Lecture2- Email API and Programming

NET 445 – Internet Programming

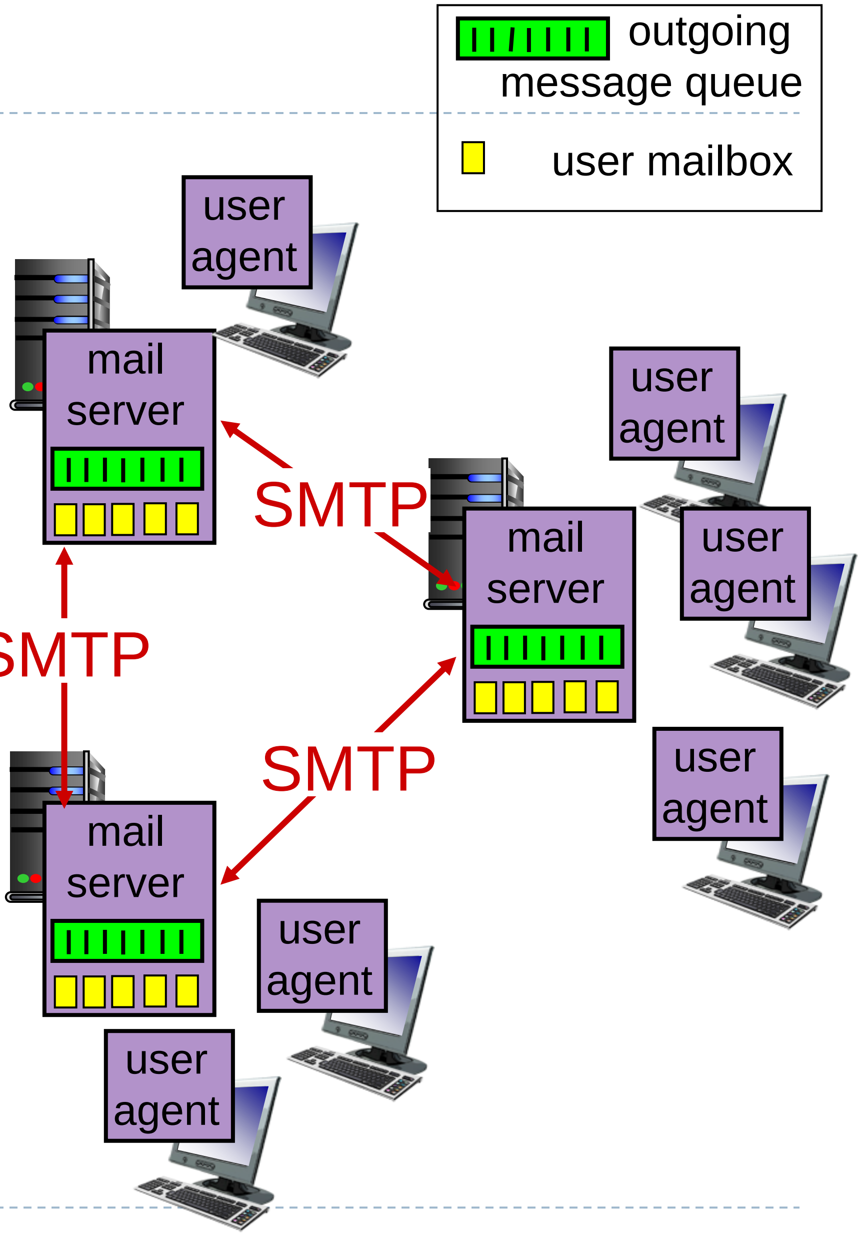
Electronic mail

Three major components:

