

---

# **Youtube Python**

*Release 2.0.0*

**Mayank Gupta**

**Jun 05, 2021**



# INSTALL

<b>1</b>	<b>Features</b>	<b>3</b>
1.1	Installation . . . . .	3
1.2	Basic Codes . . . . .	4
1.3	Advance Codes . . . . .	8
1.4	The API Documentation / Help . . . . .	10
1.5	Command Line Input . . . . .	19
1.6	YOU CAN HELP . . . . .	20
1.7	PROJECT DETAILS . . . . .	21
	<b>Python Module Index</b>	<b>23</b>
	<b>Index</b>	<b>25</b>





Current version: v2.0.0

**Youtube Python** is best and powerful python module for developers internal use *APIs* or command line *CLI* , Downloading specific video or playlist if now easy.



## FEATURES

- Supports proxy (for scraping data, proxy will not be used for downloading videos)
- Rate limit bypass
- Light module
- Command line input available
- Supports FFMPEG
- Can parsec youtube video details
- Youtube Search, Video, PlayList

## 1.1 Installation

Ways to install

### 1.1.1 Using PIP

run this command on terminal.

```
pip install -U youtube.py
```

Have multipls versions of python?

```
python3.x -m pip install -U youtube.py
```

### 1.1.2 Install FFMPEG

Windows download [link](#). After download the exe file make sure to put that on environment variables or simply move ffmpeg.exe to c:\windows\system32\

Linux

```
sudo apt update  
sudo apt install ffmpeg
```

## 1.2 Basic Codes

### 1.2.1 Class Video

#### Sample code for Video class overview

```
from youtube import Video

video_url = "https://www.youtube.com/watch?v=3AtDnEC4zak"

v = Video(video_url)

#Print Title
print(v.title)

#Print Author name
print(v.author)

#Print Channel ID
print(v.channel)

#Print video description
print(v.description)

#Print video length in second
print(v.length)

#Print video keywords in dict
print(v.keywords)

#Print high resolution thumbnail
print(v.thumbnail)

#Print video id
print(v.videoId)

#Print total video views
print(v.views)
```

#### Streams of Video class overview

```
>>> from youtube import Video
>>> v = Video("https://www.youtube.com/watch?v=3AtDnEC4zak")
>>> v.title
"Charlie Puth - We Don't Talk Anymore (feat. Selena Gomez) [Official Video]"
>>> v.length
231
>>> v.views
2550001850
>>> v.streams
[<Class Stream: itag="18" qualityLabel="360p" fps="24" mime="video/mp4" type="both" ↵
↵format="mp4" codecs="avc1.42001E,+mp4a.40.2">,
<Class Stream: itag="133" qualityLabel="240p" fps="24" mime="video/mp4" type="video" ↵
↵format="mp4" codecs="avc1.4d4015">,
(continues on next page)
```



(continued from previous page)

```

<Class Stream: itag="134" qualityLabel="360p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="avc1.4d401e">,
<Class Stream: itag="135" qualityLabel="480p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="avc1.4d401e">,
<Class Stream: itag="136" qualityLabel="720p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="avc1.4d401f">,
<Class Stream: itag="137" qualityLabel="1080p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="avc1.640028">,
<Class Stream: itag="160" qualityLabel="144p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="avc1.4d400c">,
<Class Stream: itag="242" qualityLabel="240p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="243" qualityLabel="360p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="244" qualityLabel="480p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="247" qualityLabel="720p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="248" qualityLabel="1080p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="249" bitrate="59056" mime="audio/webm" type="audio" format="webm"
↳ codecs="opus">,
<Class Stream: itag="250" bitrate="76035" mime="audio/webm" type="audio" format="webm"
↳ codecs="opus">,
<Class Stream: itag="251" bitrate="143852" mime="audio/webm" type="audio" format="webm"
↳ codecs="opus">,
<Class Stream: itag="278" qualityLabel="144p" fps="24" mime="video/webm" type="video"
↳ format="webm" codecs="vp9">,
<Class Stream: itag="394" qualityLabel="144p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.00M.08">,
<Class Stream: itag="395" qualityLabel="240p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.00M.08">,
<Class Stream: itag="396" qualityLabel="360p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.01M.08">,
<Class Stream: itag="397" qualityLabel="480p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.04M.08">,
<Class Stream: itag="398" qualityLabel="720p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.05M.08">,
<Class Stream: itag="399" qualityLabel="1080p" fps="24" mime="video/mp4" type="video"
↳ format="mp4" codecs="av01.0.08M.08">]
>>> type(v.streams)
<class 'youtube.Stream.Queue'>
>>> v.thumbnail
'https://i.ytimg.com/vi/3AtDnEC4zak/maxresdefault.jpg'
>>> v.get_dict
[{'itag': 18, 'mimeType': 'video/mp4', 'bitrate': 395026, 'width': 640, 'height': 360,
↳ 'lastModified': '1580859977121104', 'contentLength': '11378647', 'quality': 'medium',
↳ 'fps': 24, 'qualityLabel': '360p', 'projectionType': 'RECTANGULAR', 'averageBitrate':
↳ 394913, 'audioQuality': 'AUDIO_QUALITY_LOW', 'approxDurationMs': '230504',
↳ 'audioSampleRate': '44100', 'audioChannels': 2, 'type': 'both', 'format': 'mp4',
↳ 'codecs': 'avc1.42001E,+mp4a.40.2', 's':
↳ 'mAq0QJ8wRQIgRe0DKBgGi4uiqHOk6dmrhwQKtFCcrhCKAvNHps-hzk0CIQDDW_HSmfLlD3-
↳ 4G5x3XtamqDtAWRVkjYYGv_qFYUY40w====', 'sp': 'sig', 'url': 'https://r7---scontentgpe1p
↳ googlevideo.com/videoplayback?expire=1606933954&ei=YonHX_XTBIKqvQXsv4-QDg .....

