YOUTUBE CODE DOCUMENTATION

Code File: FullYTScrape.py

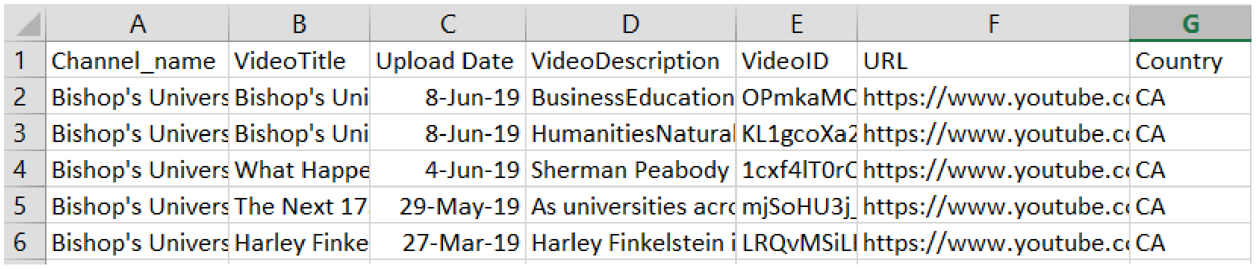
Updated on: 14/08/2019

Table of Contents

1. Overview:

The code script that is able to collect all links from the website before processing it into a CSV is done using 'FullYTScrape' Code. This script can be used to run and obtain all the Youtube videos from a list of Youtube channels and or playlists that is provided in an excel file. However, take note that out of 540 (at this time of writing, 07/08/2019) channel lists there is a total of at least 110,000 videos and it will take extensive time for the script to even finish running and collecting all the data. Additionally, if any errors were to occur such as due to loss of internet connection, the progress may be stopped and the data collection will be affected. As such, it is more practical to divide the scraping script into two main parts and run each script using multiple computers. The first script called 'YTVideoCollectLinks' will be executed first involving the collection of all video links from each Youtube channel. Once all the video links have been collected and stored in a file 'YoutubeVideoLinks.csv', the second script called 'YTCollectData Code' will then be executed to process the data from each video link such as Video Name, Video ID, description and will be stored inside a Uni_YoutubeData.csv Excel file. The multiple Uni_YoutubeData.csv excel files can be easily combined by using a script Combine Code.

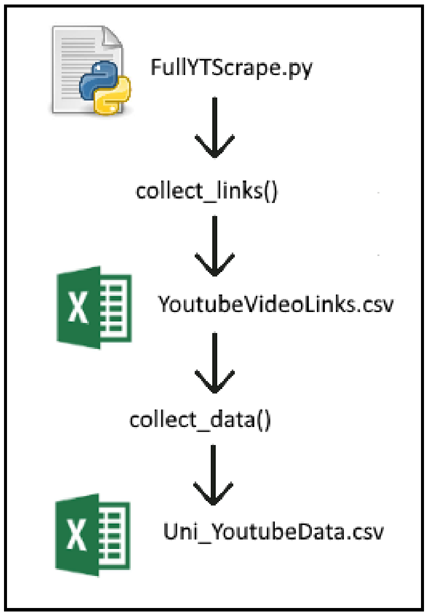
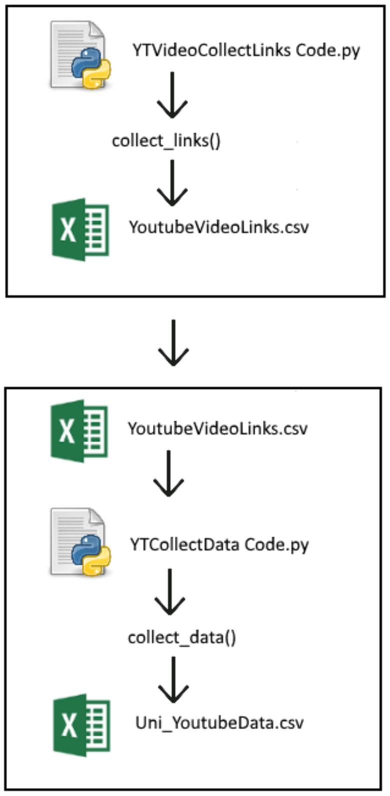
The end product is a .csv file that contains all Youtube video data from the Youtube university channels.



	A	B	C	D	E	F	G
1	Channel_name	VideoTitle	Upload Date	VideoDescription	VideoID	URL	Country
2	Bishop's Univers	Bishop's Uni	8-Jun-19	BusinessEducation	OPmkaMC	https://www.youtube.c	CA
3	Bishop's Univers	Bishop's Uni	8-Jun-19	HumanitiesNatural	KL1gcoXa2	https://www.youtube.c	CA
4	Bishop's Univers	What Happe	4-Jun-19	Sherman Peabody	1cxf4lT0rC	https://www.youtube.c	CA
5	Bishop's Univers	The Next 17	29-May-19	As universities acro	mjSoHU3j_	https://www.youtube.c	CA
6	Bishop's Univers	Harley Finke	27-Mar-19	Harley Finkelstein i	LRQvMSiLI	https://www.youtube.c	CA

The final .csv file containing all columns of data pertaining to University Youtube videos.

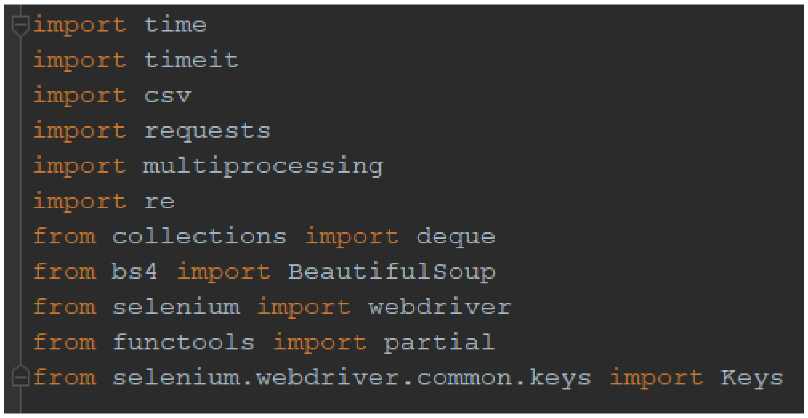
Splitting the tasks into two parts is also more beneficial than running the complete script in one go as in the case of any errors, rerunning one smaller script is much faster as compared to the full script. Any errors occurred would also only be localized within each script. For instance, if YTCollectData Code.py was running and the computer crashed, only Uni_YoutubeData.csv file is affected while YoutubeVideoLinks.csv is unaffected as it was done previously using the another code. Whereas, if any errors happened at any point of time during FullYTScrape, the process has to be executed from the beginning again. Refer to the chart in the following page for the flow of the processes visualizing the differences.

Running a single script for all methods. (FullYTScrape.py)	**Running collect_links and collect_data separately in two scripts. (YTVideoCollectLinks Code.py, followed by YTCollectData Code.py)**
 FullYTScrape.py collect_links() YoutubeVideoLinks.csv collect_data() Uni_YoutubeData.csv	 YTVideoCollectLinks Code.py collect_links() YoutubeVideoLinks.csv YoutubeVideoLinks.csv YTCollectData Code.py collect_data() Uni_YoutubeData.csv

Each box shows the extent of files affected by the script. If any files have errors, or scripts were affected, everything inside the boxes need to be redone.

This document will explain the code sections according to FullYTScrape.py. YTVideoCollectLinks Code.py and YTCollectData Code.py are basically just snippets from FullYTScrape.py and contain the same code from the full script file just that they were divided into respective scripts for multiprocessing.

2. Package Modules



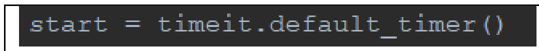
```
import time
import timeit
import csv
import requests
import multiprocessing
import re
from collections import deque
from bs4 import BeautifulSoup
from selenium import webdriver
from functools import partial
from selenium.webdriver.common.keys import Keys
```

Package modules	Explanation
Time, timeit	User for timer purposes.
csv	For usage with .csv files.
requests	Used to load web pages. Does not need to open any web browsers.
Multiprocessing	Used for processing.
re	Necessary for Regular Expression / Regex functions.
Collections, deque	For iterating the array list containing URLs.
Bs4	For usage of soup functions in order to extract elements from a page.
Selenium, webdriver	An alternative method to load web pages. Each process will open a web browser window to load the page. Much slower compared to requests method.
Functools, partial	Used to handle a method that requires multiple arguments and to be multi processed using multiprocess.Pool(). Typically multiprocess.Pool() only allows processing of a method with one argument.
Selenium.webdriver.common.keys, Keys	Used for simulating keyboard key presses. In this case, used ‘END’ key to scroll down a webpage.