- user agents
- mail servers
- simple mail transfer protocol: SMTP

User Agent

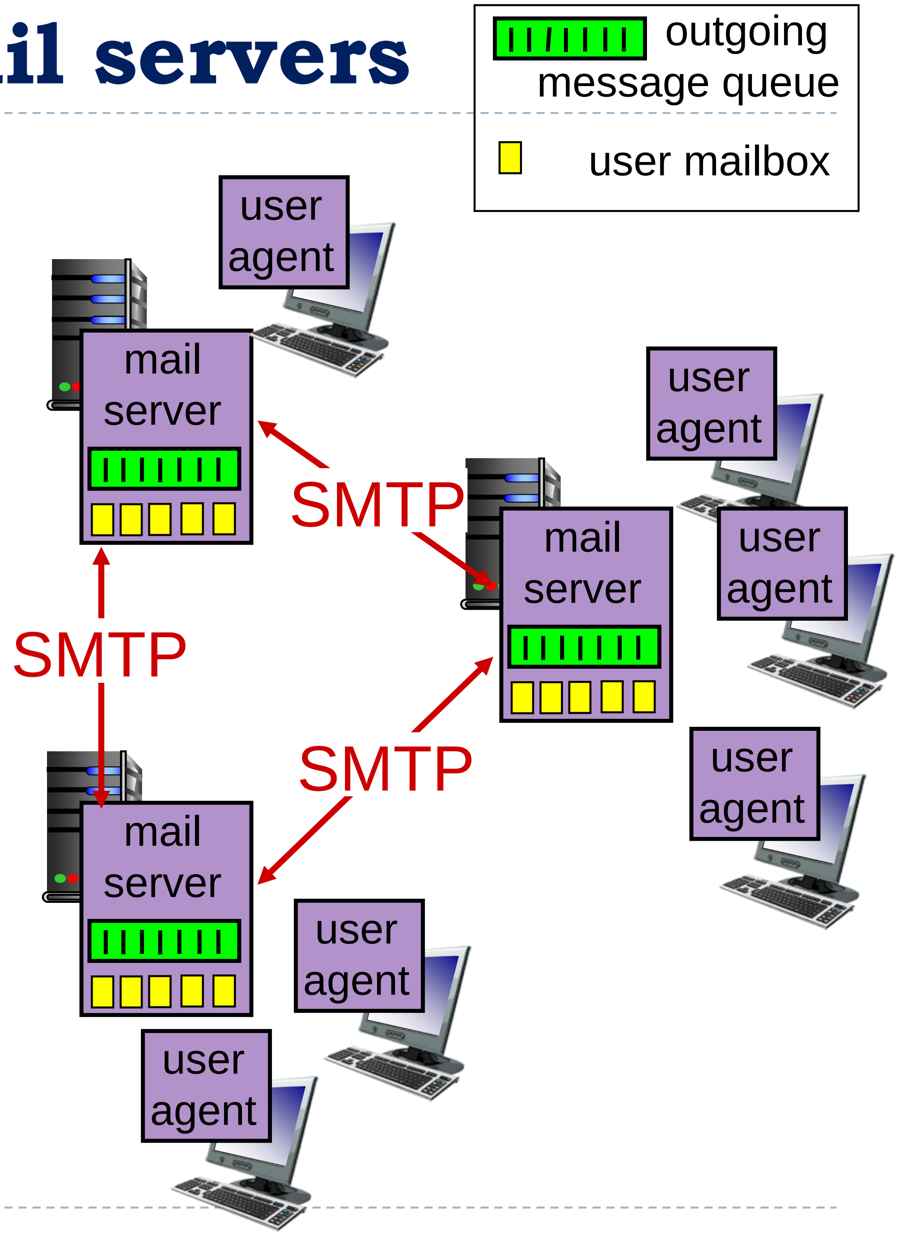
- a.k.a. "mail reader"
- composing, editing, reading mail messages
- e.g., Outlook, Thunderbird, iPhone mail client
- outgoing, incoming messages stored on server



Electronic mail: mail servers

mail servers:

- *mailbox* contains incoming messages for user
- *message queue* of outgoing (to be sent) mail messages
- *SMTP protocol* between mail servers to send email messages
  - client: sending mail server
  - "server": receiving mail server

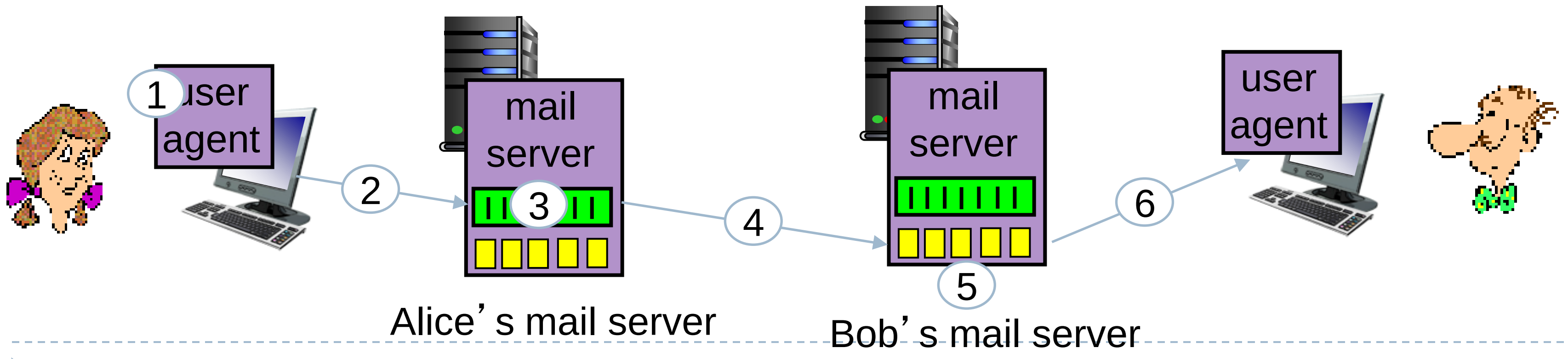


Electronic Mail: SMTP [RFC 2821]

