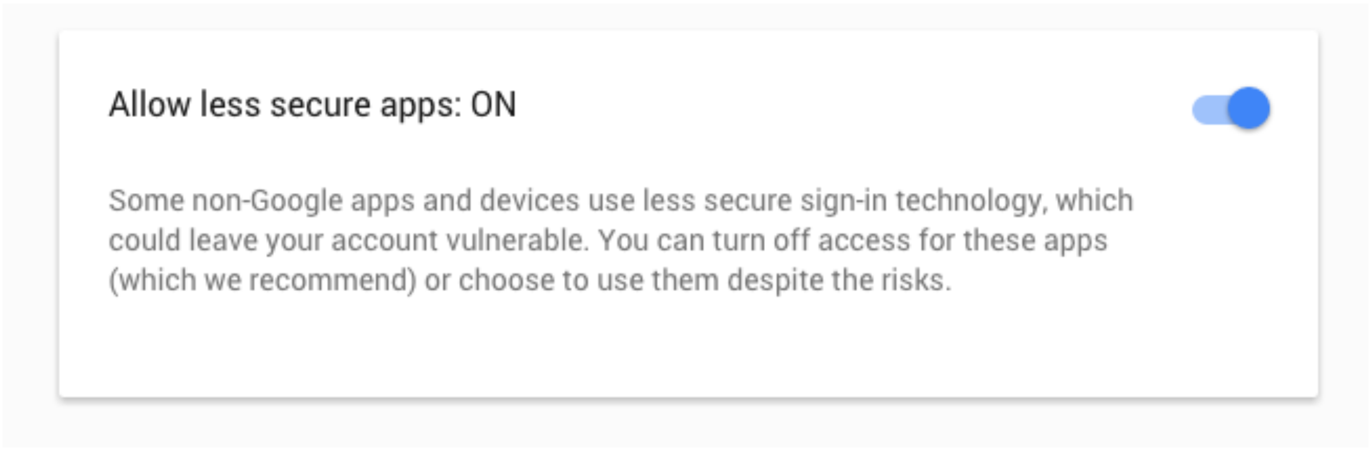
Sat 24 September 2016, by Seppe "Macuyiko" vanden Broucke

How-to: Send HTML Mails with OAuth2 and Gmail in Python (//blog.macuyiko.com/post/2016/how-to-send-html-mails-with-oauth2-and-gmail-in-python.html)

If you've been trying to send e-mails from a program using your Gmail account, you'll probably have encountered the following error message:

```
SMTP Error: Could not authenticate. Error: SMTP Error: Could not authenticate.
535 5.7.1 Username and Password not accepted. Learn more at
535 5.7.1 http://mail.google.com/support/bin/answer.py?answer=14257 p38sm2467302ybk.16
```

In fact, the internet is flooded with people searching to figure out how to get their Wordpress mailer plugin working with Gmail's SMTP. The answer most people give is to just enable the "Allow less secure apps"-option in your Google account settings:


Allow less secure apps: ON

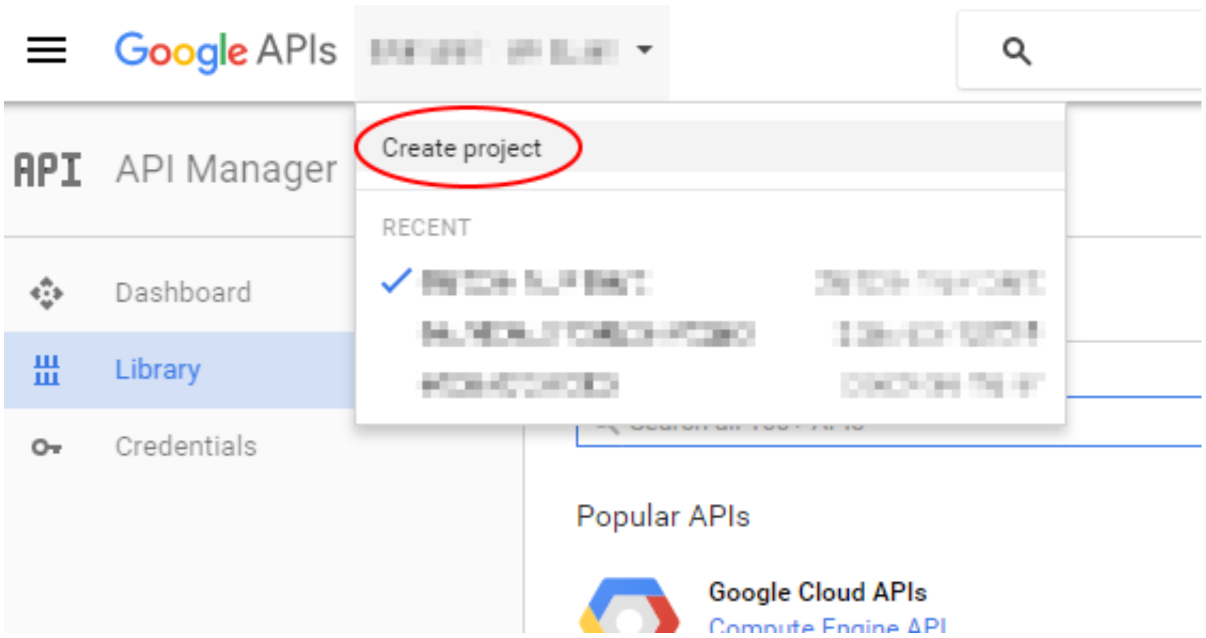
Some non-Google apps and devices use less secure sign-in technology, which could leave your account vulnerable. You can turn off access for these apps (which we recommend) or choose to use them despite the risks.

