nficano / **pytube**

A lightweight, dependency-free Python library (and command-line utility) for downloading YouTube Videos. https://python-pytube.readthedocs.io

#python #youtube #pythonic #api-wrapper

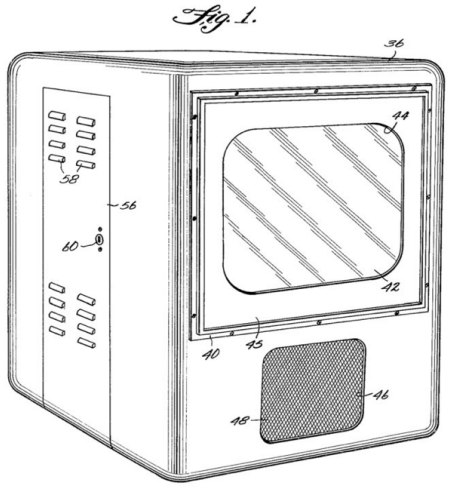
756 commits | **2** branches | **52** releases | **45** contributors

Branch: **master** | New pull request | Create new file | Upload files | Find file | Clone or download

nficano Bump version: 9.2.2 → 9.2.3 — Latest commit `05287f2` 5 hours ago

docs	Update index.rst	2 months ago
images	added logo	2 months ago
pytube	Bump version: 9.2.2 → 9.2.3	5 hours ago
tests	Added playlist location test	2 months ago
.envrc	updated envrc	2 months ago
.gitattributes	fixed language	7 months ago
.gitignore	documentation	6 months ago
.pre-commit-config.yaml	added precommit hooks	6 months ago
.travis.yml	rem 3.3 support	27 days ago
CODE_OF_CONDUCT.md	Create CODE_OF_CONDUCT.md	6 months ago
LICENSE	Merge branch 'master' of github.com:nficano/pytube	10 months ago
MANIFEST.in	fixed missing licence file	9 months ago
Makefile	rewired up flake8	27 days ago
Pipfile	come on	27 days ago
Pipfile.lock	come on	27 days ago
README.md	Update README.md	11 days ago
setup.cfg	Bump version: 9.2.2 → 9.2.3	5 hours ago
setup.py	Bump version: 9.2.2 → 9.2.3	5 hours ago

README.md



pytube

pytube is a very serious, lightweight, dependency-free Python library (and command-line utility) for downloading YouTube Videos.

Description

YouTube is the most popular video-sharing platform in the world and as a hacker you may encounter a situation where you want to script something to download videos. For this I present to you *pytube*.

pytube is a lightweight library written in Python. It has no third party dependencies and aims to be highly reliable.

pytube also makes pipelining easy, allowing you to specify callback functions for different download events, such as `on progress` or `on complete`.

Finally *pytube* also includes a command-line utility, allowing you to quickly download videos right from terminal.

Behold, a perfect balance of simplicity versus flexibility:

```
>>> YouTube('https://youtu.be/9bZkp7q19f0').streams.first().download()
>>> yt = YouTube('http://youtube.com/watch?v=9bZkp7q19f0')
>>> yt.streams
 ... .filter(progressive=True, file_extension='mp4')
 ... .order_by('resolution')
 ... .desc()
 ... .first()
 ... .download()
```

Features

- Support for Both Progressive & DASH Streams
- Support for downloading complete playlist
- Easily Register `on_download_progress` & `on_download_complete` callbacks
- Command-line Interfaced Included
- Caption Track Support
- Outputs Caption Tracks to .srt format (SubRip Subtitle)

- Ability to Capture Thumbnail URL.
- Extensively Documented Source Code
- No Third-Party Dependencies

Installation

Download using pip via pypi.

```
$ pip install pytube
```

Getting started

Let's begin with showing how easy it is to download a video with pytube:

```
>>> from pytube import YouTube
>>> YouTube('http://youtube.com/watch?v=9bZkp7q19f0').streams.first().download()
```

This example will download the highest quality progressive download stream available.

