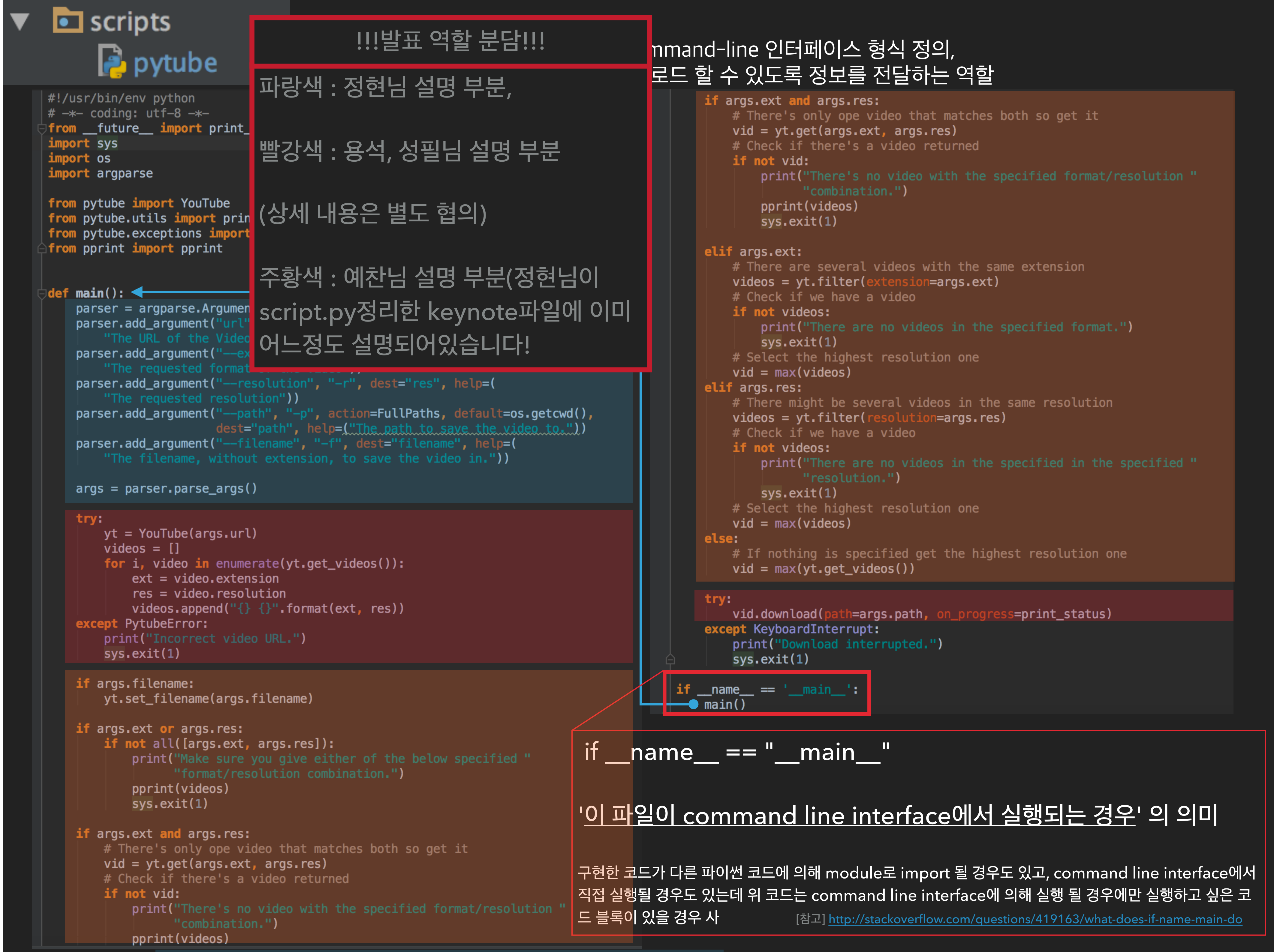
!!!발표 역할 분담!!!

파랑색 : 정현님 설명 부분,

빨강색 : 용석, 성필님 설명 부분

(상세 내용은 별도 협의)

주황색 : 예찬님 설명 부분(정현님이 script.py정리한 keynote파일에 이미 어느정도 설명되어있습니다!



```
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_
import sys
import os
import argparse

from pytube import YouTube
from pytube.utils import prin
from pytube.exceptions import
from pprint import pprint


def main():
    parser = argparse.Argumen
    parser.add_argument("url"
        "The URL of the Video
    parser.add_argument("--ex
        "The requested format
    parser.add_argument("--resolution", "-r", dest="res", help=(
        "The requested resolution"))
    parser.add_argument("--path", "-p", action=FullPaths, default=os.getcwd(),
                        dest="path", help=("The path to save the video to."))
    parser.add_argument("--filename", "-f", dest="filename", help=(
        "The filename, without extension, to save the video in."))

    args = parser.parse_args()

    try:
        yt = YouTube(args.url)
        videos = []
        for i, video in enumerate(yt.get_videos()):
            ext = video.extension
            res = video.resolution
            videos.append("{} {}".format(ext, res))
    except PytubeError:
        print("Incorrect video URL.")
        sys.exit(1)

    if args.filename:
        yt.set_filename(args.filename)

    if args.ext or args.res:
        if not all([args.ext, args.res]):
            print("Make sure you give either of the below specified "
                  "format/resolution combination.")
            pprint(videos)
            sys.exit(1)

    if args.ext and args.res:
        # There's only ope video that matches both so get it
        vid = yt.get(args.ext, args.res)
        # Check if there's a video returned
        if not vid:
            print("There's no video with the specified format/resolution "
                  "combination.")
            pprint(videos)
```

