Python

* [Introduction](https://www.devdungeon.com/content/read-and-send-email-python" \l "toc-1)
* [A note about Gmail](https://www.devdungeon.com/content/read-and-send-email-python#toc-2)
* [Read email with IMAP](https://www.devdungeon.com/content/read-and-send-email-python#toc-3)
* [Parse email contents](https://www.devdungeon.com/content/read-and-send-email-python#toc-7)
* [Send email with SMTP](https://www.devdungeon.com/content/read-and-send-email-python#toc-4)
* [Send plaintext email](https://www.devdungeon.com/content/read-and-send-email-python#toc-8)
* [Send multipart HTML email with attachments](https://www.devdungeon.com/content/read-and-send-email-python" \l "toc-9)
* [Email templates with Jinja2](https://www.devdungeon.com/content/read-and-send-email-python" \l "toc-10)
* [Send a text message (SMS/MMS) via email](https://www.devdungeon.com/content/read-and-send-email-python#toc-11)
* [Conclusion](https://www.devdungeon.com/content/read-and-send-email-python#toc-5)
* [References](https://www.devdungeon.com/content/read-and-send-email-python#toc-6)

*Submitted by [NanoDano](https://www.devdungeon.com/users/nanodano" \o "View user profile.) on Sat, 03/14/2020 - 17:44*

**Introduction**

Python 3 has built-in libraries for IMAP, POP3, and SMTP. We will focus on learning how to send mail with SMTP and read/manage email with IMAP. We will also look at how to send an SMS text message using email.

If you need your own email hosting, check out [Interserver.net hosting](https://www.interserver.net/r/181309) where you can host unlimited emails for unlimited domains as cheap as $4/month. You could also set up your own SMTP server on a VPS, but that is a hassle.

**A note about Gmail**

Gmail will not let you use IMAP or POP by default and you must enable the feature.

To do this, go to your Gmail settings, and choose "Enable IMAP" under the "Forwarding and POP/IMAP" tab. See: [Check Gmail through other email platforms](https://support.google.com/mail/answer/7126229?hl=en) for more information.

Your username is full email address at Gmail. Both IMAP and SMTP require authentication. The server names and ports are:

* **imap.gmail.com:993** (SSL/TLS enabled)
* **smtp.gmail.com:465** (SSL/TLS enabled) Port 587 for TLS/STARTTLS

**Read email with IMAP**

To fetch emails, you can use [poplib](https://docs.python.org/3/library/poplib.html) for POP3 or [imaplib](https://docs.python.org/3/library/imaplib.html) to use IMAP4. We will focus only on IMAP which give you more options.

Use [IMAP4](https://docs.python.org/3/library/imaplib.html" \l "imaplib.IMAP4) or [IMAP4\_SSL](https://docs.python.org/3/library/imaplib.html" \l "imaplib.IMAP4_SSL) class depending on whether you are using SSL. This example will use IMAP4\_SSL.

* Port 143 - Default unencrypted IMAP port
* Port 993 - Default SSL IMAP port

imaplib.IMAP4(host='', port=IMAP4\_PORT)

imaplib.IMAP4\_SSL(host='', port=IMAP4\_SSL\_PORT, keyfile=**None**, certfile=**None**, ssl\_context=**None**)

To keep the first example simple, this is a minimal simple example of checking an inbox:

**import** imaplib

# Connect to inbox

imap\_server = imaplib.IMAP4\_SSL(host='mail.example.com')

imap\_server.login('nanodano@devdungeon.com', '$ecret')

imap\_server.**select**() # Default is `INBOX`

# Find all emails in inbox and print out the raw email data

\_, message\_numbers\_raw = imap\_server.search(**None**, 'ALL')

**for** message\_number **in** message\_numbers\_raw[0].split():

\_, msg = imap\_server.fetch(message\_number, '(RFC822)')

**print**(msg[0][1])

This next example will show how to do more common operations like:

* Connect to IMAP server
* List folders (mailboxes)
* Create, rename, and delete folders (mailboxes)
* Search emails
* Fetch emails
* Mark an email as read or unread
* Move an email to a different folder
* Delete an email

**import** imaplib

# Connect and login to IMAP mail server

username = 'me@example.com'

password = 'password'

mail\_server = 'mail.example.com'

imap\_server = imaplib.IMAP4\_SSL(host=mail\_server)

imap\_server.login(username, password)

# List mailboxes (folders)

response\_code, folders = imap\_server.list()