Next, let's explore how we would view what video streams are available:

```
>>> yt = YouTube('http://youtube.com/watch?v=9bZkp7q19f0')
>>> yt.streams.all()
 [<Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.2
 <Stream: itag="43" mime_type="video/webm" res="360p" fps="30fps" vcodec="vp8.0" acodec="vorbis">,
 <Stream: itag="18" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.42001E" acodec="mp4a.40.2"
 <Stream: itag="36" mime_type="video/3gpp" res="240p" fps="30fps" vcodec="mp4v.20.3" acodec="mp4a.40.2">
 <Stream: itag="17" mime_type="video/3gpp" res="144p" fps="30fps" vcodec="mp4v.20.3" acodec="mp4a.40.2">
 <Stream: itag="137" mime_type="video/mp4" res="1080p" fps="30fps" vcodec="avc1.640028">,
 <Stream: itag="248" mime_type="video/webm" res="1080p" fps="30fps" vcodec="vp9">,
 <Stream: itag="136" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.4d401f">,
 <Stream: itag="247" mime_type="video/webm" res="720p" fps="30fps" vcodec="vp9">,
 <Stream: itag="135" mime_type="video/mp4" res="480p" fps="30fps" vcodec="avc1.4d401e">,
 <Stream: itag="244" mime_type="video/webm" res="480p" fps="30fps" vcodec="vp9">,
 <Stream: itag="134" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.4d401e">,
 <Stream: itag="243" mime_type="video/webm" res="360p" fps="30fps" vcodec="vp9">,
 <Stream: itag="133" mime_type="video/mp4" res="240p" fps="30fps" vcodec="avc1.4d4015">,
 <Stream: itag="242" mime_type="video/webm" res="240p" fps="30fps" vcodec="vp9">,
 <Stream: itag="160" mime_type="video/mp4" res="144p" fps="30fps" vcodec="avc1.4d400c">,
 <Stream: itag="278" mime_type="video/webm" res="144p" fps="30fps" vcodec="vp9">,
 <Stream: itag="140" mime_type="audio/mp4" abr="128kbps" acodec="mp4a.40.2">,
 <Stream: itag="171" mime_type="audio/webm" abr="128kbps" acodec="vorbis">,
 <Stream: itag="249" mime_type="audio/webm" abr="50kbps" acodec="opus">,
 <Stream: itag="250" mime_type="audio/webm" abr="70kbps" acodec="opus">,
 <Stream: itag="251" mime_type="audio/webm" abr="160kbps" acodec="opus">]
```

You may notice that some streams listed have both a video codec and audio codec, while others have just video or just audio, this is a result of YouTube supporting a streaming technique called Dynamic Adaptive Streaming over HTTP (DASH).

In the context of pytube, the implications are for the highest quality streams; you now need to download both the audio and video tracks and then post-process them with software like FFmpeg to merge them.

The legacy streams that contain the audio and video in a single file (referred to as "progressive download") are still available, but only for resolutions 720p and below.

To only view these progressive download streams:

```
>>> yt.streams.filter(progressive=True).all()
 [<Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.
  <Stream: itag="43" mime_type="video/webm" res="360p" fps="30fps" vcodec="vp8.0" acodec="vorbis">,
  <Stream: itag="18" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.42001E" acodec="mp4a.40.2
  <Stream: itag="36" mime_type="video/3gpp" res="240p" fps="30fps" vcodec="mp4v.20.3" acodec="mp4a.40.2"
  <Stream: itag="17" mime_type="video/3gpp" res="144p" fps="30fps" vcodec="mp4v.20.3" acodec="mp4a.40.2"
```

Conversely, if you only want to see the DASH streams (also referred to as "adaptive") you can do:

```
>>> yt.streams.filter(adaptive=True).all()
 [<Stream: itag="137" mime_type="video/mp4" res="1080p" fps="30fps" vcodec="avc1.640028">,
  <Stream: itag="248" mime_type="video/webm" res="1080p" fps="30fps" vcodec="vp9">,
  <Stream: itag="136" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.4d401f">,
  <Stream: itag="247" mime_type="video/webm" res="720p" fps="30fps" vcodec="vp9">,
  <Stream: itag="135" mime_type="video/mp4" res="480p" fps="30fps" vcodec="avc1.4d401e">,
  <Stream: itag="244" mime_type="video/webm" res="480p" fps="30fps" vcodec="vp9">,
  <Stream: itag="134" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.4d401e">,
  <Stream: itag="243" mime_type="video/webm" res="360p" fps="30fps" vcodec="vp9">,
  <Stream: itag="133" mime_type="video/mp4" res="240p" fps="30fps" vcodec="avc1.4d4015">,
  <Stream: itag="242" mime_type="video/webm" res="240p" fps="30fps" vcodec="vp9">,
  <Stream: itag="160" mime_type="video/mp4" res="144p" fps="30fps" vcodec="avc1.4d400c">,
  <Stream: itag="278" mime_type="video/webm" res="144p" fps="30fps" vcodec="vp9">,
  <Stream: itag="140" mime_type="audio/mp4" abr="128kbps" acodec="mp4a.40.2">,
  <Stream: itag="171" mime_type="audio/webm" abr="128kbps" acodec="vorbis">,
  <Stream: itag="249" mime_type="audio/webm" abr="50kbps" acodec="opus">,
  <Stream: itag="250" mime_type="audio/webm" abr="70kbps" acodec="opus">,
  <Stream: itag="251" mime_type="audio/webm" abr="160kbps" acodec="opus">]
```