```

(continued on next page)

## 1.2.2 Class Playlist

```

>>> from youtube import Playlist, HIGH
>>> p = Playlist("https://www.youtube.com/playlist?list=PLexILI4F6_4XJau4ZlJXD7-
↳K75g6rDebr")
>>> p.get_dict
[{'vid': 'C-p3HnovSbU', 'name': 'Top ABBA Songs Playlist #ABBA Best Mix'},
 {'vid': 'u44-KMrI4Hk', 'name': 'Top 100 Traditional Christmas Songs Ever - Best Classic
↳Christmas Songs 2021 Collection'},
 {'vid': 'KOICR55wlGY', 'name': 'Merry Christmas 2021 Beautiful Traditional Christmas
↳Songs Playlist Classic Christmas Songs'},
 {'vid': 'jiV7hyX65BM', 'name': 'ABBA GOLD GREATEST HITS - ABBA FULL ALBUM PLAYLIST'},
 {'vid': 'Q3YkR_Eyh-s', 'name': 'Best Christmas Songs Playlist 2020 - Top 10 Christmas
↳Songs || Merry Christmas Music Of All Time'},
 {'vid': 'fmfy9-Z4ZLc', 'name': 'Best Songs Of ABBA Collection 2020 | ABBA Greatest Hits
↳Full Album'},
 {'vid': '-PJDukRSHis', 'name': 'ABBA Best Songs Collection 2020 | Greatest Hits New
↳Playlist Of ABBA'},
 {'vid': 'ptaok2DmN0Y', 'name': 'Christmas 2020 Christmas Songs 2020 - 2021 Nonstop
↳Christmas Songs Medley 2020 - 2021'},
 {'vid': 'dLU23pGXNF8', 'name': 'NEW ABBA Full Album Playlist || The Very Best Song
↳#ABBA GOLD'}]
>>> p.get_object
[<Class Video VID=C-p3HnovSbU PROXY=None ID=0 NAME=Top ABBA Songs Playlist #ABBA Best
↳Mix 0x7f34987cd2d0>,
 <Class Video VID=u44-KMrI4Hk PROXY=None ID=1 NAME=Top 100 Traditional Christmas Songs
↳Ever - Best Classic Christmas Songs 2021 Collection 0x7f34987cdcd0>,
 <Class Video VID=KOICR55wlGY PROXY=None ID=2 NAME=Merry Christmas 2021 Beautiful
↳Traditional Christmas Songs Playlist Classic Christmas Songs 0x7f3497f60750>,
 <Class Video VID=jiV7hyX65BM PROXY=None ID=3 NAME=ABBA GOLD GREATEST HITS - ABBA FULL
↳ALBUM PLAYLIST 0x7f3497f60e50>,
 <Class Video VID=Q3YkR_Eyh-s PROXY=None ID=4 NAME=Best Christmas Songs Playlist 2020 -
↳Top 10 Christmas Songs || Merry Christmas Music Of All Time 0x7f3497f74bd0>,
 <Class Video VID=fmfy9-Z4ZLc PROXY=None ID=5 NAME=Best Songs Of ABBA Collection 2020 |
↳ABBA Greatest Hits Full Album 0x7f3497f7ae50>,
 <Class Video VID=-PJDukRSHis PROXY=None ID=6 NAME=ABBA Best Songs Collection 2020 |
↳Greatest Hits New Playlist Of ABBA 0x7f3497f8b0d0>,
 <Class Video VID=ptaok2DmN0Y PROXY=None ID=7 NAME=Christmas 2020 Christmas Songs 2020 -
↳2021 Nonstop Christmas Songs Medley 2020 - 2021 0x7f3497f8bad0>,
 <Class Video VID=dLU23pGXNF8 PROXY=None ID=8 NAME=NEW ABBA Full Album Playlist || The
↳Very Best Song #ABBA GOLD 0x7f349470ea10>]
>>> p.get_dict
[<Class Video VID=C-p3HnovSbU PROXY=None ID=0 NAME=Top ABBA Songs Playlist #ABBA Best
↳Mix 0x7f34987cd2d0>,
 <Class Video VID=u44-KMrI4Hk PROXY=None ID=1 NAME=Top 100 Traditional Christmas Songs
↳Ever - Best Classic Christmas Songs 2021 Collection 0x7f34987cdcd0>,
 <Class Video VID=KOICR55wlGY PROXY=None ID=2 NAME=Merry Christmas 2021 Beautiful
↳Traditional Christmas Songs Playlist Classic Christmas Songs 0x7f3497f60750>,
 <Class Video VID=jiV7hyX65BM PROXY=None ID=3 NAME=ABBA GOLD GREATEST HITS - ABBA FULL
↳ALBUM PLAYLIST 0x7f3497f60e50>,

