Automated Emails and Adding Attachments

Sidheswar Routray
Department of Computer Science & Engineering
School of Technology, PDEU

Introduction

- Automated emails are widely used for notifications, reports, or alerts in applications and systems.
- Python offers various libraries to automate sending emails, such as smtplib and email.
- These libraries allow the automation of routine tasks like sending emails with custom content, adding attachments, and managing email formats (HTML, plain text).

Setting Up Email Automation

Libraries Required:

- **smtplib**: Handles sending emails using the Simple Mail Transfer Protocol (SMTP).
- **email.mime**: Helps in creating email content, such as text, HTML, and attachments.

Example Workflow:

- Establish a connection with an SMTP server.
- Create an email object with the necessary content.
- Send the email using the SMTP server.

SMTP Servers

Common SMTP servers

- Gmail: smtp.gmail.com (Port 587 for TLS, Port 465 for SSL)
- Yahoo: smtp.mail.yahoo.com
- Outlook: smtp-mail.outlook.com

SMTP Setup for Gmail

```
import smtplib
# Email settings
smtp_server = 'smtp.gmail.com'
port = 587  # For TLS
sender_email = "your_email@gmail.com"
password = "your_password"
# Create connection to the SMTP server
server = smtplib.SMTP(smtp_server, port)
server.starttls()  # Secure the connection
server.login(sender_email, password)
```

SMTP Setup for Gmail

- The smtplib library in Python is used to send emails via the Simple Mail Transfer Protocol (SMTP). It provides all the necessary functions to set up an SMTP connection and send emails.
- smtp_server = 'smtp.gmail.com': This specifies the SMTP server you will use to send emails. Gmail's SMTP server is 'smtp.gmail.com'.
- port = 587: This specifies the port number to use for the SMTP connection. Port 587 is the standard port used for SMTP connections that use TLS (Transport Layer Security), which encrypts the communication.
- sender_email = "your_email@gmail.com": This is the email address from which the email will be sent.
- password = "your_password": This is the password for the sender's email account.

SMTP Setup for Gmail

- smtplib.SMTP(smtp_server, port): This creates an SMTP client session object to connect to the Gmail SMTP server using the specified port (587). This object will handle sending the emails.
- starttls(): This upgrades the connection to use TLS (Transport Layer Security). TLS ensures that the communication between your script and the SMTP server is encrypted, providing security against eavesdropping.
- login(sender_email, password): This logs in to the Gmail SMTP server using the sender's email and password. This step is necessary for Gmail to authenticate the user and allow the email to be sent.

Composing and Sending Emails

```
from email.mime.text import MIMEText
# Create the message body
subject = "Automated Email"
body = "This is an automated email sent from Python."
message = MIMEText(body, "plain")  # Email body as plain text
message["Subject"] = subject
message["From"] = sender_email
message["To"] = "recipient_email@example.com"
# Send the email
server.sendmail(sender_email,"recipient_email@example.com",
message.as_string())
server.quit()  # Close the server connection
```

Composing and Sending Emails

- MIMEText is a class from the email.mime.text module that allows you to create an email with a plain text body (or HTML if desired).
- subject = "Automated Email": This defines the subject of the email.
- body = "This is an automated email sent from Python.": This defines the content or body of the email.
- message["Subject"] = subject: This sets the subject of the email.
- message["From"] = sender_email: This sets the From field of the email to the sender's email address.
- message["To"] = "recipient_email@example.com": This sets the To field of the email to the recipient's email address.
- server.sendmail(sender_email, "recipient_email@example.com", message.as_string()): This sends the email.
- sender_email: The email address from which the email is being sent."recipient_email@example.com": The recipient's email address.
- message.as_string(): Converts the MIMEText message object to a string so that it can be sent via the sendmail() method.

Adding Attachments to Emails

- To send an email with attachments, we need to use the email.mime.multipart and email.mime.base modules.