mmand-line 인터페이스 형식 정의,
로드 할 수 있도록 정보를 전달하는 역할

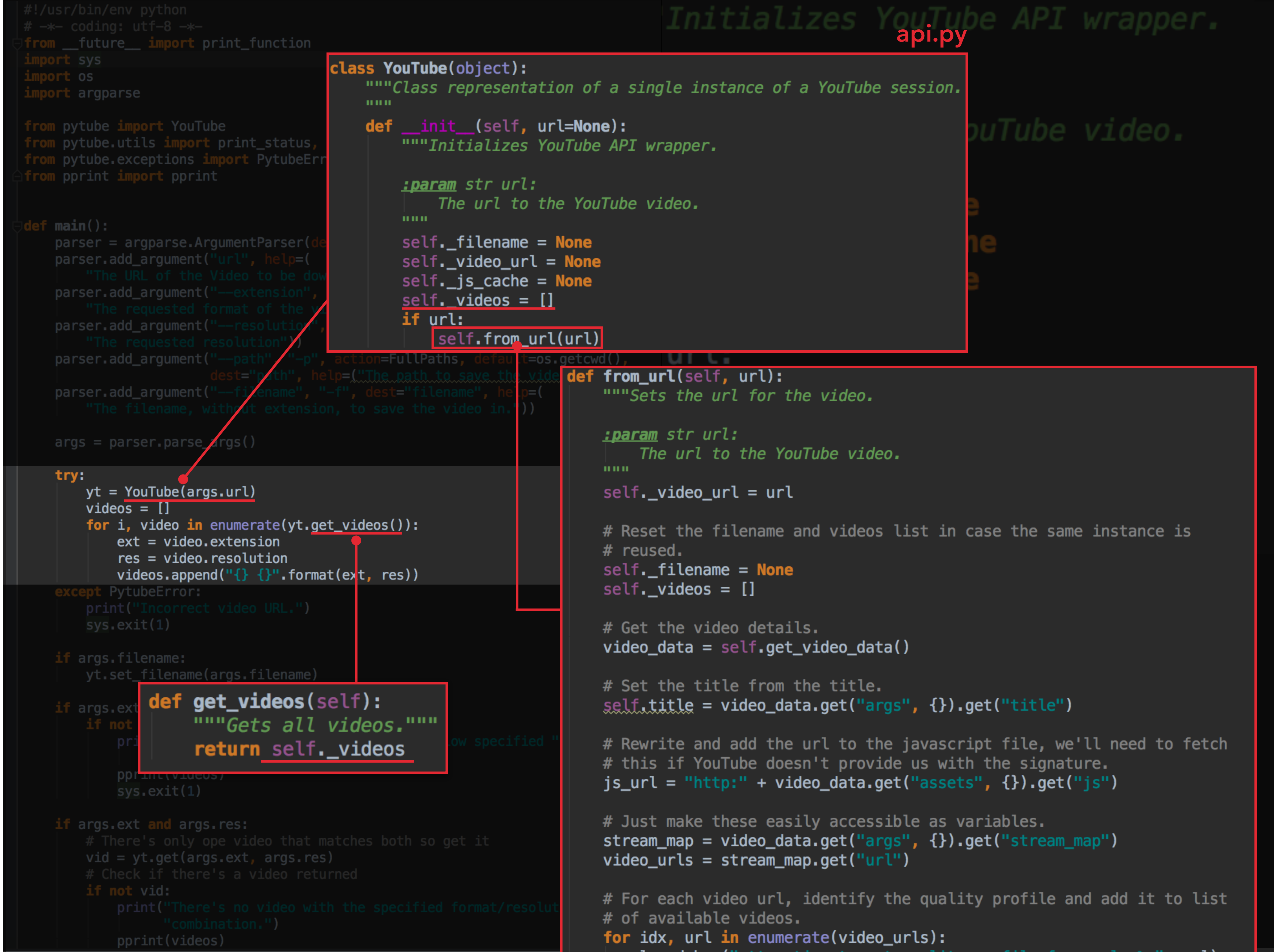
```
    if args.ext and args.res:
        # There's only ope video that matches both so get it
        vid = yt.get(args.ext, args.res)
        # Check if there's a video returned
        if not vid:
            print("There's no video with the specified format/resolution "
                  "combination.")
            pprint(videos)
            sys.exit(1)

    elif args.ext:
        # There are several videos with the same extension
        videos = yt.filter(extension=args.ext)
        # Check if we have a video
        if not videos:
            print("There are no videos in the specified format.")
            sys.exit(1)
        # Select the highest resolution one
        vid = max(videos)
    elif args.res:
        # There might be several videos in the same resolution
        videos = yt.filter(resolution=args.res)
        # Check if we have a video
        if not videos:
            print("There are no videos in the specified in the specified "
                  "resolution.")
            sys.exit(1)
        # Select the highest resolution one
        vid = max(videos)
    else:
        # If nothing is specified get the highest resolution one
        vid = max(yt.get_videos())

    try:
        vid.download(path=args.path, on_progress=print_status)
    except KeyboardInterrupt:
        print("Download interrupted.")
        sys.exit(1)

if __name__ == '__main__':
    main()
```

if __name__ == "__main__"

'이 파일이 command line interface에서 실행되는 경우' 의 의미

구현한 코드가 다른 파이썬 코드에 의해 module로 import 될 경우도 있고, command line interface에서 직접 실행될 경우도 있는데 위 코드는 command line interface에 의해 실행 될 경우에만 실행하고 싶은 코드 블록이 있을 경우 사

[참고] http://stackoverflow.com/questions/419163/what-does-if-name-main-do

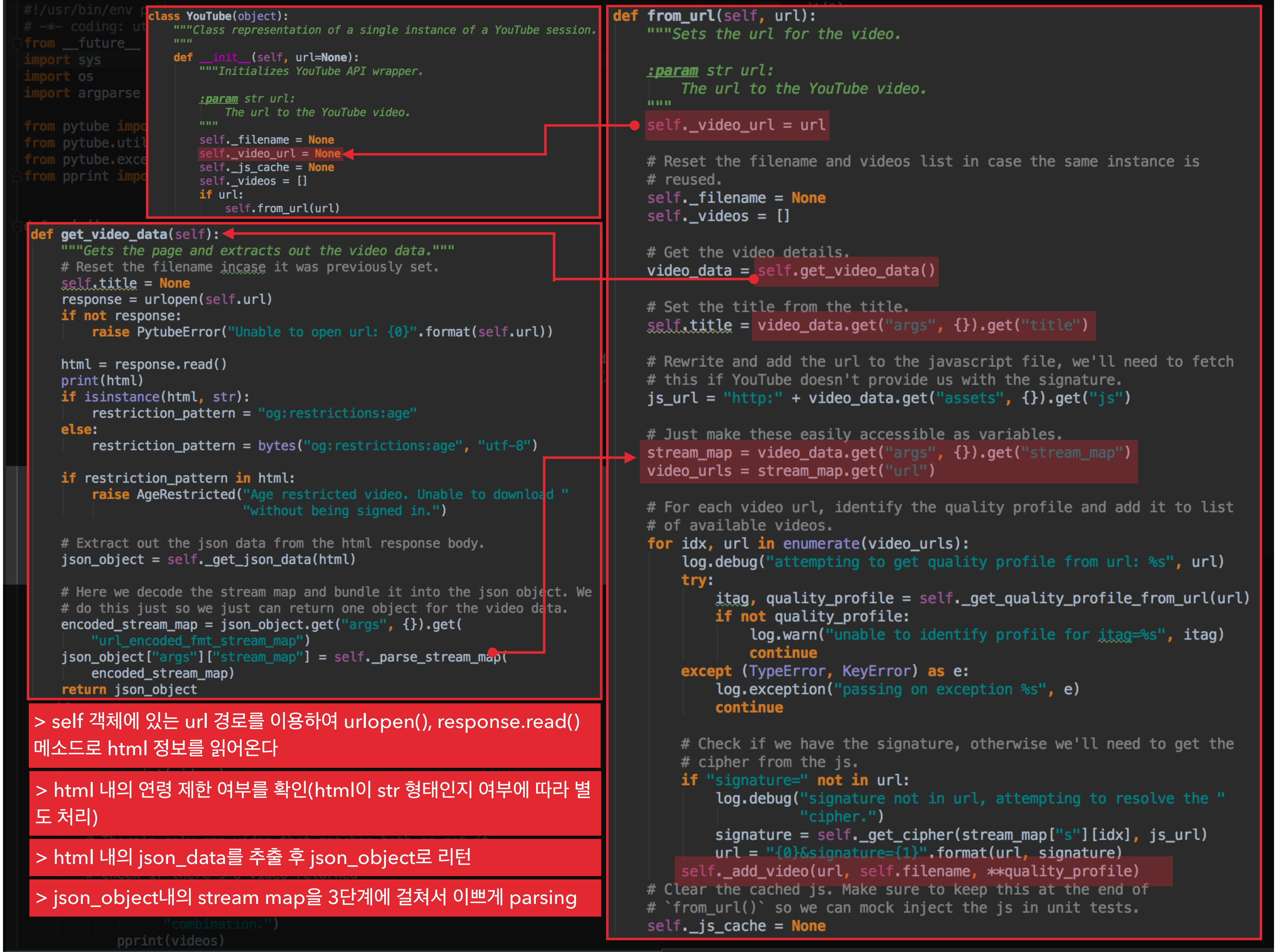


```python
#!/usr/bin/env python
# -*- coding: utf-8 -*-
from __future__ import print_function
import sys
import os
import argparse

from pytube import YouTube
from pytube.utils import print_status,
from pytube.exceptions import PytubeErr
from pprint import pprint


def main():
    parser = argparse.ArgumentParser(de
    parser.add_argument("url", help=(
        "The URL of the Video to be dow
    parser.add_argument("--extension",
        "The requested format of the v
    parser.add_argument("--resolution",
        "The requested resolution"))
    parser.add_argument("--path", "-p", action=FullPaths, default=os.getcwd(),
                        dest="path", help=("The path to save the vide
    parser.add_argument("--filename", "-f", dest="filename", help=(
        "The filename, without extension, to save the video in."))

    args = parser.parse_args()

    try:
        yt = YouTube(args.url)
        videos = []
        for i, video in enumerate(yt.get_videos()):
            ext = video.extension
            res = video.resolution
            videos.append("{} {}".format(ext, res))
    except PytubeError:
        print("Incorrect video URL.")
        sys.exit(1)

    if args.filename:
        yt.set_filename(args.filename)

    if args.ext
        if not
            pri                                    ow specified "

            pprint(videos)
            sys.exit(1)

    if args.ext and args.res:
        # There's only ope video that matches both so get it
        vid = yt.get(args.ext, args.res)
        # Check if there's a video returned
        if not vid:
            print("There's no video with the specified format/resolut
                  "combination.")
            pprint(videos)
```

```python
class YouTube(object):
    """Class representation of a single instance of a YouTube session.
    """
    def __init__(self, url=None):
        """Initializes YouTube API wrapper.

        :param str url:
            The url to the YouTube video.
        """
        self._filename = None
        self._video_url = None
        self._js_cache = None
        self._videos = []
        if url:
            self.from_url(url)
```

```python
def get_videos(self):
    """Gets all videos."""
    return self._videos
```

```python
def from_url(self, url):
    """Sets the url for the video.

    :param str url:
        The url to the YouTube video.
    """
    self._video_url = url

    # Reset the filename and videos list in case the same instance is
    # reused.
    self._filename = None
    self._videos = []

    # Get the video details.
    video_data = self.get_video_data()

    # Set the title from the title.
    self.title = video_data.get("args", {}).get("title")

    # Rewrite and add the url to the javascript file, we'll need to fetch
    # this if YouTube doesn't provide us with the signature.
    js_url = "http:" + video_data.get("assets", {}).get("js")

    # Just make these easily accessible as variables.
    stream_map = video_data.get("args", {}).get("stream_map")
    video_urls = stream_map.get("url")

    # For each video url, identify the quality profile and add it to list
    # of available videos.
    for idx, url in enumerate(video_urls):
```



```python
#!/usr/bin/env p
# -*- coding: ut
from __future__
import sys
import os
import argparse

from pytube impo
from pytube.util
from pytube.exce
from pprint impo
```

```python
class YouTube(object):
    """Class representation of a single instance of a YouTube session.
    """
    def __init__(self, url=None):
        """Initializes YouTube API wrapper.

        :param str url:
            The url to the YouTube video.
        """
        self._filename = None
        self._video_url = None
        self._js_cache = None
        self._videos = []
        if url:
            self.from_url(url)
```

```python
def get_video_data(self):
    """Gets the page and extracts out the video data."""
    # Reset the filename incase it was previously set.
    self.title = None
    response = urlopen(self.url)
    if not response:
        raise PytubeError("Unable to open url: {0}".format(self.url))

    html = response.read()
    print(html)
    if isinstance(html, str):
        restriction_pattern = "og:restrictions:age"
    else:
        restriction_pattern = bytes("og:restrictions:age", "utf-8")

    if restriction_pattern in html:
        raise AgeRestricted("Age restricted video. Unable to download "
                            "without being signed in.")

    # Extract out the json data from the html response body.
    json_object = self._get_json_data(html)

    # Here we decode the stream map and bundle it into the json object. We
    # do this just so we just can return one object for the video data.
    encoded_stream_map = json_object.get("args", {}).get(
        "url_encoded_fmt_stream_map")
    json_object["args"]["stream_map"] = self._parse_stream_map(
        encoded_stream_map)
    return json_object
```

```python
def from_url(self, url):
    """Sets the url for the video.

    :param str url:
        The url to the YouTube video.
    """
    self._video_url = url

    # Reset the filename and videos list in case the same instance is
    # reused.
    self._filename = None
    self._videos = []

    # Get the video details.
    video_data = self.get_video_data()

    # Set the title from the title.
    self.title = video_data.get("args", {}).get("title")

    # Rewrite and add the url to the javascript file, we'll need to fetch
    # this if YouTube doesn't provide us with the signature.
    js_url = "http:" + video_data.get("assets", {}).get("js")

    # Just make these easily accessible as variables.
    stream_map = video_data.get("args", {}).get("stream_map")
    video_urls = stream_map.get("url")

    # For each video url, identify the quality profile and add it to list
    # of available videos.
    for idx, url in enumerate(video_urls):
        log.debug("attempting to get quality profile from url: %s", url)
        try:
            itag, quality_profile = self._get_quality_profile_from_url(url)
            if not quality_profile:
                log.warn("unable to identify profile for itag=%s", itag)
                continue
        except (TypeError, KeyError) as e:
            log.exception("passing on exception %s", e)
            continue

        # Check if we have the signature, otherwise we'll need to get the
        # cipher from the js.
        if "signature=" not in url:
            log.debug("signature not in url, attempting to resolve the "
                      "cipher.")
            signature = self._get_cipher(stream_map["s"][idx], js_url)
            url = "{0}&signature={1}".format(url, signature)
        self._add_video(url, self.filename, **quality_profile)
    # Clear the cached js. Make sure to keep this at the end of
    # `from_url()` so we can mock inject the js in unit tests.
    self._js_cache = None
```

