# Gab.com Web Scraper Overview

This Python script scrapes information from a Gab user profile and saves the data to a CSV file. The script uses the selenium package to load the Gab profile and scroll down to the bottom of the page, the BeautifulSoup package to parse the HTML of the loaded webpage, and the pandas package to save the extracted data into a CSV file.

## Known bugs:

1. At it's current state, some posts being extracted are skipped. Possible issues are that some posts do not have text in them and are simple reposts from other users.
2. There are instances where the output would mismatch their associated engagement metrics, due to some line skipping.
3. Even if there are an x number of posts found, not all of them are extracted again due to the issues mentioned above.

## Importing packages needed for this web scraper

The script imports the following Python libraries:

1. **selenium** to use it's *webdriver* for scrolling to the bottom of the page, so we can load all posts from the user.
2. **BeautifulSoup** for parsing through our saved html files and extracting the information we will be needing.
3. **pandas** for converting our extracted data into a DataFrame, and outputting into a csv file.
4. **time** is for using it's *sleep function* to add needed delays in our python script.

In [1]:
```
from selenium import webdriver #To load all possible posts
from bs4 import BeautifulSoup #To scrape information
import pandas as pd
import time
```

## Here we initialize our automated instance of Chrome

In [ ]:
```
driver = webdriver.Chrome(executable_path="chromedriver_win32/chromedriver")
```

## Opening the Gab.com User Profile

In this section, we first create the variable `gab_url` which asks for the user's input. For this, we will be needing the url of the desired Gab User.

- Example: https://www.gab.com/a - This is the url of Gab's CEO

The user's input is then stored and used for:

- *webdriver* knowing which profile to open and save.
- A variable called `filename` which will be used for difference usecases. e.g. HTML file names, CSV file names, foreign key for merging each post to the user's information, etc.

This website is then opened by the automated Chrome instance, scrolled twice to make sure all posts are loaded, then saved as an html file with the profile's username as it's file name.

In [3]:
```
# For user input of desired profile for web scraping
gab_url = input()

# Will be for different usecases
filename = gab_url.split(".com/")[1]

# Loading the gab_url input of user, scrolling down to max scrollable post
driver.get(gab_url)
driver.execute_script("window.scrollTo(1,10000)")
driver.execute_script("window.scrollTo(1,10000)")

# To give time for webpage to fully load
time.sleep(20)

# Saving the loaded website into an html file for scraping
html = driver.page_source
with open("gab/{}.html".format(filename), "w+", encoding="utf-8") as f:
    f.write(html)
print("Gab User", str(filename)+".html saved!", end="\n")

driver.quit()
```

```
Gab User a.html saved!
```

## Using BeautifulSoup

Here's the fun part!

Now that we have our HTML file, we will now use **BeautifulSoup** to parse that HTML file and extract the information that we need.

For this project, I opted to extract the following:

1. **Date Joined** - The time user joined/registered.
2. **Username** - The profile's username.
3. **Profile Photo** - The user's profile photo which will be in the form of a url.
4. **Cover Photo** - The user's cover photo which also will be in the form of a url.
5. **About** - The user's bio/about section in their profile.
6. **Gabs** - This pertains to the number of posts they have. Can be likened to Twitter's Tweets.
7. **Followers** - How many users follow the profile we chose.
8. **Following** - How many users the profile we chose follows.
9. **Posts** - Their posts, extracting the text contained within.
10. **Post Media** - Media associated to their post, in the form of a url.
11. **Reactions** - The number of reactions their post made.
12. **Replies** - How many comments were made on their post.
13. **Reposts** - How many reposts the post had.
14. **Quotes** - How many posts were made where the post was quoted.

## Exception Handling

The script includes exception handling for cases where the profile, for example, does not have a cover photo, bio, and other use cases mentioned in the script. If the profile does not have them, the script assigns the value 'None' to the variables. This is also to avoid throwing an error wherein the specified `class ID` in our script is not found.

