**《Python自动化运维》课程**

**期末学习报告**

班级：

姓名：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

学号：\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

1. **任务背景与需求分析（宋体，加粗，小三号字）**

1．任务背景（宋体，四号字）

最近公司的员工们开始逐步返回办公室，但天气亦开始转换，公司未设备自行购买雨具，因此希望有一种方法在工作日记起床之前知道当天的天气情况是否需要携带雨具等物品。

考虑到公司需要以邮件的形式提供此类信息，你决定编写一段 Python 自动化程序。你将使用一个特定的天气 API 来获取当日的天气预报，并将其嵌入到邮件正文中。你还将编写一个配置文件，以便程序每天自动发送邮件。每天早上 8 点，你的程序将自动运行，并将当天的天气预报发送到指定的公司电子邮箱。

2. 需求分析

**本任务需要实现一个自动化测试任务，需求如下：**

**1，天气 API：选择一个合适的天气 API，确保其提供每日的天气预报数据，包括温度、降水概率等信息。你需要在该 API 上注册并获取 API 密钥。也可以使用requests库进行爬取天气日报数据。**

**2，Python 编程环境：你需要安装 Python，并确保可以运行 Python 脚本。**

**3，邮件发送库：选择一个适合的 Python 邮件发送库，例如 smtplib、email 等，用于发送邮件。确保你已经安装了所选库。**

**4，配置文件：创建一个配置文件，例如 JSON 或 INI 格式，用于存储收件人的电子邮箱地址和发送时间等信息。配置文件应该具有可读性，并在程序中能够轻松地读取和解析。**

**5，编写脚本：编写一个 Python 脚本，该脚本将完成以下任务：**

**从配置文件中读取收件人的电子邮箱地址和发送时间等信息。**

**使用天气 API 获取当天的天气预报数据。**

**构建邮件正文，将天气预报数据嵌入邮件中。**

**使用选定的邮件发送库连接到邮件服务器，并发送邮件给指定的收件人。**

**在程序运行结束后，输出日志以供参考。**

**6，定时任务：将该脚本配置为每天定时运行。你可以使用操作系统提供的定时任务工具（如 cron）或使用第三方库（如 schedule）来实现定时任务的调度。**

1. **设计方案**
2. 程序结构  
   创建一个主程序文件，例如 weather\_forecast.py。

创建其他必要的辅助模块或类，用于处理配置文件、调用天气 API、构建邮件内容和发送邮件等功能。

在主程序文件中，通过调用其他模块或类的方法，完成整个流程。

1. 业务处理流程

读取配置文件：主程序读取配置文件，获取收件人的电子邮箱地址和发送时间等信息。

获取天气数据：使用天气 API，通过发送 HTTP 请求获取当天的天气预报数据。

解析天气数据：将返回的天气数据解析为可用的格式，提取出温度、降水概率等关键信息。

构建邮件内容：使用解析后的天气数据，构建邮件的正文内容，包括当天的天气预报信息。

发送邮件：通过选定的邮件发送库，连接到邮件服务器，并将构建好的邮件内容发送给指定的收件人。

1. 数据存储

临时数据：不需要长期存储的数据，如天气 API 返回的临时数据，可以在内存中暂存，并在处理完毕后释放。

配置文件：将收件人的电子邮箱地址和发送时间等信息存储在配置文件中，以便程序读取和使用。

1. 数据处理分析

天气数据解析：根据天气 API 返回的数据格式，编写相应的解析代码，将关键的天气信息提取出来，如温度、降水概率等。

数据处理与分析：你可以根据实际需求，对天气数据进行进一步的处理和分析。例如，可以计算每周的平均温度、预测未来几天的降水趋势等。

1. **实现细节**

1,关键代码

1,使用python 爬取天气预报数据

需要使用f12控制台查找到需要获取的数据信息并且使用requests库提取查出来。

代码如下：

url = "http://www.weather.com.cn/weather/101220101.shtml"  
header = ("User-Agent",  
 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36") # 设置头部信息  
opener = urllib.request.build\_opener() # 修改头部信息  
opener.addheaders = [header] # 修改头部信息  
request = urllib.request.Request(url) # 制作请求  
response = urllib.request.urlopen(request) # 得到请求的应答包  
html = response.read() # 将应答包里面的内容读取出来  
html = html.decode('utf-8') # 使用utf-8进行编码，不重新编码就会成乱码  
  
final = [] # 初始化一个空的list，我们为将最终的的数据保存到list  
bs = BeautifulSoup(html, "html.parser") # 创建BeautifulSoup对象  
body = bs.body # 获取body部分  
data = body.find('div', {'id': '7d'}) # 找到id为7d的div  
ul = data.find('ul') # 获取ul部分  
li = ul.find\_all('li') # 获取所有的li

