**Как читать электронные письма в Python?**

В ходе данной работы было реализовано приложение, которое позволяло читать мои электронные письма, их удалять, писать и отправлять кому-либо от моего имени, как единичным письмом, так и рассылкой. Все это было реализовано с помощью встроенного модуля imaplib. Что вообще такое IMAP? IMAP — это стандартный протокол Интернета, который используется почтовыми клиентами для получения сообщений электронной почты с почтового сервера.

Начать я бы хотел с чтения отправленных мне писем. Для этого, нам не нужно ничего устанавливать, все модули, которые используются в этом руководстве, являются встроенными:

import imaplib

import email

from email.header import decode\_header

import webbrowser

import os

# учетные данные

username = "youremailaddress @ provider.com"

password = "yourpassword"

def clean(text):

# чистый текст для создания папки

return "".join(c if c.isalnum() else "\_" for c in text)

В данном блоке, мы импортировали необходимые модули, а затем указали учетные данные нашей электронной почты. Позже нам понадобится функция clean(), чтобы создавать папки без пробелов и специальных символов. Далее, мы подключаемся к серверу IMAP и так как я тестирую учетную запись gmail, то я использую сервер imap.gmail.com (для другой почты, необходим, соответственно, другой сервер):

# create an IMAP4 class with SSL

imap = imaplib.IMAP4\_SSL("imap.gmail.com")

# authenticate

imap.login(username, password)

Далее, необходимо уже начинать получать электронные письма:

status, messages = imap.select("INBOX")

# количество популярных писем для получения

N = 3

# общее количество писем

messages = int(messages[0])

В данном блоке мы использовали метод imap.select(), который выбирает почтовый ящик (входящие, спам и т. Д.) и выбрали папку **INBOX**. Переменная messages содержит общее количество сообщений в этой папке (папке «Входящие»), а статус — это просто сообщение, которое указывает, успешно ли мы получили сообщение. Затем мы преобразовали сообщения в целое число, чтобы создать цикл for. Переменная N — это количество основных сообщений электронной почты, которые вы хотите получить, сейчас я буду использовать 3, давайте переберем каждое сообщение электронной почты, извлечем все, что нам нужно, и закончим наш код:

for i in range(messages, messages-N, -1):

# fetch the email message by ID

res, msg = imap.fetch(str(i), "(RFC822)")

for response in msg:

if isinstance(response, tuple):

# parse a bytes email into a message object

msg = email.message\_from\_bytes(response[1])

# decode the email subject

subject, encoding = decode\_header(msg["Subject"])[0]

if isinstance(subject, bytes):

# if it's a bytes, decode to str

subject = subject.decode(encoding)

# decode email sender

From, encoding = decode\_header(msg.get("From"))[0]

if isinstance(From, bytes):

From = From.decode(encoding)

print("Subject:", subject)

print("From:", From)

# if the email message is multipart

if msg.is\_multipart():

# iterate over email parts

for part in msg.walk():

# extract content type of email

content\_type = part.get\_content\_type()

content\_disposition = str(part.get("Content-Disposition"))

try:

# get the email body

body = part.get\_payload(decode=True).decode()

except:

pass

if content\_type == "text/plain" and "attachment" not in content\_disposition:

# print text/plain emails and skip attachments

print(body)

elif "attachment" in content\_disposition:

# download attachment

filename = part.get\_filename()

if filename:

folder\_name = clean(subject)

if not os.path.isdir(folder\_name):

# make a folder for this email (named after the subject)

os.mkdir(folder\_name)

filepath = os.path.join(folder\_name, filename)

# download attachment and save it

open(filepath, "wb").write(part.get\_payload(decode=True))

else:

# extract content type of email

content\_type = msg.get\_content\_type()

# get the email body

body = msg.get\_payload(decode=True).decode()

if content\_type == "text/plain":

# print only text email parts

print(body)

if content\_type == "text/html":

# if it's HTML, create a new HTML file and open it in browser

folder\_name = clean(subject)

if not os.path.isdir(folder\_name):

# make a folder for this email (named after the subject)

os.mkdir(folder\_name)

filename = "index.html"

filepath = os.path.join(folder\_name, filename)

# write the file

open(filepath, "w").write(body)