In [4]:
```
# Creating lists for each field
fields = ['Date Joined', 'Username', 'User Image', 'Cover Photo', 'About', 'Gabs', 'Followers', 'Following']
date_joined = []
user_name = []
user_image = []
cover_photo = []
about = []
number_of_gabs = []
number_of_followers = []
number_of_following = []

# Now, we open the html file we just saved for parsing using BeautifulSoup package
with open("gab/{}.html".format(filename), encoding="utf-8") as f:
        page = f.read()

soup = BeautifulSoup(page, "html.parser")

# Crosschecking variable for later
NoneType = soup.find("img", class_="xxxxxxxxxx")

# Extracts when the member joined
joined = soup.find("span", class_="_33mR1 _uuSG _3_54N a8-QN _2cSLK L4pn5 RiX17").text.split("since ")[1]

# Extracts the username of the member
name = soup.find("span", class_="_3_54N _3rXHO _317eq _2xqBt _2z1u_ a8-QN L4pn5 grnpY _3tKl_").text

# Extracts the Profile Picture url
# The if statement was included since there were different values in my experience
userimage = soup.find("img", width="88")
if type(userimage) == type(NoneType):
    userimage = soup.find("img", width="150").attrs["src"]
else:
    userimage = soup.find("img", width="88").attrs["src"]

# Extracts the Cover Photo url
# The if statement was included to not throw an error on profiles with no cover photo
coverphoto = soup.find("img", alt="Header photo", title="Header photo")
if type(coverphoto) == type(NoneType):
    coverphoto = 'None'
else:
    coverphoto = soup.find("img", alt="Header photo", title="Header photo").attrs["src"]

# Extracts the "About" section of the profile
bio = soup.find("div", class_="_0utbn")
if bio == NoneType:
    bio = 'None'
else:
    bio = bio.text

# Extracts the "Gabs" of the profile
gabs = int(soup.find("a", class_="_uuSG ALevz _3Ujf8 _1o5Ge _81_1w _3dGg1 _2mtbj _1AbQQ active").attrs["title"].split(" Gabs")[0].replace(",",""))

# Extracts the number_of_followers
followers = int(soup.find("a", class_="_uuSG ALevz _3Ujf8 _1o5Ge _81_1w _3dGg1 _2mtbj _1AbQQ").attrs["title"].split(" Followers")[0].replace(",",""))

# Extracts the number of users the profile follows
following = int(soup.find("a", class_="_uuSG ALevz _3Ujf8 _1o5Ge _81_1w _3dGg1 _2mtbj _1AbQQ", href="/{}/following".format(filename)).attrs["title"].split(" Following")[0].replace(",",""))

# Extracted information are then appended to the lists created prior for later merging
date_joined.append(joined)
user_name.append(name)
user_image.append(userimage)
cover_photo.append(coverphoto)
about.append(bio)
number_of_gabs.append(gabs)
number_of_followers.append(followers)
number_of_following.append(following)
```

In [5]:
```
# All available posts are now extracted
posts_soup = soup.find_all("div", tabindex="0", lang="en")
posts = []
userpost = []

# Each post is iterated and is put into a list for later merging
for post in posts_soup:
    posts.append(post.text)
    userpost.append("@"+filename)
```

In [6]:
```
# All available imageurls are now extracted from each post
imageposts = soup.find_all("img", loading="lazy")
media = []

# Each imageurl is iterated and is pute into a list for later merging
for link in imageposts:
    media.append(link.attrs["src"])
```

In [7]:
```
# Now, we create the "epp" variable for iterating the lists created
epp = soup.find_all("div", class_="_uuSG _3dGg1 _2VJFi SslQJ _2pVfg")
reactions = []
replies = []
reposts = []
quotes = []
```

In [8]:
```
# The following for loops iterate on "epp" to get the number of reactions, replies, reposts, and quotes for each post.
# If else statements are included to take into consideration that some of them may be in the thousands, where Gab summarizes them.
# Also, getting the number of "quotes" can be tricky since replies and quotes have the same class id.
# That's where multiple if statements were used to prevent capturing the wrong data

for engagement in epp:
    reaction = engagement.find("span", class_="_3u7ZG _uuSG _3_54N a8-QN _2cSLK L4pn5 RiX17")
    if reaction == NoneType:
        reaction = int(0)
    else:
        reaction = reaction.text.split("k")[0]
        if "." in reaction:
            reaction = int(float(reaction) * 1000)
        else:
            reaction = int(reaction)
    reactions.append(reaction)

for engagement in epp:
    reply = engagement.find("span", class_="_uuSG _3_54N a8-QN _2cSLK L4pn5 RiX17")
    if reply == NoneType:
        reply = int(0)
    else:
        reply = reply.text.split(" repl")[0]
        if "repost" in reply:
            reply = int(0)
        else:
            reply = reply.split("k")[0]
            if "." in reply:
                reply = int(float(reply) * 1000)
            else:
                reply = int(reply)
    replies.append(reply)

for engagement in epp:
    repost = engagement.find("button", class_="_uuSG _3_54N L4pn5 _3Ujf8 _1o5Ge _81_1w _3TwRa _3OtSI ALevz AOUHB")
    if repost == NoneType:
        repost = int(0)
    else:
        repost = repost.text.split(" repost")[0].split("k")[0]
        if "." in repost:
            repost = int(float(repost) * 1000)
        else:
            repost = int(repost)
    reposts.append(repost)

for engagement in epp:
    quote = engagement.find("button", class_="_uuSG _3_54N L4pn5 _3Ujf8 _1o5Ge _81_1w _3TwRa _3OtSI ALevz AOUHB")
    if quote == NoneType:
        quote = int(0)
    elif quote.find_next_sibling("button") == NoneType:
        quote = int(0)
    else:
        quote = quote.find_next_sibling("button").text.split(" quote")[0].split("k")[0]
        if "." in quote:
            quote = int(float(quote) * 1000)
        else:
            quote = int(quote)
    quotes.append(quote)
```

