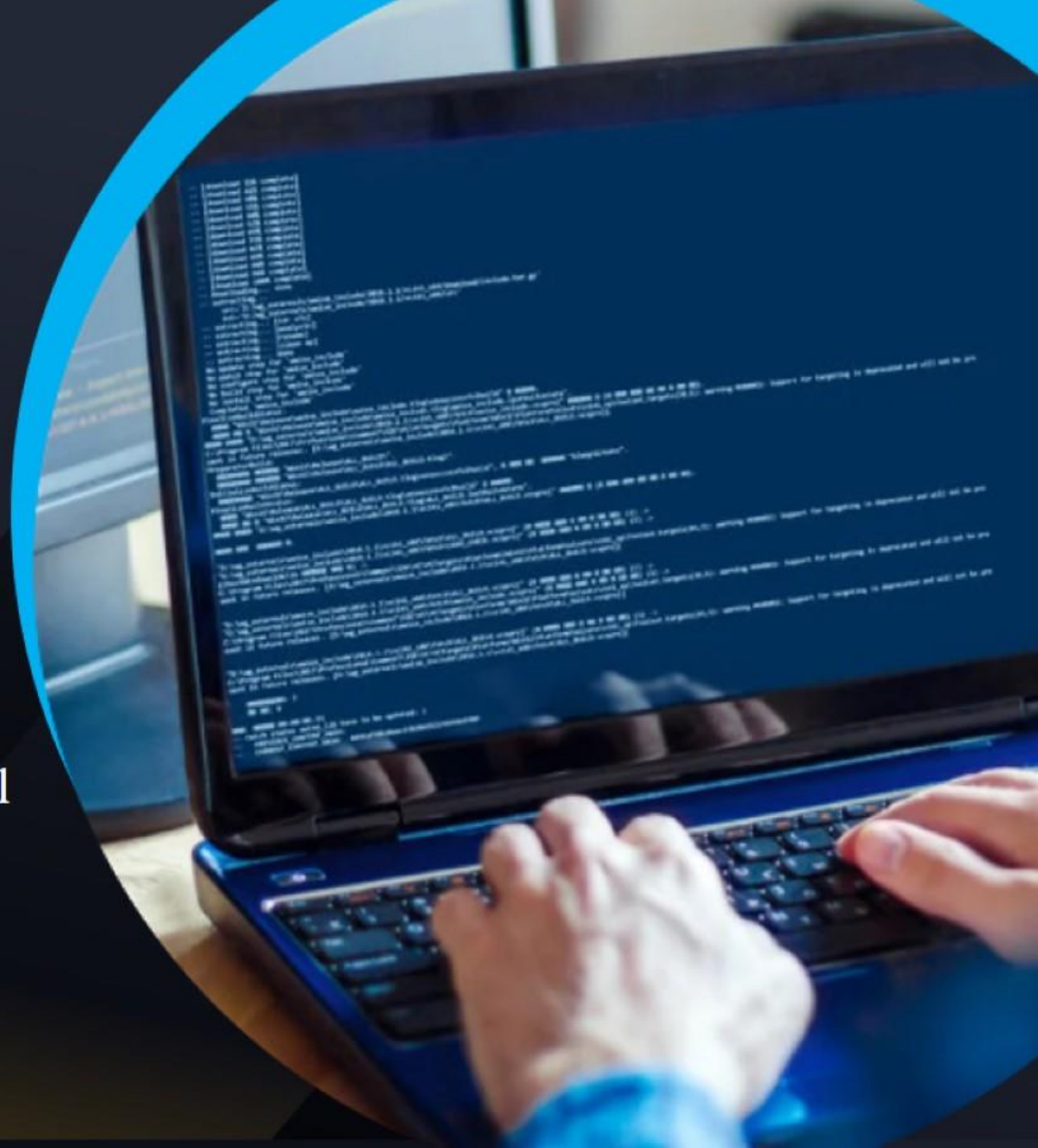
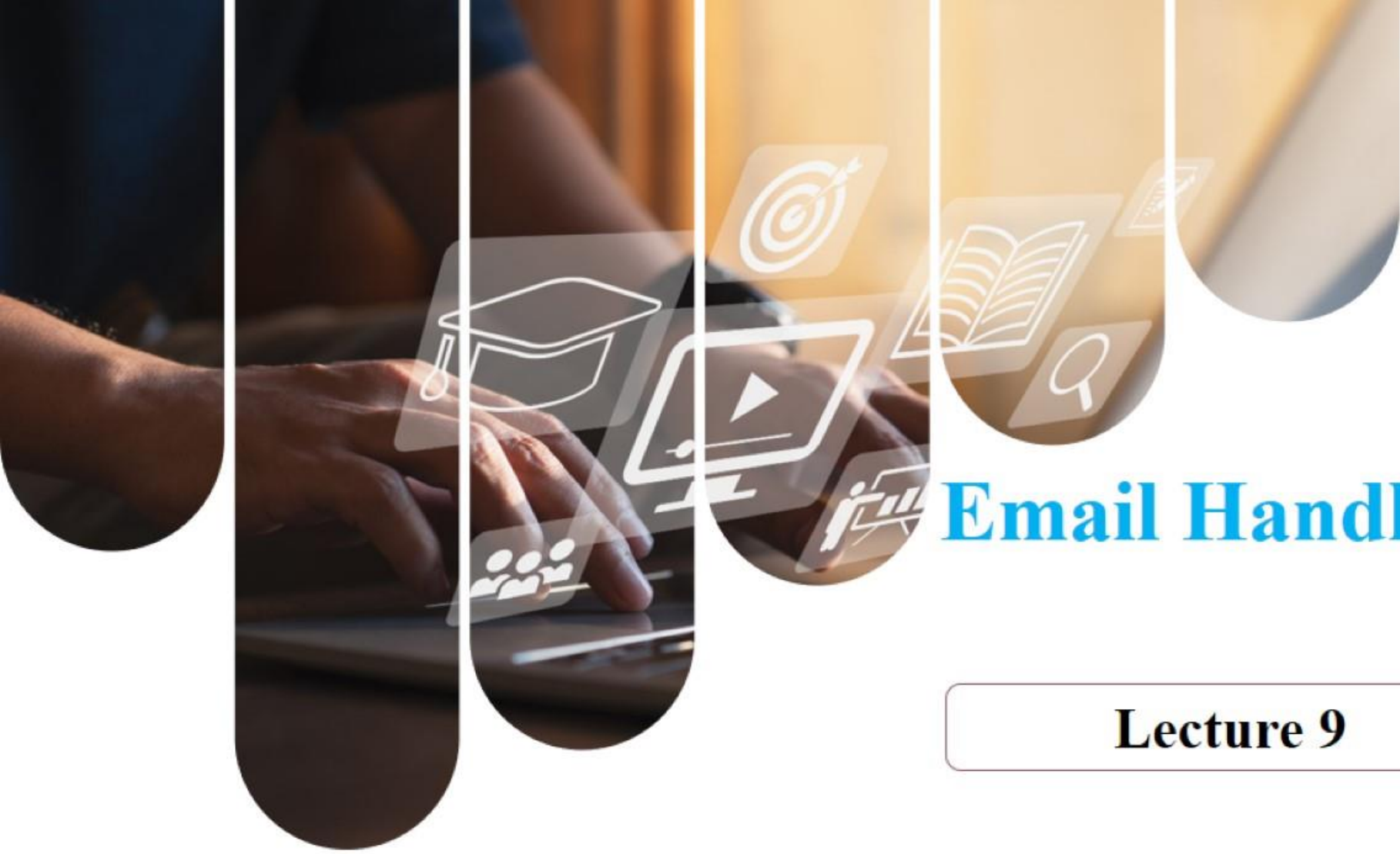


