AutomatingEmailSendingReport

January 30, 2023

1 Report of Automating Email Sending Script

1.1 Tasks:

Use the smtplib library to connect to the email server and send the emails.

Use the email library to compose the email, including the recipient's email address, the subject, and the body of the email.

Use the os library to access the report files that need to be sent.

Use a for loop to iterate through the list of recipients and send the email and attachment.

Use the schedule library to schedule the script to run daily at a specific time.

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.

1.1.1 1. Use the smtplib library to connect to the email server and send the emails.

First, we use ssl.create_default_context() to validate the hostname and its certificates and optimize the security connection (helps to send emails to the inbox, not to junk). Actually, create_default_context() is a helper function, which returns a new context with secure default settings. smtplib.SMTP("smtp.gmail.com", 587) is used to initialize a connection to the server, the address and port were taken from the Google website. starttls(context=context) puts the SMTP connection in TLS (Transport Layer Security) mode. All SMTP commands that follow will be encrypted. You should then call ehlo() again. ehlo() identifies yourself to an ESMTP server using EHLO. server.login(sender_email, password) log in on an SMTP server that requires authentication. The arguments are the username and the password to authenticate with.

[ ]:
```
sender_email = "tbrainnest@gmail.com"
password = "iufsuofxakrcgzci"
receiver_email_list = ["recipient1@example.com", "recipient2@example.com",␣
 ↪"recipient3@example.com"]
# Create secure connection with server and send email
context = ssl.create_default_context()

# Connection to the server with secure protocol
with smtplib.SMTP("smtp.gmail.com", 587) as server:
    server.starttls(context=context)
    server.ehlo()
    server.login(sender_email, password)
```

1.1.2 2. Use the email library to compose the email, including the recipient's email address, the subject, and the body of the email.

We use a built-in package, which allows us to structure more fancy emails, which can be transferred with smtp library. MIMEMultipart initializes a Multipurpose Internet Mail Extensions (MIME) object which supports multipart. A multipart option gives us an opportunity to use the method attach(). MIMEText() initialize a MIME object, which can be attached to the MIMEMultipart object

```
[ ]: 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'))
```

1.1.3 3. Use the os library to access the report files that need to be sent.

We use os.getcwd() to get current directory. MIMEBase() 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.

```
[ ]: 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",
f"attachment; filename= daily_report.pdf",
)
```

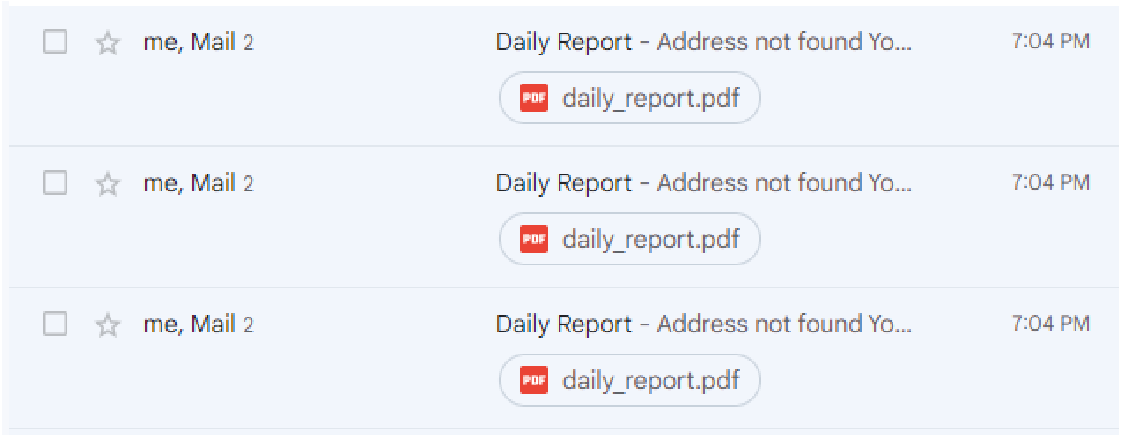
```
msg.attach(payload)
```

1.1.4 4. Use a for loop to iterate through the list of recipients and send the email and attachment.