You can also download a complete Youtube playlist:

```
>>> from pytube import Playlist
>>> pl = Playlist("https://www.youtube.com/watch?v=Edpy1szoG80&list=PL153hDY-y1E00uQtCVCVC8xJ25TYX8yPU")
>>> pl.download_all()
>>> # or if you want to download in a specific directory
>>> pl.download_all('/path/to/directory/')
```

This will download the highest progressive stream available (generally 720p) from the given playlist. Later more options would be given for user's flexibility to choose video resolution.

Pytube allows you to filter on every property available (see the documentation for the complete list), let's take a look at some of the most useful ones.

To list the audio only streams:

```
>>> yt.streams.filter(only_audio=True).all()
  [<Stream: itag="140" mime_type="audio/mp4" abr="128kbps" acodec="mp4a.40.2">,
  <Stream: itag="171" mime_type="audio/webm" abr="128kbps" acodec="vorbis">,
  <Stream: itag="249" mime_type="audio/webm" abr="50kbps" acodec="opus">,
  <Stream: itag="250" mime_type="audio/webm" abr="70kbps" acodec="opus">,
  <Stream: itag="251" mime_type="audio/webm" abr="160kbps" acodec="opus">]
```

To list only `mp4` streams:

```
>>> yt.streams.filter(subtype='mp4').all()
 [<Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.2
  <Stream: itag="18" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.42001E" acodec="mp4a.40.2
  <Stream: itag="137" mime_type="video/mp4" res="1080p" fps="30fps" vcodec="avc1.640028">,
  <Stream: itag="136" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.4d401f">,
```

```
    <Stream: itag="135" mime_type="video/mp4" res="480p" fps="30fps" vcodec="avc1.4d401e">,
    <Stream: itag="134" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.4d401e">,
    <Stream: itag="133" mime_type="video/mp4" res="240p" fps="30fps" vcodec="avc1.4d4015">,
    <Stream: itag="160" mime_type="video/mp4" res="144p" fps="30fps" vcodec="avc1.4d400c">,
    <Stream: itag="140" mime_type="audio/mp4" abr="128kbps" acodec="mp4a.40.2">]
```

Multiple filters can also be specified:

```
>>> yt.streams.filter(subtype='mp4', progressive=True).all()
>>> # this can also be expressed as:
>>> yt.streams.filter(subtype='mp4').filter(progressive=True).all()
  [<Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.
  <Stream: itag="18" mime_type="video/mp4" res="360p" fps="30fps" vcodec="avc1.42001E" acodec="mp4a.40.2
```

You also have an interface to select streams by their itag, without needing to filter:

```
>>> yt.streams.get_by_itag(22)
  <Stream: itag="22" mime_type="video/mp4" res="720p" fps="30fps" vcodec="avc1.64001F" acodec="mp4a.40.2
```

If you need to optimize for a specific feature, such as the "highest resolution" or "lowest average bitrate":

```
>>> yt.streams.filter(progressive=True).order_by('resolution').desc().all()
```

Note that `order_by` cannot be used if your attribute is undefined in any of the Stream instances, so be sure to apply a filter to remove those before calling it.

If your application requires post-processing logic, pytube allows you to specify an "on download complete" callback function:

```
>>> def convert_to_aac(stream, file_handle):
        return  # do work

>>> yt.register_on_complete_callback(convert_to_aac)
```

Similarly, if your application requires on-download progress logic, pytube exposes a callback for this as well:

```
>>> def show_progress_bar(stream, chunk, file_handle, bytes_remaining):
        return  # do work

>>> yt.register_on_progress_callback(show_progress_bar)
```

Command-line interface

pytube also ships with a tiny cli interface for downloading and probing videos.

Let's start with downloading:

```
$ pytube http://youtube.com/watch?v=9bZkp7q19f0 --itag=22
```

To view available streams:

```
$ pytube http://youtube.com/watch?v=9bZkp7q19f0 --list
```

Finally, if you're filing a bug report, the cli contains a switch called `--build-playback-report`, which bundles up the state, allowing others to easily replay your issue.