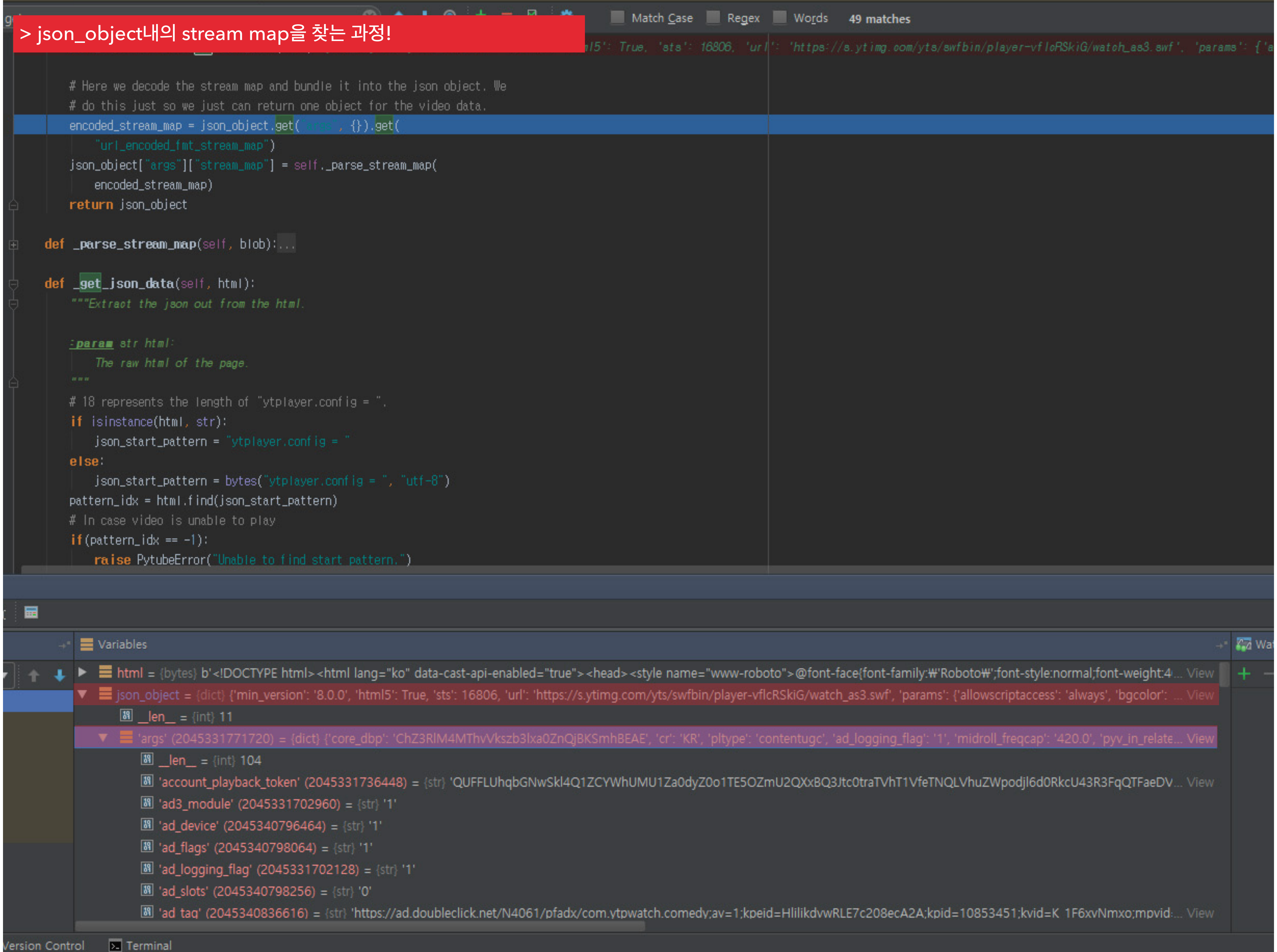
```python
                "combination.")
        pprint(videos)
```