Network Programming in Python

Prepared by Dr. Hamada I. AbdulWakel

Lecture 9



The background features a series of vertical panels with rounded ends. The first panel on the left shows a person's hands typing on a laptop. The subsequent panels are semi-transparent and contain various white line-art icons: a graduation cap, a play button, a target with an arrow, an open book, a magnifying glass, and a group of people. The title 'Email Handling' is written in a large, bold, blue font across the middle of these panels.

Email Handling

Lecture 9

Prepared by
Dr. Hamada I. AbdulWakel

www.midocean.edu.km

Lecture

09

RECAP: LECTURE 8

- Email Messages.
- E-mail Construction in Python.

Lecture

09

LECTURE MAIN POINTS

- Email Protocols.
- Sending Emails.
- Securing Emails.
- Retrieving Emails.

Lecture

09

EMAIL PROTOCOLS

- **Simple Mail Transfer Protocol (SMTP):** It can only be used by the MTA for sending e-mails from one host to another.
- **Post Office Protocol 3 (POP3):** It provides a simple and standardized way for the users to gain access to the mailboxes and then download the messages.
- **Internet Message Access Protocol (IMAP):** It provides a simple and standardized way for accessing your e-mail. It requires only a small data transfer; this scheme works well even over a slow connection.

Lecture

09

POP3 VS. IMAP

- With IMAP:
 - All your mail stays on the server in multiple folders.
 - You can connect to any computer and see all your mail and mail folders.
 - If you have a dedicated connection to the Internet or you like to check your mail from various locations.
- With POP3:
 - You only have one folder, the Inbox folder.
 - When you open your mailbox, new mail is moved from the host server and saved on your computer.

Lecture

09

EMAIL MODULES

- Python has three modules:
 - **smtplib** for SMTP protocol.
 - **poplib** for POP3 protocol.
 - **imaplib** for IMAP protocol.
- Each module has options for transmitting the information securely by using the Transport Layer Security (TLS) protocol.
- Each protocol also uses some form of authentication for ensuring the confidentiality of the data.