```

(continues on next page)

(continued from previous page)

```
<Class Video VID=Q3YkR_Eyh-s PROXY=None ID=4 NAME=Best Christmas Songs Playlist 2020 -
↳Top 10 Christmas Songs || Merry Christmas Music Of All Time 0x7f3497f74bd0>,
<Class Video VID=fmfy9-Z4ZLc PROXY=None ID=5 NAME=Best Songs Of ABBA Collection 2020 |
↳ABBA Greatest Hits Full Album 0x7f3497f7ae50>,
<Class Video VID=-PJDUkRSHis PROXY=None ID=6 NAME=ABBA Best Songs Collection 2020 |
↳Greatest Hits New Playlist Of ABBA 0x7f3497f8b0d0>,
<Class Video VID=ptaok2DmN0Y PROXY=None ID=7 NAME=Christmas 2020 Christmas Songs 2020 -
↳2021 Nonstop Christmas Songs Medley 2020 - 2021 0x7f3497f8bad0>,
<Class Video VID=dLU23pGXNF8 PROXY=None ID=8 NAME=NEW ABBA Full Album Playlist || The
↳Very Best Song #ABBA GOLD 0x7f349470ea10>]
```

p.downloadall(HIGH) will download full playlist in HIGH quality

### 1.2.3 Class Search

```
>>> from youtube import Search
>>> s = Search("taylor swift - willow")
>>> s.videos
[{'vid': 'RsEZmictANA', 'name': 'taylor swift - willow (official music video)'}, {'vid':
↳'7EvwIw4gLyk', 'name': 'Taylor Swift - willow (Official Lyric Video)'}, {'vid':
↳'zI4DS5GmQWE', 'name': 'Taylor Swift - dorothea (Official Lyric Video)'}, {'vid':
↳'hP6QpMeSG6s', 'name': 'Taylor Swift - marjorie (Official Lyric Video)'}, {'vid':
↳'wDw8RCwmcKg', 'name': 'Taylor Swift - Willow (Lyrics)'}, {'vid': 'IEPomqor2A8', 'name
↳': 'Taylor Swift - no body, no crime (Official Lyric Video) ft. HAIM'}, {'vid': '-
↳qQogoNwJdM', 'name': "New TAYLOR SWIFT?! 'Willow' - My First Watch/Reaction!"}, {'vid
↳': 'Oi2Vw0n2EfM', 'name': 'Taylor Swift - willow | Video Explained (Analysis, Easter
↳eggs And More)'}, {'vid': 'OEd32AL-exA', 'name': 'Taylor Swift - Willow - Guitar Lesson
↳'}, {'vid': 'jepo9tm22ig', 'name': 'Taylor Swift - Willow Cover'}, {'vid': 'qPOw4p360Lc
↳', 'name': 'Taylor Swift - willow | Piano Cover by Pianella Piano'}, {'vid':
↳'2TgkjFVqdJ0', 'name': 'Taylor Swift - willow - Music Video - REACTION'}, {'vid':
↳'ThOVwAmmL0o', 'name': 'Taylor Swift - Willow (Lyrics)'}, {'vid': 'NLIacgCnwFA', 'name
↳': 'Taylor Swift - Willow - Reaction/Review'}, {'vid': '7B_DjCNKgHU', 'name': 'Taylor
↳Swift - willow (Lyrics)'}, {'vid': '06IayU4FPmI', 'name': "Vocal Coach Reacts to
↳Taylor Swift 'Willow' Evermore"}, {'vid': 'ufozmQSSy7c', 'name': 'Taylor Swift -
↳Willow Reaction'}, {'vid': '1PKgktWMzNY', 'name': 'Taylor Swift - willow | Piano
↳Tutorial'}, {'vid': '0ldUWIRURuw', 'name': 'taylor swift - willow - Piano Karaoke
↳Instrumental Cover with Lyrics'}, {'vid': 'SqX8vbkG5_Q', 'name': 'Taylor Swift -
↳willow (Piano Tutorial Easy)'}]
>>> s.first
<Class Video VID=RsEZmictANA PROXY=None ID=1 NAME=taylor swift - willow (official music
↳video) 0x7f15a34c5a90>
>>>
```

## 1.2.4 How to get all title from playlist videos

A sample code to get all titles of playlist videos

```
from youtube import PlayList

video_url = "https://www.youtube.com/playlist?list=PLexILI4F6_4XJau4ZlJXD7-K75g6rDebr"

p = PlayList(video_url)

for n in p.get_dict: # This will return the raw dict with vid and name since we didnt_
    ↪used process=True
    print(n.get("name")) # This will print the title name
```