> self 객체에 있는 url 경로를 이용하여 urlopen(), response.read() 메소드로 html 정보를 읽어온다

> html 내의 연령 제한 여부를 확인(html이 str 형태인지 여부에 따라 별도 처리)

> html 내의 json_data를 추출 후 json_object로 리턴

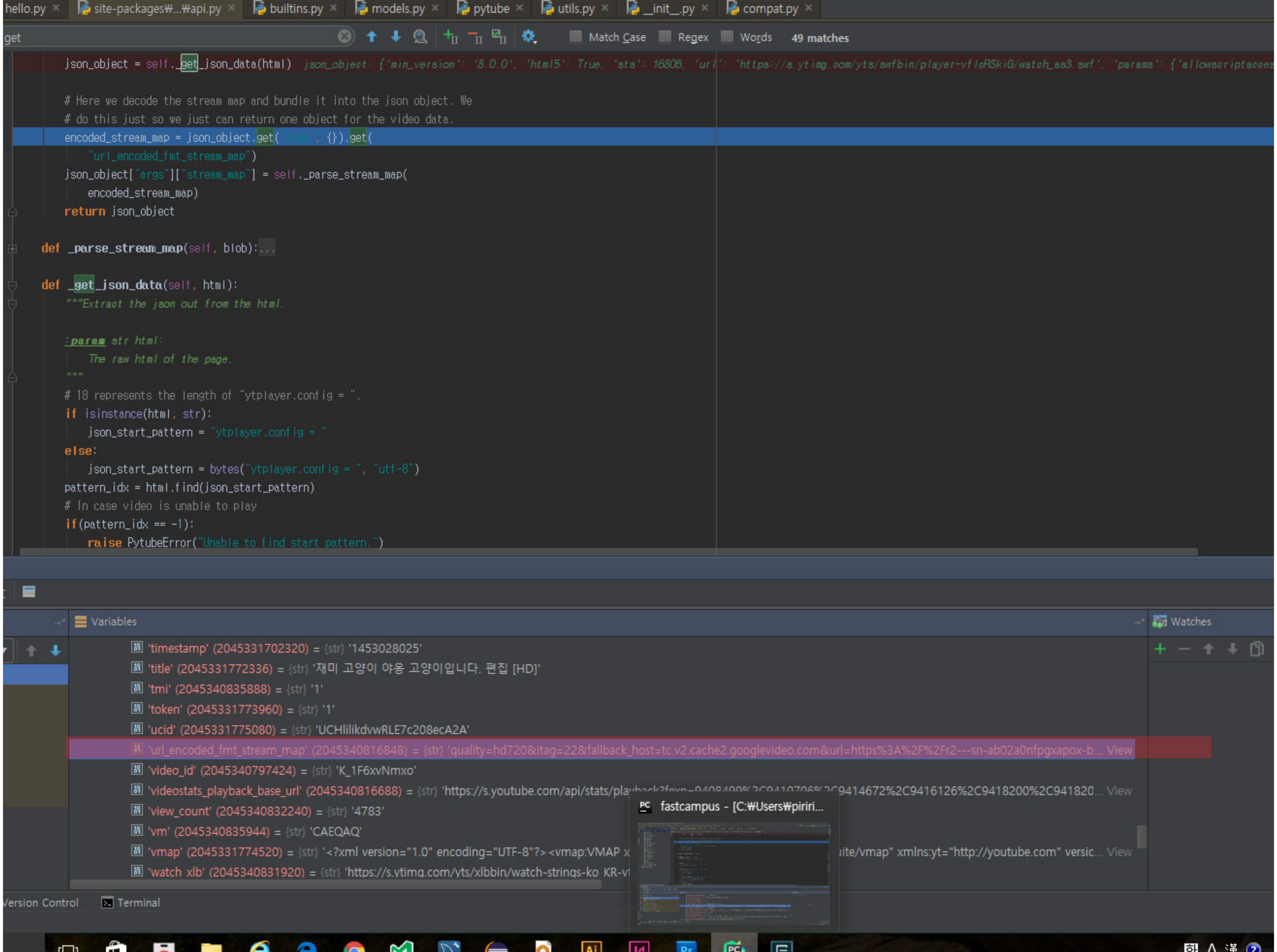
> json_object내의 stream map을 3단계에 걸쳐서 이쁘게 parsing



> json_object내의 stream map을 찾는 과정!

```
l5': True, 'sts': 16806, 'url': 'https://s.ytimg.com/yts/swfbin/player-vflcRSkiG/watch_as3.swf', 'params': {'a

        # Here we decode the stream map and bundle it into the json object. We
        # do this just so we just can return one object for the video data.
        encoded_stream_map = json_object.get("args", {}).get(
            "url_encoded_fmt_stream_map")
        json_object["args"]["stream_map"] = self._parse_stream_map(
            encoded_stream_map)
        return json_object

    def _parse_stream_map(self, blob):...

    def _get_json_data(self, html):
        """Extract the json out from the html.

        :param str html:
            The raw html of the page.
        """
        # 18 represents the length of "ytplayer.config = ".
        if isinstance(html, str):
            json_start_pattern = "ytplayer.config = "
        else:
            json_start_pattern = bytes("ytplayer.config = ", "utf-8")
        pattern_idx = html.find(json_start_pattern)
        # In case video is unable to play
        if(pattern_idx == -1):
            raise PytubeError("Unable to find start pattern.")
```