- uses TCP to reliably transfer email message from client to server, port 25
- direct transfer: sending server to receiving server
- three phases of transfer
  - handshaking (greeting)
  - transfer of messages
  - closure
- command/response interaction (like HTTP, FTP)
  - commands: ASCII text
  - response: status code and phrase
- messages must be in 7-bit ASCI

Scenario: Alice sends message to Bob

1) Alice uses UA to compose message "to" `bob@someschool.edu`
2) Alice's UA sends message to her mail server; message placed in message queue
3) client side of SMTP opens TCP connection with Bob's mail server
4) SMTP client sends Alice's message over the TCP connection
5) Bob's mail server places the message in Bob's mailbox
6) Bob invokes his user agent to read message



Sample SMTP interaction

```
S: 220 hamburger.edu
    C: HELO crepes.fr
    S: 250  Hello crepes.fr, pleased to meet you
    C: MAIL FROM: <alice@crepes.fr>
    S: 250 alice@crepes.fr... Sender ok
    C: RCPT TO: <bob@hamburger.edu>
    S: 250 bob@hamburger.edu ... Recipient ok
    C: DATA
    S: 354 Enter mail, end with "." on a line by itself
    C: Do you like ketchup?
    C: How about pickles?
    C: .
    S: 250 Message accepted for delivery
    C: QUIT
    S: 221 hamburger.edu closing connection
```