## 1.2.5 How to get description from all playlist video

A sample code to get description from all playlist video

```
from youtube import PlayList

video_url = "https://www.youtube.com/playlist?list=PLexILI4F6_4XJau4ZlJXD7-K75g6rDebr"

p = PlayList(video_url, process=True)

for n in p.get_dict: # This will return the Video class object since we used process=True
    print(n.description, end="\n\n") # This will print the description
```

## 1.2.6 How to get videos search query

Sample code to get videos by search

```
from youtube import Search

r = Search("taylor swift - willow")

print(r.get_dict)
```

## 1.3 Advance Codes

### 1.3.1 How to download mass videos with ffmpeg.

```
from youtube import Video

video_url = ["3AtDnEC4zak", "3AtDnEC4zak"]

for vdo in video_url:
    v = Video(f"https://youtu.be/{vdo}").streams
    v.ffhigh.download() # ffhigh will download stream with highest quality
```

For download low and mid quality use `ffmid` and `fflow`

Use `status=True` in download to show process bar

### 1.3.2 How to download mass videos with ffmpeg and resolutions.

```
from youtube import Video

video_url = ["3AtDnEC4zak", "3AtDnEC4zak"]

for vdo in video_url:
    v = Video(f"https://youtu.be/{vdo}").streams
    v.ffresolution(quality=480).download() # ffresolution will return the FFMPEG_
↳class
```

480 is stream quality/resolution.

### 1.3.3 How to download all playlist videos concurrently/fast.

```
from youtube import Playlist

video_url = "https://www.youtube.com/playlist?list=PLexILI4F6_4XJau4ZlJXD7-K75g6rDebr"

p = Playlist(video_url, process=True) # process=True to generate all Video objects

for n in p.get_object:
    print(n) # n is the Video object
```

`p.get_object` will return the list of Video

now we will use `concurrent.futures` for concurrently download

```
from youtube import Playlist
import concurrent.futures

MAX_CONCURRENT_DOWNLOADS = 1 #max concurrent download at ones

video_url = "https://www.youtube.com/playlist?list=PLexILI4F6_4XJau4ZlJXD7-K75g6rDebr"

p = Playlist(video_url, process=True) # process=True to generate all Video objects

func = lambda x: x.streams.get_audios.low().download()

with concurrent.futures.ThreadPoolExecutor(max_workers=MAX_CONCURRENT_DOWNLOADS) as
↳executor:
    for n in executor.map(func,p.get_object):
        print(f'{n} Downloaded')
```

**Note:** Be careful when using concurrent set `MAX_CONCURRENT_DOWNLOADS` according to your internet speed connection will get close after 5 second if no data recv

### 1.3.4 Other examples

```
from youtube import Video

video_url = ["3AtDnEC4zak", "3AtDnEC4zak"]

for vdo in video_url:
    v = Video(f"https://youtu.be/{vdo}").streams
    v.get_both.high()
```

get\_both will return the streams with audio/video both and high will return the first stream from the list example return self.data[0] like get\_both.high() or get\_both[0] are the same.

high() will return the first stream so ofcause it will a Stream

with .url you can even call the stream url of that perticular stream ie. v.get\_both.high().url

```
from youtube import Video

video_url = ["3AtDnEC4zak", "3AtDnEC4zak"]

for vdo in video_url:
    v = Video(f"https://youtu.be/{vdo}").streams
    url = v.get_both.high().url
    print(url)
```

## 1.4 The API Documentation / Help

If you want in depth details of functions and class here is the docs.

### 1.4.1 YouTube Class

```
class youtube.Video(url: str, proxy: str = "", id: int = 1, http: Optional[youtube.Connection.HTTP] = None,
                    name: str = "")
```

Construct a *Video* object.

#### Parameters

- **url** (*str*) – Pass youtube valid video url. ie. <https://www.youtube.com/watch?v=9NQqaKz7eyI>
- **proxy** (*str*) – (optional) pass the proxy url example:  
socks5://admin:admin@127.0.0.1:9050, http://admin:admin@127.0.0.1:8000, https://127.0.0.1:8000
- **id** (*object*) – (optional) Video class id just to identify easily.
- **id** – (optional) Pass the HTTP object.
- **name** (*int*) – (optional) if you want specify name.

**Return type** None

**Returns** None

**property author: str**

This will return the video author.

**Return type** str

**Returns** str

**property channel: str**

This will return the video channel id.

**Return type** str

**Returns** str

**property description: str**

This will return the video description.

**Return type** str

**Returns** str

**property get\_dict**

This will return the raw dict of video streams urls

**property keywords: List[str]**

This will return the video keywords.

**Return type** List[str]

**Returns** List[str]

**property length: int**

This will return the video time in second.

**Return type** int

**Returns** int

**property streams: youtube.Stream.Queue**

Will returns *Queue*

**Return type** *Queue*

**Returns** Returns the Queue object

**property thumbnail: str**

This will return the video thumbnail.