3. Major Methods

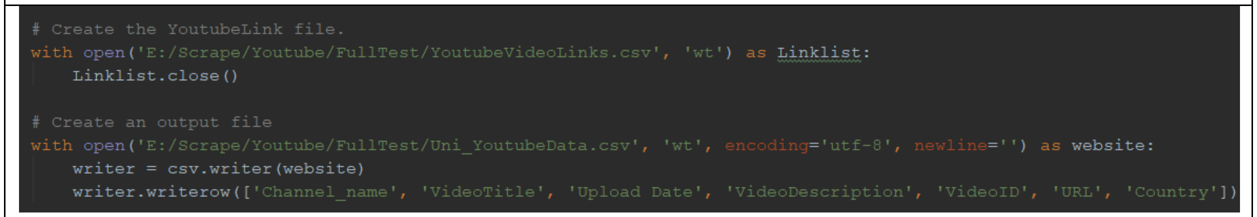
3.1 main()

This section will explain each section of the main() method. This method forms as the basis for the entire program and will execute the necessary methods in order to perform web scraping of the Youtube website.



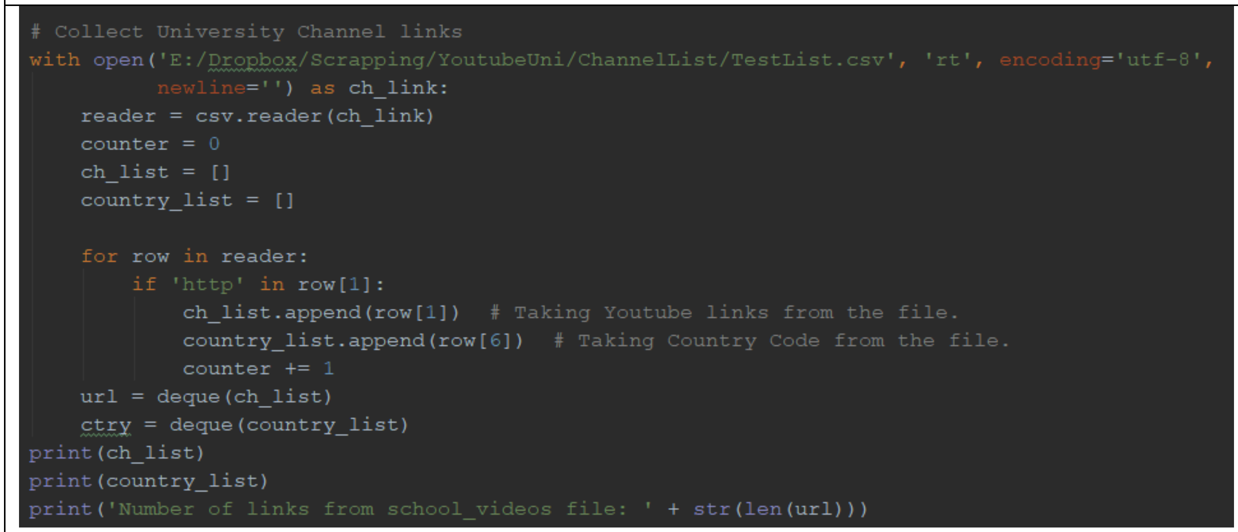
```
start = timeit.default_timer()
```

This shows the time for how long the code was executed for.



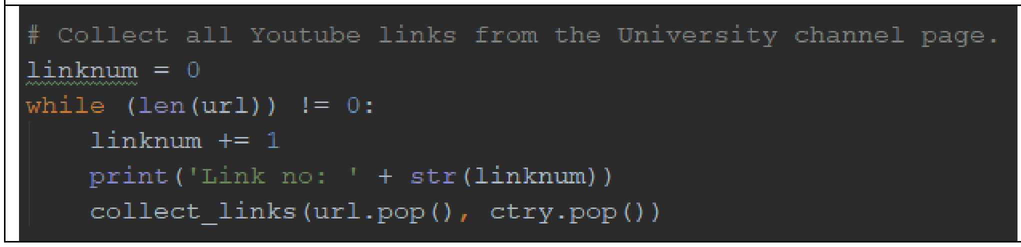
```
# Create the YoutubeLink file.
with open('E:/Scrape/Youtube/FullTest/YoutubeVideoLinks.csv', 'wt') as Linklist:
    Linklist.close()

# Create an output file
with open('E:/Scrape/Youtube/FullTest/Uni_YoutubeData.csv', 'wt', encoding='utf-8', newline='') as website:
    writer = csv.writer(website)
    writer.writerow(['Channel_name', 'VideoTitle', 'Upload Date', 'VideoDescription', 'VideoID', 'URL', 'Country'])
```

Creates two files at the directory specified. ‘YoutubeVideoLinks.csv’ file is for the storage of each Youtube video URL while ‘Uni_YoutubeData.csv’ file is the final Excel file containing all details for each Youtube video.



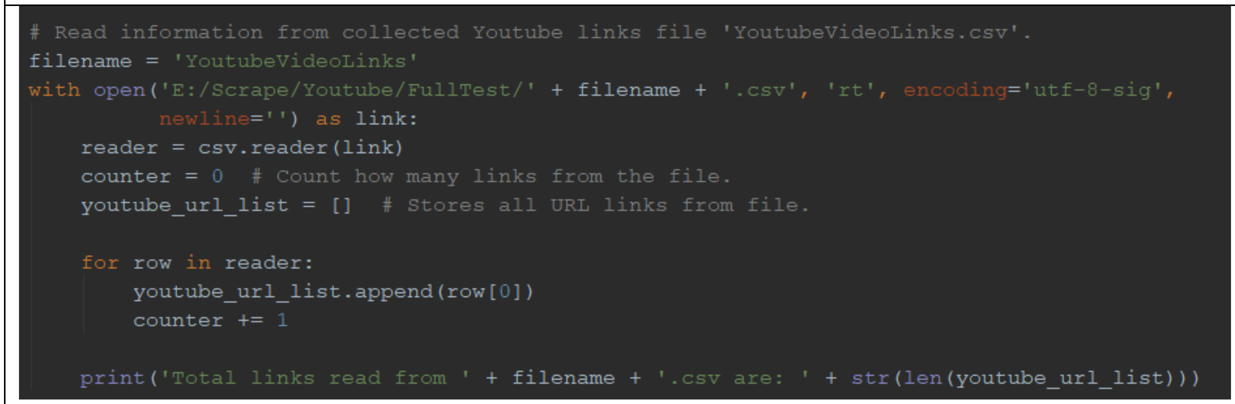
```
# Collect University Channel links
with open('E:/Dropbox/Scrapping/YoutubeUni/ChannelList/TestList.csv', 'rt', encoding='utf-8',
          newline='') as ch_link:
    reader = csv.reader(ch_link)
    counter = 0
    ch_list = []
    country_list = []

    for row in reader:
        if 'http' in row[1]:
            ch_list.append(row[1])  # Taking Youtube links from the file.
            country_list.append(row[6])  # Taking Country Code from the file.
            counter += 1
    url = deque(ch_list)
    ctry = deque(country_list)
print(ch_list)
print(country_list)
print('Number of links from school_videos file: ' + str(len(url)))
```

Reading from a source file and taking the url links of each Youtube channel as well as the country code. The data read will be stored in array lists ‘ch_list’ and ‘country_list’. These lists are then deque’d for iteration purposes in the next section.



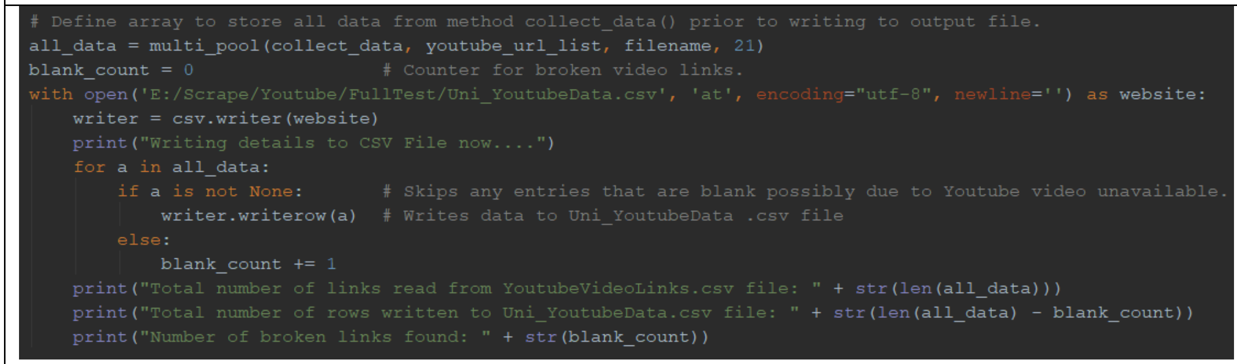
```
# Collect all Youtube links from the University channel page.
linknum = 0
while (len(url)) != 0:
    linknum += 1
    print('Link no: ' + str(linknum))
    collect_links(url.pop(), ctry.pop())
```

This section involves collecting each Youtube video link present in the Uploads section of the Youtube channel. A loop is created to execute the 'collect_links()' method until the list of Youtube Channel URL links is exhausted. 'url' and 'ctry' corresponds to the deque'd 'ch_list' and 'country_list' created previously and removes a row every time the row is read in each iteration. When the url row has been completely read and cleared, the loop ends.