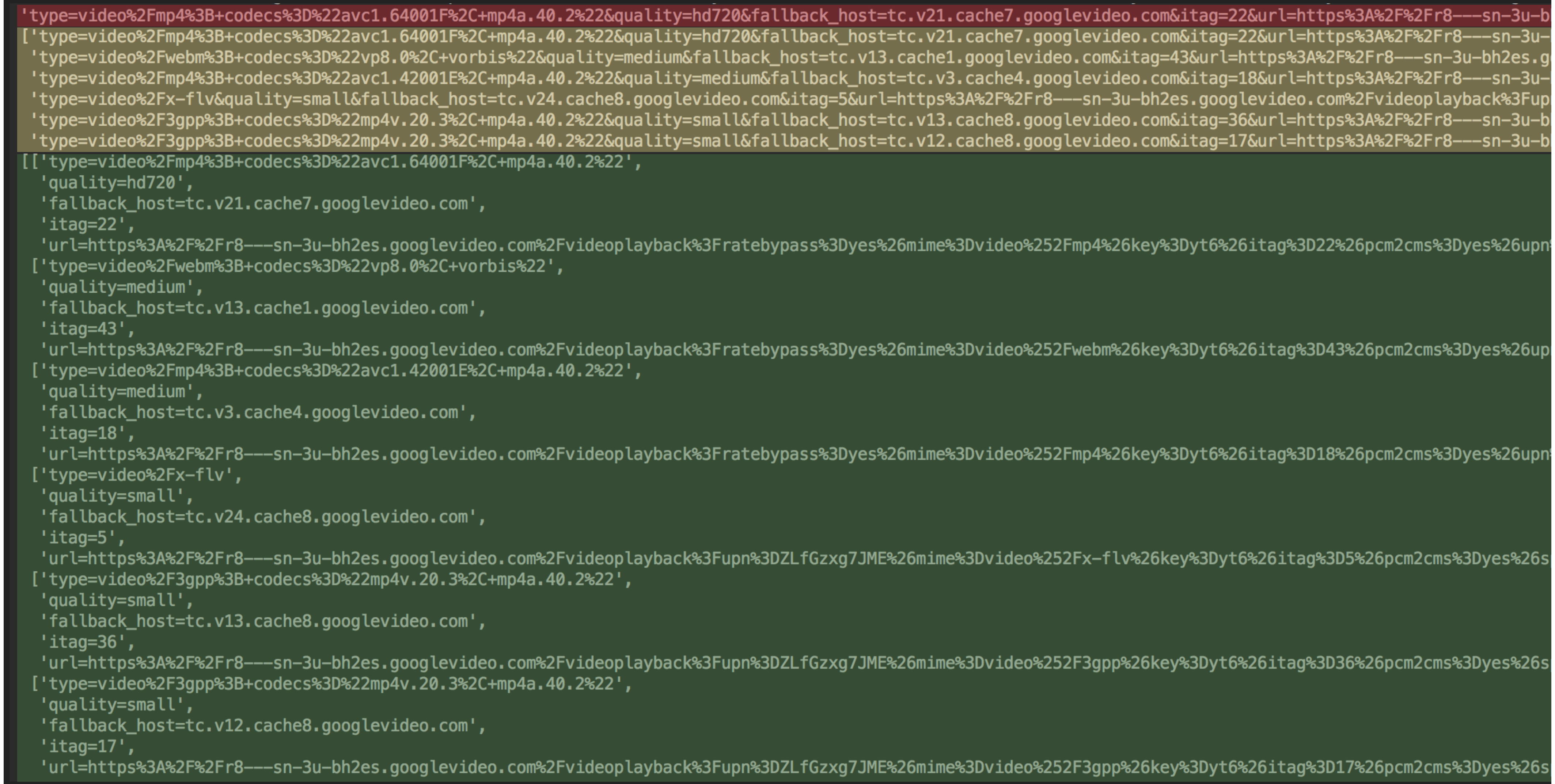
```
▶ html = {bytes} b'<!DOCTYPE html><html lang="ko" data-cast-api-enabled="true"><head><style name="www-roboto">@font-face{font-family:₩'Roboto₩';font-style:normal;font-weight:4... View
▼ json_object = {dict} {'min_version': '8.0.0', 'html5': True, 'sts': 16806, 'url': 'https://s.ytimg.com/yts/swfbin/player-vflcRSkiG/watch_as3.swf', 'params': {'allowscriptaccess': 'always', 'bgcolor': ... View
    __len__ = {int} 11
  ▼ 'args' (2045331771720) = {dict} {'core_dbp': 'ChZ3RlM4MThvVkszb3lxa0ZnQjBKSmhBEAE', 'cr': 'KR', 'pltype': 'contentugc', 'ad_logging_flag': '1', 'midroll_freqcap': '420.0', 'pyv_in_relate... View
      __len__ = {int} 104
      'account_playback_token' (2045331736448) = {str} 'QUFFLUhqbGNwSkl4Q1ZCYWhUMU1Za0dyZ0o1TE5OZmU2QXxBQ3Jtc0traTVhT1VfeTNQLVhuZWpodjl6d0RkcU43R3FqQTFaeDV... View
      'ad3_module' (2045331702960) = {str} '1'
      'ad_device' (2045340796464) = {str} '1'
      'ad_flags' (2045340798064) = {str} '1'
      'ad_logging_flag' (2045331702128) = {str} '1'
      'ad_slots' (2045340798256) = {str} '0'
      'ad_tag' (2045340836616) = {str} 'https://ad.doubleclick.net/N4061/pfadx/com.ytpwatch.comedy;av=1;kpeid=HlilikdvwRLE7c208ecA2A;kpid=10853451;kvid=K_1F6xvNmxo;mpvid:... View
```



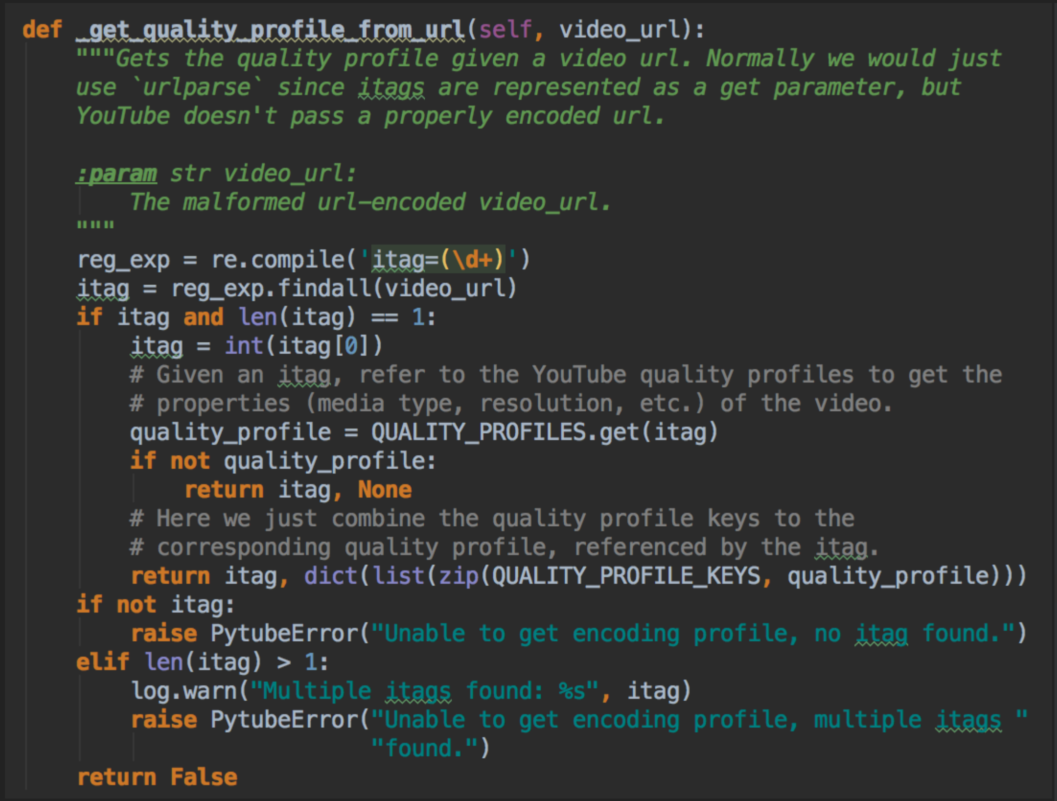
```
        json_object = self._get_json_data(html)  json_object: {'min_version': '8.0.0', 'html5': True, 'sts': 16806, 'url': 'https://s.ytimg.com/yts/swfbin/player-vfloRSkiG/watch_as3.swf', 'params': {'allowscriptacces

        # Here we decode the stream map and bundle it into the json object. We
        # do this just so we just can return one object for the video data.
        encoded_stream_map = json_object.get("args", {}).get(
            "url_encoded_fmt_stream_map")
        json_object["args"]["stream_map"] = self._parse_stream_map(
            encoded_stream_map)
        return json_object

    def _parse_stream_map(self, blob):...

    def _get_json_data(self, html):
        """Extract the json out from the html.

        :param str html:
            The raw html of the page.
        """
        # 18 represents the length of "ytplayer.config = ".
        if isinstance(html, str):
            json_start_pattern = "ytplayer.config = "
        else:
            json_start_pattern = bytes("ytplayer.config = ", "utf-8")
        pattern_idx = html.find(json_start_pattern)
        # In case video is unable to play
        if(pattern_idx == -1):
            raise PytubeError("Unable to find start pattern.")
```

```
'timestamp' (2045331702320) = {str} '1453028025'
'title' (2045331772336) = {str} '재미 고양이 야옹 고양이입니다. 편집 [HD]'
'tmi' (2045340835888) = {str} '1'
'token' (2045331773960) = {str} '1'
'ucid' (2045331775080) = {str} 'UCHlilikdvwRLE7c208ecA2A'
'url_encoded_fmt_stream_map' (2045340816848) = {str} 'quality=hd720&itag=22&fallback_host=tc.v2.cache2.googlevideo.com&url=https%3A%2F%2Fr2---sn-ab02a0nfpgxapox-b... View
'video_id' (2045340797424) = {str} 'K_1F6xvNmxo'
'videostats_playback_base_url' (2045340816688) = {str} 'https://s.youtube.com/api/stats/pla                9414672%2C9416126%2C9418200%2C941820... View
'view_count' (2045340832240) = {str} '4783'
'vm' (2045340835944) = {str} 'CAEQAQ'
'vmap' (2045331774520) = {str} '<?xml version="1.0" encoding="UTF-8"?><vmap:VMAP x                ite/vmap" xmlns:yt="http://youtube.com" versic... View
'watch_xlb' (2045340831920) = {str} 'https://s.ytimg.com/yts/xlbbin/watch-strings-ko_KR-v
```