## Saving the Extracted Data using pandas

The extracted data is saved to a CSV file using the **pandas** package. The **pd.DataFrame()** function is used to create a DataFrame from the lists containing the extracted data. The DataFrame is then saved to a CSV file using the **to_csv()** function with the **index=False** argument to exclude the index column from the CSV file.

In addition to our extracted data, we also added:

1. **Total Engagement** - This is the total of adding each post's Reactions, Replies, Reposts, and Quotes.
2. **Avg Engagement** - This uses `Total Engagement` and divides the number by 4 to get this metric.
3. **Avg Post Engagement** - This metric is calculated by getting the total of all post engagements and dividing it by the number of posts we have extracted.

In [9]:
```
# pandas DataFrames are now created for both user information and user posts information.
# The Total Engagement column and Avg Engagement Column were also added based on the merge information for posts.
# An Avg Post Engagement column was also added in the user information to show the average engagement per post the user gets

dfposts = pd.DataFrame(data = zip(userpost, posts, media, reactions, replies, reposts, quotes), columns = ['Username', 'Post', 'Media', 'Reactions', 'Replies', 'Reposts', 'Quotes'])
dfposts['Total Engagement'] = dfposts.Reactions + dfposts.Replies + dfposts.Reposts + dfposts.Quotes
dfposts['Avg Engagement'] = dfposts['Total Engagement']/4
dfposts['Avg Engagement'].round(2)

dfuser = pd.DataFrame(data = zip(date_joined, user_name, user_image, cover_photo, about, number_of_gabs, number_of_followers, number_of_following), columns = fields)
dfuser['Avg Post Engagement'] = dfposts['Total Engagement'].sum()/len(posts)
```

## Now let's look at our extracted data!

In [10]:
```
dfposts.head()
```

Out[10]:

| | Username | Post | Media | Reactions | Replies | Reposts | Quotes | Total Engagement | Avg Engagement |
|---|---|---|---|---|---|---|---|---|---|
| 0 | @a | "In a September 2022 audit of the seven sites ... | https://media.gab.com/cdn-cgi/image/width=410,... | 1200 | 180 | 255 | 19 | 1654 | 413.50 |
| 1 | @a | Gab is a First Amendment company which means w... | https://media.gab.com/cdn-cgi/image/width=420,... | 4 | 439 | 1100 | 47 | 1590 | 397.50 |
| 2 | @a | Just some of my Original, Miniature, wearable ... | https://media.gab.com/cdn-cgi/image/width=420,... | 17100 | 3400 | 4300 | 199 | 24999 | 6249.75 |
| 3 | @a | Proverbs 22:6 Train up a child in the way he s... | https://media.gab.com/system/media_attachments... | 45 | 6 | 9 | 0 | 60 | 15.00 |
| 4 | @a | Serenity of Stillness:... | https://media.gab.com/cdn-cgi/image/width=420,... | 80 | 4 | 7 | 0 | 91 | 22.75 |

In [11]:
```
dfuser.head()
```

Out[11]:

| | Date Joined | Username | User Image | Cover Photo | About | Gabs | Followers | Following | Avg Post Engagement |
|---|---|---|---|---|---|---|---|---|---|
| 0 | August 2016 | @a | https://media.gab.com/system/accounts/avatars/... | https://media.gab.com/system/accounts/headers/... | Saved servant soldier of Jesus Christ. Husband... | 69270 | 3878881 | 2745 | 1962.625 |

## Finally, let's save all of that into one csv file.

In [12]:
```
# The dataframes are then merged and saved into one csv file for further use and/or analysis.

df = pd.merge(dfuser, dfposts, on='Username').to_csv("gab/{}.csv".format(filename), index=False)
```