This doesn't feel right

Fair enough, though a more secure way is to use OAuth2 in combination with Google's API console. This page (https://developers.google.com/gmail/xoauth2_protocol) describes the protocol, whereas this GitHub repo (https://github.com/google/gmail-oauth2-tools) provides sample code, including for Python.

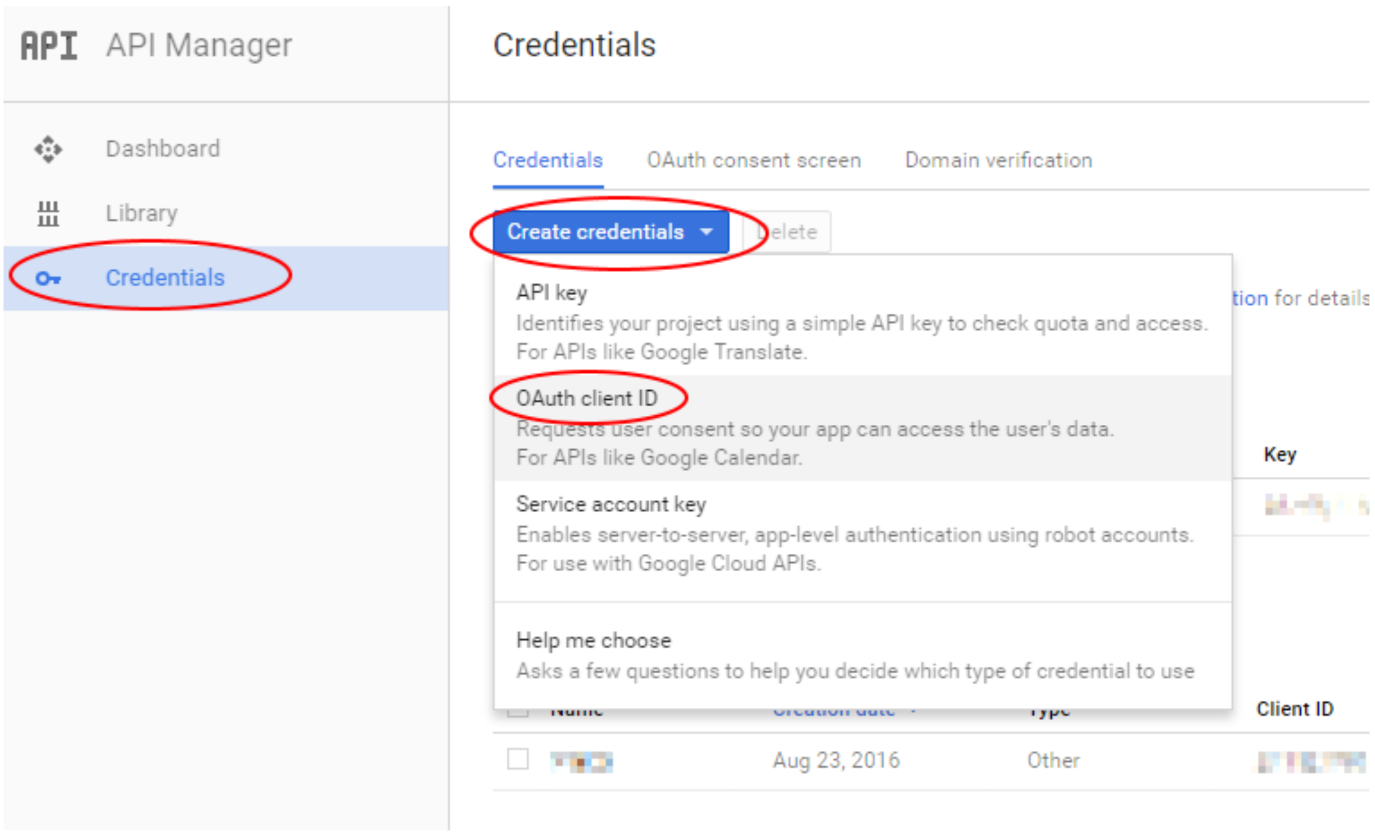
Since the code has been written for Python 2 and I wanted to use 3, I've performed a quick conversion and outline of the steps…