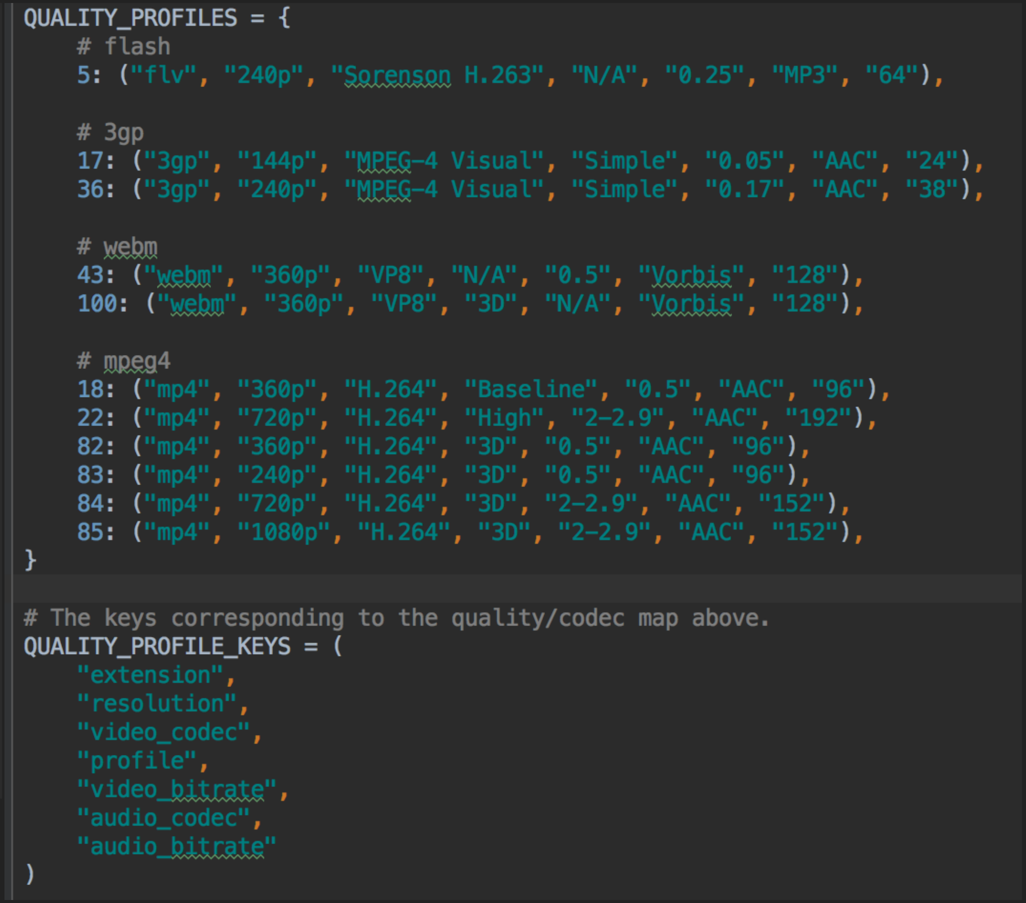
> 힘들게 찾아온 stream map을 3단계에 걸쳐서 이쁘게 parsing!