Lecture

09

SMTPLIB METHODS

- **SMTP()**: returns an object of SMTP class.
- **ehlo()**: identifies the client to the server. An extended hello message will be sent by the client to the server.
- **starttls()**: starts unsecured SMTP connection with encryption for the traffic.
- **login()**: takes username and password arguments to login to SMTP server.

Lecture

09

SMTPLIB METHODS

Cont'd

- **sendmail()**: performs an entire mail transaction.
- **quit()**: terminates SMTP session.

Lecture

09

LOCAL SMTP SERVER

- Python comes with simple `smtpd` module which allows you to run a simple SMTP server easily.
- When doing email debugging, `smtpd` has extreme useful `DebuggingServer` feature making it dump all relayed email to stdout instead of relaying them forward for email delivery.
- Running a local debugging server means it's not necessary to deal with encryption of messages or use credentials to log in to an email server.

LOCAL SMTP SERVER

Cont'd

- Code:

```
import smtplib
session = smtplib.SMTP('localhost', 1025)
subject = 'Test'
body = "A test message to check the email sending using STMP"
msg = f"Subject: {subject} \n\n{body}"
```

```
session.sendmail("hamada@midocean.km", "hamada_mbsoft@gmail.com", msg)
```

Lecture

09

LOCAL SMTP SERVER

Cont'd

```
drhamada@drhamada-Latitude-E6520: ~  
File Edit View Search Terminal Help  
drhamada@drhamada-Latitude-E6520:~$ python3 -m smtpd -c DebuggingServer -n local  
host:1025  
/usr/lib/python3.10/smtpd.py:105: DeprecationWarning: The asyncore module is dep  
recated and will be removed in Python 3.12. The recommended replacement is asyn  
io  
    import asyncore  
/usr/lib/python3.10/smtpd.py:106: DeprecationWarning: The asynchat module is dep  
recated and will be removed in Python 3.12. The recommended replacement is asyn  
io  
    import asynchat  
----- MESSAGE FOLLOWS -----  
b'Subject: Test '  
b'X-Peer: 127.0.0.1'  
b''  
b'A test message to check the email sending using STMP'  
----- END MESSAGE -----
```

Lecture

09

EMAIL CLIENT APP

- Code:

```
import smtplib, imghdr
from email.message import EmailMessage
from email.utils import make_msgid
sender = input("Enter sender email address: ")
recipient = input("Enter recipient email address: ")
msg = EmailMessage()
msg['subject'] = 'A new message with attachment'
msg['from'] = sender
msg['to'] = recipient
```

- Code:

```
msg['Message-ID'] = make_msgid()
msg.set_content("Image attached ...")
with open('image.jpg', 'rb') as f:
    file = f.read()
    fileType = imghdr.what(f.name)
    print(fileType)
    fileName = f.name
msg.add_attachment(file, maintype='image', subtype=fileType, filename
= fileName)
```

Lecture

09

EMAIL CLIENT APP

Cont'd

- Code:

Incoming Mail (IMAP) Server

imap.gmail.com Requires SSL: Yes Port: 993

Outgoing Mail (SMTP) Server

smtp.gmail.com Requires SSL: Yes Requires
TLS: Yes (if available) Requires Authentication:
Yes Port for SSL: 465 Port for TLS/STARTTLS:
587

```
mailserver = 'smtp.gmail.com'  
mailport = 587  
session = smtplib.SMTP(mailserver, mailport)  
session.ehlo()  
session.sendmail(sender, recipient, msg.as_string())  
print(f"You email is sent to {recipient}.")  
session.quit()
```

Lecture

09

EMAIL CLIENT APP

Cont'd

```
drhamada@drhamada-Latitude-E6520:/mnt/3126DF085226C5FC$ /bin/python3 /mnt/3126
Enter sender email address: hamada@esu.ac.ae
Enter recipient email address: hamada_mbsoft@yahoo.com
Enter your email subject: test
Enter your email message. Press Enter when finished. test
Traceback (most recent call last):
  File "/mnt/3126DF085226C5FC/Codes/server.py", line 31, in <module>
    send_email(sender, recipient)
  File "/mnt/3126DF085226C5FC/Codes/server.py", line 24, in send_email
    session.sendmail(sender, recipient, msg.as_string())
  File "/usr/lib/python3.10/smtplib.py", line 887, in sendmail
    raise SMTPSenderRefused(code, resp, from_addr)
smtplib.SMTPSenderRefused: (530, b'5.7.0 Must issue a STARTTLS command first.
.47 - gsmtplib', 'hamada@esu.ac.ae')
```

Lecture

09

LESS SECURE APPS

- An App Password gives permission to a non-Google app to access Google's SMTP server.
- App Passwords can only be used with accounts that have 2-step verification turned on.

To help keep your account secure, from **May 30, 2022**, Google no longer supports the use of third-party apps or devices which ask you to sign in to your Google Account using only your username and password.

Important: This deadline does not apply to Google Workspace or Google Cloud Identity customers. The enforcement date for these customers will be announced on the Workspace blog at a later date.