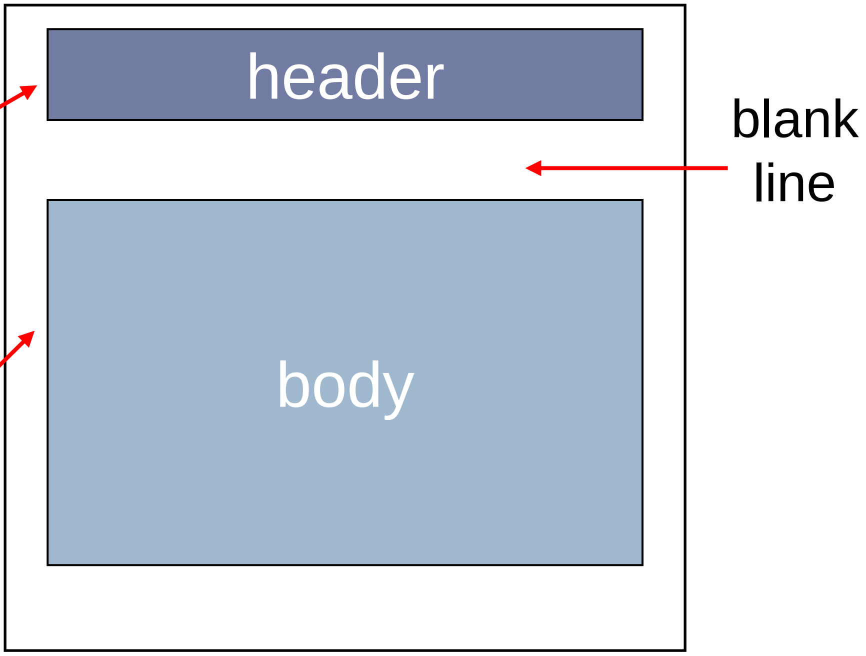
Mail message format

SMTP: protocol for exchanging email msgs

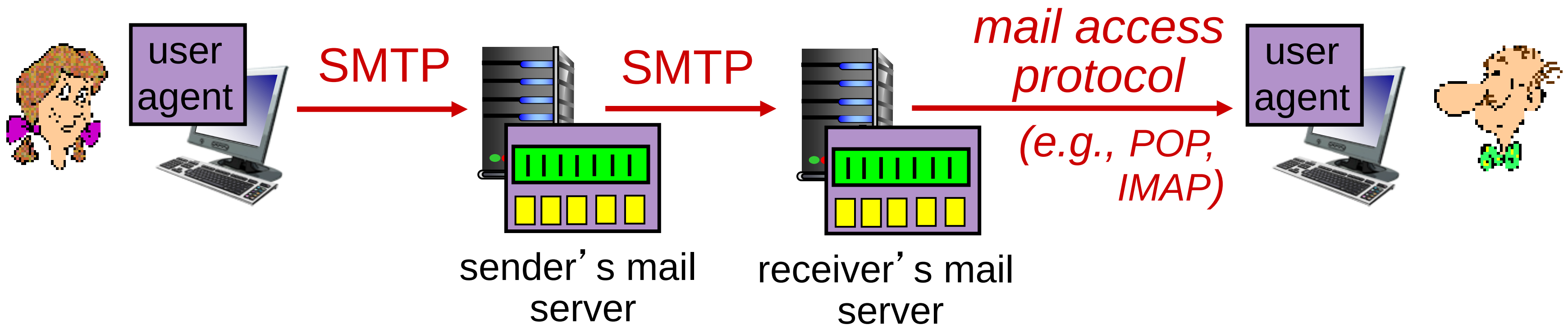
RFC 822: standard for text message format:

- header lines, e.g.,
  - To:
  - From:
  - Subject:

  different from SMTP MAIL FROM, RCPT TO: commands!
- Body: the "message"
  - ASCII characters only



Mail access protocols



- SMTP: delivery/storage to receiver's server
- mail access protocol: retrieval from server
  - POP: Post Office Protocol [RFC 1939]: authorization, download
  - IMAP: Internet Mail Access Protocol [RFC 1730]: more features, including manipulation of stored msgs on server
  - HTTP: gmail, Hotmail, Yahoo! Mail, etc.

POP3 (more) and IMAP

more about POP3

- previous example uses POP3 "download and delete" mode
  - Bob cannot re-read e-mail if he changes client
- POP3 "download-and-keep": copies of messages on different clients
- POP3 is stateless across sessions

IMAP

- keeps all messages in one place: at server
- allows user to organize messages in folders
- keeps user state across sessions:
  - names of folders and mappings between message IDs and folder name

Python Local SMTP server

- Starting a local smtp server

```
python3 -m smtpd -c DebuggingServer -n localhost:1025
```

- If you prefer working in the local environment, the local SMTP debugging server might be an option. For this purpose.
- Python offers an smtpd module.
- It has a DebuggingServer feature, which will discard messages you are sending out and will print them to stdout.
- It is compatible with all operations systems.

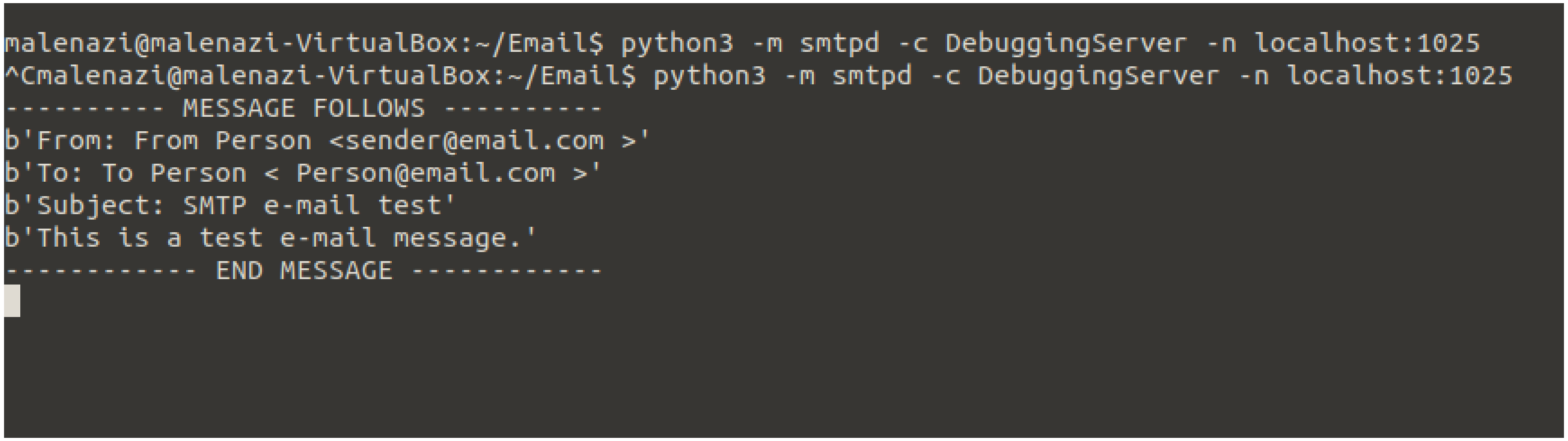
Example: Sending an email via Python

- using localhost sever

```
#!/usr/bin/python3
import smtplib
from smtplib import SMTPException
sender = 'sender@email.com'
receivers = ['Person@email.com']

message = """From: From Person <sender@email.com >
To: To Person < Person@email.com >
Subject: SMTP e-mail test
This is a test e-mail message.
"""
try:
   smtpObj = smtplib.SMTP('localhost',1025)
   smtpObj.sendmail(sender, receivers, message)
   print ("Successfully sent email")
except SMTPException:
   print ("Error: unable to send email")
```

