```python
import os
import googleapiclient.discovery
from googleapiclient.errors import HttpError
from datetime import datetime
import pandas as pd
import requests
import time
```

```python
youtube = googleapiclient.discovery.build(
    "youtube", "v3",
    developerKey = )
```

```python
def get_comments_from_video(video_id: str, max_comments = 0) -> None:
    """
    Takes the video's ID, the string at the end of the url, after "v=",
    and compiles the comments, appending them to a global dataframe df.

    Optionally accepts a limit to comments gathered, default value of
    zero returns all comments. If you specify a max, you may still get
    more than specified due to how the API responds.

    Comments are returned with the most recently interacted
    with first, such as a brand new comment, or an old comment with a new
    reply added.
    """
    # Define a function to reset dict_ with, to facilitate appending
    def reset_dict():
        return {
            'video_id':[], 'text': [], 'likes': [],
            'date': [], 'channel_id': [], 'viewer_rating':[],
            'mentions':[], 'comment_id':[]
            }
    # Initialize the dict_ and access the global df reference
    dict_ = reset_dict()
    global df

    # Adds data from the comment thread's top comment to the dict_
    def read_top_level_comment(comment):
        snip = comment['snippet']['topLevelComment']['snippet']
        dict_['video_id'].append(['videoId'])
        dict_['text'].append(snip['textOriginal'])
        dict_['likes'].append(snip['likeCount'])
        dict_['date'].append(snip['publishedAt'])
        dict_['channel_id'].append(snip['authorChannelId']['value'])
        dict_['viewer_rating'].append(snip['viewerRating'])
        dict_['mentions'].append('')
        dict_['comment_id'].append(comment['id'])

    # Adds the relevent data to a reply comment to the dict
    # Is a seperate function, the data isnt located in identical locations
    def read_reply(reply):
        # Reply comments do not carry their own videoId reference
        snip = reply['snippet']
        dict_['video_id'].append(dict_['video_id'][-1])
        dict_['text'].append(snip['textOriginal'])
        dict_['likes'].append(snip['likeCount'])
        dict_['date'].append(snip['publishedAt'])
        dict_['channel_id'].append(snip['authorChannelId']['value'])
        dict_['viewer_rating'].append(snip['viewerRating'])
        dict_['mentions'].append(snip['parentId'])
        dict_['comment_id'].append(reply['id'])

    # Determine if a max number of comments
    # is called for and create initial request
    max_comments = max_comments if max_comments > 0 else float('inf')
    request = youtube.commentThreads().list(
        part = "snippet,replies",
        videoId = video_id,
        maxResults = min(100, max_comments))

    # Loop persists until all comments gathered/max is exceeded
    while request is not None and max_comments > 0:
        # Try most recent request, ending function if an error occurs
        try:
            response = request.execute()
        except HttpError as err:
            print(err)
            return

        # After positive response, loop through top level comments
        for comment_thread in response['items']:
            read_top_level_comment(comment_thread)
            # If no replies to the top level comment, skip
            if comment_thread['snippet']['totalReplyCount'] < 1:
                continue
