```
#%% md
# Report of Automating Email Sending Script
#%%
import json
import smtplib, ssl
import time
import os
import schedule
from datetime import datetime
from email.mime.base import MIMEBase
from email.mime.text import MIMEText
from email.mime.multipart import MIMEMultipart
from email import encoders
#%% md
## Tasks:
<ul>
    <li>Use the smtplib library to connect to the email server and send the emails.</li>
    <li>Use the email library to compose the email, including the recipient's email address, the subject, and the body of the email.</li>
    <li>Use the os library to access the report files that need to be sent.</li>
    <li>Use a for loop to iterate through the list of recipients and send the email and attachment.</li>
    <li>Use the schedule library to schedule the script to run daily at a specific time.</li>
    <li>You can also set up a log file to keep track of the emails that have been sent and any errors that may have occurred during the email sending process.</li>
</ul>
#%% md
### 1. Use the smtplib library to connect to the email server and send the emails.
#%% md
<p style="font-size:20">   First, we use <i>ssl.create_default_context()</i> to validate the host name and its certificates and optimize the security connection (helps to send mails to inbox, not to junk). Actually, <i>create_default_context()</i> is a
```

```
27 helper function, which returns a new context with
   secure default settings.
28 <i>smtplib.SMTP("smtp.gmail.com", 587)</i> is used to
    initialize connection to the server, address and
   port were taken from Google web-site.
29 <i>starttls(context=context)</i> puts the SMTP
   connection in TLS (Transport Layer Security) mode.
   All SMTP commands that follow will be encrypted. You
   should then call <i>ehlo()</i> again. <i>ehlo()</i>
   identifies yourself to an ESMTP server using EHLO.
30 <i>server.login(sender_email, password)</i> log in on
    an SMTP server that requires authentication. The
   arguments are the username and the password to
   authenticate with.</p>
31 #%%
32 sender_email = "tbrainnest@gmail.com"
33 password = "iufsuofxakrcgzci"
34 receiver_email_list = ["recipient1@example.com", "
   recipient2@example.com", "recipient3@example.com"]
35 # Create secure connection with server and send email
36 context = ssl.create_default_context()
37
38 # Connection to the server with secure protocol
39 with smtplib.SMTP("smtp.gmail.com", 587) as server:
40     server.starttls(context=context)
41     server.ehlo()
42     server.login(sender_email, password)
43 #%% md
44 ### 2. Use the email library to compose the email,
   including the recipient's email address, the subject
   , and the body of the email.
45 #%% md
46 <p style="font-size:20">   We use built-in
   package, which allows us to structure more fancy
   emails, which can be transferred with smtp library. <
   i>MIMEMultipart</i> initialize an Multipurpose
   Internet Mail Extensions (MIME) object which supports
    multiparts. Multipart option gives us opportunity to
    use method <i>attach()</i>. <i>MIMEText()</i>
   initialize a MIME object, which can be attached to
   the MIMEMultipart object</p>
```

```
#%%
body = f'''\
Hello,

Please find a report in the attachment.

Best regards,
{sender}
'''

msg = MIMEMultipart()
msg['From'] = sender
msg['To'] = recipient #Or if we do not want for loops , we can use ", ".join(recipiets)
msg['Subject'] = "Daily Report"

msg.attach(MIMEText(body, 'plain'))
#%% md
### 3. Use the os library to access the report files that need to be sent.
#%% md
<p style="font-size:20">   We use <i>os.getcwd()</i> to get current directory. <i>MIMEBase()</i> is  used to create a base MIME object, to which we can set our attachment as a payload. Then we convert binary to ASCII symbols. Give a header, description and filename to our MIME based and attach it to the maim MIME object.<p>
#%%
filename = str(os.getcwd()) + "\\daily_report.pdf"
with open(filename, "rb") as attachment:
    payload = MIMEBase('application', 'octate-stream')
    payload.set_payload(attachment.read())

# enconding the binary into ASCII
encoders.encode_base64(payload)

# add header with pdf name
payload.add_header(
"Content-Disposition",
```

```
 79 f"attachment; filename= daily_report.pdf",
 80 )
 81
 82 msg.attach(payload)
 83 #%% md
 84 ### 4. Use a for loop to iterate through the list of
     recipients and send the email and attachment.
