youtube-transcript-api 1.0.3

Latest version

Released: Mar 25, 2025

```
pip install youtube-transcript-api
```

This is an python API which allows you to get the transcripts/subtitles for a given YouTube video. It also works for automatically generated subtitles, supports translating subtitles and it does not require a headless browser, like other selenium based solutions do!

Navigation

Project description

Release history

Download files

Verified details

These details have been verified by PyPI

Project description

✨ YouTube Transcript API ✨

This is a python API which allows you to retrieve the transcript/subtitles for a given YouTube video. It also works for automatically generated subtitles, supports translating subtitles and it does not require a headless browser, like other selenium based solutions do!

Maintenance of this project is made possible by all the contributors and sponsors. If you'd like to sponsor this project and have your avatar or company logo appear below click here. 💖

jdepoix

Unverified details

*These details have **not** been verified by PyPI*

Project links

- Homepage
- Repository

Meta

- **License:** MIT License (MIT)
- **Author:** Jonas Depoix
- cli, subtitle, subtitles, transcript, transcripts, youtube, youtube-api, youtube-subtitles, youtube-transcripts
- **Requires:** Python <3.14, >=3.8

Classifiers

License

- OSI Approved :: MIT License

Operating System

- OS Independent

Programming Language

- Python :: 3
- Python :: 3.8
- Python :: 3.9
- Python :: 3.10

Install

It is recommended to install this module by using pip:

```
pip install youtube-transcript-api
```

You can either integrate this module into an existing application or just use it via a CLI.

API

The easiest way to get a transcript for a given video is to execute:

```
from youtube_transcript_api import YouTubeTranscriptAp

ytt_api = YouTubeTranscriptApi()
ytt_api.fetch(video_id)
```

> ***Note:*** *By default, this will try to access the English transcript of the video. If your video has a different language, or you are interested in fetching a transcript in a different language, please read the section below.*

> ***Note:*** *Pass in the video ID, NOT the video URL. For a video with the URL `https://www.youtube.com/watch?v=12345` the ID is `12345`.*

This will return a `FetchedTranscript` object looking somewhat like this:

```
FetchedTranscript(
    snippets=[
        FetchedTranscriptSnippet(
            text="Hey there",
```

```
            FetchedTranscriptSnippet(
                text="how are you",
                start=1.54,
                duration=4.16,
            ),
            # ...
        ],
        video_id="12345",
        language="English",
        language_code="en",
        is_generated=False,
)
```

This object implements most interfaces of a `List`:

```
ytt_api = YouTubeTranscriptApi()
fetched_transcript = ytt_api.fetch(video_id)

# is iterable
for snippet in fetched_transcript:
    print(snippet.text)

# indexable
last_snippet = fetched_transcript[-1]

# provides a length
snippet_count = len(fetched_transcript)
```

If you prefer to handle the raw transcript data you can call `fetched_transcript.to_raw_data()`, which will return a list of dictionaries:

```
[
    {
        'text': 'Hey there',
        'start': 0.0,
        'duration': 1.54
    },
    {
        'text': 'how are you',
        'start': 1.54
        'duration': 4.16
    },
```

Retrieve different languages

You can add the `languages` param if you want to make sure the transcripts are retrieved in your desired language (it defaults to english).

```
YouTubeTranscriptApi().fetch(video_id, languages=['de'
```

It's a list of language codes in a descending priority. In this example it will first try to fetch the german transcript (`'de'`) and then fetch the english transcript (`'en'`) if it fails to do so. If you want to find out which languages are available first, have a look at `list()`.

If you only want one language, you still need to format the `languages` argument as a list

```
YouTubeTranscriptApi().fetch(video_id, languages=['de']
```

Preserve formatting

You can also add `preserve_formatting=True` if you'd like to keep HTML formatting elements such as `<i>` (italics) and `<b>` (bold).

```
YouTubeTranscriptApi().fetch(video_ids, languages=['de
```

List available transcripts

If you want to list all transcripts which are available for a given video you can call:

This will return a `TranscriptList` object which is iterable and provides methods to filter the list of transcripts for specific languages and types, like:

```
transcript = transcript_list.find_transcript(['de', 'en
```

By default this module always chooses manually created transcripts over automatically created ones, if a transcript in the requested language is available both manually created and generated. The `TranscriptList` allows you to bypass this default behaviour by searching for specific transcript types:

```
# filter for manually created transcripts
transcript = transcript_list.find_manually_created_tran

# or automatically generated ones
transcript = transcript_list.find_generated_transcript(
```

The methods `find_generated_transcript`, `find_manually_created_transcript`, `find_transcript` return `Transcript` objects. They contain metadata regarding the transcript:

```
print(
    transcript.video_id,
    transcript.language,
    transcript.language_code,
    # whether it has been manually created or generated
    transcript.is_generated,
    # whether this transcript can be translated or not
    transcript.is_translatable,
    # a list of languages the transcript can be transla
    transcript.translation_languages,
)
```

```
transcript.fetch()
```

This returns a `FetchedTranscript` object, just like `YouTubeTranscriptApi().fetch()` does.