Example: Sending an email via Python

- After sending the email
- The SMTP server will show the message

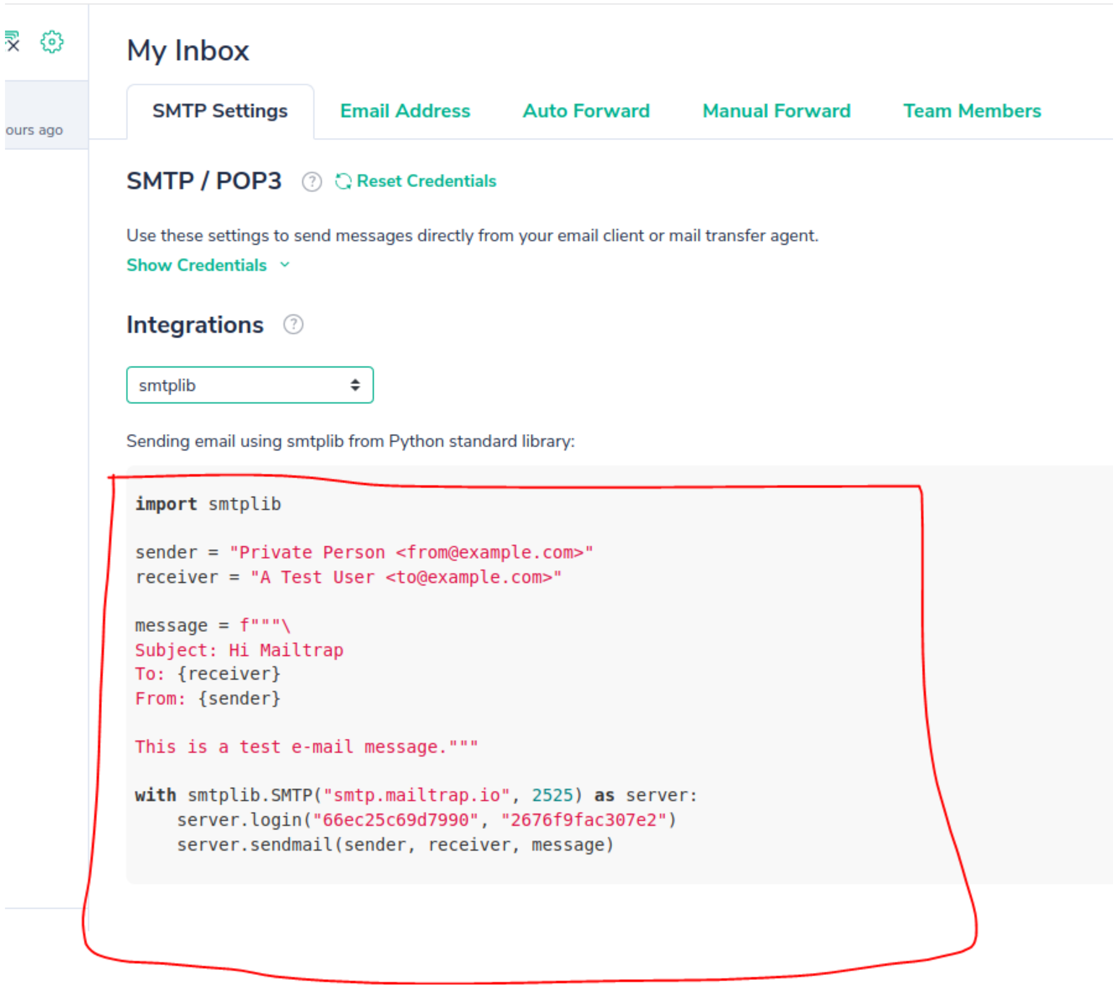


```
malenazi@malenazi-VirtualBox:~/Email$ python3 -m smtpd -c DebuggingServer -n localhost:1025
^Cmalenazi@malenazi-VirtualBox:~/Email$ python3 -m smtpd -c DebuggingServer -n localhost:1025
---------- MESSAGE FOLLOWS ----------
b'From: From Person <sender@email.com >'
b'To: To Person < Person@email.com >'
b'Subject: SMTP e-mail test'
b'This is a test e-mail message.'
------------ END MESSAGE ------------
```