            # If there are up to five replies, they are all included in
            # the original response and can be read before moving on
            if comment_thread['snippet']['totalReplyCount'] <= 5:
                for reply in comment_thread['replies']['comments']:
                    read_reply(reply)
                continue
            # Finally, if there are more than five replies,
            # a new request must be made to retrieve them all
            reply_req = youtube.comments().list(
                part = "snippet",
                parentId = comment_thread['id'],
                maxResults = 100)
            # A new loop is necessary in case they exceed 100
            while reply_req is not None:
                # Try most recent reply request, breaking the loop if
```

```python
                # HttpError. This loop might cause you to exceed your
                #  daily limit, so we will just move on from errors
                try:
                    reply_resp = reply_req.execute()
                except HttpError as err:
                    break
                for reply in reply_resp['items']:
                    read_reply(reply)
                reply_req = youtube.comments().list_next(reply_req,
                                                        reply_resp)

        # Decrement max and request next page if possible/needed
        max_comments -= len(dict_['text'])
        request = youtube.commentThreads().list_next(request, response)

        # Append the dataframe with the comments
        #  gathered in this loop and reset the dict for the next loop
        df = df.append(pd.DataFrame.from_dict(dict_))
        df = df.reset_index(drop=True)
        dict_ = reset_dict()
```

```python
# making a list containing all of the videos we've scraped so that we don't waste any time
completed_videos =   ['JWeR_F4uyE0', 'VIMV6E8OxG8', 'THqtAQOicQI',
                      '6VBCxWcAPXw', 'PQnvjGN91Mg', 'Xj1tzy_lTyU',
                      'kmFOBoy2MZ8', '76sJ7C0QEJs', 'C30gxc6TWuY',
                      'W9olSzNOh8s', 'kS0Jg6hlUSs', 't9c7aheZxls',
                      't_n0yhhuJBs', 'NtQkz0aRDe8', '-9lBVznUuHk',
                      'X8bBP_cLrl0', 'b3D7QlMVa5s', 'KQqHDEYpIvI',
                      'wYDJ0vxg1lU', 'NtQkz0aRDe8', '-dL28N5yPmQ',
                      'R_LqgcndmAo', 'eH-xm9G9QBk', 'DMMPYkRrd4o'
                      'eXRdZ_qnZTA', '2emC9xPKh_Q', 'h8T9mVkGh3s',
                      'gWKyTYEFVGY', '-YebEDmbG_M', 'llh8rfwWqqY',
                      '60V0_-AHfyM', 'w6J7FteaW2Q', 'H0CBLw0xOlo',
                      'QKq4sLERZ4M', '9T6WqdHq7JY', 'kS1J-ZSaecw',
                      'w_4D6xKqH9w', 'D1KPZOK-iHg', 'ji5i4gXBcSk',
                      'LdIW_bOaspg', 'AKLnXeFDQ1A', 'TOivsknjD0k',
                      'njESY1JxNcM', 'P5BNNA97LEc', 'VOD_uugAlJw',
                      'xg_jyUDsLpU', 'N_SjGaiuGoU', '1xWbCcaJnIQ',
                      'db4cEuLpPsQ', 'aTci511TD4A', 'ego91VOyObw',
                      '88QDCJkNLlE', 'LOkqR4CK7Qc', 'hgE-v1OEJFM',
                      'hzp7vqgprCc', 'uFhsagtKtwM', 'QKq4sLERZ4M',
                      'JzeYsRt7axc', 'nhimQHsTo0s', '8ydvxFu6bJ8',
                      '9ot3bCkhjTM', 'mKAIL8DDemg', 'kPd56OY2ED8',
                      'FTcXKFZcToM', '-R2x02n-o64', 'vS7aidy2bwk',
                      'iB0ilH7yrfU', 'XVqPwcnRGBU', 'OyrFddzsymQ',
                      '0kZ-EcGt39s', '0pGzSKohRJo', 'e7o4ct0Z8tI',
                      'VYD0DleJn7U', 'OYAgcS31-p0', 'Zo62S0ulqhA',