 85 #%% md
 86 <p style="font-size:20">   <i>get_message()</i
    > returns us a MIME object, body of the message and
    file directory. Then we use <i>.sendmail(sender,
    reciever, message)</i> to send our message</p>
 87 #%%
 88  for receiver in receiver_email_list:
 89     # Get a message context
 90     message, body_msg, dir_file = get_message(
    sender_email, receiver)
 91     # Send a message via email
 92     server.sendmail(sender_email, receiver, message.
    as_string())
 93 #%% md
 94 Here is an example of sending message:
 95 #%% md
 96 <img src="images\mails.PNG" alt="Alternative text
    " />
 97 #%% md
 98 ### 5. Use the schedule library to schedule the
    script to run daily at a specific time.
 99 #%% md
100 <p style="font-size:20">   We use <i>schedule
    library</i> to schedule an event at some moment. Our
     event is a method <i>send_daily_report()</i>, which
     send a report to our receivers. We schedule this
    event every day at 20:00 of local machine time. Then
     we use infinite loop to not finish the execution
    and send daily report, when our conditions are met.
101 #%%
102 # Set up schedule
103 schedule.every().day.at("20:00").do(
    send_daily_report)
104 while True:
```

```
    schedule.run_pending()
    time.sleep(1)
#%% md
### 6. You can also set up a log file to keep track of the emails that have been sent and any errors that may have occurred during the email sending process.
#%% md
#### Log tracking
<p style="font-size:20">   We use a json format to write our logs. So, first, we open the json file <i>open(str(os.getcwd()) + "\\log.json", 'r')</i> and load json from this file <i>json.load()</i>. JSON file contains a list mails, which contains dictinaries with message information (From, To, Subject, Message and which file we attach). So, every time when we send a message, we append to a list our dictinary list. When a sending is done, we convert data to JSON format <i>json.dumps()</i> and save it <i>log.write(json_object)</i>.
#%%
 # open log file
    with open(str(os.getcwd()) + "\\log.json", 'r') as log:
        data = json.load(log)

    for receiver in receiver_email_list:
        # Get a message context
        message, body_msg, dir_file = get_message(sender_email, receiver)
        # Send a message via email
        server.sendmail(sender_email, receiver, message.as_string())
        # Add a message to log data
        data['Mails'].append({"From": message['From'], "To": message['To'], "Subject": message['Subject'], "Message": body_msg, "File directory": dir_file})
    json_object = json.dumps(data)

    # Write sended messages to log file
    with open(str(os.getcwd()) + "\\log.json", 'w')
```

```
as log:
        log.write(json_object)
#%% md
Here is an example of JSON file:
#%% md
<img src="images\json.PNG" alt="Alternative text" />
#%% md
#### Error tracking
#%% md
<p style="font-size:20">   We use try and except to catch any errors, which appear in our code. When an error is occured, we get local machine time and error description and then creat and save all information.
#%%
try:
    # Set up schedule
    schedule.every().day.at("20:00").do(send_daily_report)
    while True:
        schedule.run_pending()
        time.sleep(1)
except Exception as e: # Catch errors in the code

    # Get current time
    now = datetime.now()
    dt_string = now.strftime("%d/%m/%Y %H:%M:%S")

    # Write an error with the time and description
    with open('error.txt', 'w') as f:
        f.write(f'''The error is occurred at {dt_string}.
        The reason is {e}''')
#%% md
Here is an example of Error file:
#%% md
<img src="images\error.PNG" alt="Alternative text" />
```