Steps:

1. Create a MIMEMultipart object to hold the email content.
2. Attach the email body (text or HTML) to the object.
3. Add files using MIMEBase and encode them with base64 encoding.
4. Attach these files to the email.

Adding Attachments to Emails

```
import os
From email.mime.multipart import MIMEMultipart
from email.mime.base import MIMEBase
from email import encoders
# Create the email object
message = MIMEMultipart()
message["From"] = sender_email
message["To"]= "recipient_email@example.com"
message["Subject"] = "Automated Email with Attachment"
# Add body text
body = "Please find the attachment below."
message.attach(MIMEText(body, "plain"))
```

```
# Specify the file to attach
filename = "example.pdf"  # File to be attached
filepath = os.path.join("path/to/file", filename)
# Open the file in binary mode and attach it
with open(filepath, "rb") as attachment:
    part = MIMEBase("application", "octet-stream")
    part.set_payload(attachment.read())
# Encode the file in base64 and add it to the email
encoders.encode_base64(part)
part.add_header("Content-Disposition", f"attachment; filename={filename}")
message.attach(part)
# Send the email
server.sendmail(sender_email, "recipient_email@example.com", message.as_string())
server.quit()
```

Adding Attachments to Emails

- os: This library is used to manipulate file paths and handle operating system functions.
- MIMEMultipart: This class is used to create a container (multipart email) that can hold multiple parts (like the email body and attachments).
- MIMEBase: A generic container class for attachments, it specifies the type of file being attached (e.g., application, audio, image).
- encoders: Used to encode the attachment into a format suitable for sending (like base64).
- message["From"], message["To"], message["Subject"]: These lines set the "From", "To", and "Subject" headers for the email.
- MIMEText(body, "plain"): This creates a plain text part for the email body. In this case, the body just contains a message asking the recipient to find the attached file.
- message.attach(): This attaches the plain-text body to the MIMEMultipart email object.
- filename = "example.pdf": Specifies the name of the file to be attached.
- filepath = os.path.join("path/to/file", filename): Uses os.path.join() to construct the full path to the file. This is useful for ensuring compatibility across different operating systems.

Adding Attachments to Emails

- with open(filepath, "rb") as attachment:: Opens the file in binary mode ("rb" stands for "read binary") so that it can be attached.
- MIMEBase("application", "octet-stream"): Creates a new MIMEBase object to hold the file. The "application" and "octet-stream" arguments specify that this is a generic binary file (PDF, in this case).
- part.set_payload(attachment.read()): Reads the contents of the file and sets it as the payload of the MIMEBase object.
- encoders.encode_base64(part): Encodes the file content in base64, which is a standard encoding format used to safely send binary data as plain text in emails.part.
- add_header("Content-Disposition", f"attachment; filename={filename}"): Adds a header specifying that this part is an attachment. The filename tells the recipient's email client the name of the file being attached.
- message.attach(part): Finally, this attaches the encoded file to the email.
- server.sendmail(sender_email, "recipient_email@example.com", message.as_string()): This sends the email using the sendmail() function from the smtplib server object. The message.as_string() method converts the entire MIMEMultipart object (with the body and attachment) into a string that can be sent via SMTP.

Sending HTML Emails

```
from email.mime.text import MIMEText
# HTML content
html_content = """
<html>
  <body>
    <h1>This is an HTML email</h1>
    <p>This email was sent from a Python script.</p>
  </body>
</html>
"""
# Create the MIMEText object for HTML content
message = MIMEMultipart("alternative")
message["Subject"] = "Automated HTML Email"
message["From"] = sender_email
```

```
# Attach the HTML content to the email
message.attach(MIMEText(html_content
, "html"))
# Send the email
server.sendmail(sender_email,
"recipient_email@example.com",
message.as_string())
server.quit()
```

Sending HTML Emails

- **MIMEText("html")** specifies that the email content is in HTML format.
- **MIMEMultipart("alternative")** is used when we have different formats for the same content (e.g., plain text vs. HTML).