**Return type** str

**Returns** str

**property title: str**

This will return the video title.

**Return type** str

**Returns** str

**property videoId: str**

This will return the video id.

**Return type** str

**Returns** str

**property views: int**

This will return the total video views.

**Return type** `int`

**Returns** `int`

## 1.4.2 PlayList Class

**class** `youtube.PlayList`(*url: str, proxy: str = "", process: bool = False*)

Construct a *PlayList*

### Parameters

- **url** (*str*) – Pass youtube valid playlist url. ie. <https://www.youtube.com/watch?v=9NQqaKz7eyI>
- **proxy** (*str*) – (optional) pass the proxy url example:  
`socks5://admin:admin@127.0.0.1:9050`    `http://admin:admin@127.0.0.1:8000`    `https://127.0.0.1:8080`
- **process** (*bool*) – (optional) Process will trigger the function for making objects of *Video* with the list of playlist videos default (False), When (True) it will take some extra time depend on your CPU and Internet speed. Process will use 10 workers to create objects fasts.

**downloadall**(*quality: int*) → `None`

This is to download all playlist videos, remember this will download all videos synchronizly. Thiw function will download one file (audio/video) you may not satisfied with the quality.

Quality	Value
HIGH	0
MID	1
LOW	2 or n

**Parameters** **quality** (*int*) – Pass the quality type

**property** `get_dict`

This will return the list of *Video* if available or will just return the dict of vid and name, Availability of *Video* objects is depend on your `PlayList(url, process=False)`, process will process all videos to *Video* objects.

**property** `get_object`

This will return the list of *Video* objects, even when you `process=False`.

**processing**()

Processing will process all videos to *Video* object

## 1.4.3 Stream Class

**class** `youtube.Stream`(*stream: Dict[Any, Any], name: str*)

Here we will process *Stream*

### Parameters

- **stream** (*Dict [Any, Any]*) – singal dict of stream json
- **name** (*str*) – Pass the video title



**check**(*a: str*) → *str*

Here we will take a and return the values, a could be qualityLabel or type qualityLabel: will return resolution of the video type: will return video, audio or both

**download**(*dire: str = "*, *name: str = "*, *status: bool = False*, *connection: int = 8*, *chunk: int = 5120*) → *youtube.Downloader.Download*

This will return the *Download*

#### Parameters

- **name** (*str*) – (optional) Pass the name of file name.
- **dire** (*str*) – (optional) Pass the dir for output file with excluding filename.
- **status** (*bool*) – (optional) Pass if you want to enable process bar. Default[False]
- **connection** (*int*) – (optional) Pass number is connection to be create. Default[8]
- **chunk** (*int*) – (optional) Pass the chunk/buffer size for accepting packets. Default[5120]

**Return type** *Download*

**Returns** *Download* object

**property format: str**

This will return the stream format

**Return type** *str*

**Returns** format of video in str

**property is\_audio: bool**

Check is the stream is audio

**Return type** *bool*

**Returns** check the stream and return bool(true, false)

**property is\_both: bool**

Check is the stream has audio/video

**Return type** *bool*

**Returns** check the stream and return bool(true, false)

**property is\_video: bool**

Check is the stream is video

**Return type** *bool*

**Returns** check the stream and return bool(true, false)

**property itag: int**

This will return the stream format

**Return type** *int*

**Returns** itag of video in int

**property url: str**

This will return the URL of this Stream

**Return type** *str*

**Returns** url of video in str

## 1.4.4 Queue Class

**class** `youtube.Queue`(*data: List[youtube.Stream.Stream]*, *repeat: bool = False*)

We will manage the *Stream* queues here

### Parameters

- **data** (*List[Stream]*) – pass the dict of *Stream* objects.
- **repeat** (*bool*) – if you re generating Queue object from inside the function.

**property** `all: List[youtube.Stream.Stream]`

all will return the *Stream* objects in list

**Return type** List of *Stream*

**Returns** list of *Stream* objects

**property** `ffhigh: youtube.Stream.FFMPEG`

Processing with highest quality video and audio

**Return type** *FFMPEG*

**Returns** *FFMPEG* object

**property** `fflow: youtube.Stream.FFMPEG`

Processing with lowest quality video and audio

**Return type** *FFMPEG*

**Returns** *FFMPEG* object

**property** `ffmid: youtube.Stream.FFMPEG`

Processing with mid quality video and audio

**Return type** *FFMPEG*

**Returns** *FFMPEG* object

**ffresolution**(*quality: str*) → *youtube.Stream.FFMPEG*

Processing audio/video depend on resolution

**Parameters** **quality** (*int*) – pass the quality type, Examples. ('144', '240', '360', '480', '720', '1080') more/less depend on your video.

**Return type** *FFMPEG*

**Returns** *FFMPEG* object

**format**(*f: str*) → *youtube.Stream.Queue*

This will return all streams with format match

**Parameters** **f** (*str*) – pass the format type ie. 'mp4'.

**Return type** *Queue*

**Returns** *Queue* object

**property** `get_audios: youtube.Stream.Queue`

This will return all audios stream without video.