Testing with Fake Email servers

- Testing with real email server is difficult since it has security issues.
- Some websites provide fake email servers to test your script before send real email.
- Mailtrap.io
  - provides a fake SMTP server to test, view and share emails sent.
  - You need to sign up and get your user authentication information from Mailtrap.io

Mailtrap.io



Use these settings to send messages directly from your email client or mail transfer agent.

smtplib

Sending email using smtplib from Python standard library:

```
import smtplib

sender = "Private Person <from@example.com>"
receiver = "A Test User <to@example.com>"

message = f"""\
Subject: Hi Mailtrap
To: {receiver}
From: {sender}

This is a test e-mail message."""

with smtplib.SMTP("smtp.mailtrap.io", 2525) as server:
    server.login("66ec25c69d7990", "2676f9fac307e2")
    server.sendmail(sender, receiver, message)
```

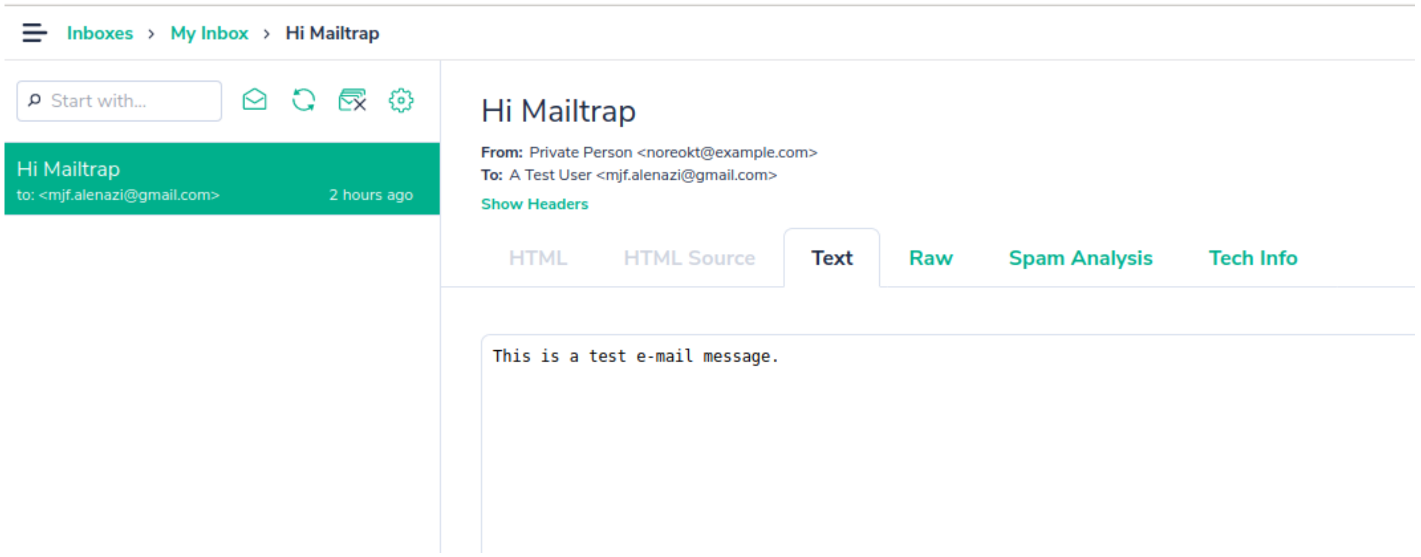
Example: Sending an email via Mailtrap.io

- Sending email

```
import smtplib

sender = "Private Person <from@example.com>"
receiver = "A Test User <to@example.com>"

message = f"""\
Subject: Hi Mailtrap
To: {receiver}
From: {sender}

This is a test e-mail message."""

with smtplib.SMTP("smtp.mailtrap.io", 2525) as server:
    server.login("66ec25c69d7990", "2676f9fac307e2")
    server.sendmail(sender, receiver, message)
```

Example: Sending an email via Mailtrap.io

- Once the email is sent using Python
- The email will appear the inbox



Hi Mailtrap
to: <mjf.alenazi@gmail.com> 2 hours ago

Hi Mailtrap

From: Private Person <noreokt@example.com>
To: A Test User <mjf.alenazi@gmail.com>

```
This is a test e-mail message.
```