For more information, continue to read.

- Visit: <https://support.google.com/accounts/answer/6010255?hl=en>

LESS SECURE APPS

Cont'd

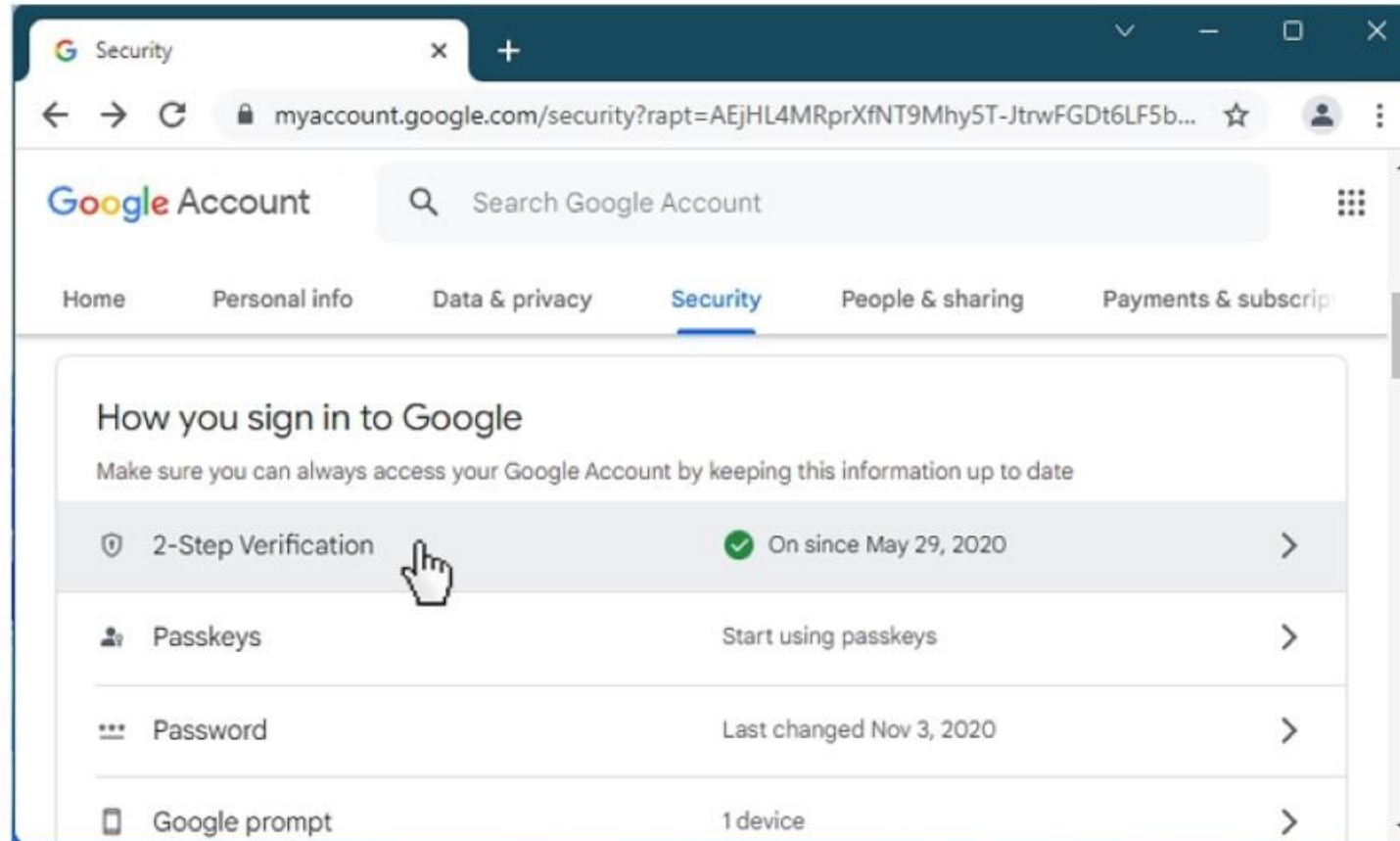
- To use Google's SMTP server for sending email with Automation Workshop, follow the very simple set-up process:
 - Open Google Account Security settings.
 - Select 2-Step Verification option from the How to sign in to Google section.
 - Select App passwords option (at the very bottom of the page).
 - Click on the Select App field, and choose the Other (Custom name) option.
 - Enter an easily recognizable name, and click the Generate button.
 - Write down the generated App password.
 - From now on, for the SMTP settings use this 16-character password (without spaces) along with your complete Gmail address.