**Return type** *Queue*

**Returns** *Queue* object

**property** `get_both: youtube.Stream.Queue`

This will return all audios/video stream.

**Return type** *Queue*

**Returns** *Queue* object

**get\_itag**(*i: int*) → Optional[*youtube.Stream.Stream*]

This will return the stream, matched with i <arg>

**Parameters** **i** (*int*) – pass the itag id for match.

**Return type** Union[*Stream*, None]

**Returns** Will return *Stream* or None

**property get\_videos:** **youtube.Stream.Queue**

This will return all videos stream without audio.

**Return type** *Queue*

**Returns** *Queue* object

**high**(*ff: bool = False*) → *youtube.Stream.Stream*

High will return the highest quality stream.

**Parameters** **ff** (*bool*) – this input is not for the user please ignore.

**Return type** *Stream*

**Returns** *Stream* object

**static init**(*data: List[youtube.Stream.Stream]*) → List[*youtube.Stream.Stream*]

Here we will return the List[Stream] after removing 'ignore\_itag'

**low**(*ff: bool = False*) → *youtube.Stream.Stream*

Low will return the lowest quality stream.

**Parameters** **ff** (*bool*) – this input is not for the user please ignore.

**Return type** *Stream*

**Returns** *Stream* object

**mid**(*ff: bool = False*) → *youtube.Stream.Stream*

Mid will return the normal quality stream.

**Parameters** **ff** (*bool*) – this input is not for the user please ignore.

**Return type** *Stream*

**Returns** *Stream* object

**resolution**(*quality*) → *youtube.Stream.FFMPEG*

Processing audio/video depend on resolution

**Return type** *FFMPEG*

**Returns** *FFMPEG* object

### 1.4.5 FFMPEG Class

**class** youtube.FFMPEG(*av: List[youtube.Stream.Stream]*)

Here we will download 2 stream audio/video, and copy them in one file with ffmpeg

**Parameters** *av* (*List[Stream]*) – pass the stream of audio and video in list,

**Return type** *None*

**Returns** Nothing will get return

**check\_ffmpeg()**

Checking if FFMPEG installed or not

**Return type** *bool*

**Returns** check status in (True/False)

**download**(*dire: str = "", name: str = "", status: bool = False, connection: int = 8, chunk: int = 5120*) → *str*

This will return bool value as video download convert status`

**Parameters**

- **name** (*str*) – (optional) Pass the name of file name.
- **dire** (*str*) – (optional) Pass the dir for output file with excluding filename.
- **status** (*bool*) – (optional) Pass if you want to enable process bar. **Default[False]**
- **connection** (*int*) – (optional) Pass number is connection to be create. **Default[8]**
- **chunk** (*int*) – (optional) Pass the chunk/buffer size for accepting packets. **Default[5120]**

**Return type** *str*

**Returns** name of the file

### 1.4.6 Cipher Class

**class** youtube.Cipher(*js: str*)

Here we will decrypt the signature of youtube videos

js is the raw string javascript of base.js

**Parameters** *js* (*str*) – pass javascript of base.js

**Return type** *None*

**Returns** nothing

**static Func\_check**(*s: str*) → *Callable[[List[str], int], List[str]]*

This will check compair the js function and return the python object

**Parameters** *s* (*str*) – pass the js fuction

**Return type** *object*

**Returns** will return python object

example:

```
>>> Func_check('TY:function(a){a.reverse()}')
<function Reverse at 0x7fd281937710>
```

**static Get\_funs**(*js: str*) → List[str]

Will return the method of encryption

**Parameters** **js** (*str*) – pass the raw js of base.js

**Return type** List[str]

**Returns** the decipher method/algorithm in list of strings

example:

```
>>> Get_funs(js)
... ['wv.TY(a,37)', 'wv.vd(a,10)', 'wv.TY(a,7)', 'wv.A3(a,1)', 'wv.vd(a,13)',
↪ 'wv.A3(a,2)', 'wv.vd(a,15)']
```

**Get\_funs\_objects**(*funs: List[str]*) → Dict[str, Callable[[List[str], int], List[str]]]

This function will return the dict with javascript function name and object

**Parameters** **funs** (*List[str]*) – pass the list of function

**Return type** Dict[str, Callable[[List[str],int], List[str]]]

**Returns** dict of function name and function object

example:

```
>>> Get_funs_objects(['TY:function(a){a.reverse()}', 'A3:function(a,b){a.
↪splice(0,b)}'],
    'vd:function(a,b){var c=a[0];a[0]=a[b%a.length];a[b%a.length]=c}'])
{'TY': <function Reverse at 0x7f0f9cfce7a0>, 'A3': <function Splice at
↪0x7f0f9cfd63b0>,
    'vd': <function Switch at 0x7f0f9cfd6710>}
```