# open in the default browser

webbrowser.open(filepath)

print("="\*100)

# close the connection and logout

imap.close()

imap.logout()

Во-первых, range(messages, messages-N, -1), означает переход сверху вниз, то есть самые новые сообщения электронной почты имеют наивысший идентификационный номер, а первое сообщение электронной почты имеет идентификатор 1. Во-вторых, мы использовали метод imap.fetch(), который извлекает сообщение электронной почты по идентификатору, используя стандартный формат, указанный в RFC 822. После этого, мы анализируем байты, возвращаемые методом fetch(), на правильный объект Message и используем функцию decode\_header() из email.header для декодирования темы электронного адреса в удобочитаемый юникод. После того, как мы напечатали отправителя электронной почты и тему, мы хотим извлечь тело сообщения. Мы проверяем, является ли электронное сообщение составным, а это означает, что оно состоит из нескольких частей. Например, сообщение электронной почты может содержать содержимое text/html и части text/plain, что означает, что у него есть HTML-версия и текстовая версия сообщения. Он также может содержать вложения файлов, мы обнаруживаем это по заголовку Content-Disposition, поэтому мы загружаем его в новую папку, созданную для каждого сообщения электронной почты, названного в честь темы. Объект msg, который является объектом сообщения модуля электронной почты, имеет много других полей для извлечения, в этом примере мы использовали только **From** и **Subject**, напишите msg.keys() и посмотрите доступные поля для извлечения, вы можете, например, получить дату отправки сообщения с помощью msg ["Date"]. Если же вам ничего не отправили, то он будет выводить «Писем нет».

**Как удалить электронные письма в Python?**

Чтобы автоматический удалять электронные письма в Python, мы все также используем встроенный модуль imaplib и начинаем с импорта необходимых модулей и указания учетных данных нашей учетной записи:

import imaplib

import email

from email.header import decode\_header

# учетные данные

username = "youremailaddress @provider.com"

password = "yourpassword"

После этого, мы подключаемся к IMAP-серверу почтового провайдера:

# create an IMAP4 class with SSL

imap = imaplib.IMAP4\_SSL("imap.gmail.com")

# authenticate

imap.login(username, password)

Далее, когда мы вошли в нашу учетную запись электронной почты, давайте выберем наш целевой почтовый ящик:

# select the mailbox I want to delete in

# if you want SPAM, use imap.select("SPAM") instead

imap.select("INBOX")

Итак, мы используем выбор почтового ящика с помощью метода imap.select(), после чего я выбрал папку INBOX. Теперь давайте выполним поиск писем, которые мы хотим удалить, по дате:

# для получения писем после определенной даты

status, messages = imap.search(None, 'SINCE "01-JAN-2020"')

status содержит строку, указывающую, был ли поиск успешно выполнен, messages возвращаются как список строки [[ASCII] почтовых идентификаторов, разделенных пробелом, преобразуем его в список целых чисел:

# преобразовать сообщения в список адресов электронной почты

messages = messages[0].split(b' ')

Теперь давайте переберем выбранные письма и отметим их как удаленные:

for mail in messages:

\_, msg = imap.fetch(mail, "(RFC822)")

# вы можете удалить цикл for для повышения производительности, если у вас длинный список писем

# потому что он предназначен только для печати SUBJECT целевого электронного письма, которое нужно удалить

for response in msg:

if isinstance(response, tuple):

msg = email.message\_from\_bytes(response[1])

# расшифровать тему письма

subject = decode\_header(msg["Subject"])[0][0]

if isinstance(subject, bytes):

# if it's a bytes type, decode to str

subject = subject.decode()

print("Deleting", subject)

# отметить письмо как удаленное

imap.store(mail, "+FLAGS", "\\Deleted")

Наконец, мы выполняем метод imap.expunge(), который навсегда удаляет сообщения, помеченные как удаленные, так же закрываем почтовый ящик и выходим из аккаунта:

# навсегда удалить письма, помеченные как удаленные

# из выбранного почтового ящика (в данном случае INBOX)

imap.expunge()

# закрыть почтовый ящик

imap.close()

# выйти из аккаунта

imap.logout()

**Как отправлять электронные письма с Python?**

В данном разделе мы будем использовать сразу несколько модулей, а именно: [smtplib](https://docs.python.org/3/library/smtplib.html), [email](https://docs.python.org/3/library/email.html), предназначенный для отправки различных типов вложений, таких как HTML-контент и двоичные файлы, а также библиотека [BeautifulSoup](https://pypi.org/project/beautifulsoup4/) (которую нужно будет заранее установить), позволяющая автоматически извлекать простой текст из HTML, не беспокоясь о регулярных выражениях.