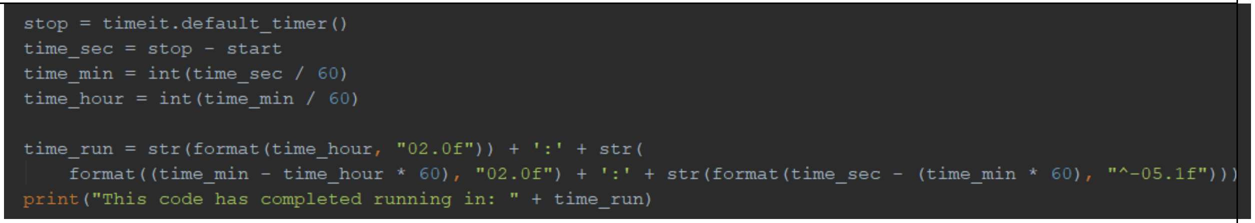
```python
# Read information from collected Youtube links file 'YoutubeVideoLinks.csv'.
filename = 'YoutubeVideoLinks'
with open('E:/Scrape/Youtube/FullTest/' + filename + '.csv', 'rt', encoding='utf-8-sig',
          newline='') as link:
    reader = csv.reader(link)
    counter = 0  # Count how many links from the file.
    youtube_url_list = []  # Stores all URL links from file.

    for row in reader:
        youtube_url_list.append(row[0])
        counter += 1

    print('Total links read from ' + filename + '.csv are: ' + str(len(youtube_url_list)))
```

Reads the created 'YoutubeVideoLinks.csv' file that have collected all Youtube video links from the Youtube channels. The links are stored in youtube_url_list which will be used to scrape each video details using the collect_data() method in the next section.



```python
# Define array to store all data from method collect_data() prior to writing to output file.
all_data = multi_pool(collect_data, youtube_url_list, filename, 21)
blank_count = 0                  # Counter for broken video links.
with open('E:/Scrape/Youtube/FullTest/Uni_YoutubeData.csv', 'at', encoding="utf-8", newline='') as website:
    writer = csv.writer(website)
    print("Writing details to CSV File now....")
    for a in all_data:
        if a is not None:       # Skips any entries that are blank possibly due to Youtube video unavailable.
            writer.writerow(a)  # Writes data to Uni_YoutubeData .csv file
        else:
            blank_count += 1
    print("Total number of links read from YoutubeVideoLinks.csv file: " + str(len(all_data)))
    print("Total number of rows written to Uni_YoutubeData.csv file: " + str(len(all_data) - blank_count))
    print("Number of broken links found: " + str(blank_count))
```

collect_data() method is executed in parallel by utilizing multiprocessing.Pool() inside the defined multi_pool() method. Refer to section for further details regarding each method. Each time collect_data() is processed, it will return a row of Youtube video details which will then be appended to all_data. Eventually at the end, all_data will possess all Youtube video details in each row from all Youtube video url links. The data from this list will then be written into the 'Uni_YoutubeData.csv' file that was created at the beginning of main().

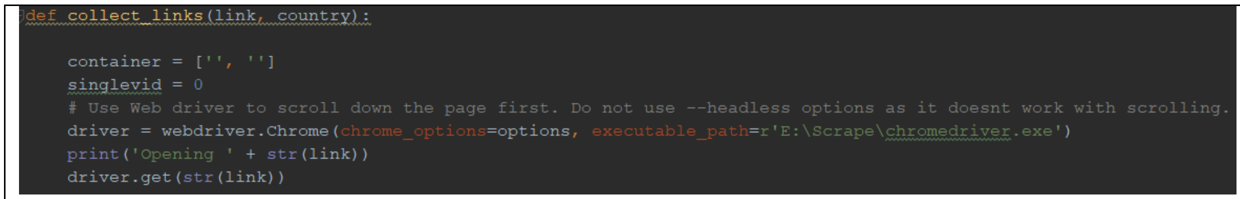


```python
stop = timeit.default_timer()
time_sec = stop - start
time_min = int(time_sec / 60)
time_hour = int(time_min / 60)

time_run = str(format(time_hour, "02.0f")) + ':' + str(
    format((time_min - time_hour * 60), "02.0f") + ':' + str(format(time_sec - (time_min * 60), "^-05.1f")))
print("This code has completed running in: " + time_run)
```

Code for showing the time that was taken to complete all processes. Typically no change required to this section.

3.2 collect_links()

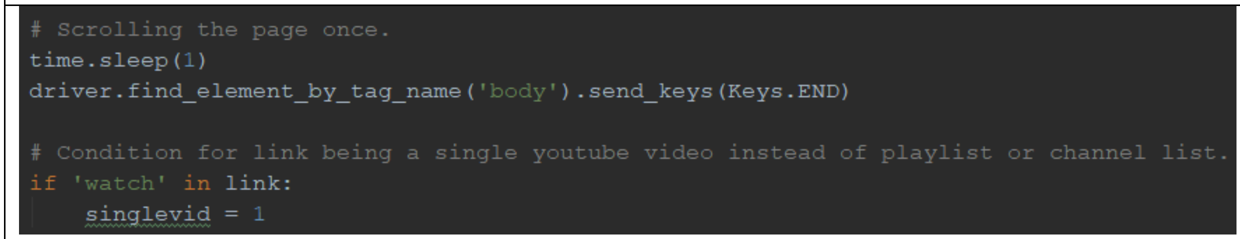
This method is used to collect the Youtube Video links from the uploads section of Youtube Channels.



```
def collect_links(link, country):

    container = ['', '']
    singlevid = 0
    # Use Web driver to scroll down the page first. Do not use --headless options as it doesnt work with scrolling.
    driver = webdriver.Chrome(chrome_options=options, executable_path=r'E:\Scrape\chromedriver.exe')
    print('Opening ' + str(link))
    driver.get(str(link))
```

This section details the section inside collect_links() method. This method takes two arguments: link, country which corresponds to the Youtube channel list URLs and Country Code from the previous section.

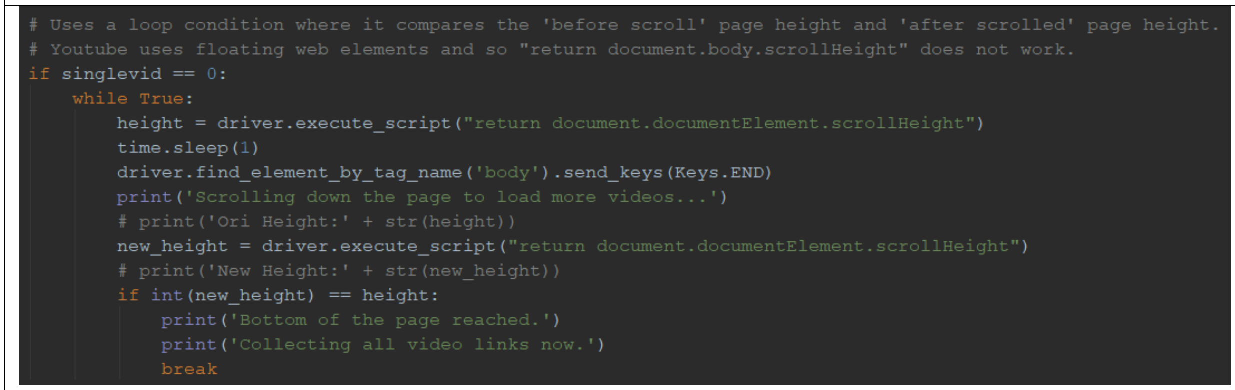
The URL link is opened using Selenium / Web Driver.



```
# Scrolling the page once.
time.sleep(1)
driver.find_element_by_tag_name('body').send_keys(Keys.END)

# Condition for link being a single youtube video instead of playlist or channel list.
if 'watch' in link:
    singlevid = 1
```

The page is scrolled down once in order to check whether the page is scrollable. If the page is scrollable, then it should be done until the bottom of the page is reached. This is because there may be Youtube Videos on the page that is hidden, unless the page was scrolled into view. This is done using the selenium module.