Step 1: navigate to your Google API console (https://console.developers.google.com/). You might want to create a new project (or make sure you have the right project selected):



Create a new project if you want

Step 2: under "Credentials", select "Create credentials" and create a new "OAuth client ID". If you would already have such an ID, you can skip this step and just re-use the one you have.



API Manager — Credentials

- Dashboard
- Library
- Credentials

Credentials | OAuth consent screen | Domain verification

Create credentials ▾ | Delete

API key
Identifies your project using a simple API key to check quota and access.
For APIs like Google Translate.

OAuth client ID
Requests user consent so your app can access the user's data.
For APIs like Google Calendar.

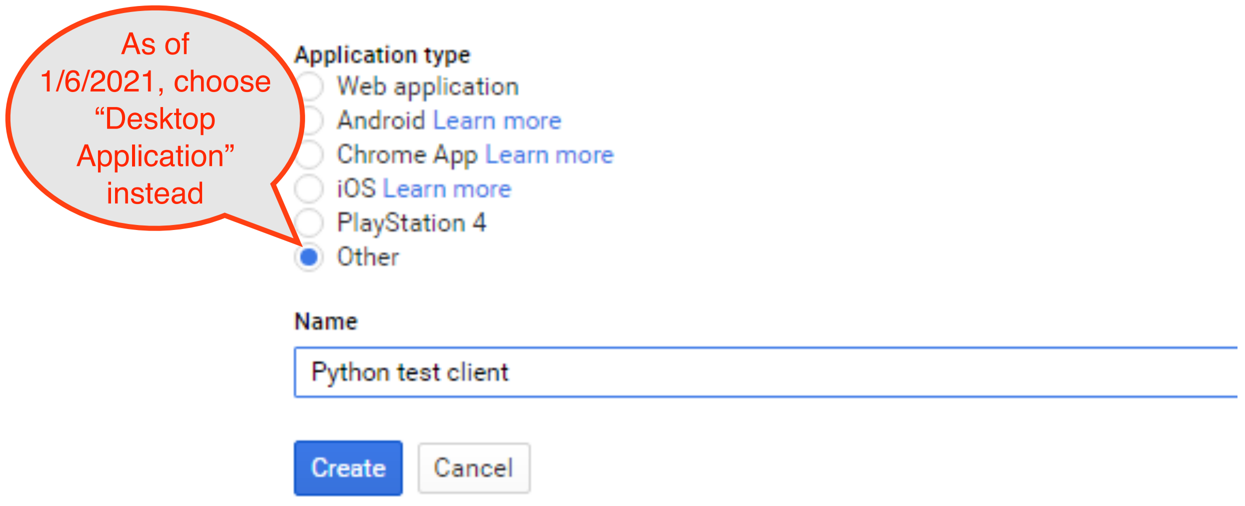
Service account key
Enables server-to-server, app-level authentication using robot accounts.
For use with Google Cloud APIs.