**print**(response\_code) # OK

**print**('Available folders(mailboxes) to select:')

**for** folder\_details\_raw **in** folders:

folder\_details = folder\_details\_raw.decode().split()

**print**(f'- {folder\_details[-1]}')

# Create, rename, and delete mailboxes (folders)

# This format is the one my email provider interserver.net uses

# Create a mailbox

response\_code, response\_details = imap\_server.create('INBOX.myfavorites')

**print**(response\_code) # `OK` on success or `NO` on failure

**print**(response\_details) # Create completed/Mailbox already exists

# Rename a mailbox

imap\_server.rename('INBOX.myfavorites', 'INBOX.faves')

# Delete a mailbox

imap\_server.**delete**('INBOX.faves')

# Choose the mailbox (folder) to search

# Case sensitive!

imap\_server.**select**('INBOX') # Default is `INBOX`

# Search for emails in the mailbox that was selected.

# First, you need to search and get the message IDs.

# Then you can fetch specific messages with the IDs.

# Search filters are explained in the RFC at:

# https://tools.ietf.org/html/rfc3501#section-6.4.4

search\_criteria = 'ALL'

charset = **None** # All

respose\_code, message\_numbers\_raw = imap\_server.search(charset, search\_criteria)

**print**(f'Search response: {respose\_code}') # e.g. OK

**print**(f'Message numbers: {message\_numbers\_raw}') # e.g. ['1 2'] A list, with string of message IDs

message\_numbers = message\_numbers\_raw[0].split()

# Fetch full message based on the message numbers obtained from search

**for** message\_number **in** message\_numbers:

response\_code, message\_data = imap\_server.fetch(message\_number, '(RFC822)')

**print**(f'Fetch response for message {message\_number}: {response\_code}')

**print**(f'Raw email data:\n{message\_data[0][1]}')

# Mark an email read/unread.

# Other flags you can set with store() from RFC3501 include:

# \Seen \Answered \Flagged \Deleted \Draft \Recent

imap\_server.store(message\_number, '+FLAGS', '\SEEN') # Mark as read

imap\_server.store(message\_number, '-FLAGS', '\SEEN') # Mark as unread

# Copy an email to a different

imap\_server.create('INBOX.mykeepers')

imap\_server.copy(message\_number, 'INBOX.mykeepers')

# Delete an email

imap\_server.store(message\_number, '+FLAGS', '\Deleted')

# Expunge after marking emails deleted

imap\_server.expunge()

imap\_server.close()

imap\_server.logout()

PARSE EMAIL CONTENTS

In the previous example we showed how to fetch the raw email data, but it includes the headers, the body, and everything in a single blob. That raw content is the equivalent of a .eml message. Python has an email package that will parse this raw data and provide us a useful object.

You can parse the email with [email.parser](https://docs.python.org/3/library/email.parser.html). There is also a function named [email.message\_from\_bytes()](https://docs.python.org/3/library/email.parser.html" \l "email.message_from_bytes) that you can use to parse directly from the raw bytes like we will have. Once you have the [email.message.Message](https://docs.python.org/2/library/email.message.html" \l "email.message.Message) you can check various aspects like if it is multipart, content type, and get the payload.

This example will build on top of the simple inbox check example above and demonstrate how to:

* Parse email message
  + Get to/from/cc/bcc email addresses
  + Get plain text version
  + Get html version
  + Get attachments

**import** imaplib

**import** email

# Connect to inbox

imap\_server = imaplib.IMAP4\_SSL(host='mail.example.com')

imap\_server.login('nanodano@devdungeon.com', '$ecret')

imap\_server.**select**() # Default is `INBOX`

# Find all emails in inbox

\_, message\_numbers\_raw = imap\_server.search(**None**, 'ALL')

**for** message\_number **in** message\_numbers\_raw[0].split():

\_, msg = imap\_server.fetch(message\_number, '(RFC822)')

# Parse the raw email message in to a convenient object

message = email.message\_from\_bytes(msg[0][1])