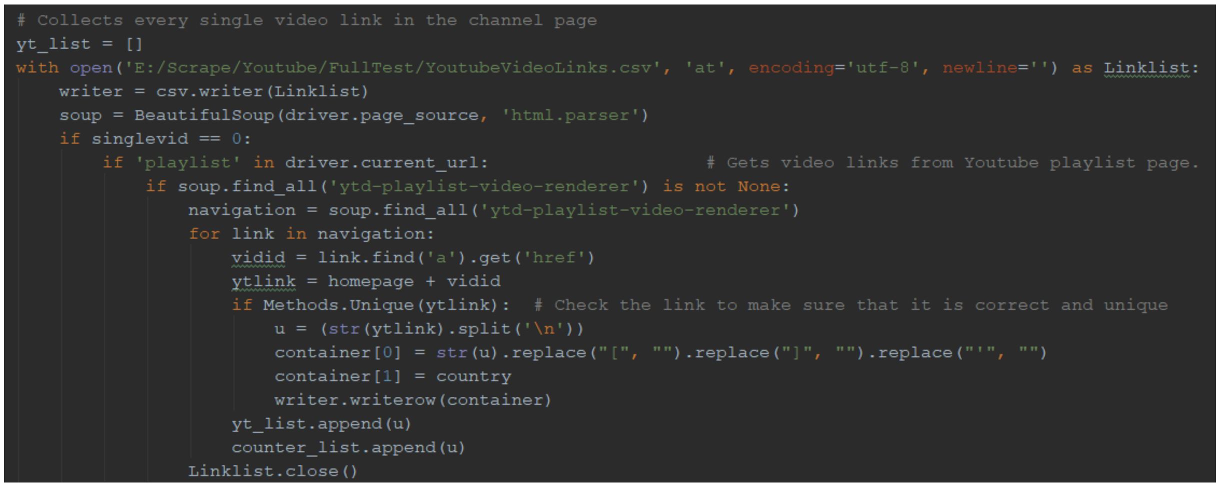
Singlevid is a variable used to set conditions where pages are a direct Youtube Video Page. These types of pages have a URL that contains the term ‘watch’. For example: https://www.youtube.com/watch?v=vkG2CfmqE8A.



```
# Uses a loop condition where it compares the 'before scroll' page height and 'after scrolled' page height.
# Youtube uses floating web elements and so "return document.body.scrollHeight" does not work.
if singlevid == 0:
    while True:
        height = driver.execute_script("return document.documentElement.scrollHeight")
        time.sleep(1)
        driver.find_element_by_tag_name('body').send_keys(Keys.END)
        print('Scrolling down the page to load more videos...')
        # print('Ori Height:' + str(height))
        new_height = driver.execute_script("return document.documentElement.scrollHeight")
        # print('New Height:' + str(new_height))
        if int(new_height) == height:
            print('Bottom of the page reached.')
            print('Collecting all video links now.')
            break
```

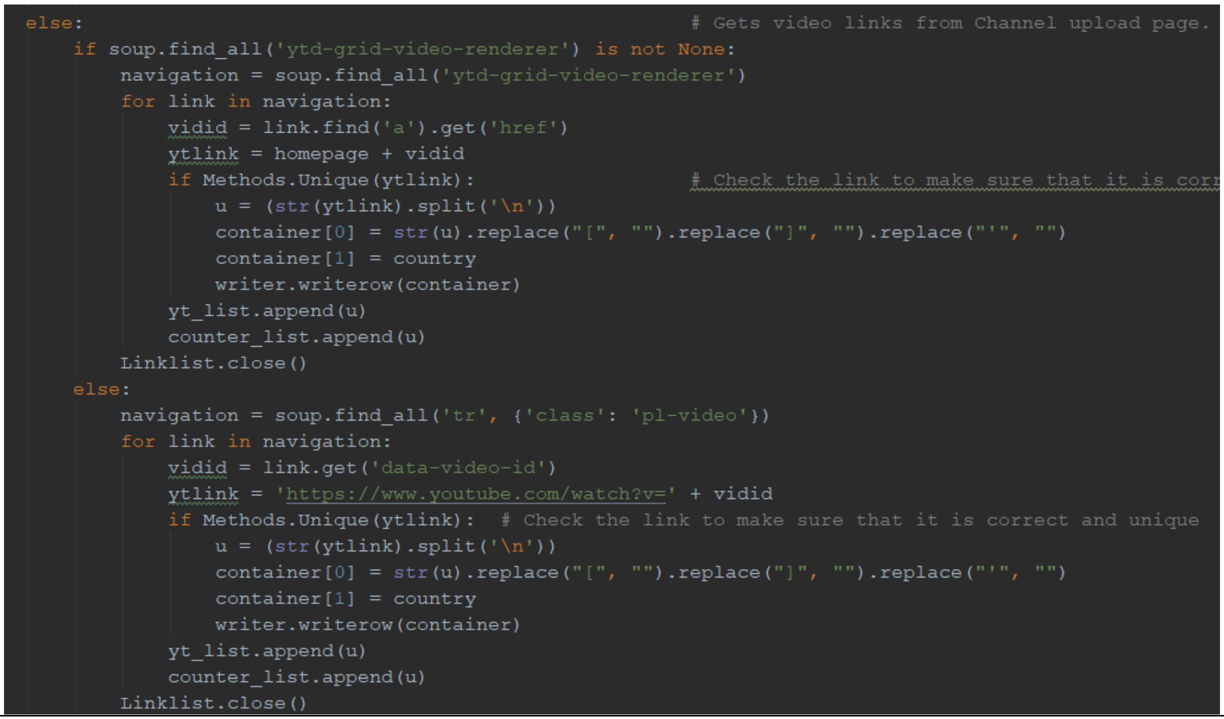
A loop for scrolling the page. ‘singlevid’ as previously explained is used here to ensure the code only runs for channel / playlist pages and not single page videos. The logic of the loop is essentially to find the bottom of the page in order to load all Youtube videos in the channel list page. The code does this by recording the height of the page, then simulating the keyboard button ‘End’ to scroll down the page once, and then recording the new height of the page in another variable. If the heights do not match, the loop continues. At the bottom of the page, the ‘height’ and ‘new_height’ should have the same value since there should be no change in page height before and after pressing the ‘End’ key.

This section contains the code for collecting the Youtube video links. It is essentially divided into three separate sub sections nested within ‘if-else’ statements. One section is for obtaining links if the page is a Youtube Channel, one section is for Youtube Playlists and one section is for direct Youtube Video page itself.



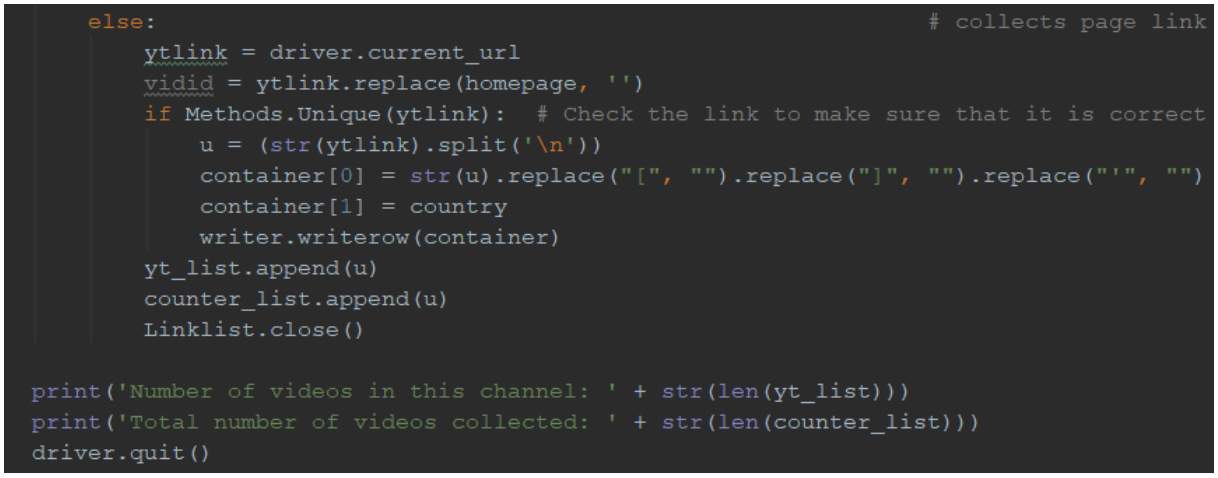
```
# Collects every single video link in the channel page
yt_list = []
with open('E:/Scrape/Youtube/FullTest/YoutubeVideoLinks.csv', 'at', encoding='utf-8', newline='') as Linklist:
    writer = csv.writer(Linklist)
    soup = BeautifulSoup(driver.page_source, 'html.parser')
    if singlevid == 0:
        if 'playlist' in driver.current_url:                    # Gets video links from Youtube playlist page.
            if soup.find_all('ytd-playlist-video-renderer') is not None:
                navigation = soup.find_all('ytd-playlist-video-renderer')
                for link in navigation:
                    vidid = link.find('a').get('href')
                    ytlink = homepage + vidid
                    if Methods.Unique(ytlink):  # Check the link to make sure that it is correct and unique
                        u = (str(ytlink).split('\n'))
                        container[0] = str(u).replace("[", "").replace("]", "").replace("'", "")
                        container[1] = country
                        writer.writerow(container)
                    yt_list.append(u)
                    counter_list.append(u)
                Linklist.close()
```