Help me choose
Asks a few questions to help you decide which type of credential to use

Name	Creation date	Type	Client ID
[redacted]	Aug 23, 2016	Other	[redacted]

Create a new OAuth client ID

You will be asked which type of app will use this ID, choose “Other” and give your app an easy-to-remember name:



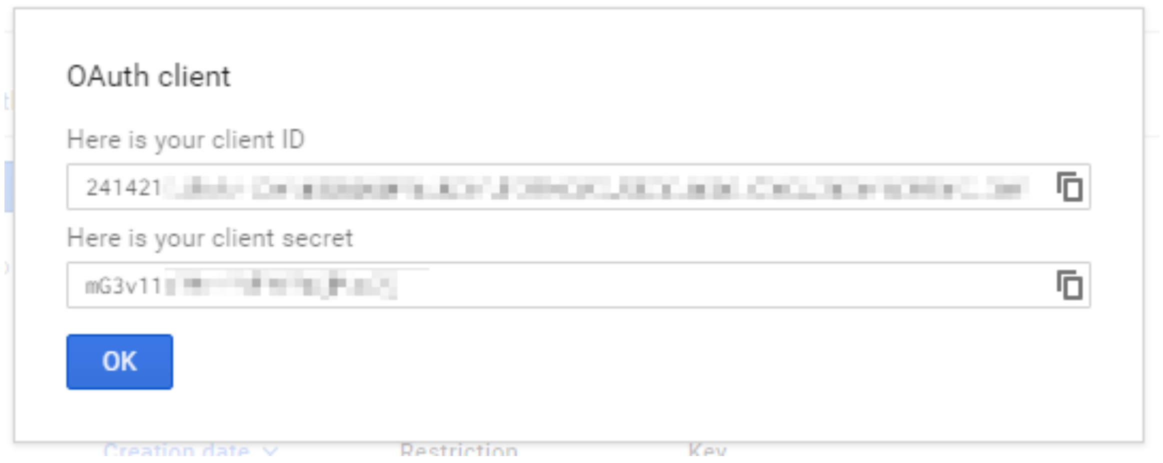
Application type
- ( ) Web application
- ( ) Android Learn more
- ( ) Chrome App Learn more
- ( ) iOS Learn more
- ( ) PlayStation 4
- [✓ Other]

Name: Python test client

Create | Cancel

Select "Other"

Next up comes the most important screen, giving you your client ID and secret key. Write both of these down and do not give these out to others!



OAuth client

Here is your client ID

241421[redacted]

Here is your client secret

mG3v11[redacted]