**print**('== Email message =====')

# print(message) # print FULL message

**print**('== Email details =====')

**print**(f'From: {message["from"]}')

**print**(f'To: {message["to"]}')

**print**(f'Cc: {message["cc"]}')

**print**(f'Bcc: {message["bcc"]}')

**print**(f'Urgency (1 highest 5 lowest): {message["x-priority"]}')

**print**(f'Object type: {type(message)}')

**print**(f'Content type: {message.get\_content\_type()}')

**print**(f'Content disposition: {message.get\_content\_disposition()}')

**print**(f'Multipart?: {message.is\_multipart()}')

# If the message is multipart, it basically has multiple emails inside

# so you must extract each "submail" separately.

**if** message.is\_multipart():

**print**('Multipart types:')

**for** part **in** message.walk():

**print**(f'- {part.get\_content\_type()}')

multipart\_payload = message.get\_payload()

**for** sub\_message **in** multipart\_payload:

# The actual text/HTML email contents, or attachment data

**print**(f'Payload\n{sub\_message.get\_payload()}')

**else**: # Not a multipart message, payload is simple string

**print**(f'Payload\n{message.get\_payload()}')

# You could also use `message.iter\_attachments()` to get attachments only

Note that if you have an email on disk and you want to parse it directly from a file, you can use the email.parser.BytesParser like this:

**from** email.parser **import** BytesParser

**with** open('some\_email.eml', 'rb') **as** email\_file:

message = BytesParser().parse(email\_file)

If you want to pull attachments only from an email ignoring the body, you can use [iter\_attachments()](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage.iter_attachments).

**Send email with SMTP**

Let's look at how to send an email using Python. First, we'll look at sending a very basic plaintext email using [smtplib](https://docs.python.org/3/library/smtplib.html). Then we'll craft a multipart email message using the [email.message](https://docs.python.org/3/library/email.message.html) with text, HTML, and attachments.

These examples will use an encrypted SSL SMTP server. The default port for SMTP with SSL is 587.

* Port 25 - Default unencrypted SMTP port
* Port 587 - Default encrypted SSL SMTP port
* Port 465 - Non-standard port for SSL SMTP that is rarely used

Note that your from address can be very important. Some firewalls and email servers will prevent your email from going through if you use a domain name that does not match the sending host, so you can't pretend to be @google.com.

SEND PLAINTEXT EMAIL

This first example will show the simplest example of sending a mail with SMTP. The email will be crafted by hand, with the headers first, followed by a blank line, followed by the plain-text body.

**from** smtplib **import** SMTP\_SSL, SMTP\_SSL\_PORT

SMTP\_HOST = 'mail.example.com'

SMTP\_USER = 'nanodano@devdungeon.com'

SMTP\_PASS = 'Secret!'

# Craft the email by hand

from\_email = 'John Leon <nanodano@devdungeon.com>' # or simply the email address

to\_emails = ['nanodano@devdungeon.com', 'admin@devdungeon.com']

body = "Hello, world!"

headers = f"From: {from\_email}\r\n"

headers += f"To: {', '.join(to\_emails)}\r\n"

headers += f"Subject: Hello\r\n"

email\_message = headers + "\r\n" + body # Blank line needed between headers and body

# Connect, authenticate, and send mail

smtp\_server = SMTP\_SSL(SMTP\_HOST, port=SMTP\_SSL\_PORT)

smtp\_server.set\_debuglevel(1) # Show SMTP server interactions

smtp\_server.login(SMTP\_USER, SMTP\_PASS)

smtp\_server.sendmail(from\_email, to\_emails, email\_message)

# Disconnect

smtp\_server.quit()

Instead of creating the email as a big raw string, you can use the [email.message.EmailMessage](https://docs.python.org/3/library/email.message.html" \l "email.message.EmailMessage) class to manage en email easier. This example will show how to

* Create an email message object
* Set to and from addresses
* Set the subject
* Add the urgent flag
* Set body of email

**from** smtplib **import** SMTP\_SSL, SMTP\_SSL\_PORT

**from** email.message **import** EmailMessage

# Craft the email using email.message.EmailMessage

from\_email = 'John Leon <nanodano@devdungeon.com>' # or simply the email address

to\_emails = ['nanodano@devdungeon.com', 'admin@devdungeon.com']

email\_message = EmailMessage()

email\_message.add\_header('To', ', '.join(to\_emails))

email\_message.add\_header('From', from\_email)

email\_message.add\_header('Subject', 'Hello!')

email\_message.add\_header('X-Priority', '1') # Urgency, 1 highest, 5 lowest

email\_message.set\_content('Hello, world!')