This first section involves checking if the page is a playlist. If it is, then the tags for the playlist page ‘href’ tags will be located. The ‘href’ tags on the Youtube page is typically in the form of ‘/watch?=…..’ where the full youtube video link can be obtained by combining it with the homepage url. This is done in ‘ytlink’ where ‘www.youtube.com’ in ‘homepage’ variable is combined with ‘vidid’ containing the ‘href’ link. Once that is done, the container list will store the full URL link as well as the country code from the previous section.



```
else:                                                   # Gets video links from Channel upload page.
    if soup.find_all('ytd-grid-video-renderer') is not None:
        navigation = soup.find_all('ytd-grid-video-renderer')
        for link in navigation:
            vidid = link.find('a').get('href')
            ytlink = homepage + vidid
            if Methods.Unique(ytlink):                  # Check the link to make sure that it is corr
                u = (str(ytlink).split('\n'))
                container[0] = str(u).replace("[", "").replace("]", "").replace("'", "")
                container[1] = country
                writer.writerow(container)
            yt_list.append(u)
            counter_list.append(u)
        Linklist.close()
    else:
        navigation = soup.find_all('tr', {'class': 'pl-video'})
        for link in navigation:
            vidid = link.get('data-video-id')
            ytlink = 'https://www.youtube.com/watch?v=' + vidid
            if Methods.Unique(ytlink):  # Check the link to make sure that it is correct and unique
                u = (str(ytlink).split('\n'))
                container[0] = str(u).replace("[", "").replace("]", "").replace("'", "")
                container[1] = country
                writer.writerow(container)
            yt_list.append(u)
            counter_list.append(u)
        Linklist.close()
```

The second section is for locating 'href' tags if the page is a Youtube Channel page. There are two types of Channel page layouts and as such a nested 'if-else' statement is used to check for both layouts and grab the information accordingly.



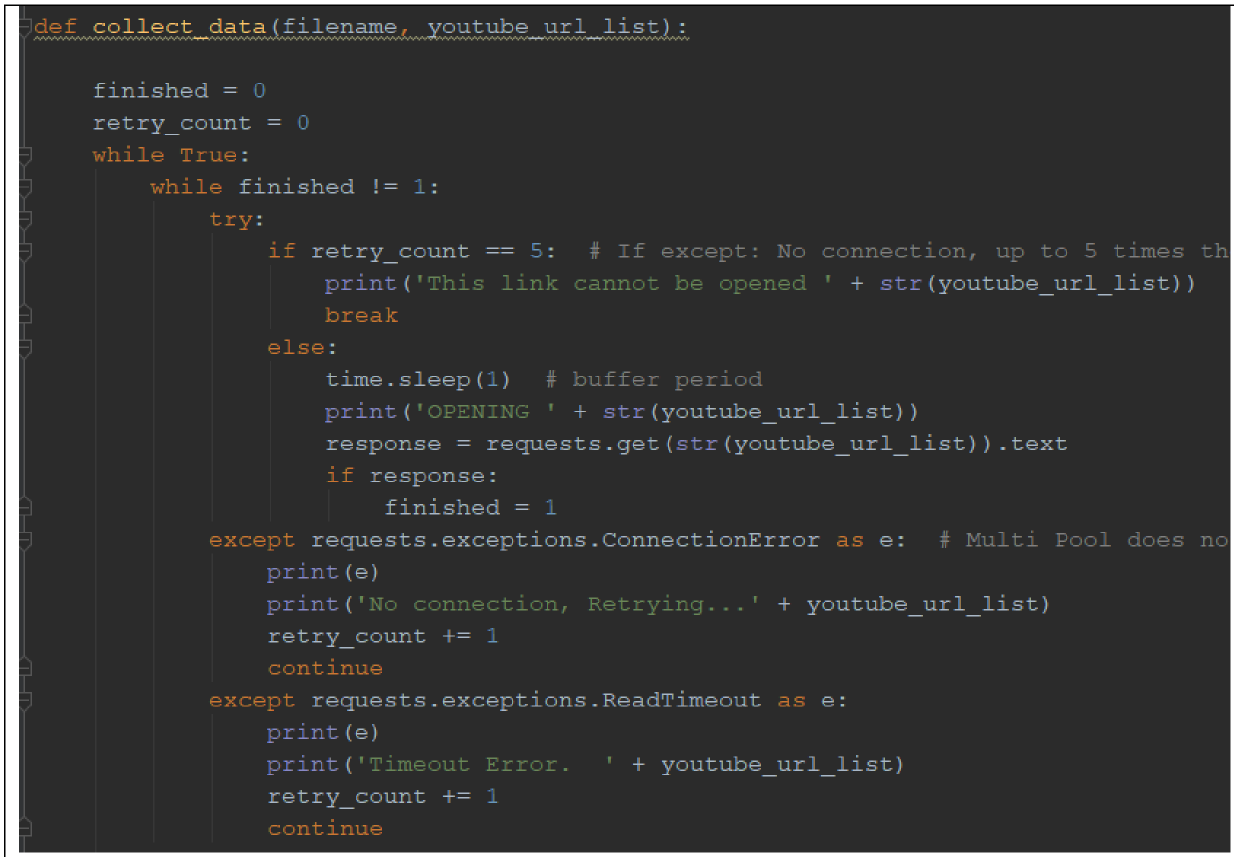
```
    else:                                                    # collects page link
        ytlink = driver.current_url
        vidid = ytlink.replace(homepage, '')
        if Methods.Unique(ytlink):  # Check the link to make sure that it is correct
            u = (str(ytlink).split('\n'))
            container[0] = str(u).replace("[", "").replace("]", "").replace("'", "")
            container[1] = country
            writer.writerow(container)
        yt_list.append(u)
        counter_list.append(u)
        Linklist.close()

print('Number of videos in this channel: ' + str(len(yt_list)))
print('Total number of videos collected: ' + str(len(counter_list)))
driver.quit()
```

The last section of this method covers the final condition where the page is a single direct Youtube Video page. As the page itself is the Youtube Video page, the link is collected by using driver.current_url which refers to the url of the page itself.

Driver.quit() is used to close the browser as there are no further actions required and the video link collection process is completely.

3.3 collect_data()

