2.3.6 Channel . . . 10
 2.3.7 Utilities . . . 10

3 Indices and tables **11**

Youtube Scraping API is a lightweight, easy-to-use library packed with Youtube data scraper and video downloader. This scraping api doesn't depend on Youtube Official API, since there are strict quota when using it. With this API, you can get access to all data of **Youtube videos**, **playlists**, **channels**, and **search results**. Simply download the package using pip, and you're ready to go. For those who never use this API before, *Quickstart Guide* can help you get started with using this API quickly.

Here is a simple demonstration

```
from youtube_scraping_api import YoutubeAPI
api = YoutubeAPI()

# search for videos, playlists, channels, and etc.
api.search('python tutorial')

# download videos
api.video('a1EYnngNHIA').download()
```

Of course these are just a tip of the iceberg. We have a lot more features available in this API.

CHAPTER ONE

FEATURES

- Scrape YouTube search results
- Scrape YouTube suggestion queries
- Useful filters for YouTube searching
- Download videos with any resolutions
- Get metadata of channels, videos, and playlists
- Scrape thumbnail urls of videos, channels, and playlists
- Scrape captions of videos
- Ability to log progress when executing code
- Nicely formed object structure
- Well documented source code
- You can literally create a fully working YouTube clone with this API :)

CHAPTER

TWO

TODO

- Callback function for video downloading
- Unit test for each features
- Ability to merge high quality videos and sound
- Turn video download data into object
- **Extract from channel:**
    - all videos
    - all playlist
    - channel page
    - community posts
    - all external links in about page
    - sections in channel homepage
- Search results filtering system
- Search results sorting system

2.1 Installation

Installing Youtube Scraping API is easy like a piece of cake! There are a few options for installing:

2.1.1 PyPI

Installing from PyPI is the easiest and recommended method. It will contain the most up-to-date *stable* distribution:

```
python -m pip install youtube-scraping-api
```

2.1.2 Source

If for some reason you hate using PyPI to install packages, you may also install using git.

```
python -m pip install git+https://github.com/melvinchia3636/YouTube-Scraping-API
```

Warning: If you download the code directly from GitHub, sometimes you might find some experimental features after installing it. For those who are facing any problems when running the code, kindly drop your issue in the Issue page of the repository.

2.2 Quickstart

Although using Youtube Scraping API is simple enough, but some tutorials are still needed. This guide will walk you through the basic usage of Youtube Scraping API.

Let's get started with some examples, I mean a lot of examples.

2.2.1 Initialize

To use a library, first of all you need to import it, no doubt for that. To import Youtube Scraping API, all you need is one line of code:

```
>>> from youtube_scraping_api import YoutubeAPI
```

Personally speaking, the name of the library is a little bit long, but unfortunately we have no choice. When I was registering this library on PyPi, all other names like 'youtube-api', 'youtube-scraper' have all been taken by others. Maybe there are some ways to make the name shorter, let's just hope that I figure it out as soon as possible.

Let's get back to the point. Now how can you start using it after import? That's a good question, but the answer is stupidly simple: call the class. Here is how you can do it:

```
>>> api = YoutubeAPI()
```

Here you go! Initialization completed. Now you're all set to do whatever the heck you want with this API. But please don't DDOS lol, I'm not sure if you'll get arrested by FBI (just kidding) if you do so.

2.2.2 Search

Tired of searching on YouTube webpage? Or you just wanna try to be cool? We have the perfect solution for you. With this API, you can do basically everything you can perform on youtube webpage by using codes. Sounds cool, right? Let's try searching for your favourite videos, channel, or playlist:

```
>>> query = 'Minecraft Survival Guide'
>>> api.search(query)
<SearchResult {'Playlist': 3, 'Video': 16, 'Channel': 1, 'Shelf': 1}>
```

So you'll probably ask: What is this? Where is my result? So let me tell you: this thing is just a wrapper. The real result is inside this wrapper. This is a special datatype that inherits from Python builtin list, so what you can do is:

```
>>> for result in api.search(query):
...     print(result)
```

And now you should be able to see all results get printed out one by one. For more information on types of result, check out *API Reference*.

2.2.3 Video

Here come the most important part of this API - The Ultimate Video Object. With this object, you can retrieve almost all metadata of the video, parse the caption, download the video, and do whatever the heck you want with them. So here comes the basic usage of it:

```
>>> videoId = 'WzxSiEWK3cg'
>>> api.video(videoId)
<Video id="WzxSiEWK3cg" title="PARIS | A380 LANDING 4K EXTENDED" author="High Pressure␣
↪Aviation Films">
```

There you go! You've just created a video Object with contains everything about the video. Suppose you have a video object with the name `video` on hand right now, let's see some examples of how we can use it:

2.2.3.1 Retrieve metadata

```
>>> video.title
"PARIS | A380 LANDING 4K EXTENDED"

>>> video.view_count
1374685

>>> video.length #in second
1025
```

These are just a tip of the iceberg of the metadata you can get from the video object. Check out *API Reference* for all the available metadata that you can retrieve from this object.

2.2.3.2 Get video captions

Check out *this section*.

