# Praktijk Emails beheren:

This post aims to someone that is starting with Python but also can be useful to knowledgeable users who haven’t worked with email before. The version of Python used to make this post was 3.6.x.

## The email Protocols

Email systems are robust because they rely upon well-established protocols used by all email platforms across the internet. These protocols are defined and published by documents called [RFC](https://en.wikipedia.org/wiki/Request_for_Comments) that stands for Request for Comments and resemble the [PEPs](https://www.python.org/dev/peps/) from Python, but for protocols and patterns that define the operation of communication systems through the internet.

To send emails, we use the [SMTP](https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol) protocol (RFCs [821](https://tools.ietf.org/html/rfc821) e [5321](https://tools.ietf.org/html/rfc5321)), and to receive them, we use [IMAP](https://en.wikipedia.org/wiki/Internet_Message_Access_Protocol) (RFC [3501](https://tools.ietf.org/html/rfc3501)) or [POP](https://en.wikipedia.org/wiki/Post_Office_Protocol) (RFCs [918](https://tools.ietf.org/html/rfc918) e [1081](https://tools.ietf.org/html/rfc1081)) protocols.

## Sending emails

Simple Mail Transfer Protocol or **SMTP** is a protocol from the application layer used to send emails. It operates over [TCP/IP](https://en.wikipedia.org/wiki/Internet_protocol_suite), and your communication is often made using the port 25 or 587 for unencrypted connections and 465 for [TLS/SSL](https://en.wikipedia.org/wiki/Transport_Layer_Security) encrypted connections.

Python comes with batteries included, so we don’t need to install any external package to deal with **SMTP**. We use a library called smtplib:

import smtplib from email.mime.text import MIMEText

connect with Google’s servers smtp\_ssl\_host = ‘smtp.gmail.com’ smtp\_ssl\_port = 465

use username or email to log in

username = ‘origin@gmail.com’ password = ‘password’

from\_addr = ‘origin@gmail.com’ to\_addrs = [‘destiny@gmail.com’]

the email lib has a lot of templates for different message formats, on our case we will use MIMEText to send only text

message = MIMEText(‘Hello World’) message[‘subject’] = ‘Hello’ message[‘from’] = from\_addr message[‘to’] = ‘,’.join(to\_addrs)

we’ll connect using SSL

server = smtplib.SMTP\_SSL(smtp\_ssl\_host, smtp\_ssl\_port)

to interact with the server, first we log in and then we send the message

server.login(username, password) server.sendmail(from\_addr, to\_addrs, message.as\_string()) server.quit()

In case you have problems to connect at Google, you need to enable the “Less secure app access”.

Google blocks access from apps that it judges as not following its security standards, the problem is they don’t have a clear explanation of what these standards are, besides that it’s not a trivial task for whom is starting and doing the first test to struggle with that.

That said if you are facing this issue, you can enable access to make your tests by accessing <https://myaccount.google.com/u/0/security?hl=en>

Now we can use Google as our **gateway** to send the messages, but keep in mind that even with Google as your **gateway** there is a policy of email sending, so be careful with **SPAMS**.

## Receiving emails

The **IMAP** Internet Message Access Protocol is used to receive emails, and as on **SMTP** it operates at the application layer over **TCP/IP**. The port used for its connections is 143 for unencrypted and 993 for encrypted.

Another protocol that works for this task is **POP** Post Office Protocol, but **IMAP** is better due to its synchronization between the client and the server and also the ability to access more than the email inbox.

The process of receiving emails is more complicated than sending because you also have to search for the message and decode it:

import email  
import imaplib  
  
EMAIL = 'mymail@mail.com'  
PASSWORD = 'password'  
SERVER = 'imap.gmail.com'  
  
# connect to the server and go to its inbox  
mail = imaplib.IMAP4\_SSL(SERVER)  
mail.login(EMAIL, PASSWORD)  
# we choose the inbox but you can select others  
mail.select('inbox')  
  
# we'll search using the ALL criteria to retrieve  
# every message inside the inbox  
# it will return with its status and a list of ids  
status, data = mail.search(None, 'ALL')  
# the list returned is a list of bytes separated  
# by white spaces on this format: [b'1 2 3', b'4 5 6']  
# so, to separate it first we create an empty list  
mail\_ids = []  
# then we go through the list splitting its blocks  
# of bytes and appending to the mail\_ids list  
for block in data:  
 # the split function called without parameter  
 # transforms the text or bytes into a list using  
 # as separator the white spaces:  
 # b'1 2 3'.split() => [b'1', b'2', b'3']  
 mail\_ids += block.split()  
  