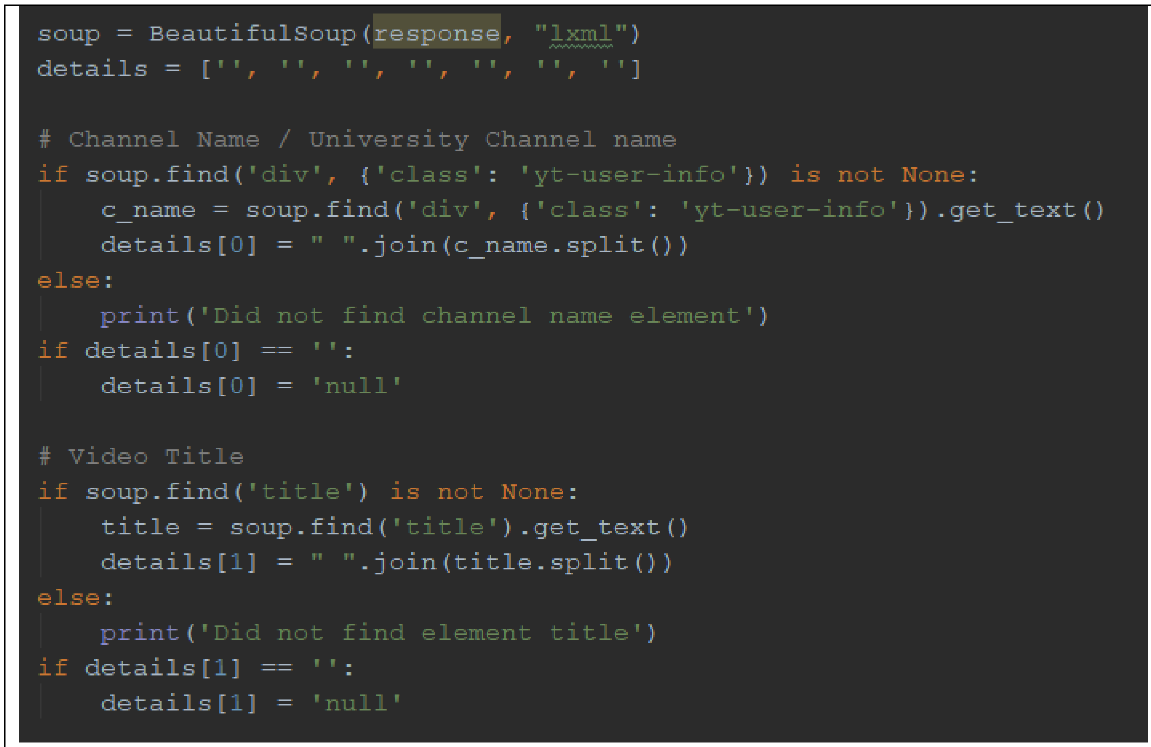
This method contains the code for the scraping of the Youtube video information from the video page. Basically, the code will create an array called 'details' and will store all relevant information within this array. (Refer to Section 3.1 to see what column names were made when ExtractedData.csv was created) Once the 'details' array is populated with the required data, the method is complete. When this method is called, the output will return the 'details' array.



```
def collect_data(filename, youtube_url_list):

    finished = 0
    retry_count = 0
    while True:
        while finished != 1:
            try:
                if retry_count == 5:  # If except: No connection, up to 5 times th
                    print('This link cannot be opened ' + str(youtube_url_list))
                    break
                else:
                    time.sleep(1)  # buffer period
                    print('OPENING ' + str(youtube_url_list))
                    response = requests.get(str(youtube_url_list)).text
                    if response:
                        finished = 1
            except requests.exceptions.ConnectionError as e:  # Multi Pool does no
                print(e)
                print('No connection, Retrying...' + youtube_url_list)
                retry_count += 1
                continue
            except requests.exceptions.ReadTimeout as e:
                print(e)
                print('Timeout Error.  ' + youtube_url_list)
                retry_count += 1
                continue
```

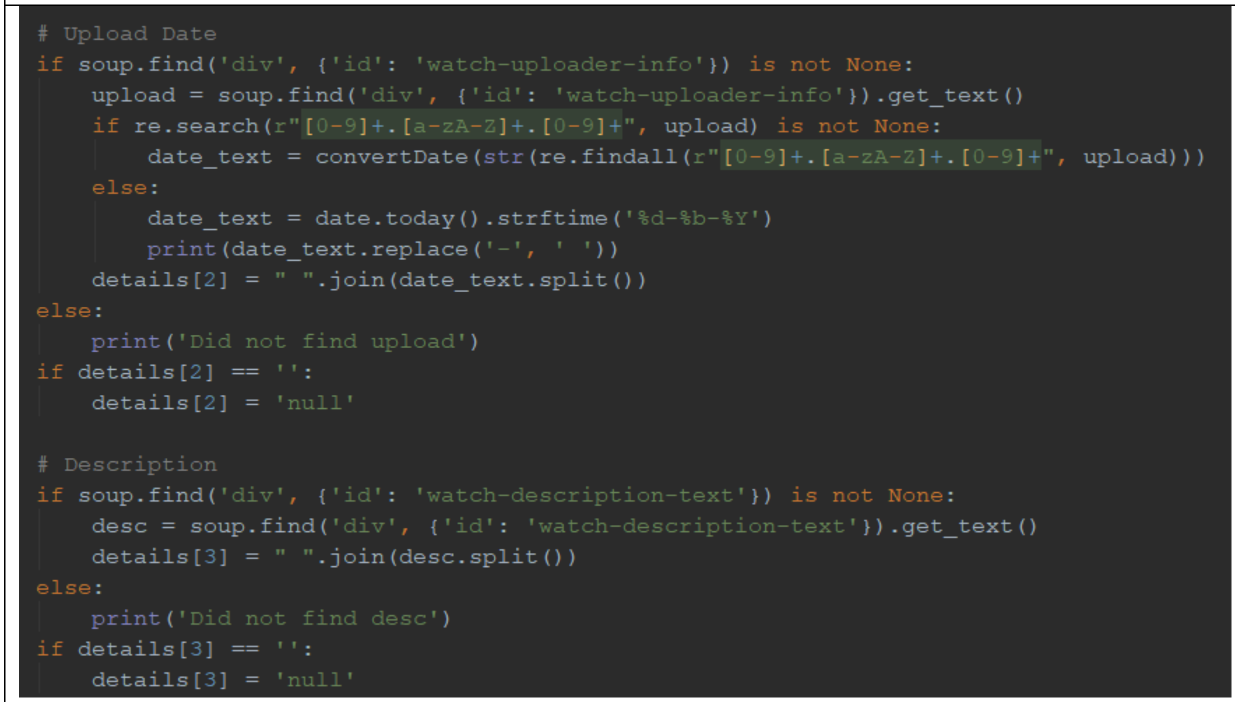
The collect_data() method is defined to accept two arguments when called. ‘filename’ is used to refer to the source file containing the Country Code information while ‘youtube_url_list’ is the collection of Youtube video links.

The rest of this section is basically to utilize requests to open each Youtube Video links from ‘youtube_url_list’ while handling errors that may result due to connection problems. By default, this code will handle up to 5 attempts at opening a link. After the 5th attempt, if the page could not be accessed then the link will be skipped.



```python
soup = BeautifulSoup(response, "lxml")
details = ['', '', '', '', '', '', '']

# Channel Name / University Channel name
if soup.find('div', {'class': 'yt-user-info'}) is not None:
    c_name = soup.find('div', {'class': 'yt-user-info'}).get_text()
    details[0] = " ".join(c_name.split())
else:
    print('Did not find channel name element')
if details[0] == '':
    details[0] = 'null'

# Video Title
if soup.find('title') is not None:
    title = soup.find('title').get_text()
    details[1] = " ".join(title.split())
else:
    print('Did not find element title')
if details[1] == '':
    details[1] = 'null'
```

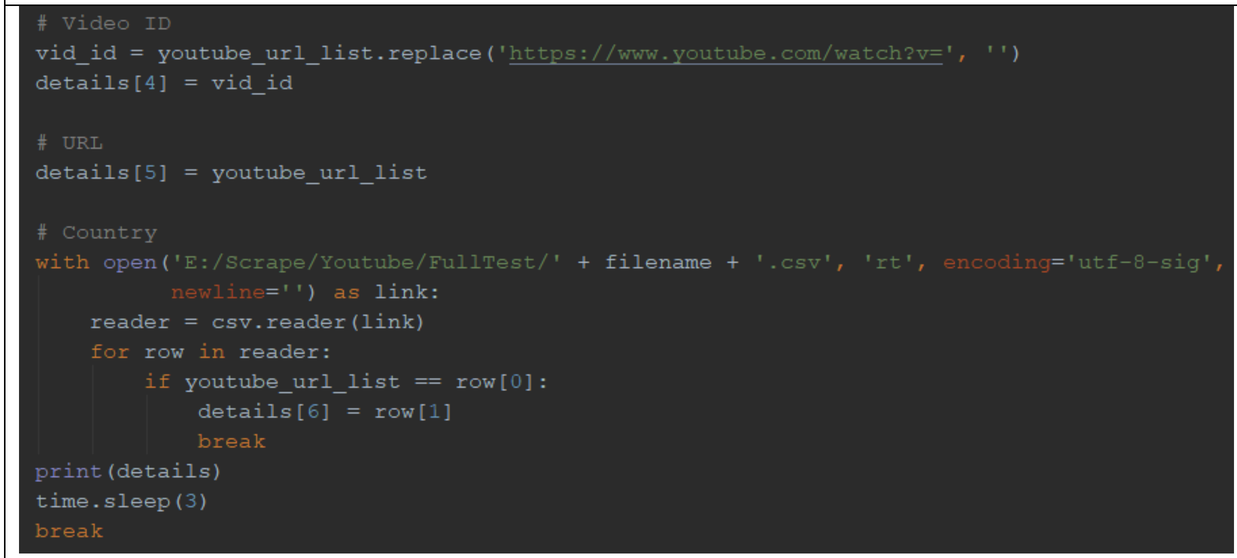
The 'details' array shall be filled with the respective information obtained from the Youtube Video page. The information is extracted by utilizing requests to access the page and Beautiful Soup for html tag location commands. This section shows the location of the tags containing the Channel Name and Video Title. The information is extracted and stored in the respective 'details' location.



```python
# Upload Date
if soup.find('div', {'id': 'watch-uploader-info'}) is not None:
    upload = soup.find('div', {'id': 'watch-uploader-info'}).get_text()
    if re.search(r"[0-9]+.[a-zA-Z]+.[0-9]+", upload) is not None:
        date_text = convertDate(str(re.findall(r"[0-9]+.[a-zA-Z]+.[0-9]+", upload)))
    else:
        date_text = date.today().strftime('%d-%b-%Y')
        print(date_text.replace('-', ' '))
    details[2] = " ".join(date_text.split())
else:
    print('Did not find upload')
if details[2] == '':
    details[2] = 'null'

# Description
if soup.find('div', {'id': 'watch-description-text'}) is not None:
    desc = soup.find('div', {'id': 'watch-description-text'}).get_text()
    details[3] = " ".join(desc.split())
else:
    print('Did not find desc')
if details[3] == '':
    details[3] = 'null'
```