# Connect, authenticate, and send mail

smtp\_server = SMTP\_SSL('mail.example.com', port=SMTP\_SSL\_PORT)

smtp\_server.set\_debuglevel(1) # Show SMTP server interactions

smtp\_server.login('user@example.com', 'pass')

smtp\_server.sendmail(from\_email, to\_emails, email\_message.as\_bytes())

# Disconnect

smtp\_server.quit()

SEND MULTIPART HTML EMAIL WITH ATTACHMENTS

To create a multipart email that contains text and HTML versions along with attachments, you can use the [email.mime.multipart.MIMEMultipart](https://docs.python.org/3.9/library/email.mime.html" \l "email.mime.multipart.MIMEMultipart) class.

email.mime.multipart.MIMEMultipart(\_subtype='mixed', boundary=**None**, \_subparts=**None**, \*, policy=compat32, \*\*\_params)

To use a MIMEMultipart, first create the object just like a normal email.message.EmailMessage. Instead of setting the content though, we will attach() all of the parts, including the text version, html version, and any attachments.

This example will show how to create a multipart MIME email that has

* Plain-text version of email
* HTML version of email
* Attachments

**from** smtplib **import** SMTP\_SSL, SMTP\_SSL\_PORT

**from** email.mime.multipart **import** MIMEMultipart, MIMEBase

**from** email.mime.text **import** MIMEText

**from** email.encoders **import** encode\_base64

from\_email = 'John Leon <nanodano@devdungeon.com>' # or simply the email address

to\_emails = ['nanodano@devdungeon.com', 'johndleon@gmail.com']

# Create multipart MIME email

email\_message = MIMEMultipart()

email\_message.add\_header('To', ', '.join(to\_emails))

email\_message.add\_header('From', from\_email)

email\_message.add\_header('Subject', 'Hello!')

email\_message.add\_header('X-Priority', '1') # Urgent/High priority

# Create text and HTML bodies for email

text\_part = MIMEText('Hello world plain text!', 'plain')

html\_part = MIMEText('<html><body><h1>HTML!</h1></body></html>', 'html')

# Create file attachment

attachment = MIMEBase("application", "octet-stream")

attachment.set\_payload(b'\xDE\xAD\xBE\xEF') # Raw attachment data

encode\_base64(attachment)

attachment.add\_header("Content-Disposition", "attachment; filename=myfile.dat")

# Attach all the parts to the Multipart MIME email

email\_message.attach(text\_part)

email\_message.attach(html\_part)

email\_message.attach(attachment)

# Connect, authenticate, and send mail

smtp\_server = SMTP\_SSL('mail.example.com', port=SMTP\_SSL\_PORT)

smtp\_server.set\_debuglevel(1) # Show SMTP server interactions

smtp\_server.login('user@email.com', 'password')

smtp\_server.sendmail(from\_email, to\_emails, email\_message.as\_bytes())

# Disconnect

smtp\_server.quit()

EMAIL TEMPLATES WITH JINJA2

If you want to create a text or HTML template for re-use, I recommend [Jinja2 templates](https://jinja.palletsprojects.com/en/2.11.x/).

Here is a *very* basic example of how a Jinja2 template can be used. Refer to the [Jinja2 documentation](https://jinja.palletsprojects.com/en/2.11.x/) for more details.

# pip install jinja2

**from** jinja2 **import** Template

**template** = Template('Hello, {{ name }}!')

**print**(**template**.render({'name': 'NanoDano'}))

SEND A TEXT MESSAGE (SMS/MMS) VIA EMAIL

Most cell phone service providers also offer an email gateway that lets you email an address and it will send an SMS/MMS to the cell phone.

For a detailed list of SMS email gateways listed by provider, check out [SMS gateways on Wikipedia](https://en.wikipedia.org/wiki/SMS_gateway).

For example, to text the number 888-123-4567 on AT&T, I could send an email to:

8881234567@txt.att.net

**Conclusion**

After reading this guide, you should understand how to use Python to read mail with IMAP and how to send mail using SMTP with plain-text or HTML emails along with attachments.

**References**

* [email package](https://docs.python.org/3/library/email.html)
* [smtplib package](https://docs.python.org/3/library/smtplib.html)
* [imaplib](https://docs.python.org/3/library/imaplib.html)
* [poplib package](https://docs.python.org/3/library/poplib.html)
* [SMS gateways on Wikipedia](https://en.wikipedia.org/wiki/SMS_gateway)
* [message headers](https://www.iana.org/assignments/message-headers/message-headers.xhtml)
* [Jinja2 templates](https://jinja.palletsprojects.com/en/2.11.x/)