**static Get\_js\_funcs**(*js: str, var: str*) → List[str]

will return the js raw function as string

**Parameters**

- **js** (*str*) – raw js html
- **var** (*str*) – js variable name

**Return type** List[str]

**Returns** the variable content/program from base.js

Example.

```
>>> Get_js_funcs(js, var)
... ['TY:function(a){a.reverse()}', 'A3:function(a,b){a.splice(0,b)',
    'vd:function(a,b){var c=a[0];a[0]=a[b%a.length];a[b%a.length]=c}']
```

**Get\_signature**(*sig: List[str]*) → str

This function will return decipher signature

**Parameters** **sig** (*list[str]*) – cipher signature provided by youtube

**Return type** str

**Returns** will return decipher signature

**static Parsec\_js\_methods**(*s: str*) → Tuple[str, int]

This the parse the methods

Example:

```
>>> Parsec_js_methods("wv.TY(a,37)")
('TY', 37)
```

### 1.4.7 Download Class

```
class youtube.Download(url: str, dire: str = "", name: str = "", status: bool = False, connection: int = 8, chunk: int = 5120)
```

This *Download* will download streams with multi-connections.

#### Parameters

- **url** (*str*) – Pass the download link
- **name** (*str*) – (optional) Pass the name of file name.
- **dire** (*str*) – (optional) Pass the dir for output file with excluding filename.
- **status** (*bool*) – (optional) Pass if you want to enable process bar. **Default[False]**
- **connection** (*int*) – (optional) Pass number is connection to be create. **Default[8]**
- **chunk** (*int*) – (optional) Pass the chunk/buffer size for accepting packets. **Default[5120]**

**Return type** *str*

**Returns** the file name

**start()** → *str*

Start will fire up the downloading

### 1.4.8 Search Class

```
class youtube.Search(query: str, country: str = "", proxy: str = "", process: bool = False)
```

This will search your query on youtube and will return videos.

#### Parameters

- **query** (*str*) – Pass the query for search.
- **country** (*str*) – (optional) Pass the ISO 2 country code.
- **proxy** (*str*) – (optional) pass the proxy url example:  

```
socks5://admin:admin@127.0.0.1:9050 http://admin:admin@127.0.0.1:8000 https://127.0.0.1:8080
```
- **process** (*bool*) – (optional) Process will trigger the function for making objects of *Video* with the list of playlist videos default (False), When (True) it will take some extra time depend on your CPU and Internet speed. Process will use 10 workers to create objects fasts.

**Return type** *None*

**Returns** *None*

**property first**

Will return the first search video.

**Return type** *Video*

**Returns** This will return *Video* of first search video.

**property get\_dict**

This will return the list of *Video* if available or will just return the dict of vid and name, Availability of *Video* objects is depend on your Search(query, process=False), process will process all videos to *Video* objects.

**property get\_object**

This will return the list of *Video* objects, even when you process=False.

**processing()**

Processing will process all videos to *Video* object

## 1.5 Command Line Input

### 1.5.1 Help

```
usage: youtube [-h] [-o OUTPUT] [-l] [-ff [FFMPEG] | -v [VIDEO] | -i ITAG | -s
              | -a [AUDIO] | -r [RESOLUTION] | --version]
              url

Definitions:
  Quality Types: ('HIGH', 'MID', 'LOW')
  Video resolution examples. ('144', '240', '360', '480', '720', '1080') more/less
↳ depend on your video

Note:
  Default values for '--ffmpeg', '--video', '--audio' is 'HIGH'
  Default values for '--resolution' is 720

positional arguments:
  url                  Enter video or playlist url

optional arguments:
  -h, --help          show this help message and exit
  -o OUTPUT, --output OUTPUT
                      Output location for downloading streams.
  -l, --logs          To enable extra logs.
  -ff [FFMPEG], --ffmpeg [FFMPEG]
                      FFMPEG downloads audio/video both then copy in one file, Pass
↳ the quality type.
  -v [VIDEO], --video [VIDEO]
                      Downlaod video (only progressive), Pass quality.
  -i ITAG, --itag ITAG
                      Download stream with itag, Pass itag id.
  -s, --streams       This argument will list the all available streams.
  -a [AUDIO], --audio [AUDIO]
                      Download audio, Pass quality.
  -r [RESOLUTION], --resolution [RESOLUTION]
                      Download stream with quality type, Pass video resolution.
  --version           Check the current version of youtube.py
```

### 1.5.2 Example

`-r` and `--resolution` have default resolution 720

```
>>> youtube https://www.youtube.com/watch?v=3AtDnEC4zak -r
Downloading : video : mp4: 720
Process: [] 100.0% Complete 0.0 Kb/s /s
Downloading : audio : webm: 76035
Process: [] 100.0% Complete 0.0 Kb/s /s
Charlie_Puth_-_We_Don't_Talk_Anymore_(feat.mp4 Downloaded
```

`-r` and `--resolution` value can be overwritten

```
>>> youtube https://www.youtube.com/watch?v=3AtDnEC4zak -r 1080
Downloading : video : mp4: 1080
Process: [] 100.0% Complete 0.0 Kb/s /s
Downloading : audio : webm: 76035
Process: [] 100.0% Complete 0.0 Kb/s /s
Charlie_Puth_-_We_Don't_Talk_Anymore_(feat.mp4 Downloaded
```