                      '5OLtteIwwNs', 'ZL4yYHdDSWs', 'UNEFDynNw-Q',
                      't59Ge4O70iM', 'AhF44UT2AIk', 'biSWmzIg-2k',
                      'N-1gzo3Pyvo', '0kZ-EcGt39s', 'OyF1ByhjSv0',
                      'g_ROkapCj14', 'Bsgrbd_Yv4Y', 'fXcmzmWXZmw',
                      'fzyYAVz3IGg', 'ep3GlvxUUew', '2QI7z46LWLY',
                      '8Oh5ARY_MCM', 'fniq8Wuw60A', 'H0CBLw0xOlo',
                      '5nE3UO1kqv0', '-nwbLls-PCs', 'vTNP01Sg-Ss',
                      'PHY_vAKLzzo', 'BpPmP8DUh4s', 'F-c5iAyfgAU',
                      'lQS8cnsZuGU', 'LmfaVwAXZy4', 'wtlUnI1fe8Y',
                      '35b2tAMxQXg', 'hTbQF6UBe5Q', 'IW9A-uWM0JU',
                      '85vvVZ4jSZM', 'xudplVZgGV0', '76sJ7C0QEJs',
                      '8Uvgh4gYzlw', 'ySKIm7k1-18', 'oAqhNmLmY7g',
                      'C30gxc6TWuY', '1iGriklFHHQ', 'GQ7v2dI2RF4',
                      'lCCKdcL_h3Y', 'oBXmUP3Jq8A', 'SM1vXb6J7gE',
                      'dTEIL19FLYI', 'VZpN7hd1ybI', 'C9GiZDoZvxE',
                      '-qov7HlrvbM', 'KZXjFdrct-w', 'NyLPPXaGl5A',
                      'C2jh7dCwGRs', 'XL1ehbG9EL8', '42Je9Xczu0o',
                      'D-J9maAnhwg', '_Ihdb8-h5Ek', '4cv3SjVK-n0',
                      'hYyg8JC-6ew', 'RcXBuYwm3xk', '-YebEDmbG_M',
                      'TNRQFKVV68I', 'pxa0IrZCNzg', 'vFdx1Hs71iA',
                      'fM-JHvg-ZCM', 'aCCR5qBsD0c', 'cb6sdimG8GE',
                      '0ENabNTQwNg', 'LqoYtBZAKO0', 'H2f0Wd3zNj0',
                      'JkeLIAd2Nd0', 'TmLWxptFFYc', 'S0dqd72ALkQ',
                      '0Ap4JhPoPQY', 'P4aXmnQzJ0o', 'PQnvjGN91Mg',
                      'HdpRxGjtCo0', 'BI-old7YI4I', 'kmFOBoy2MZ8',
                      'bGcvv3683Os', 'JgxkilF5XUM', 's6BQSgidbmc',
                      '6VBCxWcAPXw', '2zaIy1TARPE', '3y3MmmfZmP8',
                      'xe4Kkbq4An8', 'X4C5fbcYSNg', 'U09K0bQT5PE',
                      'X8bBP_cLrl0', 'oyKnBTIoC5E', 'EVicgFd25D4',
                      'Ox6pqjQiuJ0', 'fwCl9Ce7MDM', 'aPuDNDZZ6-U',
                      '_9MKYKR8lFA', 'vOpH3xnzFJE', 'bq220dgUb0I',
                      'lLTdBJsU8N8', 'qXZdRDoGSHo', 'I7yCAmLEDdo',
                      'Gogn3p8aDEs', 'TYB8dvCNCQc', 'g_m5VRiKy_E',
                      'Gcnf5BdLXxw', '1bJKAu11Ni4', 'OYAgcS31-p0',
                      'PPqI-Sk7vsw', 'YWKWkuJwHj4', 'JVhJcXBTl3Y',
                      'wfAoq89LNRQ', 'ZSNxaWkuoRo', '4cvZ9NWgsws',
                      '-n9uz_cOjT8', '17i2kyEgjWE', '5nE3UO1kqv0',
                      'JmF-OOuOxKg', 'WREUb8T4r8o', 'Li7_yFiNaIA',
                      'FxrAe5N1xu0', 'CIf6VJH4dZk', 'W77xm6f2sJI',
                      '8VzSqYooxmw', 'c7OeeGcMFMc', 'xsMAY4_ICdM',
                      'N6wq2eHOZYU', '5VfesP3p0xc', 'X_m1mPtYzTk',
                      'H7Uyfqi_TE8', 'UkAVtEoSnoE', 'XQXF3PnSROk',
                      'HPTNbPgB5eg', 'JaimO7nvzzQ', '68bu0AeCHm8',
                      'QodPNv_XIow', 'j9SdeW5UqTY']
```

```
# creating a list of videos to get the comments from
completed_videos = [ ]
# starting a dataframe for us to add additional comments onto
df = pd.DataFrame()

#iterating over our list of videos to get their comments and add them to our starting data
for item in completed_videos:
    # checking if video is in our completed list, easier than remembering or checking visu
    if item not in video_list:
        get_comments_from_video(video_id=item)
```

```python
        video_list.append(item)
```

```python
df.to_csv('4.7.yt.csv')
```