2.循环获取近七天的天气状况

i = 0  
for day in li: # 对每个li标签中的内容进行遍历  
 if i < 7:  
 temp = []  
 date = day.find('h1').string # 找到日期  
 # print (date)  
 temp.append(date) # 添加到temp中  
 # print (temp)  
 inf = day.find\_all('p') # 找到li中的所有p标签  
 # print(inf)  
 # print (inf[0])  
 temp.append(inf[0].string) # 第一个p标签中的内容（天气状况）加到temp中  
 if inf[1].find('span') is None:  
 temperature\_highest = None # 天气预报可能没有当天的最高气温（到了傍晚，就是这样），需要加个判断语句,来输出最低气温  
 else:  
 temperature\_highest = inf[1].find('span').string # 找到最高温度  
 temperature\_highest = temperature\_highest.replace('℃', '') # 到了晚上网站会变，最高温度后面也有个℃  
 temperature\_lowest = inf[1].find('i').string # 找到最低温度  
 temperature\_lowest = temperature\_lowest.replace('℃', '') # # 最低温度后面有个℃，去掉这个符号  
 temp.append(temperature\_highest)  
 temp.append(temperature\_lowest)  
 final.append(temp)  
 i = i + 1

3,数据存储

with open('weather.csv', 'a', errors='ignore', newline='') as f:  
 f\_csv = csv.writer(f)  
 f\_csv.writerows(final)