---

**Note:** Downloading with `--resolution`, `--ffmpeg` or with `--video` is different, resolution and ffmpeg uses FFmpeg to download files and copy them in one but `--video` download a single file made/generate by youtube which has audio/video both. but `--video` cant always download files in better quality so preffer `--resolution` or `--ffmpeg` use `--video` only when your system do not support *FFMPEG*

---

Example with `--video`

```
>>> youtube https://www.youtube.com/watch?v=3AtDnEC4zak --video high
Process: [] 100.0% Complete 0.0 Kb/s s
44Charlie_Puth_-_We_Don't_Talk_Anymore_(feat._Selena_Gomez)_[Official_Video].mp4_
↳Downloaded
```

Example with `--audio`

```
>>> youtube https://www.youtube.com/watch?v=3AtDnEC4zak --audio high
Process: [] 100.0% Complete 0.0 Kb/s /s
71Charlie_Puth_-_We_Don't_Talk_Anymore_(feat._Selena_Gomez)_[Official_Video].webm_
↳Downloaded
```

## 1.6 YOU CAN HELP

You just simple fork the project from github and fix the bugs and push with full explanation of bug and fix.

We do love when you invest your time and contribute us for making us more better/stable.

Your each bug report on github or contribute is much appreciated



## 1.7 PROJECT DETAILS

**Authors** Mayank Gupta

**Version** 2.0.0

**License** MIT License

**Copyright** Copyright (c) 2020 YouTube.py (Mayank Gupta)

**Website** <https://youtube-python.mayankfawkes.xyz>

**Github** <https://github.com/youtube-py/youtube.py>

**Github Profile** <https://github.com/youtube-py>



## PYTHON MODULE INDEX

y

youtube, 10



## A

all (*youtube.Queue* property), 14  
 author (*youtube.Video* property), 10

## C

channel (*youtube.Video* property), 11  
 check() (*youtube.Stream* method), 12  
 check\_ffmpeg() (*youtube.FFMPEG* method), 16  
 Cipher (*class in youtube*), 16

## D

description (*youtube.Video* property), 11  
 Download (*class in youtube*), 18  
 download() (*youtube.FFMPEG* method), 16  
 download() (*youtube.Stream* method), 13  
 downloadall() (*youtube.PlayList* method), 12

## F

ffhigh (*youtube.Queue* property), 14  
 fflow (*youtube.Queue* property), 14  
 ffmid (*youtube.Queue* property), 14  
 FFMPEG (*class in youtube*), 16  
 ffresolution() (*youtube.Queue* method), 14  
 first (*youtube.Search* property), 18  
 format (*youtube.Stream* property), 13  
 format() (*youtube.Queue* method), 14  
 Func\_check() (*youtube.Cipher* static method), 16

## G

get\_audios (*youtube.Queue* property), 14  
 get\_both (*youtube.Queue* property), 14  
 get\_dict (*youtube.PlayList* property), 12  
 get\_dict (*youtube.Search* property), 18  
 get\_dict (*youtube.Video* property), 11  
 Get\_funcs() (*youtube.Cipher* static method), 16  
 Get\_funcs\_objects() (*youtube.Cipher* method), 17  
 get\_itag() (*youtube.Queue* method), 15  
 Get\_js\_funcs() (*youtube.Cipher* static method), 17  
 get\_object (*youtube.PlayList* property), 12  
 get\_object (*youtube.Search* property), 19  
 Get\_signature() (*youtube.Cipher* method), 17

get\_videos (*youtube.Queue* property), 15

## H

high() (*youtube.Queue* method), 15

## I

init() (*youtube.Queue* static method), 15  
 is\_audio (*youtube.Stream* property), 13  
 is\_both (*youtube.Stream* property), 13  
 is\_video (*youtube.Stream* property), 13  
 itag (*youtube.Stream* property), 13

## K

keywords (*youtube.Video* property), 11

## L

length (*youtube.Video* property), 11  
 low() (*youtube.Queue* method), 15

## M

mid() (*youtube.Queue* method), 15  
 module  
     youtube, 10

## P

Parsec\_js\_methods() (*youtube.Cipher* static method),  
     17  
 PlayList (*class in youtube*), 12  
 processing() (*youtube.PlayList* method), 12  
 processing() (*youtube.Search* method), 19

## Q

Queue (*class in youtube*), 14

## R

resolution() (*youtube.Queue* method), 15

## S

Search (*class in youtube*), 18  
 start() (*youtube.Download* method), 18  
 Stream (*class in youtube*), 12

streams (*youtube.Video* property), 11

## T

thumbnail (*youtube.Video* property), 11

title (*youtube.Video* property), 11

## U

url (*youtube.Stream* property), 13

## V

Video (*class in youtube*), 10

videoId (*youtube.Video* property), 11

views (*youtube.Video* property), 11

## Y

youtube

    module, 10