get_message() returns us a MIME object, the body of the message and the file directory. Then we use .sendmail(sender, receiver, message) to send our message

[ ]:
```
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())
```

Here is an example of sending messages:


me, Mail 2 Daily Report - Address not found Yo... 7:04 PM daily_report.pdf

me, Mail 2 Daily Report - Address not found Yo... 7:04 PM daily_report.pdf

me, Mail 2 Daily Report - Address not found Yo... 7:04 PM daily_report.pdf

1.1.5 5. Use the schedule library to schedule the script to run daily at a specific time.

We use the schedule library to schedule an event at some moment. Our event is a method send_daily_report(), which sends a report to our receivers. We schedule this event every day at 20:00 of local machine time. Then we use an infinite loop to not finish the execution and send daily reports, when our conditions are met.

[ ]:
```
# Set up schedule
schedule.every().day.at("20:00").do(send_daily_report)
while True:
    schedule.run_pending()
    time.sleep(1)
```

1.1.6 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.

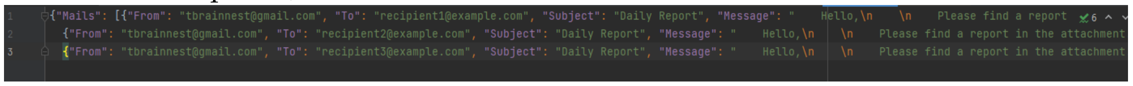
Log tracking We use a JSON format to write our logs. So, first, we open the JSON file open(str(os.getcwd()) + "\log.json", 'r') and load json from this file json.load(). JSON file contains a list of mails, which contains dictionaries with message information (From, To, Subject, Message and which file we attach). So, every time when we send a message, we append it to a list our dictionary list. When a sending is done, we convert data to JSON format json.dumps() and save it log.write(json_object).

```
[ ]: # 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)
```

Here is an example of JSON file:



```
{"Mails": [{"From": "tbrainnest@gmail.com", "To": "recipient1@example.com", "Subject": "Daily Report", "Message": "    Hello,\n    \n    Please find a report
  {"From": "tbrainnest@gmail.com", "To": "recipient2@example.com", "Subject": "Daily Report", "Message": "    Hello,\n    \n    Please find a report in the attachment
  {"From": "tbrainnest@gmail.com", "To": "recipient3@example.com", "Subject": "Daily Report", "Message": "    Hello,\n    \n    Please find a report in the attachment
```

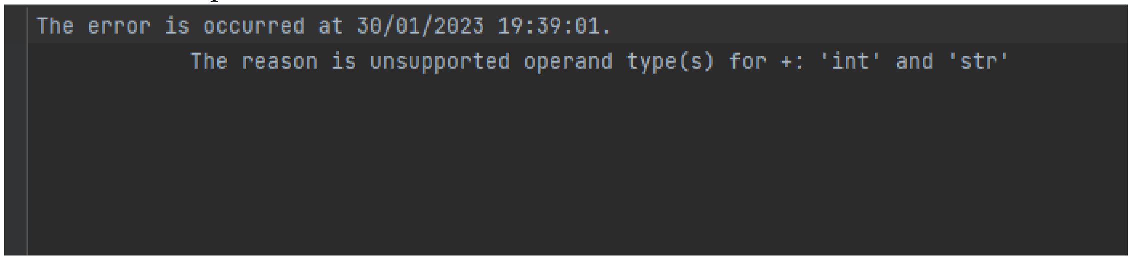
Error tracking We use to try and except to catch any errors, which appear in our code. When an error occurs, we get the local machine time and error description and then create and save all information.

```
[ ]: 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}''')
```

Here is an example of an Error file:



```
The error is occurred at 30/01/2023 19:39:01.
            The reason is unsupported operand type(s) for +: 'int' and 'str'
```