Ну и как обычно, начинаем мы с импорта всех библиотек:

import smtplib

from email import encoders

from email.mime.text import MIMEText

from email.mime.multipart import MIMEMultipart

from email.mime.base import MIMEBase

from bs4 import BeautifulSoup as bs

Далее, мы определяем наши параметры:

# ваши учетные данные

email = "email @ example.com"

password = "password"

# электронная почта отправителя

FROM = "from @example.com"

# адрес электронной почты получателя

TO = "to @example.com"

# тема письма (тема)

subject = "Just a subject"

Переменные email и password — это учетные данные адреса электронной почты, который вы хотите отправить, FROM и TO — это адрес электронной почты отправителя и адрес получателя соответственно (адрес электронной почты и FROM обычно совпадают), а subject — это заголовок (или тема) письма, которое мы отправим.

Убедившись в том, что значения заданы правильно, вы можете отправить сообщения на несколько адресов электронной почты, просто используя их список переменной TO, а не единственный адрес.

Кроме того, мы будем использовать объект MIMEMultipart для одновременной передачи «альтернативы» в качестве подтипа, чтобы объединить две версии почты в одно сообщение с двумя вариантами отображения:

# инициализируем сообщение, которое хотим отправить

msg = MIMEMultipart("alternative")

# установить адрес электронной почты отправителя

msg["From"] = FROM

# установить адрес электронной почты получателя

msg["To"] = TO

# задаем тему

msg["Subject"] = subject

Также, устанавливаем адреса электронной почты From, To и subject. Создадим тело сообщения:

# установить тело письма как HTML

html = """

This email is sent using <b>Python</b>!

"""

# делаем текстовую версию HTML

text = bs(html, "html.parser").text

В этом примере мы установили html как простое сообщение HTML, но вы можете прочитать его из почтового HTML‑шаблона, например:

# установить тело письма как HTML

html = open("mail.html").read()

# делаем текстовую версию HTML

text = bs(html, "html.parser").text

Далее, мы завершаем построение сообщения и смотрим, как оно выглядит:

text\_part = MIMEText(text, "plain")

html\_part = MIMEText(html, "html")

# прикрепить тело письма к почтовому сообщению

# сначала прикрепите текстовую версию

msg.attach(text\_part)

msg.attach(html\_part)

print(msg.as\_string())

Теперь, когда у нас есть письмо, готовое к отправке, нужно создать функцию, которая принимает адреса электронной почты FROM и TO, а также фактическое сообщение, которое нужно отправить, и она отправляет нам электронное письмо:

def send\_mail(email, password, FROM, TO, msg):

# инициализировать SMTP-сервер

server = smtplib.SMTP("smtp.gmail.com", 587)

# подключиться к SMTP-серверу в режиме TLS (безопасный) и отправить EHLO

server.starttls()

# войти в учетную запись, используя учетные данные

server.login(email, password)

# отправить электронное письмо

server.sendmail(FROM, TO, msg.as\_string())

# завершить сеанс SMTP

server.quit()

Итак, сначала мы подключаемся к SMTP-серверу с помощью **smtplib**, в этом примере мы использовали SMTP-сервер Gmail с портом 587. Затем мы переводим соединение с SMTP-сервером в режим TLS для безопасности, и, наконец, мы входим в систему, используя данные учётной записи и завершаем сеанс после отправки нашего электронного письма. Server.sendmail() выполняет всю почтовую транзакцию. У метода есть несколько аргументов: адрес, отправляющий письмо (FROM), список адресов получателей письма (если это строка, то она будет рассматриваться как список с 1 адресом) и сообщение для отправки. Вызовем только что созданную функцию:

# отправить почту

send\_mail(email, password, FROM, TO, msg)

Как итог, проделав все вышесказанные действия, вы сможете с легкостью взаимодействовать со своим Email благодаря и используя Python.