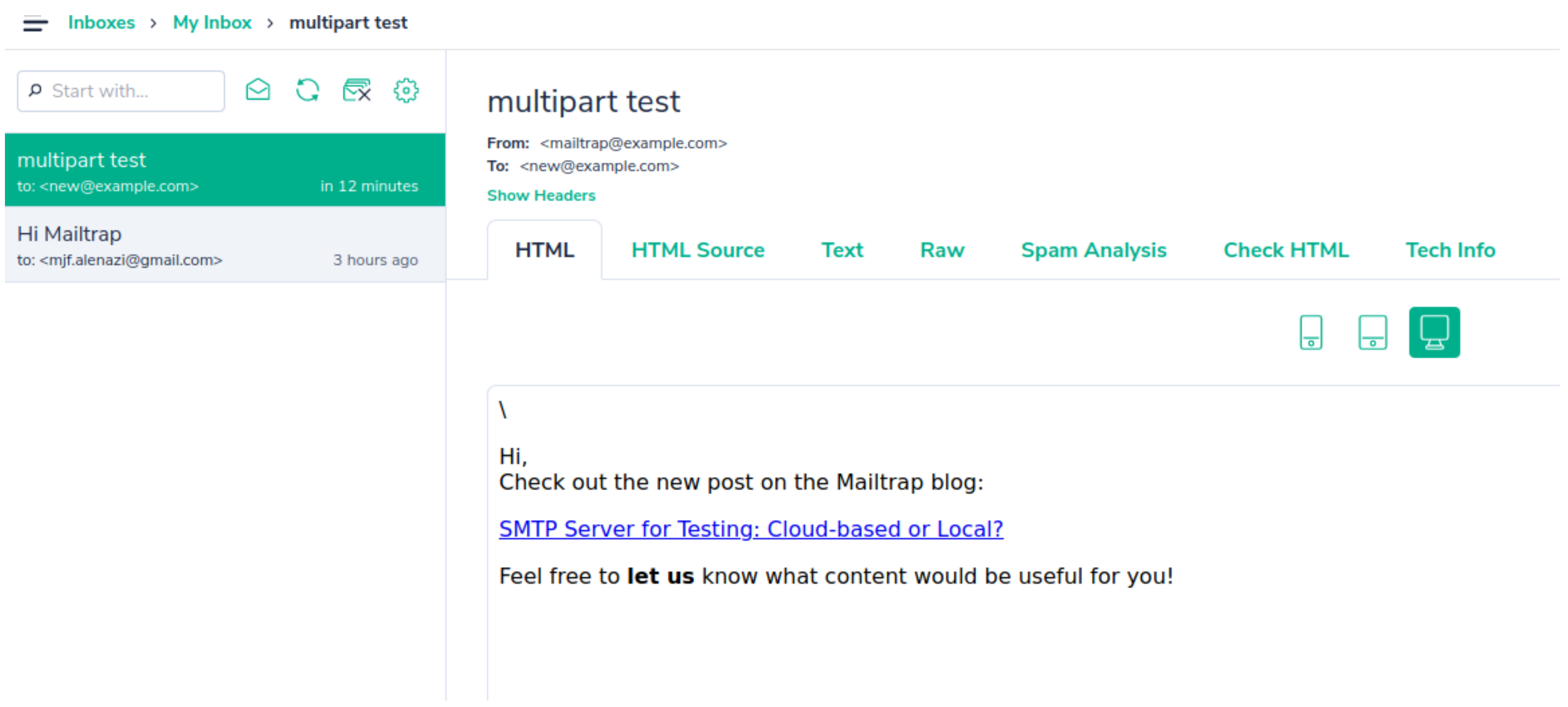
Example: Sending an email with HTML content

- Sending an email with HTML content

```
import smtplib
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart

sender = "Private Person <from@example.com>"
receiver = "A Test User <to@example.com>"

sender_email = "mailtrap@example.com"
receiver_email = "new@example.com"

message = MIMEMultipart("alternative")
message["Subject"] = "multipart test"
message["From"] = sender_email
message["To"] = receiver_email
# Write the plain text part
text = """\ Hi, Check out the new post on the Mailtrap blog: SMTP Server for Testing: Cloud-based or Local?
https://blog.mailtrap.io/2018/09/27/cloud-or-local-smtp-server/ Feel free to let us know what content would be useful for you!"""

# write the HTML part
html = """\ <html> <body> <p>Hi,<br> Check out the new post on the Mailtrap blog:</p> <p><a
href="https://blog.mailtrap.io/2018/09/27/cloud-or-local-smtp-server">SMTP Server for Testing: Cloud-based or Local?</a></p> <p> Feel free
to <strong>let us</strong> know what content would be useful for you!</p> </body> </html> """

# convert both parts to MIMEText objects and add them to the MIMEMultipart message
part1 = MIMEText(text, "plain")
part2 = MIMEText(html, "html")
message.attach(part1)
message.attach(part2)



with smtplib.SMTP("smtp.mailtrap.io", 2525) as server:
                server.login("66ec25c69d7990", "2676f9fac307e2")
                server.sendmail( sender_email, receiver_email, message.as_string() )
```

Example: Sending an email via Mailtrap.io

- An email testing using HTML content



multipart test
to: <new@example.com> in 12 minutes

Hi Mailtrap
to: <mjf.alenazi@gmail.com> 3 hours ago

multipart test

From: <mailtrap@example.com>
To: <new@example.com>

\

Hi,
Check out the new post on the Mailtrap blog:

SMTP Server for Testing: Cloud-based or Local?