# now for every id we'll fetch the email  
# to extract its content  
for i in mail\_ids:  
 # the fetch function fetch the email given its id  
 # and format that you want the message to be  
 status, data = mail.fetch(i, '(RFC822)')  
  
 # the content data at the '(RFC822)' format comes on  
 # a list with a tuple with header, content, and the closing  
 # byte b')'  
 for response\_part in data:  
 # so if its a tuple...  
 if isinstance(response\_part, tuple):  
 # we go for the content at its second element  
 # skipping the header at the first and the closing  
 # at the third  
 message = email.message\_from\_bytes(response\_part[1])  
  
 # with the content we can extract the info about  
 # who sent the message and its subject  
 mail\_from = message['from']  
 mail\_subject = message['subject']  
  
 # then for the text we have a little more work to do  
 # because it can be in plain text or multipart  
 # if its not plain text we need to separate the message  
 # from its annexes to get the text  
 if message.is\_multipart():  
 mail\_content = ''  
  
 # on multipart we have the text message and  
 # another things like annex, and html version  
 # of the message, in that case we loop through  
 # the email payload  
 for part in message.get\_payload():  
 # if the content type is text/plain  
 # we extract it  
 if part.get\_content\_type() == 'text/plain':  
 mail\_content += part.get\_payload()  
 else:  
 # if the message isn't multipart, just extract it  
 mail\_content = message.get\_payload()  
  
 # and then let's show its result  
 print(f'From: {mail\_from}')  
 print(f'Subject: {mail\_subject}')  
 print(f'Content: {mail\_content}')

On this code, we extracted only the plain text of the email, but there’s much more you can extract, like the **HTML** content and the annexed files, but this can be a topic for a next post.

### BONUS: Automate the cleaning of bloated mailboxes

Recently, my mailbox got bloated, and I had to do a clean up to get some space, but the high amount of messages overloaded the Gmail interface.

So, to solve that, and also for automating this tedious manual task, I wrote this:

import imaplib  
  
# caso sua caixa de emails esteja muito cheia  
# use esta variável para aumentar o limite do  
# tamanho da resposta  
imaplib.\_MAXLINE = 1000000  
  
EMAIL = 'mymail@gmail.com'  
PASSWORD = 'password'  
SERVER = 'imap.gmail.com'  
  
mail = imaplib.IMAP4\_SSL(SERVER)  
mail.login(EMAIL, PASSWORD)  
# select the box you want to clean  
mail.select('bloated\_box')  
  
status, search\_data = mail.search(None, 'ALL')  
  
mail\_ids = []  
  
for block in search\_data:  
 mail\_ids += block.split()  
  
# define the range for the operation  
start = mail\_ids[0].decode()  
end = mail\_ids[-1].decode()  
  
# move the emails to the trash  
# this step is Gmail specific because  
# it doesn't allow excluding messages  
# outside the trash  
mail.store(f'{start}:{end}'.encode(), '+X-GM-LABELS', '\\Trash')  
  
# access the Gmail trash  
mail.select('[Gmail]/Trash')  
# mark the emails to be deleted  
mail.store("1:\*", '+FLAGS', '\\Deleted')  
  
# remove permanently the emails  
mail.expunge()  
  
# close the mailboxes  
mail.close()  
# close the connection  
mail.logout()

import smtplib  
from email.mime.text import MIMEText  
  
# connect with Google's servers  
smtp\_ssl\_host = 'smtp.gmail.com'  
smtp\_ssl\_port = 465  
# use username or email to log in  
username = 'origin@gmail.com'  
password = 'password'  
  
from\_addr = 'origin@gmail.com'  
to\_addrs = ['destiny@gmail.com']  
  
# the email lib has a lot of templates  
# for different message formats,  
# on our case we will use MIMEText  
# to send only text  
message = MIMEText('Hello World')  
message['subject'] = 'Hello'  
message['from'] = from\_addr  
message['to'] = ', '.join(to\_addrs)  
  
# we'll connect using SSL  
server = smtplib.SMTP\_SSL(smtp\_ssl\_host, smtp\_ssl\_port)  
# to interact with the server, first we log in  
# and then we send the message  
server.login(username, password)  
server.sendmail(from\_addr, to\_addrs, message.as\_string())  
server.quit()