Keep it secret, keep it safe

Step 3: the Python code we'll be using is adapted from (https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py) but works in Python 3 and has been made simpler:

```
"""
Adapted from:
https://github.com/google/gmail-oauth2-tools/blob/master/python/oauth2.py
https://developers.google.com/identity/protocols/OAuth2

1. Generate and authorize an OAuth2 (generate_oauth2_token)
2. Generate a new access tokens using a refresh token(refresh_token)
3. Generate an OAuth2 string to use for login (access_token)
"""

import base64
import imaplib
import json
import smtplib
import urllib.parse
import urllib.request
from email.mime.multipart import MIMEMultipart
from email.mime.text import MIMEText
import lxml.html

GOOGLE_ACCOUNTS_BASE_URL = 'https://accounts.google.com'
REDIRECT_URI = 'urn:ietf:wg:oauth:2.0:oob'

GOOGLE_CLIENT_ID = '<FILL ME IN>'
GOOGLE_CLIENT_SECRET = '<FILL ME IN>'
GOOGLE_REFRESH_TOKEN = None


def command_to_url(command):
    return '%s/%s' % (GOOGLE_ACCOUNTS_BASE_URL, command)


def url_escape(text):
    return urllib.parse.quote(text, safe='~-._')


def url_unescape(text):
    return urllib.parse.unquote(text)


def url_format_params(params):
    param_fragments = []
    for param in sorted(params.items(), key=lambda x: x[0]):
        param_fragments.append('%s=%s' % (param[0], url_escape(param[1])))
    return '&'.join(param_fragments)


def generate_permission_url(client_id, scope='https://mail.google.com/'):
    params = {}
    params['client_id'] = client_id
    params['redirect_uri'] = REDIRECT_URI
    params['scope'] = scope
    params['response_type'] = 'code'
    return '%s?%s' % (command_to_url('o/oauth2/auth'), url_format_params(params))


def call_authorize_tokens(client_id, client_secret, authorization_code):
    params = {}
    params['client_id'] = client_id
    params['client_secret'] = client_secret
    params['code'] = authorization_code
    params['redirect_uri'] = REDIRECT_URI
    params['grant_type'] = 'authorization_code'
    request_url = command_to_url('o/oauth2/token')
    response = urllib.request.urlopen(request_url, urllib.parse.urlencode(params).encode('UTF-8')).read
().decode('UTF-8')
    return json.loads(response)


def call_refresh_token(client_id, client_secret, refresh_token):
    params = {}
    params['client_id'] = client_id
    params['client_secret'] = client_secret
    params['refresh_token'] = refresh_token
    params['grant_type'] = 'refresh_token'
    request_url = command_to_url('o/oauth2/token')
    response = urllib.request.urlopen(request_url, urllib.parse.urlencode(params).encode('UTF-8')).read
().decode('UTF-8')
    return json.loads(response)


def generate_oauth2_string(username, access_token, as_base64=False):
    auth_string = 'user=%s\1auth=Bearer %s\1\1' % (username, access_token)
    if as_base64:
        auth_string = base64.b64encode(auth_string.encode('ascii')).decode('ascii')
    return auth_string


def test_imap(user, auth_string):
    imap_conn = imaplib.IMAP4_SSL('imap.gmail.com')
    imap_conn.debug = 4
    imap_conn.authenticate('XOAUTH2', lambda x: auth_string)
    imap_conn.select('INBOX')


def test_smpt(user, base64_auth_string):
    smtp_conn = smtplib.SMTP('smtp.gmail.com', 587)
    smtp_conn.set_debuglevel(True)
    smtp_conn.ehlo('test')
    smtp_conn.starttls()
    smtp_conn.docmd('AUTH', 'XOAUTH2 ' + base64_auth_string)


def get_authorization(google_client_id, google_client_secret):
    scope = "https://mail.google.com/"
    print('Navigate to the following URL to auth:', generate_permission_url(google_client_id, scope))
    authorization_code = input('Enter verification code: ')
    response = call_authorize_tokens(google_client_id, google_client_secret, authorization_code)
    return response['refresh_token'], response['access_token'], response['expires_in']


def refresh_authorization(google_client_id, google_client_secret, refresh_token):
    response = call_refresh_token(google_client_id, google_client_secret, refresh_token)
    return response['access_token'], response['expires_in']


def send_mail(fromaddr, toaddr, subject, message):
    access_token, expires_in = refresh_authorization(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SECRET, GOOGLE_REF
RESH_TOKEN)
    auth_string = generate_oauth2_string(fromaddr, access_token, as_base64=True)

    msg = MIMEMultipart('related')
    msg['Subject'] = subject
    msg['From'] = fromaddr
    msg['To'] = toaddr
    msg.preamble = 'This is a multi-part message in MIME format.'
    msg_alternative = MIMEMultipart('alternative')
    msg.attach(msg_alternative)
    part_text = MIMEText(lxml.html.fromstring(message).text_content().encode('utf-8'), 'plain', _charse
t='utf-8')
    part_html = MIMEText(message.encode('utf-8'), 'html', _charset='utf-8')
    msg_alternative.attach(part_text)
    msg_alternative.attach(part_html)
    server = smtplib.SMTP('smtp.gmail.com:587')
    server.ehlo(GOOGLE_CLIENT_ID)
    server.starttls()
    server.docmd('AUTH', 'XOAUTH2 ' + auth_string)
    server.sendmail(fromaddr, toaddr, msg.as_string())
    server.quit()



if __name__ == '__main__':
    if GOOGLE_REFRESH_TOKEN is None:
        print('No refresh token found, obtaining one')
        refresh_token, access_token, expires_in = get_authorization(GOOGLE_CLIENT_ID, GOOGLE_CLIENT_SEC
RET)
        print('Set the following as your GOOGLE_REFRESH_TOKEN:', refresh_token)
        exit()

    send_mail('--------@gmail.com', '--------@gmail.com',
              'A mail from you from Python',
              '<b>A mail from you from Python</b><br><br>' +
              'So happy to hear from you!')
```

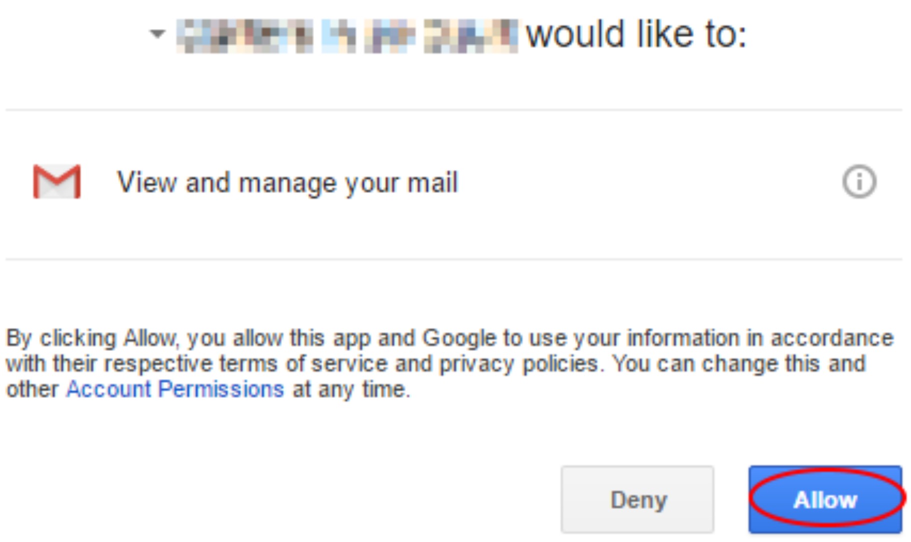
Don't forget to change:

```
GOOGLE_CLIENT_ID = '<FILL ME IN>'
GOOGLE_CLIENT_SECRET = '<FILL ME IN>'
```