Lecture

09

LESS SECURE APPS

Cont'd

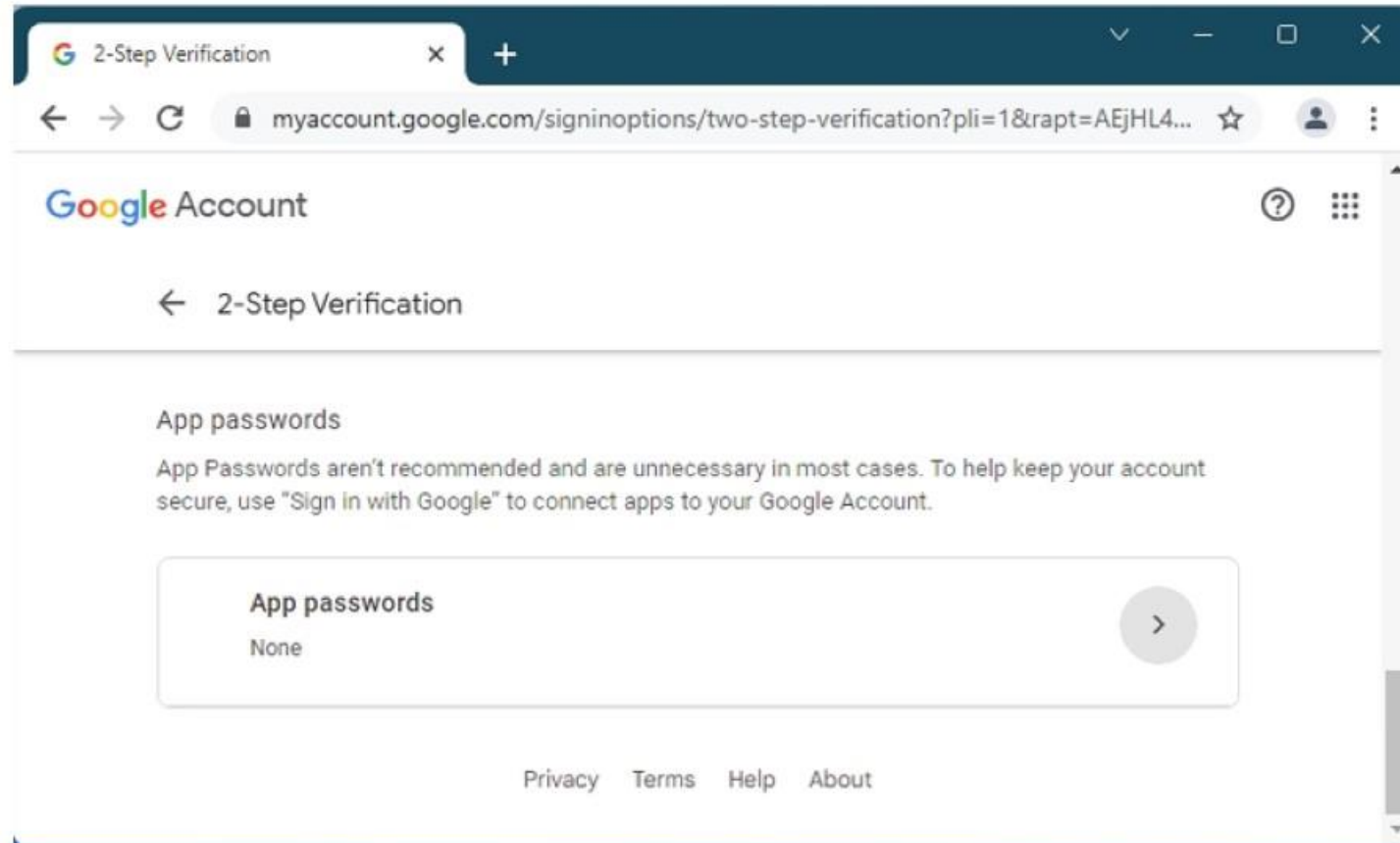


Lecture

09

LESS SECURE APPS

Cont'd

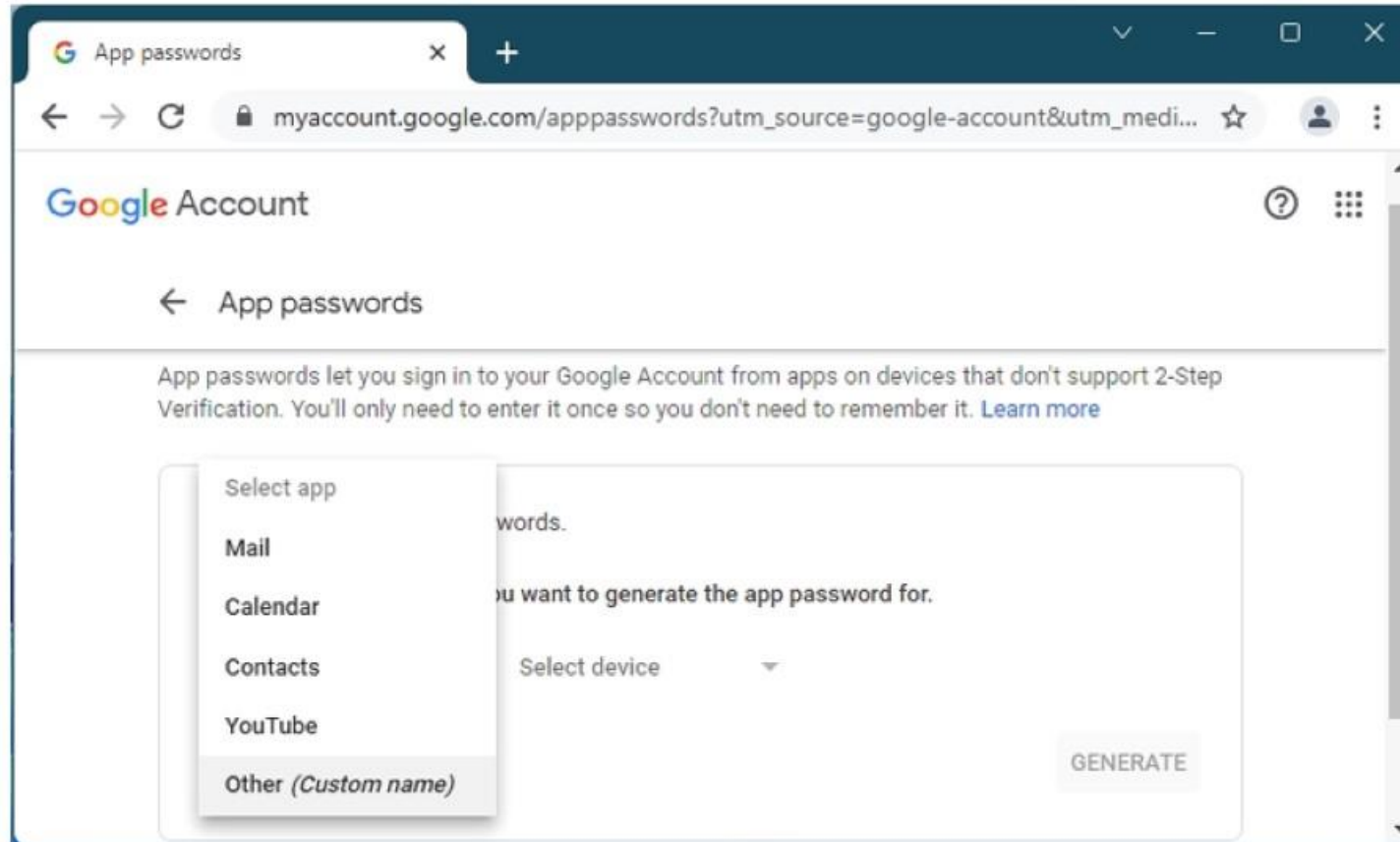


Lecture

09

LESS SECURE APPS

Cont'd



The screenshot shows a web browser window with the address bar displaying "myaccount.google.com/apppasswords?utm_source=google-account&utm_medi...". The page title is "App passwords" and the Google Account logo is visible. The main heading is "App passwords" with a back arrow. Below this, a paragraph explains: "App passwords let you sign in to your Google Account from apps on devices that don't support 2-Step Verification. You'll only need to enter it once so you don't need to remember it. [Learn more](#)".

The form contains two sections:

- Select app:** A dropdown menu with the following options: Mail, Calendar, Contacts, YouTube, and Other (Custom name). The "Other (Custom name)" option is currently selected.
- Select device:** A dropdown menu with a downward arrow, currently empty.

