1. Preparation for the API:
   1.1. Install the Google client library with preferred installer program (pip) in terminal
   1.2. Set up API project in Google Cloud Console, then get the credential file in json format
   1.3. Configure sample: use the code provide by google to quick start (in quickstart.py) the authorization process of my gmail account so that a token (token.json) can be obtain to extract information later
   1.4. Run the quickstart.py
2. *get_service()* Function:
   - refined version of quickstart.py, but put into a function
   - A google provided function to access the data in a certain Gmail address
   - https://developers.google.com/gmail/api/quickstart/python
   - Libraries:

```python
import os.path

from google.auth.transport.requests import Request
from google.oauth2.credentials import Credentials
from google_auth_oauthlib.flow import InstalledAppFlow
from googleapiclient.discovery import build
from googleapiclient.errors import HttpError
```

← Google - provided

```python
def get_service():

    creds = None
    # The file token.json stores the user's access and refresh tokens, and is
    # created automatically when the authorization flow completes for the first
    # time.
    if os.path.exists('token.json'):
        creds = Credentials.from_authorized_user_file('token.json', SCOPES)
    # If there are no (valid) credentials available, let the user log in.
    if not creds or not creds.valid:
        if creds and creds.expired and creds.refresh_token:
            creds.refresh(Request())
        else:
            flow = InstalledAppFlow.from_client_secrets_file(
                'credentials.json', SCOPES)
            creds = flow.run_local_server(port=0)
        # Save the credentials for the next run
        with open('token.json', 'w') as token:
            token.write(creds.to_json())

        # Call the Gmail API
    service = build('gmail', 'v1', credentials=creds)
    results = service.users().labels().list(userId='me').execute()
    labels = results.get('labels', [])

    return service
```

- token.js get created automatically
  ↳ there are total of used json file:
  + token.js
  + credentials.js

- 'v1' is one of the many service provide by google API

   2.1. Brief understanding of the syntax:
   - From the summary, to use a syntax, one need to start with 'v1', which is declared in the variable 'service' as seen.
   - Add the *'.execute()'* at the end of a function to get the return value
   - Check the 'JSON representation' in the documentation to see the expected return and its value

3. *get_id(service, user_id, search_string)* function
   3.1. The *try:* and *except…:* method: it is recommended by the Google whence there is an error of not getting request to the http. So, whatever intended code, it's put into the *try:* block.

```python
def get_id (service, user_id, search_string):
    service = get_service()
    try:
        search_id=service.users().messages().list(userId=user_id, q=search_string).execute()
        number_result = search_id["resultSizeEstimate"] ## a returned element from the dictionary of user().list()
        msg_id_list = []
        if number_result > 0:
            messages_id = search_id["messages"] ## access the "messages"
            for ids in messages_id:
                msg_id_list.append(ids['id'])

        #else:
            #print ("There was no result for that search, returning an empty string")
            #return ''

    except HttpError as error:
        print ('An error occured: {}'.format(error))

    #print (msg_id_list)
    return msg_id_list
```

- list all the msg ID and the number of msg
- print (search_id)

*Link to "id checking.docx": show "search_id"

   3.2. This function returns a list of message IDs.

4. *get_msg(service, user_id, msg_id)*
   4.1. Function that take in message id (*msg_id*) and return a list of the sender, subject and truncated ID to differentiate subject duplication.

```python
def get_msg (service, user_id, msg_id):
    service = get_service()
    try:
        message_encode=service.users().messages().get(userId=user_id, id = msg_id).execute() ##get the encoeded message
        trunc_id = '***'+msg_id[12:]
        for header in reversed(message_encode["payload"]["headers"]):
            if header['name'] == 'Subject':
                o_subject = header['value']
            if header['name'] == 'From':
                o_sender = header['value']

    except HttpError as error:
        print ('An error occured: {}'.format(error))
    return [o_sender, o_subject, trunc_id]
```

- Id_checking.docx
- Link of dictionary
- Filter through the dict to get variable info

Go through message_encode["payload"]["headers"] and search for specific dictionary's key