2.2.3.3 Download video

Here comes the most interesting part of this API: Video download. For those who has used PyTube before or you're currently using it, uninstall it right now! Use Youtube Scraping API - a much better version of pytube. Of course I'm just kidding. Currently our API still have a lot of unfinished part, since there is only one developer - myself, so the progress speed won't be fast, but still decent. Wanna download a video, try this line of code:

```
>>> video.download()
```

There you go! Now you should see a progress bar being loaded on your screen. After the progress is finished, you should see your video waiting for you in your working directory. Wanna change where to save it? Or perhaps you want a custom name instead of the video title? Here are how to make it work:

```
>>> video.download(path='your/desired/path/')
>>> video.download(name='your/desired/name/without/extension/')
```

And maybe you want to download video with different resolutions? No problem. Here is how you can get all available resolutions:

```
>>> video.download_data
{18: {'url': 'https://...', 'signature_cipher': None, 'mime_type': 'video/mp4; codecs=
↪"avc1.42001E, mp4a.40.2"', 'bitrate': 539970, 'width': 640, 'height': 360, 'size':
↪'69197263', 'fps': 25, 'quality': 'medium', 'quality_label': '360p', 'duration':
↪'1025253'}
```

The output is a dictionary, with the key of each element being the itag of the downloadable content, and the value of it being its metadata. This output will be converted into object format soon, so you'll be able to query it using some functions.

And now you've found the itag of your choice. To download it, simply put the itag as the first argument of the download function:

```
>>> itag = 251
>>> video.download(itag)
```

And now the video with your desired quality should start downloading pretty quickly.

2.2.4 Caption

Sometimes when you're watching a video with caption, you'll probably wonder if you can download the caption for some personal usage. So let me tell you: Yes, it's possible, and you can download them easily with using this API. Below are some common usage examples. If you want to take a deep look at all its available usage, check out *API Reference*.

To fetch all captions of a video, simply do like so:

```
>>> captions = video.captions
<CaptionQuery [<Caption lang="English" code="en" is_translatable=True>, <Caption lang=
↪"French" code="fr" is_translatable=True>, <Caption lang="French (auto-generated)" code=
↪"fr" is_translatable=True>]>
```

To get caption of specific language, you can do like this:

```
>>> caption = captions.get_caption('en')
<Caption lang="English" code="en" is_translatable=True>
```

If you use `get_caption` without passing anything into it, it will return you the caption with default language, most likely English unless the content creator has specified another default language:

```
>>> caption = captions.get_caption()
<Caption lang="English" code="en" is_translatable=True>
```

And now we've just got our caption object here. Here are all available output format of this caption object:

2.2.4.1 XML

```
>>> caption.xml
<?xml version="1.0" encoding="utf-8" ?><transcript><text start="6.76" dur="1.86">CPT:␣
↪Idle, Open Des, perfect.</text><text start="8.62" dur="0.76">Glide</text><text start=
↪"9.5" dur="0.98">FO: We can set QNH</text><text start="10.48" dur="1.6">CPT: 5000 set␣
↪QNH</text><text start="14.6" dur="3.3">FO: I activate approach phase...</transcript>
```

2.2.4.2 Dictionary

```
>>> caption.dict
[{'start_from': 6.76, 'duration': 1.86, 'text': 'CPT: Idle, Open Des, perfect.'}, {
↪'start_from': 8.62, 'duration': 0.76, 'text': 'Glide'}, ...]
```

2.2.4.3 Text

```
>>> caption.get_text()
CPT: Idle, Open Des, perfect.
Glide
FO: We can set QNH
CPT: 5000 set QNH
FO: I activate approach phase, even if it will be done automatically
....
FO: I think they don't like cold
CPT: Kilo 59
```

2.2.5 Playlist

And now let's move on to bigger collection - Playlist. To fetch a playlist, you just need the playlist id, which is located somewhere in the youtube playlist URL, like:

```
"https://www.youtube.com/watch?v=4c2hzzZz978&list=PL7VmhWGNRxKgtwHFgDMCnutcmiafoP1c4 <--␣
↪here"
"https://www.youtube.com/playlist?list=PL7VmhWGNRxKgtwHFgDMCnutcmiafoP1c4 <-- here"
```

Basically you just need to copy the snippet that comes after `list=` in the url. After that, simply do like so:

```
>>> playlist_id = "PL7VmhWGNRxKgtwHFgDMCnutcmiafoP1c4"
>>> playlist = api.Playlist(playlist_id)
<Playlist title="Hermitcraft VII" video_count=119>
```

This features hasn't quite finished yet, so I can't talk too much about it here. Simply put, it's just a list of video object, and you can threat is as a normal list, do stuff like for loop or something with it, and you can also get some information about the playlist like this:

```
>>> playlist.description
Season Seven of Hermitcraft! This is the official playlist for my series on the␣
↪Hermitcraft 7 Server
>>> playlist.view_count
3820735
```

For more details on the playlist object, unfortunately there are no API reference for now. But this features will be completed as soon as I can, so stay tuned on it.

2.2.6 Channel

2.3 API References

2.3.1 YoutubeAPI

2.3.2 Filter

2.3.3 Video

2.3.4 Caption

2.3.5 Playlist

2.3.6 Channel

2.3.7 Utilities

CHAPTER THREE

INDICES AND TABLES

- genindex
- modindex
- search