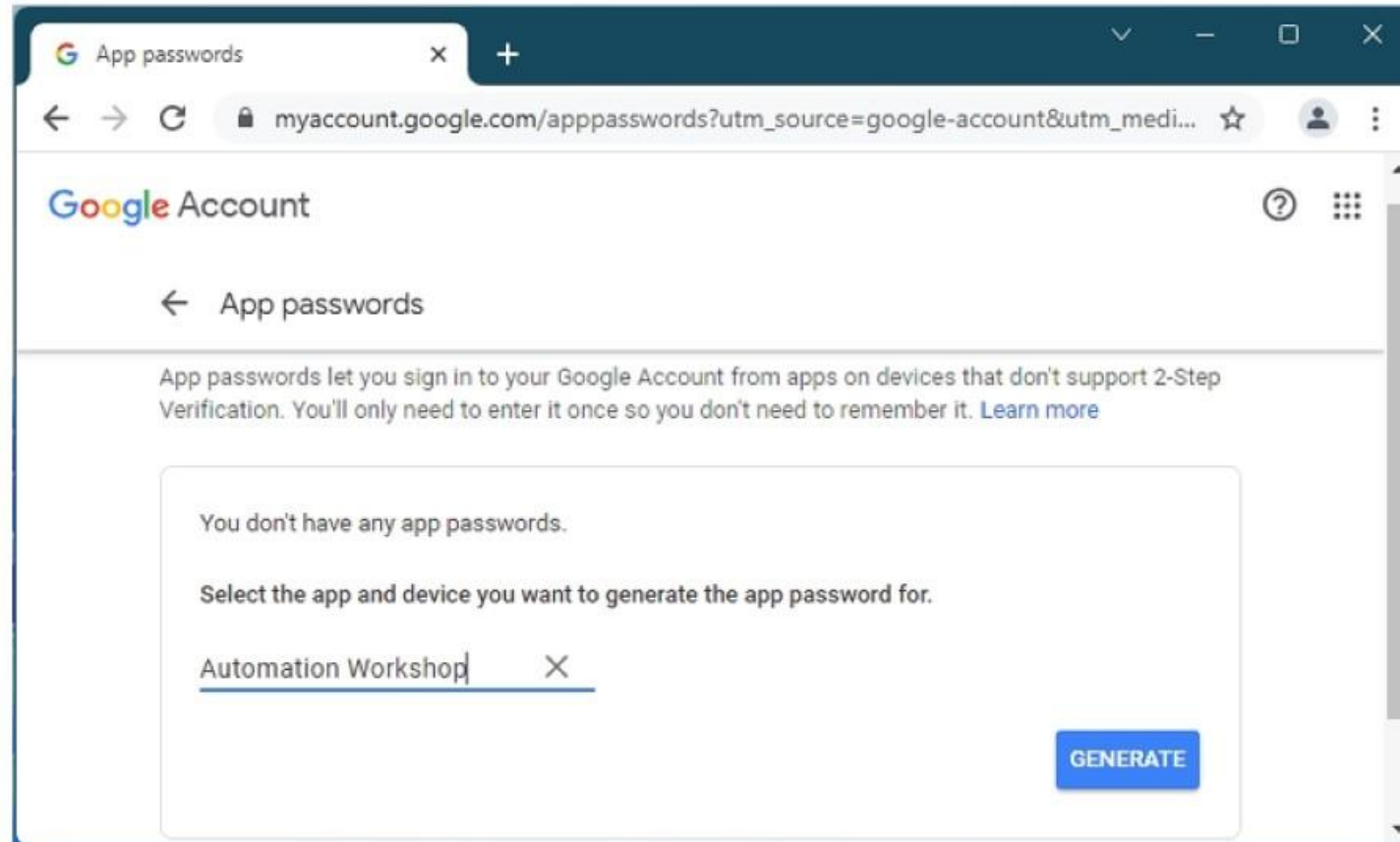
A "GENERATE" button is located at the bottom right of the form.