Feel free to **let us** know what content would be useful for you!

Example: Sending Email using Gmail

- IMAP protocol

```
import smtplib
from email.mime.text import MIMEText

smtp_ssl_host = 'smtp.gmail.com'
smtp_ssl_port = 465

from_addr = 'net445ksu@gmail.com'
to_addrs = ['mjalenazi@ksu.edu.sa']

# 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. Hi from Ubuntu Mate 18')
message['subject'] = 'Hello'
message['from'] = from_addr
message['to'] = ', '.join(to_addrs)

username = 'net445ksu@gmail.com'
password = 'Ksu@12345'
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()
```

IMAP

- IMAP allows the client program to manipulate the e-mail message on the server without downloading them on the local computer.
- The e-mail is hold and maintained by the remote server.
- It enables us to take any action such as downloading, delete the mail without reading the mail.
- It enables us to create, manipulate and delete remote message folders called mail boxes.
- IMAP enables the users to search the e-mails.
- It allows concurrent access to multiple mailboxes on multiple mail servers.

IMAP Commands

S.N.	Command Description
1	**IMAP_LOGIN** This command opens the connection.
2	**CAPABILITY** This command requests for listing the capabilities that the server supports.
3	**NOOP** This command is used as a periodic poll for new messages or message status updates during a period of inactivity.
4	**SELECT** This command helps to select a mailbox to access the messages.
5	**EXAMINE** It is same as SELECT command except no change to the mailbox is permitted.
6	**CREATE** It is used to create mailbox with a specified name.
7	**DELETE** It is used to permanently delete a mailbox with a given name.
8	**RENAME** It is used to change the name of a mailbox.
9	**LOGOUT** This command informs the server that client is done with the session. The server must send BYE untagged response before the OK response and then close the network connection.

Example: Sending Email using Gmail

- IMAP protocol

```
import imaplib
import pprint

imap_host = 'imap.gmail.com'
imap_user = 'net445ksu@gmail.com'
imap_pass = 'Ksu@12345'

# connect to host using SSL
imap = imaplib.IMAP4_SSL(imap_host)

## login to server
imap.login(imap_user, imap_pass)

imap.select('Inbox')

tmp, data = imap.search(None, 'ALL')
for num in data[0].split():
            tmp, data = imap.fetch(num, '(RFC822)')
            print('Message: {0}\n'.format(num))
            pprint.pprint(data[0][1])
            break
imap.close()
```

POP3

- POP is an application layer internet standard protocol.
- Since POP supports offline access to the messages, thus requires less internet usage time.
- POP does not allow search facility.
- In order to access the messaged, it is necessary to download them.
- It allows only one mailbox to be created on server.
- POP commands are generally abbreviated into codes of three or four letters. Eg. STAT.

POP3 Commands

S.N.	Command Description
1	**LOGIN** This command opens the connection.
2	**STAT** It is used to display number of messages currently in the mailbox.
3	**LIST** It is used to get the summary of messages where each message summary is shown.
4	**RETR** This command helps to select a mailbox to access the messages.
5	**DELE** It is used to delete a message.
6	**RSET** It is used to reset the session to its initial state.
7	**QUIT** It is used to log off the session.

Example: Sending Email using Gmail

- POP3 protocol

```
import  poplib

user = 'net445ksu@gmail.com'
# Connect to the mail box
Mailbox = poplib.POP3_SSL('pop.googlemail.com', '995')
Mailbox.user(user)
Mailbox.pass_('Ksu@12345')
NumofMessages = len(Mailbox.list()[1])
for i in range(NumofMessages):
    for msg in Mailbox.retr(i+1)[1]:
        print (msg)
Mailbox.quit()
```

References:

- Foundations of Python Network Programming Third Edition by Brandon Rhodes (2014)
- James F. Kurose, and Keith W Ross, Computer Networking: A Top-Down Approach, 6th Edition
- Python 3 documentation
- https://wiki.python.org/moin/UdpCommunication
- https://www.w3schools.com/python/
- https://www.tutorialspoint.com/python/