Next, just simply run the code. Since we don't have a `GOOGLE_REFRESH_TOKEN` yet, the following will appear on the screen:

```
No refresh token found, obtaining one
Navigate to the following URL to auth: https://accounts.google.com/o/oauth2/auth?client_id= [...]
Enter verification code:
```

Open the URL in your browser, you will be greeted with this:

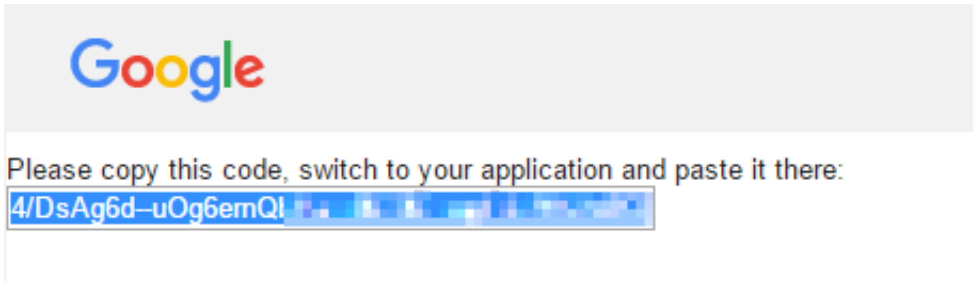

[redacted] would like to:

View and manage your mail

By clicking Allow, you allow this app and Google to use your information in accordance with their respective terms of service and privacy policies. You can change this and other Account Permissions at any time.

Press accept to accept

After pressing "Accept", Google will present you with a verification code:


Google

Please copy this code, switch to your application and paste it there:

4/DsAg6d–uOg6emQl[redacted]

Press accept to accept

Enter this in the running prompt:

```
Enter verification code: <ENTER YOUR VERIFICATION CODE>
Set the following as your GOOGLE_REFRESH_TOKEN: [...] 1bA [...]
```

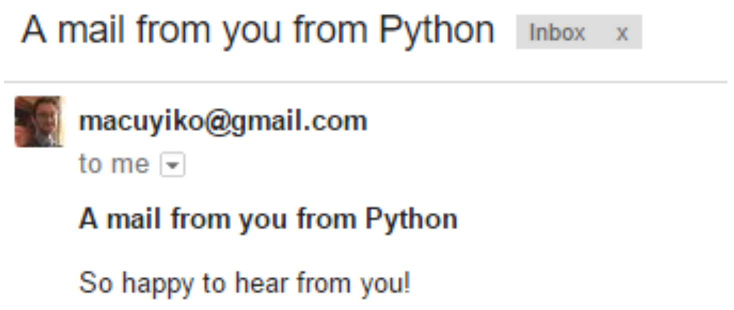
Now modify the Python script to fill in the refresh token with the value you got back:

```
GOOGLE_REFRESH_TOKEN = '[...] 1bA [...]'
```

Step 4: start sending mail. Change the following line with your Gmail address:

```
send_mail('--------@gmail.com', '--------@gmail.com',
```

And run the script again. If all goes well, it should just terminate and you should see an e-mail popping up in your inbox, formatted with HTML, too:


A mail from you from Python Inbox x

macuyiko@gmail.com
to me

A mail from you from Python

So happy to hear from you!

Done!

You don't need to perform a manual reverification as the same refresh token can be used multiple times. If you do start to get exceptions, the timeout for the refresh token has been reached and you manually need to get a new refresh token, though normally refresh tokens are valid for a long time span as long as you wouldn't create additional ones (older ones will stop working) or do not use a refresh token for a long amount of time.