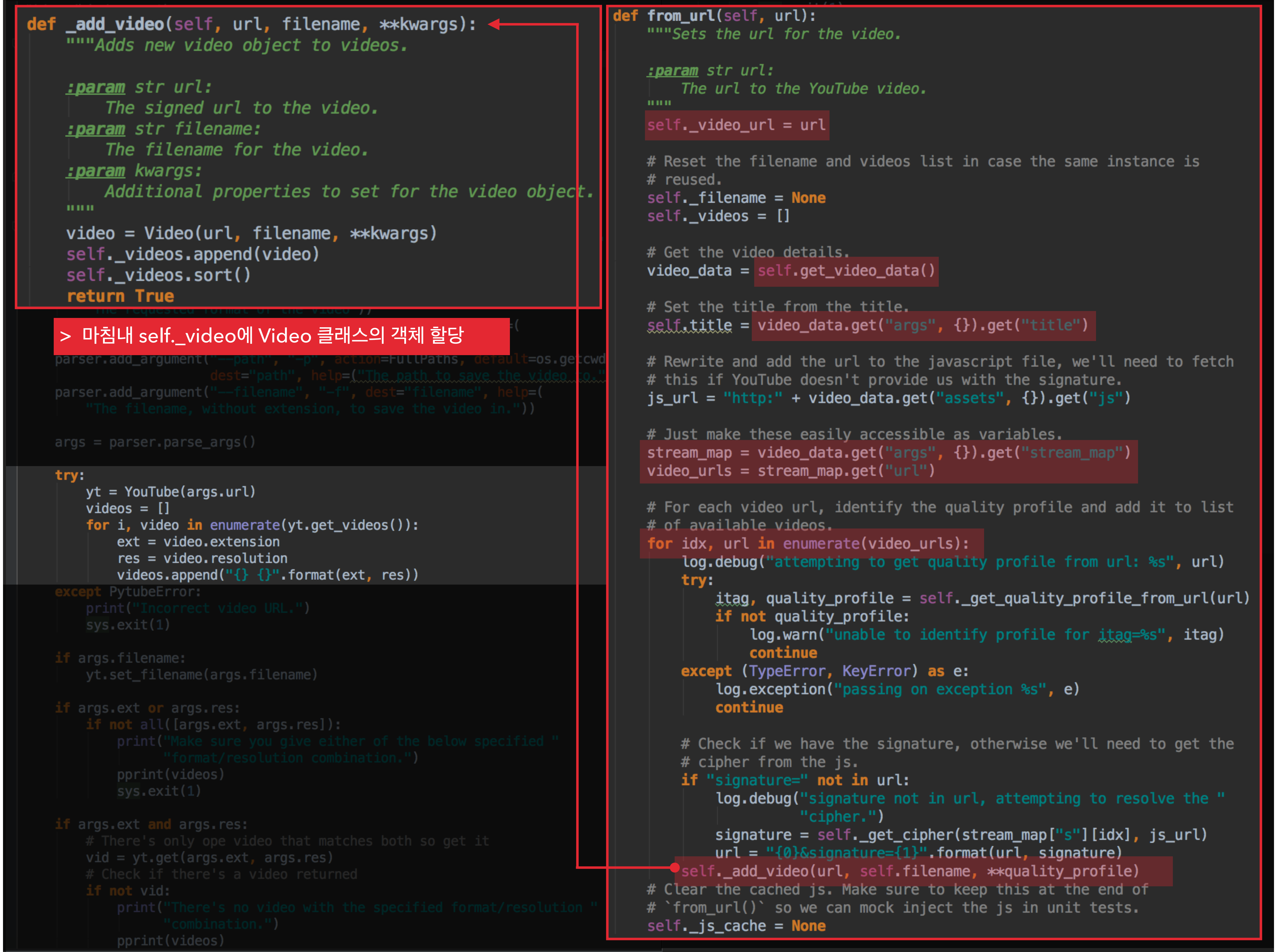
```
'type=video%2Fmp4%3B+codecs%3D%22avc1.64001F%2C+mp4a.40.2%22&quality=hd720&fallback_host=tc.v21.cache7.googlevideo.com&itag=22&url=https%3A%2F%2Fr8---sn-3u-b
['type=video%2Fmp4%3B+codecs%3D%22avc1.64001F%2C+mp4a.40.2%22&quality=hd720&fallback_host=tc.v21.cache7.googlevideo.com&itag=22&url=https%3A%2F%2Fr8---sn-3u-
 'type=video%2Fwebm%3B+codecs%3D%22vp8.0%2C+vorbis%22&quality=medium&fallback_host=tc.v13.cache1.googlevideo.com&itag=43&url=https%3A%2F%2Fr8---sn-3u-bh2es.g
 'type=video%2Fmp4%3B+codecs%3D%22avc1.42001E%2C+mp4a.40.2%22&quality=medium&fallback_host=tc.v3.cache4.googlevideo.com&itag=18&url=https%3A%2F%2Fr8---sn-3u-
 'type=video%2Fx-flv&quality=small&fallback_host=tc.v24.cache8.googlevideo.com&itag=5&url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fup
 'type=video%2F3gpp%3B+codecs%3D%22mp4v.20.3%2C+mp4a.40.2%22&quality=small&fallback_host=tc.v13.cache8.googlevideo.com&itag=36&url=https%3A%2F%2Fr8---sn-3u-b
 'type=video%2F3gpp%3B+codecs%3D%22mp4v.20.3%2C+mp4a.40.2%22&quality=small&fallback_host=tc.v12.cache8.googlevideo.com&itag=17&url=https%3A%2F%2Fr8---sn-3u-b
[['type=video%2Fmp4%3B+codecs%3D%22avc1.64001F%2C+mp4a.40.2%22',
  'quality=hd720',
  'fallback_host=tc.v21.cache7.googlevideo.com',
  'itag=22',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fratebypass%3Dyes%26mime%3Dvideo%252Fmp4%26key%3Dyt6%26itag%3D22%26pcm2cms%3Dyes%26upn
 ['type=video%2Fwebm%3B+codecs%3D%22vp8.0%2C+vorbis%22',
  'quality=medium',
  'fallback_host=tc.v13.cache1.googlevideo.com',
  'itag=43',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fratebypass%3Dyes%26mime%3Dvideo%252Fwebm%26key%3Dyt6%26itag%3D43%26pcm2cms%3Dyes%26up
 ['type=video%2Fmp4%3B+codecs%3D%22avc1.42001E%2C+mp4a.40.2%22',
  'quality=medium',
  'fallback_host=tc.v3.cache4.googlevideo.com',
  'itag=18',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fratebypass%3Dyes%26mime%3Dvideo%252Fmp4%26key%3Dyt6%26itag%3D18%26pcm2cms%3Dyes%26upn
 ['type=video%2Fx-flv',
  'quality=small',
  'fallback_host=tc.v24.cache8.googlevideo.com',
  'itag=5',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fupn%3DZLfGzxg7JME%26mime%3Dvideo%252Fx-flv%26key%3Dyt6%26itag%3D5%26pcm2cms%3Dyes%26s
 ['type=video%2F3gpp%3B+codecs%3D%22mp4v.20.3%2C+mp4a.40.2%22',
  'quality=small',
  'fallback_host=tc.v13.cache8.googlevideo.com',
  'itag=36',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fupn%3DZLfGzxg7JME%26mime%3Dvideo%252F3gpp%26key%3Dyt6%26itag%3D36%26pcm2cms%3Dyes%26s
 ['type=video%2F3gpp%3B+codecs%3D%22mp4v.20.3%2C+mp4a.40.2%22',
  'quality=small',
  'fallback_host=tc.v12.cache8.googlevideo.com',
  'itag=17',
  'url=https%3A%2F%2Fr8---sn-3u-bh2es.googlevideo.com%2Fvideoplayback%3Fupn%3DZLfGzxg7JME%26mime%3Dvideo%252F3gpp%26key%3Dyt6%26itag%3D17%26pcm2cms%3Dyes%26s
```



```python
def _get_quality_profile_from_url(self, video_url):
    """Gets the quality profile given a video url. Normally we would just
    use `urlparse` since itags are represented as a get parameter, but
    YouTube doesn't pass a properly encoded url.

    :param str video_url:
        The malformed url-encoded video_url.
    """
    reg_exp = re.compile('itag=(\d+)')
    itag = reg_exp.findall(video_url)
    if itag and len(itag) == 1:
        itag = int(itag[0])
        # Given an itag, refer to the YouTube quality profiles to get the
        # properties (media type, resolution, etc.) of the video.
        quality_profile = QUALITY_PROFILES.get(itag)
        if not quality_profile:
            return itag, None
        # Here we just combine the quality profile keys to the
        # corresponding quality profile, referenced by the itag.
        return itag, dict(list(zip(QUALITY_PROFILE_KEYS, quality_profile)))
    if not itag:
        raise PytubeError("Unable to get encoding profile, no itag found.")
    elif len(itag) > 1:
        log.warn("Multiple itags found: %s", itag)
        raise PytubeError("Unable to get encoding profile, multiple itags "
                          "found.")
    return False
```



```python
QUALITY_PROFILES = {
    # flash
    5: ("flv", "240p", "Sorenson H.263", "N/A", "0.25", "MP3", "64"),

    # 3gp
    17: ("3gp", "144p", "MPEG-4 Visual", "Simple", "0.05", "AAC", "24"),
    36: ("3gp", "240p", "MPEG-4 Visual", "Simple", "0.17", "AAC", "38"),

    # webm
    43: ("webm", "360p", "VP8", "N/A", "0.5", "Vorbis", "128"),
    100: ("webm", "360p", "VP8", "3D", "N/A", "Vorbis", "128"),

    # mpeg4
    18: ("mp4", "360p", "H.264", "Baseline", "0.5", "AAC", "96"),
    22: ("mp4", "720p", "H.264", "High", "2-2.9", "AAC", "192"),
    82: ("mp4", "360p", "H.264", "3D", "0.5", "AAC", "96"),
    83: ("mp4", "240p", "H.264", "3D", "0.5", "AAC", "96"),
    84: ("mp4", "720p", "H.264", "3D", "2-2.9", "AAC", "152"),
    85: ("mp4", "1080p", "H.264", "3D", "2-2.9", "AAC", "152"),
}

# The keys corresponding to the quality/codec map above.
QUALITY_PROFILE_KEYS = (
    "extension",
    "resolution",
    "video_codec",
    "profile",
    "video_bitrate",
    "audio_codec",
    "audio_bitrate"
)
```



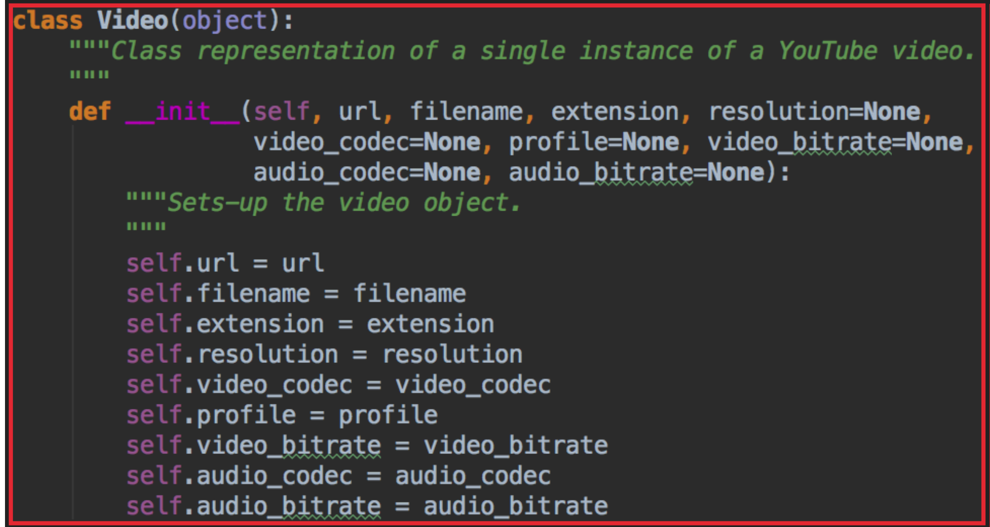
```python
def _add_video(self, url, filename, **kwargs):
    """Adds new video object to videos.

    :param str url:
        The signed url to the video.
    :param str filename:
        The filename for the video.
    :param kwargs:
        Additional properties to set for the video object.
    """
    video = Video(url, filename, **kwargs)
    self._videos.append(video)
    self._videos.sort()
    return True
```