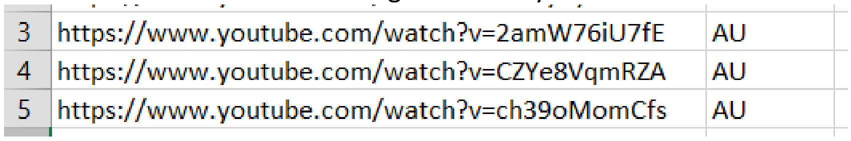
This section shows the location of the tags containing the Upload Date and Description. The information is extracted and stored in the respective 'details' location. Regular expression is used to filter the information from the Upload Date tags as they contain unnecessary words such as

‘Published on’ or ‘Streamed on’. The date formatting on the Youtube pages are consistently in the form of DD-MMM-YY and so utilizing the Regex pattern is suitable for extracting the exact date. Once extracted, the Months need to be translated into English format as the dates extracted shows the months in Malay. Utilizing convertDate() defined method, the month will also be formatted into the English version. (Refer to Section 4.2 for further information regarding convertDate()).

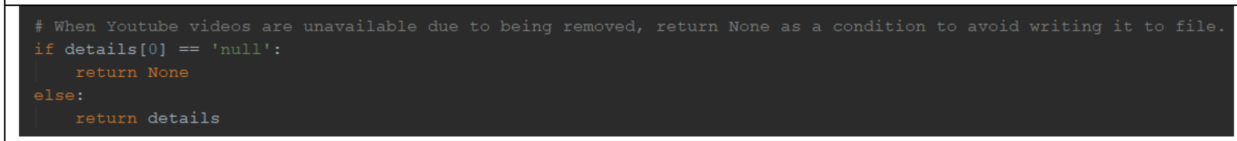


```python
# Video ID
vid_id = youtube_url_list.replace('https://www.youtube.com/watch?v=', '')
details[4] = vid_id

# URL
details[5] = youtube_url_list

# Country
with open('E:/Scrape/Youtube/FullTest/' + filename + '.csv', 'rt', encoding='utf-8-sig',
          newline='') as link:
    reader = csv.reader(link)
    for row in reader:
        if youtube_url_list == row[0]:
            details[6] = row[1]
            break
print(details)
time.sleep(3)
break
```

This section shows the location of the tags containing the Video ID, URL and Country Code. The information is extracted and stored in the respective ‘details’ location. Youtube’s video ID can be obtained at the far end of a youtube video page. For example: ‘https://www.youtube.com/watch?v=7wX41Gqbg-A’ in this link, the video ID is 7wX41Gqbg-A. The URL is just obtained by referencing the link that was used to open the page, youtube_url_list. The country code is obtained from the source file which was written onto the same file as the one written in collect_links(). The file, ‘filename’ lists the country code corresponding to every Youtube link and hence, the loop in the code is based on matching the current page link with the link in the file and extracting the corresponding country code before storing in ‘details’. Refer to the image below for how the ‘filename’ containing the country code and URL looks like:



3	https://www.youtube.com/watch?v=2amW76iU7fE	AU
4	https://www.youtube.com/watch?v=CZYe8VqmRZA	AU
5	https://www.youtube.com/watch?v=ch39oMomCfs	AU



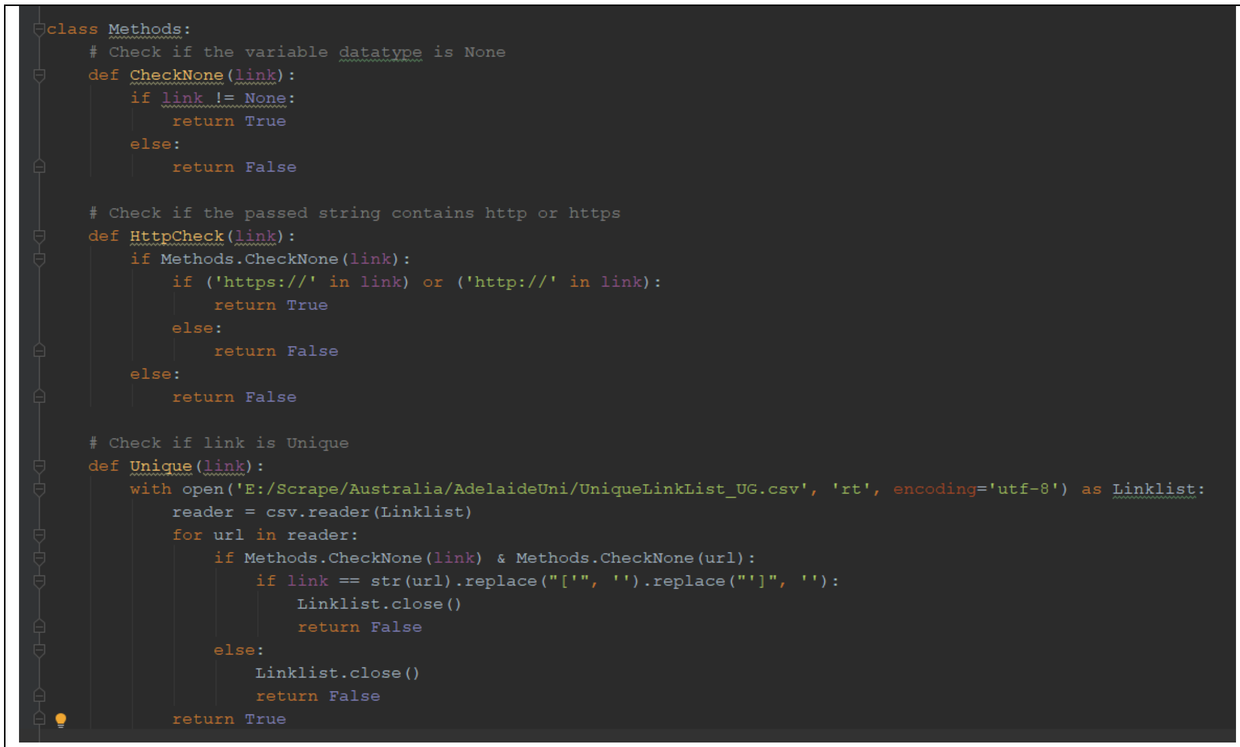
```python
# When Youtube videos are unavailable due to being removed, return None as a condition to avoid writing it to file.
if details[0] == 'null':
    return None
else:
    return details
```

The last section of collect_data() involves checking for broken Youtube video links before returning ‘details’ as an output of this method. When Youtube video links are broken, there is no information that can be obtained except the Channel name which would be listed as YouTube itself. As such, the condition details[0] == ‘null’ can be used to avoid returning broken links and empty info since only Channel name is populated and details[0] refers to the video title which is typically ‘null’.

4. Other Methods

This section will cover the remaining methods used in the code in general. They are mainly used for data validation and formatting data to support the main methods listed in Section 3.