Lecture

09

LESS SECURE APPS

Cont'd

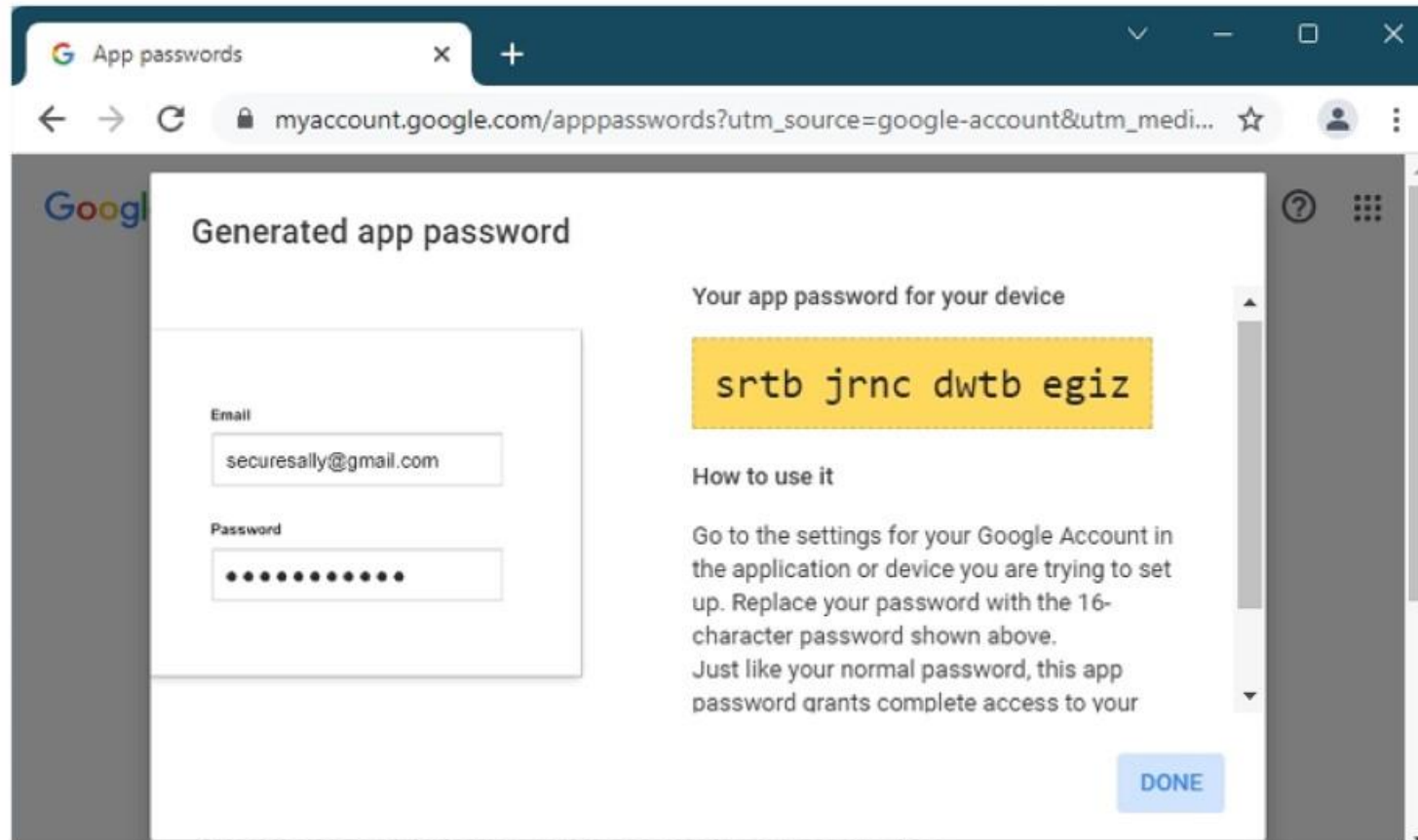


Lecture

09

LESS SECURE APPS

Cont'd



Lecture

09

EMAIL CLIENT APP

- Code:

```
import getpass
session = smtplib.SMTP(mailserver, mailport)
session.ehlo()
session.starttls()
session.ehlo()      # re-identify themselves as encrypted connection
# "tgrjnuccuhsjjzji"
password = getpass.getpass(prompt="Enter you email password: ")
session.login(sender, password)
```

Lecture

09

POP3 METHODS

- **POP3()**: creates a POP3 object.
- **POP3_SSL()**: creates a secured POP3 object.
- **user()**: sets the username to get the mailbox.
- **pass_()**: sets the password for the username.
- **stat()**: returns mailbox status in a tuple format (message count, mailbox size).

POP3 METHODS

Cont'd

- **list():**
 - Retrieves a list of messages.
 - Returns a tuple of:
(responsecode, msglist, sizeofresponse)
- **retr():**
 - Retrieves a single message.
 - Returns a tuple of:
(responsecode, lines, size)
 - Marks the Email as read.

Lecture

09

EMAIL ACCESS CLIENT APP

- Code:

```
import getpass, poplib
popserver = 'pop.gmail.com'
popport = 995
username = input("Enter your email user ID: ")
# "tgrjnuccuhsjjzji"
password = getpass.getpass(prompt="Enter your email password: ")
mailbox = poplib.POP3_SSL(popserver , popport)
mailbox.user(username)
mailbox.pass_(password)
```

EMAIL ACCESS CLIENT APP

Cont'd

- Code:

```
num_messages = len(mailbox.list()[1])
print(f"Total emails: {num_messages}")
print("Getting last message")
for msg in mailbox.retr(num_messages)[1]:
    print(msg)
mailbox.quit()
```

Lecture

09

IMAP METHODS

- **IMAP4_SSL()**: creates a secure IMAP object.
- **login()**: logs in to the server.
- **list()**:
 - Lists the folders present in mailbox.
 - Returns a tuple of:
(**'Ok'**, **[list]**)

- **select():**
 - Selects a specific mail folder to get messages from it.
 - Returns a tuple of:
(`'Ok'`, [list of one item specify number of emails in this folder])
- **fetch():**
 - Returns a particular message.
 - The 1st argument is the number of message, the second is the parsing format.
 - Returns a tuple of:
(`'Ok'`, [list of message header]).

Lecture

09

EMAIL ACCESS CLIENT APP

- Code:

```
import imaplib, email
imapServer = 'imap.gmail.com'
sender = 'hamada.mbsoft@gmail.com'
password = 'tgrjnuccuhsjjzji'
conn = imaplib.IMAP4_SSL(imapServer)
conn.login(sender, password)
print(conn.list())
print(conn.select('INBOX'))
```

EMAIL ACCESS CLIENT APP

Cont'd

- Code:

```
conn.select('INBOX')
typ, data = conn.fetch(b'6433','(RFC822)')
print(data[0])
print(data[0][1])
# convert data into email message object
msg = email.message_from_string(data[0][1].decode('utf-8'))
print("To:", msg['to'])
print("From:", msg['from'])
```

EMAIL ACCESS CLIENT APP

Cont'd

- Code:

```
print("Subject: ", msg['subject'])
# use walk() to get message parts and check its content type
for p in msg.walk():
    if p.get_content_type() == 'text/plain':
        body = p.get_payload()
print(body)
```

Lecture

09

QUESTIONS FOR ONLINE DISCUSSION

- Write a program to create your own Mail Client and use it to send e-mail with attachment to "hamada@midocean.km".



Thank You