Translate transcript

YouTube has a feature which allows you to automatically translate subtitles. This module also makes it possible to access this feature. To do so `Transcript` objects provide a `translate()` method, which returns a new translated `Transcript` object:

```
transcript = transcript_list.find_transcript(['en'])
translated_transcript = transcript.translate('de')
print(translated_transcript.fetch())
```

By example

```
from youtube_transcript_api import YouTubeTranscriptAp

ytt_api = YouTubeTranscriptApi()

# retrieve the available transcripts
transcript_list = ytt_api.list('video_id')

# iterate over all available transcripts
for transcript in transcript_list:

    # the Transcript object provides metadata propertie
    print(
        transcript.video_id,
        transcript.language,
        transcript.language_code,
        # whether it has been manually created or gener
        transcript.is_generated,
        # whether this transcript can be translated or
        transcript.is_translatable,
        # a list of languages the transcript can be tra
        transcript.translation_languages,
```

```
    print(transcript.fetch())

    # translating the transcript will return another tr
    print(transcript.translate('en').fetch())

# you can also directly filter for the language you are
transcript = transcript_list.find_transcript(['de', 'en

# or just filter for manually created transcripts
transcript = transcript_list.find_manually_created_tran

# or automatically generated ones
transcript = transcript_list.find_generated_transcript
```

Working around IP bans (`RequestBlocked` or `IpBlocked` exception)

Unfortunately, YouTube has started blocking most IPs that are known to belong to cloud providers (like AWS, Google Cloud Platform, Azure, etc.), which means you will most likely run into `ReuquestBlocked` or `IpBlocked` exceptions when deploying your code to any cloud solutions. Same can happen to the IP of your self-hosted solution, if you are doing too many requests. You can work around these IP bans using proxies. However, since YouTube will ban static proxies after extended use, going for rotating residential proxies provide is the most reliable option.

There are different providers that offer rotating residential proxies, but after testing different offerings I have found Webshare to be the most reliable and have therefore integrated it into this module, to make setting it up as easy as possible.

Using Webshare

Once you have created a Webshare account and purchased a "Residential" proxy package that suits your workload (make sure NOT to purchase "Proxy Server" or "Static Residential"!), open the Webshare Proxy Settings to retrieve your "Proxy Username" and "Proxy Password". Using this information you can initialize the `YouTubeTranscriptApi` as follows:

```
ytt_api = YouTubeTranscriptApi(
    proxy_config=WebshareProxyConfig(
        proxy_username="<proxy-username>",
        proxy_password="<proxy-password>",
    )
)

# all requests done by ytt_api will now be proxied thr
ytt_api.fetch(video_id)
```

Using the `WebshareProxyConfig` will default to using rotating residential proxies and requires no further configuration.

Note that referral links are used here and any purchases made through these links will support this Open Source project, which is very much appreciated! 💖 😊 🙏 💖

However, you are of course free to integrate your own proxy solution using the `GenericProxyConfig` class, if you prefer using another provider or want to implement your own solution, as covered by the following section.

Using other Proxy solutions

Alternatively to using Webshare, you can set up any generic HTTP/HTTPS/SOCKS proxy using the `GenericProxyConfig` class:

```
from youtube_transcript_api import YouTubeTranscriptAp
from youtube_transcript_api.proxies import GenericProxy

ytt_api = YouTubeTranscriptApi(
    proxy_config=GenericProxyConfig(
        http_url="http://user:pass@my-custom-proxy.org
        https_url="https://user:pass@my-custom-proxy.o
    )
)

# all requests done by ytt_api will now be proxied usi
ytt_api.fetch(video_id)
```

a pool of proxy addresses, if you want to maximize reliability.

Overwriting request defaults

When initializing a `YouTubeTranscriptApi` object, it will create a `requests.Session` which will be used for all HTTP(S) request. This allows for caching cookies when retrieving multiple requests. However, you can optionally pass a `requests.Session` object into its constructor, if you manually want to share cookies between different instances of `YouTubeTranscriptApi`, overwrite defaults, set custom headers, specify SSL certificates, etc.

```
from requests import Session

http_client = Session()

# set custom header
http_client.headers.update({"Accept-Encoding": "gzip, c

# set path to CA_BUNDLE file
http_client.verify = "/path/to/certfile"

ytt_api = YouTubeTranscriptApi(http_client=http_client)
ytt_api.fetch(video_id)

# share same Session between two instances of YouTubeTr
ytt_api_2 = YouTubeTranscriptApi(http_client=http_clien
# now shares cookies with ytt_api
ytt_api_2.fetch(video_id)
```

Cookie Authentication

Some videos are age restricted, so this module won't be able to access those videos without some sort of authentication. To do this, you will need to have access to the desired video in a browser. Then, you will need to download that pages cookies into a text file. You can use the Chrome extension Cookie-Editor and select "Netscape" during export, or the Firefox extension cookies.txt.

```
ytt_api = YouTubeTranscriptApi(cookie_path='/path/to/yo
ytt_api.fetch(video_id)
```

Using Formatters

Formatters are meant to be an additional layer of processing of the transcript you pass it. The goal is to convert a `FetchedTranscript` object into a consistent string of a given "format". Such as a basic text (`.txt`) or even formats that have a defined specification such as JSON (`.json`), WebVTT (`.vtt`), SRT (`.srt`), Comma-separated format (`.csv`), etc...