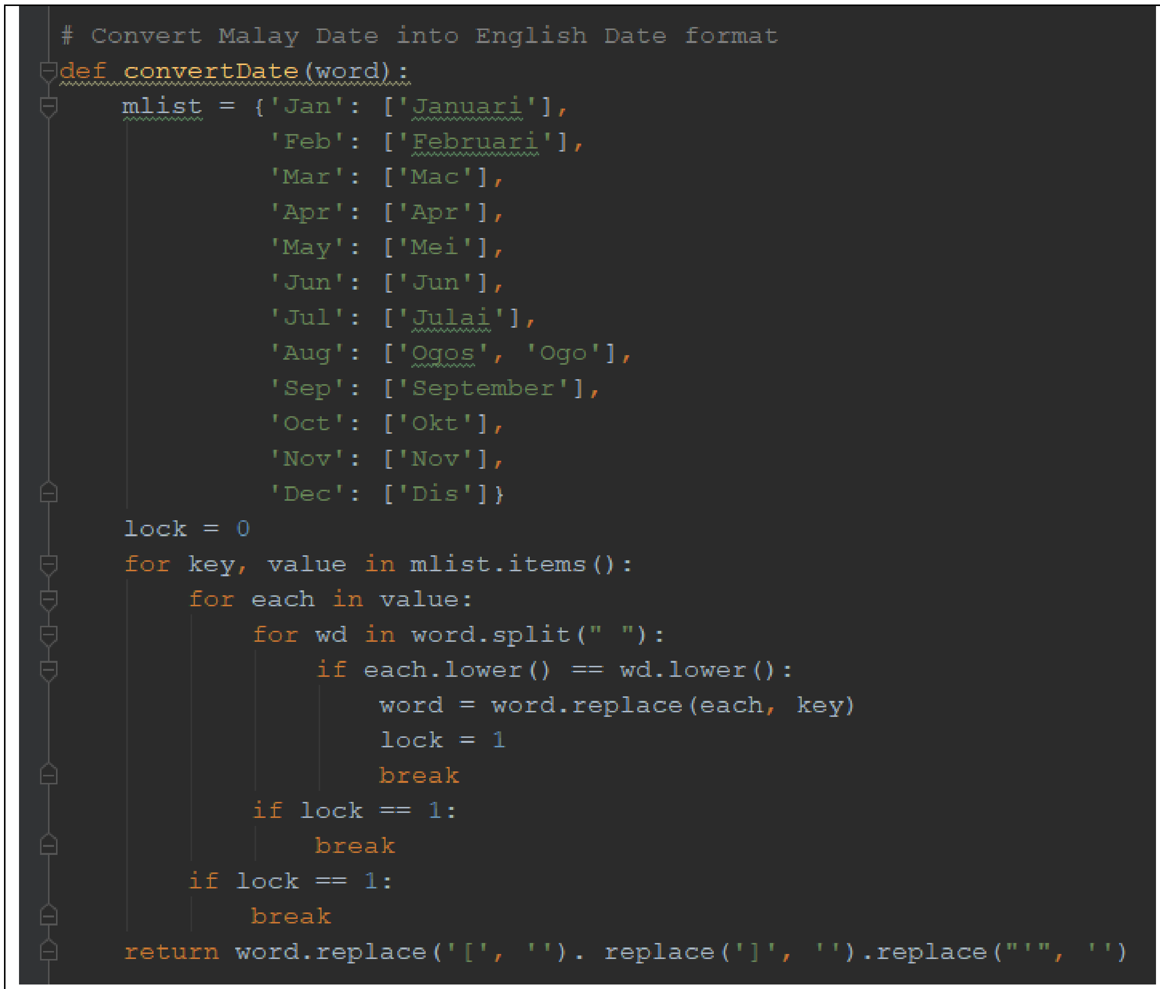
4.1 checkNone(), HttpCheck(), Unique()



```python
class Methods:
    # Check if the variable datatype is None
    def CheckNone(link):
        if link != None:
            return True
        else:
            return False

    # Check if the passed string contains http or https
    def HttpCheck(link):
        if Methods.CheckNone(link):
            if ('https://' in link) or ('http://' in link):
                return True
            else:
                return False
        else:
            return False

    # Check if link is Unique
    def Unique(link):
        with open('E:/Scrape/Australia/AdelaideUni/UniqueLinkList_UG.csv', 'rt', encoding='utf-8') as Linklist:
            reader = csv.reader(Linklist)
            for url in reader:
                if Methods.CheckNone(link) & Methods.CheckNone(url):
                    if link == str(url).replace("['", '').replace("']", ''):
                        Linklist.close()
                        return False
                else:
                    Linklist.close()
                    return False
            return True
```

These 3 methods are used mainly for collect_links() method. CheckNone() is to check for empty links. HttpCheck() will determine if https or http is in the url link. These two methods will then be used with Unique() method to write the links inside the YoutubeVideoLinks.csv file.

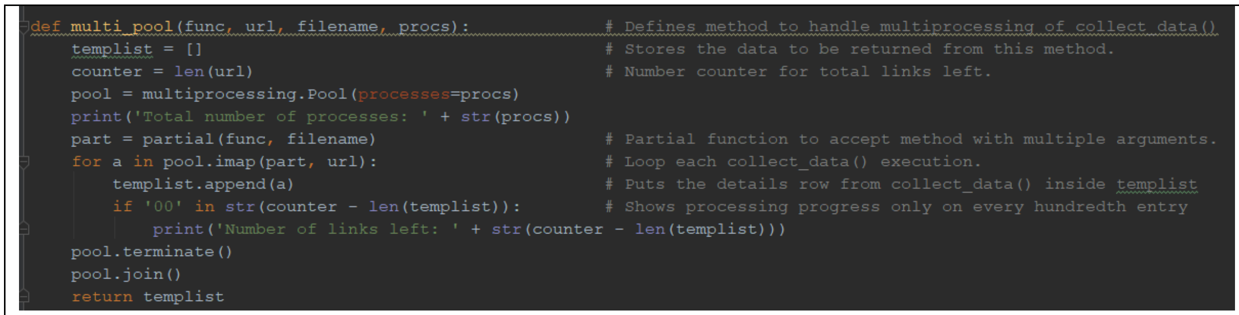
4.2 convertDate()



```
# Convert Malay Date into English Date format
def convertDate(word):
    mlist = {'Jan': ['Januari'],
             'Feb': ['Februari'],
             'Mar': ['Mac'],
             'Apr': ['Apr'],
             'May': ['Mei'],
             'Jun': ['Jun'],
             'Jul': ['Julai'],
             'Aug': ['Ogos', 'Ogo'],
             'Sep': ['September'],
             'Oct': ['Okt'],
             'Nov': ['Nov'],
             'Dec': ['Dis']}
    lock = 0
    for key, value in mlist.items():
        for each in value:
            for wd in word.split(" "):
                if each.lower() == wd.lower():
                    word = word.replace(each, key)
                    lock = 1
                    break
            if lock == 1:
                break
        if lock == 1:
            break
    return word.replace('[', ''). replace(']', '').replace("'", '')
```

This method is used to change the month name from Malay into English. It takes one argument ‘word’ which should be the date containing the Malay format month that needs to be converted into an English format month. Extracting the data from Youtube’s page from Malaysia defaults the date into Malay language. By creating a dictionary of keywords containing the months in Malay, the months could be converted into English and returned as an output.

4.3 multi_pool()



```
def multi_pool(func, url, filename, procs):           # Defines method to handle multiprocessing of collect_data()
    templist = []                                     # Stores the data to be returned from this method.
    counter = len(url)                                # Number counter for total links left.
    pool = multiprocessing.Pool(processes=procs)
    print('Total number of processes: ' + str(procs))
    part = partial(func, filename)                    # Partial function to accept method with multiple arguments.
    for a in pool.imap(part, url):                    # Loop each collect_data() execution.
        templist.append(a)                            # Puts the details row from collect_data() inside templist
        if '00' in str(counter - len(templist)):      # Shows processing progress only on every hundredth entry
            print('Number of links left: ' + str(counter - len(templist)))
    pool.terminate()
    pool.join()
    return templist
```

This method defines the multiple processing of a function. In this code the function is collect_data(). collect_data() is focused on extracting information from a single Youtube Page and hence multi_pool() will allow multiple collect_data() processes run in parallel to extract the data at a much

higher rate. This method utilizes Pool from multiprocessing and partial modules. The goal of this method is to store all ‘details’ from multiple processing of collect_data() into ‘templist’ which will be returned as an output from this method. This method takes four arguments: ‘func’ for the method to be multiprocessed, url’ for the list of youtube video links, ‘filename’ for the file containing all youtube video links and country code and finaly ‘procs’ which is the number of processes to be ran in parallel. First partial is used to allow pool.imap to take in multiple arguments apart from ‘url’. In this case, ‘filename’ is the 2nd argument to be taken before running it inside the for loop for pool.imap. In each iteration of the loop, collect_data() is processed in parallel according to iterable ‘url’ and each iteration, the ‘details’ will be collected and appended to ‘templist’.
The ‘if’ statement is just to check the progress of the links in the Run terminal in Pycharm and was set to display the remaining links in hundreds. Once completed, pool.terminate() and pool.join() is used to end the parallel processing and collect all results respectively before returning templist as a result.