> 마침내 self._video에 Video 클래스의 객체 할당

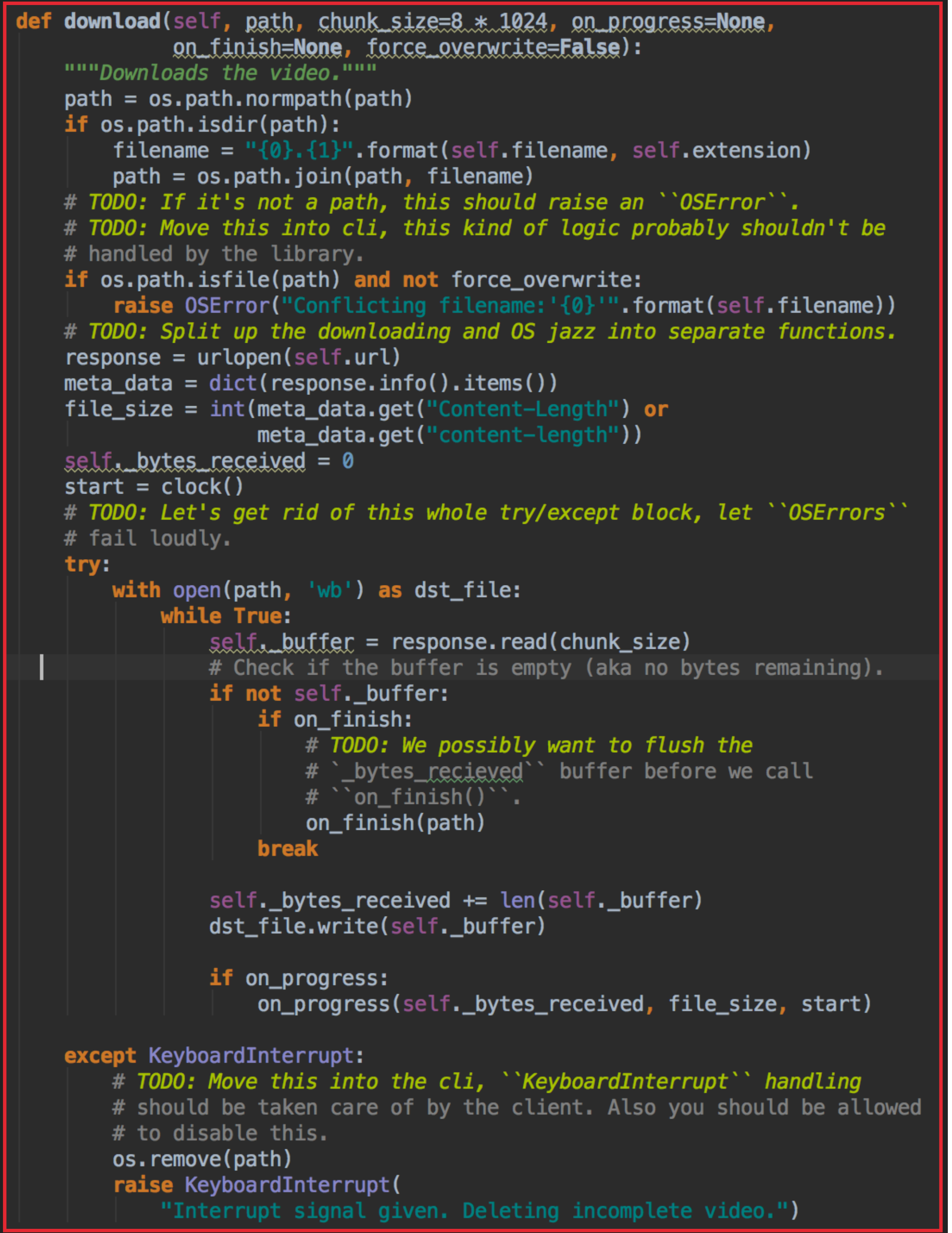
```python
parser.add_argument("--path", "-p", action=FullPaths, default=os.getcwd
                    dest="path", help=("The path to save the video to."
parser.add_argument("--filename", "-f", dest="filename", help=(
    "The filename, without extension, to save the video in."))

args = parser.parse_args()

try:
    yt = YouTube(args.url)
    videos = []
    for i, video in enumerate(yt.get_videos()):
        ext = video.extension
        res = video.resolution
        videos.append("{} {}".format(ext, res))
except PytubeError:
    print("Incorrect video URL.")
    sys.exit(1)

if args.filename:
    yt.set_filename(args.filename)

if args.ext or args.res:
    if not all([args.ext, args.res]):
        print("Make sure you give either of the below specified "
              "format/resolution combination.")
        pprint(videos)
        sys.exit(1)

if args.ext and args.res:
    # There's only ope video that matches both so get it
    vid = yt.get(args.ext, args.res)
    # Check if there's a video returned
    if not vid:
        print("There's no video with the specified format/resolution "
              "combination.")
        pprint(videos)
```

```python
def from_url(self, url):
    """Sets the url for the video.

    :param str url:
        The url to the YouTube video.
    """
    self._video_url = url

    # Reset the filename and videos list in case the same instance is
    # reused.
    self._filename = None
    self._videos = []

    # Get the video details.
    video_data = self.get_video_data()

    # Set the title from the title.
    self.title = video_data.get("args", {}).get("title")

    # Rewrite and add the url to the javascript file, we'll need to fetch
    # this if YouTube doesn't provide us with the signature.
    js_url = "http:" + video_data.get("assets", {}).get("js")

    # Just make these easily accessible as variables.
    stream_map = video_data.get("args", {}).get("stream_map")
    video_urls = stream_map.get("url")

    # For each video url, identify the quality profile and add it to list
    # of available videos.
    for idx, url in enumerate(video_urls):
        log.debug("attempting to get quality profile from url: %s", url)
        try:
            itag, quality_profile = self._get_quality_profile_from_url(url)
            if not quality_profile:
                log.warn("unable to identify profile for itag=%s", itag)
                continue
        except (TypeError, KeyError) as e:
            log.exception("passing on exception %s", e)
            continue

        # Check if we have the signature, otherwise we'll need to get the
        # cipher from the js.
        if "signature=" not in url:
            log.debug("signature not in url, attempting to resolve the "
                      "cipher.")
            signature = self._get_cipher(stream_map["s"][idx], js_url)
            url = "{0}&signature={1}".format(url, signature)
        self._add_video(url, self.filename, **quality_profile)
    # Clear the cached js. Make sure to keep this at the end of
    # `from_url()` so we can mock inject the js in unit tests.
    self._js_cache = None
```

> 모든 준비를 마친 self._video는 다운로드를 하고자 할 때, Video 클래스의 객체를 생성합니다.



```python
class Video(object):
    """Class representation of a single instance of a YouTube video.
    """
    def __init__(self, url, filename, extension, resolution=None,
                 video_codec=None, profile=None, video_bitrate=None,
                 audio_codec=None, audio_bitrate=None):
        """Sets-up the video object.
        """
        self.url = url
        self.filename = filename
        self.extension = extension
        self.resolution = resolution
        self.video_codec = video_codec
        self.profile = profile
        self.video_bitrate = video_bitrate
        self.audio_codec = audio_codec
        self.audio_bitrate = audio_bitrate
```



```python
def download(self, path, chunk_size=8 * 1024, on_progress=None,
             on_finish=None, force_overwrite=False):
    """Downloads the video."""
    path = os.path.normpath(path)
    if os.path.isdir(path):
        filename = "{0}.{1}".format(self.filename, self.extension)
        path = os.path.join(path, filename)
    # TODO: If it's not a path, this should raise an ``OSError``.
    # TODO: Move this into cli, this kind of logic probably shouldn't be
    # handled by the library.
    if os.path.isfile(path) and not force_overwrite:
        raise OSError("Conflicting filename:'{0}'".format(self.filename))
    # TODO: Split up the downloading and OS jazz into separate functions.
    response = urlopen(self.url)
    meta_data = dict(response.info().items())
    file_size = int(meta_data.get("Content-Length") or
                    meta_data.get("content-length"))
    self._bytes_received = 0
    start = clock()
    # TODO: Let's get rid of this whole try/except block, let ``OSErrors``
    # fail loudly.
    try:
        with open(path, 'wb') as dst_file:
            while True:
                self._buffer = response.read(chunk_size)
                # Check if the buffer is empty (aka no bytes remaining).
                if not self._buffer:
                    if on_finish:
                        # TODO: We possibly want to flush the
                        # `_bytes_recieved`` buffer before we call
                        # ``on_finish()``.
                        on_finish(path)
                    break

                self._bytes_received += len(self._buffer)
                dst_file.write(self._buffer)

                if on_progress:
                    on_progress(self._bytes_received, file_size, start)

    except KeyboardInterrupt:
        # TODO: Move this into the cli, ``KeyboardInterrupt`` handling
        # should be taken care of by the client. Also you should be allowed
        # to disable this.
        os.remove(path)
        raise KeyboardInterrupt(
            "Interrupt signal given. Deleting incomplete video.")
```