The `formatters` submodule provides a few basic formatters, which can be used as is, or extended to your needs:

- JSONFormatter
- PrettyPrintFormatter
- TextFormatter
- WebVTTFormatter
- SRTFormatter

Here is how to import from the `formatters` module.

```
# the base class to inherit from when creating your own
from youtube_transcript_api.formatters import Formatter

# some provided subclasses, each outputs a different st
from youtube_transcript_api.formatters import JSONForma
from youtube_transcript_api.formatters import TextForma
from youtube_transcript_api.formatters import WebVTTFor
from youtube_transcript_api.formatters import SRTFormat
```

Formatter Example

Let's say we wanted to retrieve a transcript and store it to a JSON file. That would look something like this:

```
from youtube_transcript_api import YouTubeTranscriptAp
from youtube_transcript_api.formatters import JSONForma

ytt_api = YouTubeTranscriptApi()
transcript = ytt_api.fetch(video_id)

formatter = JSONFormatter()

# .format_transcript(transcript) turns the transcript
json_formatted = formatter.format_transcript(transcrip

# Now we can write it out to a file.
with open('your_filename.json', 'w', encoding='utf-8')
    json_file.write(json_formatted)

# Now should have a new JSON file that you can easily
```

Passing extra keyword arguments

Since JSONFormatter leverages `json.dumps()` you can also forward keyword arguments into `.format_transcript(transcript)` such as making your file output prettier by forwarding the `indent=2` keyword argument.

```
json_formatted = JSONFormatter().format_transcript(tra
```

Custom Formatter Example

You can implement your own formatter class. Just inherit from the `Formatter` base class and ensure you implement the `format_transcript(self, transcript: FetchedTranscript, **kwargs) -> str` and `format_transcripts(self, transcripts: List[FetchedTranscript], **kwargs) -> str` methods which should ultimately return a string when called on your formatter instance.

```
class MyCustomFormatter(Formatter):
    def format_transcript(self, transcript: FetchedTra
        # Do your custom work in here, but return a st
```

```
        # Do your custom work in here to format a list
        return 'your processed output data as a string
```

CLI

Execute the CLI script using the video ids as parameters and the results will be printed out to the command line:

```
youtube_transcript_api <first_video_id> <second_video_
```

The CLI also gives you the option to provide a list of preferred languages:

```
youtube_transcript_api <first_video_id> <second_video_
```

You can also specify if you want to exclude automatically generated or manually created subtitles:

```
youtube_transcript_api <first_video_id> <second_video_
youtube_transcript_api <first_video_id> <second_video_
```

If you would prefer to write it into a file or pipe it into another application, you can also output the results as json using the following line:

```
youtube_transcript_api <first_video_id> <second_video_
```

Translating transcripts using the CLI is also possible:

If you are not sure which languages are available for a given video you can call, to list all available transcripts:

```
youtube_transcript_api --list-transcripts <first_video_
```

If a video's ID starts with a hyphen you'll have to mask the hyphen using `\` to prevent the CLI from mistaking it for a argument name. For example to get the transcript for the video with the ID `-abc123` run:

```
youtube_transcript_api "\-abc123"
```

Working around IP bans using the CLI

If you are running into `ReqestBlocked` or `IpBlocked` errors, because YouTube blocks your IP, you can work around this using residential proxies as explained in Working around IP bans. To use Webshare "Residential" proxies through the CLI, you will have to create a Webshare account and purchase a "Residential" proxy package that suits your workload (make sure NOT to purchase "Proxy Server" or "Static Residential"!). Then you can use the "Proxy Username" and "Proxy Password" which you can find in your Webshare Proxy Settings, to run the following command:

```
youtube_transcript_api <first_video_id> <second_video_
```

If you prefer to use another proxy solution, you can set up a generic HTTP/HTTPS proxy using the following command:

```
youtube_transcript_api <first_video_id> <second_video_
```

To authenticate using cookies through the CLI as explained in Cookie Authentication run:

```
youtube_transcript_api <first_video_id> <second_video_
```

Warning

This code uses an undocumented part of the YouTube API, which is called by the YouTube web-client. So there is no guarantee that it won't stop working tomorrow, if they change how things work. I will however do my best to make things working again as soon as possible if that happens. So if it stops working, let me know!

Contributing

To setup the project locally run the following (requires poetry to be installed):

```
poetry install --with test,dev
```

There's poe tasks to run tests, coverage, the linter and formatter (you'll need to pass all of those for the build to pass):

```
poe test
poe coverage
poe format
poe lint
```

If you just want to make sure that your code passes all the necessary checks to get a green build, you can simply run:

```
poe precommit
```

If this project makes you happy by reducing your development time, you can make me happy by treating me to a cup of coffee, or become a Sponsor of this project :)