   4.2. Check in reverse because the information is somewhat at the end, so the finding is more efficient.

5. *generate_list(searching)*

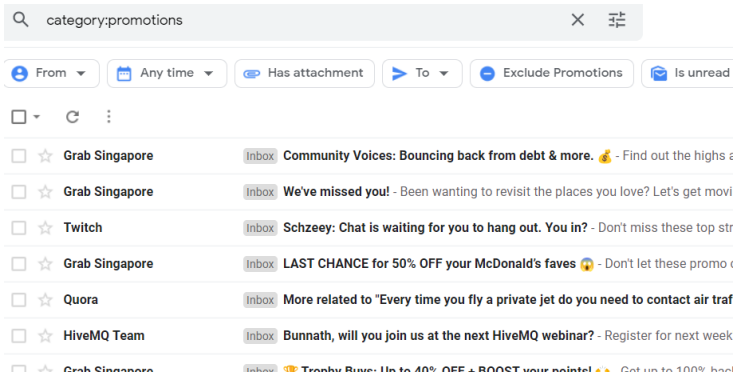
```python
def generate_list(searching):
    service = get_service()
    i_msg_id_list = get_id (service, user_id, searching)
    msg_info_list = []
    for i_msg_id in i_msg_id_list:
    #i_msg_id = "18160e83d24ddc25"
        msg_info_list.append(get_msg (service, user_id, i_msg_id))

    final_msg_list = []
    if searching == "in:sent":
        final_msg_list = msg_info_list
    else:
        for j in range (len(msg_info_list)):
            if 'Re: ' not in msg_info_list[j][1]:
                final_msg_list.append (msg_info_list[j])

    return final_msg_list
```

- Normal python function
- list
- search engine
- Remove replies cuz this app only summary the main msg

5.1. putting all the returned value of *get_msg(…)* of each inbox in the email in to 1 single nested list where each element is a list.

5.2. Ignore all the reply, since they are sent by us and the program only to summary the email inbox

5.3. But for the category “sent”, it’s all a reply, so user can check in that way with a *searching* keyword of *“in:sent”* (it’s a gmail searching talgorithm)



6. generate_sheet (searching)

6.1. use the nested list in generate_list to put into an excel spreadsheet

6.2. library needed: openpyxl

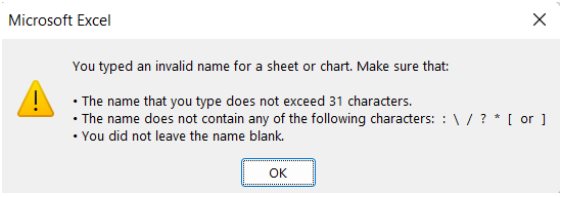
6.3. set up:

```python
from openpyxl import Workbook, load_workbook
from openpyxl.utils import get_column_letter
from openpyxl.styles import Alignment

workbook = load_workbook ('Excel\Email Data.xlsx')

for k in workbook.sheetnames:
    del_sheet = workbook[k]
    workbook.remove(del_sheet)
###################
```

Remove every sheet first

```python
def generate_sheet (searching):
    the_list = generate_list(searching)
    searching = ":"+searching
    indi, key_word = searching.rsplit(':', 1)

    if key_word == '':
        key_word = "BLANK_named"
    elif len(key_word) > 31:
        key_word = key_word[:28]+"..."
    elif key_word.isalnum() == 0:
        for char in key_word:
            if char in ":\/?*[]":
                key_word = key_word.replace (char," ")
    else:
        key_word = key_word.capitalize()

    sheet = workbook.create_sheet(key_word)

    headings = ["From", "Subject", "ID (Technical)"]
    heading_cell = ['A1', 'B1', 'C1']
    sheet.append (headings)
    for email_data in the_list:
        sheet.append(email_data)
```


6.4. putting the returned nested list of generate_list(searching) into *"the_list"*. Then clean the *"searching"* so that it can be use as a name of the sheet in excel with validation checking base on this:



6.5. append the elements in the list into each cell, and it ends up looking somewhat like this:

From	Subject	ID (Technical)
PayScale <	Verify You	***5e74
The Googl	☑ Bunna	***3158
Google <n	Security al	***a944
3R4_2022	another te	***dc25
3R4_2022	test	***4940
Bunnath Ti	test	***6122

6.6. Resize in into proper scaling.

```
for cell in heading_cell:
    sheet[cell].alignment = Alignment(horizontal='center')

sheet.column_dimensions['A'].width = 40
sheet.column_dimensions['B'].width = 80
sheet.column_dimensions['C'].width = 13
for row in sheet.iter_rows():
    for cell in row:
        cell.alignment = Alignment(wrap_text=True)



workbook.save ('Excel\Email Data.xlsx')
workbook.close()
```