4,发送邮件

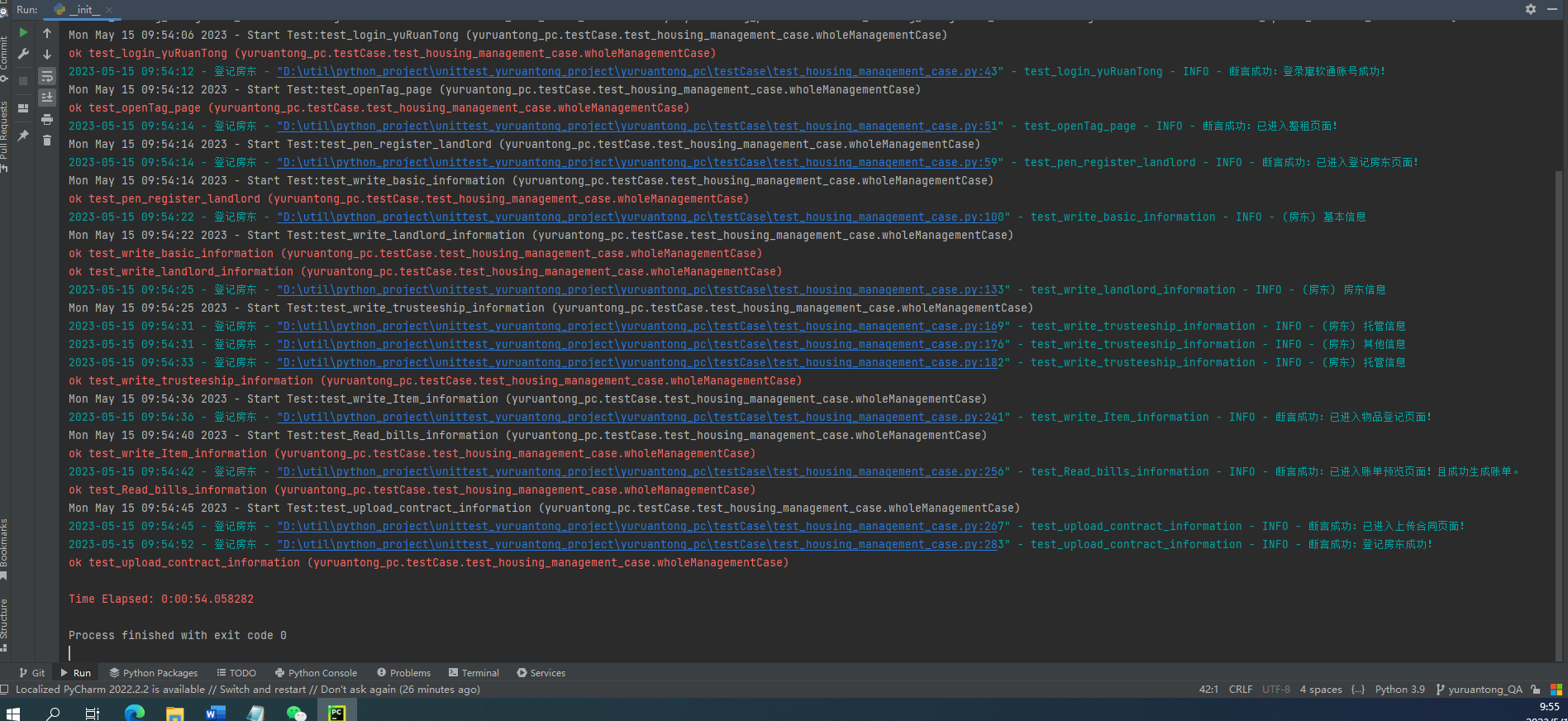
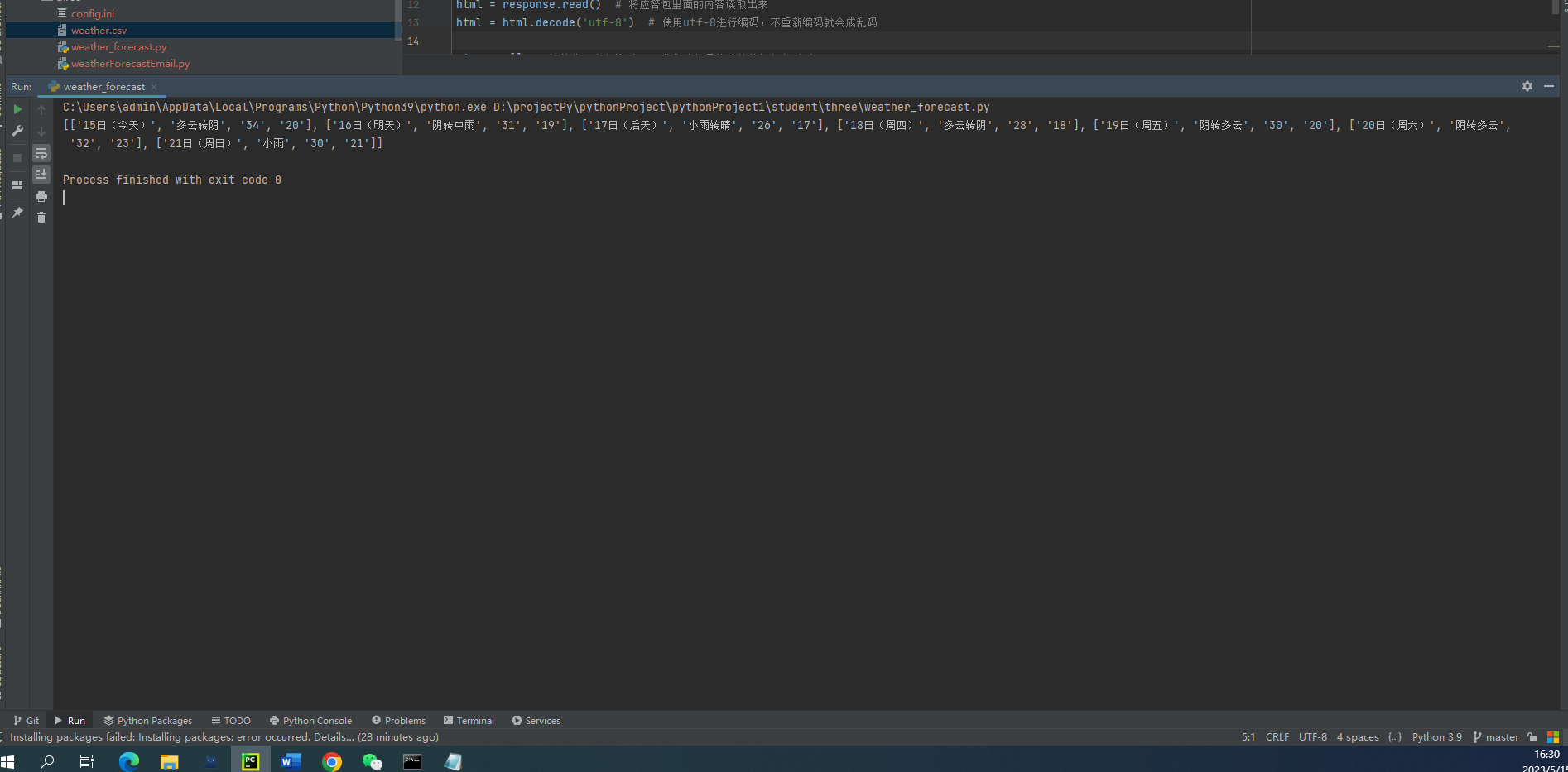
def send(self, title: str, text: str, types: str = "plain", name: str = None, file: str = None, ):  
 *"""  
 title: 邮件标题  
 text: 邮件正文  
 types: 文本格式 plain(纯文本)/html(html代码)  
 name : 自定义邮件名称  
 file: 附件路径  
 """* try:  
 print("邮件发送中..")  
 mtpObj = smtplib.SMTP\_SSL(self.serverAddress)  
 # 建立连接  
 mtpObj.connect(self.serverAddress, 587)  
 # 登录--发送者账号和口令  
 mtpObj.login(self.sender, self.password)  
 # 设置发送信息  
 message = self.message(title, text, self.sender, self.recipient, types=types, name=name,  
 file=file, )  
 # 发送邮件  
 mtpObj.sendmail(self.sender, self.recipient, message.as\_string())  
 print("邮件发送成功")  
 mtpObj.quit()  
 except smtplib.SMTPException:  
 print("无法发送邮件")

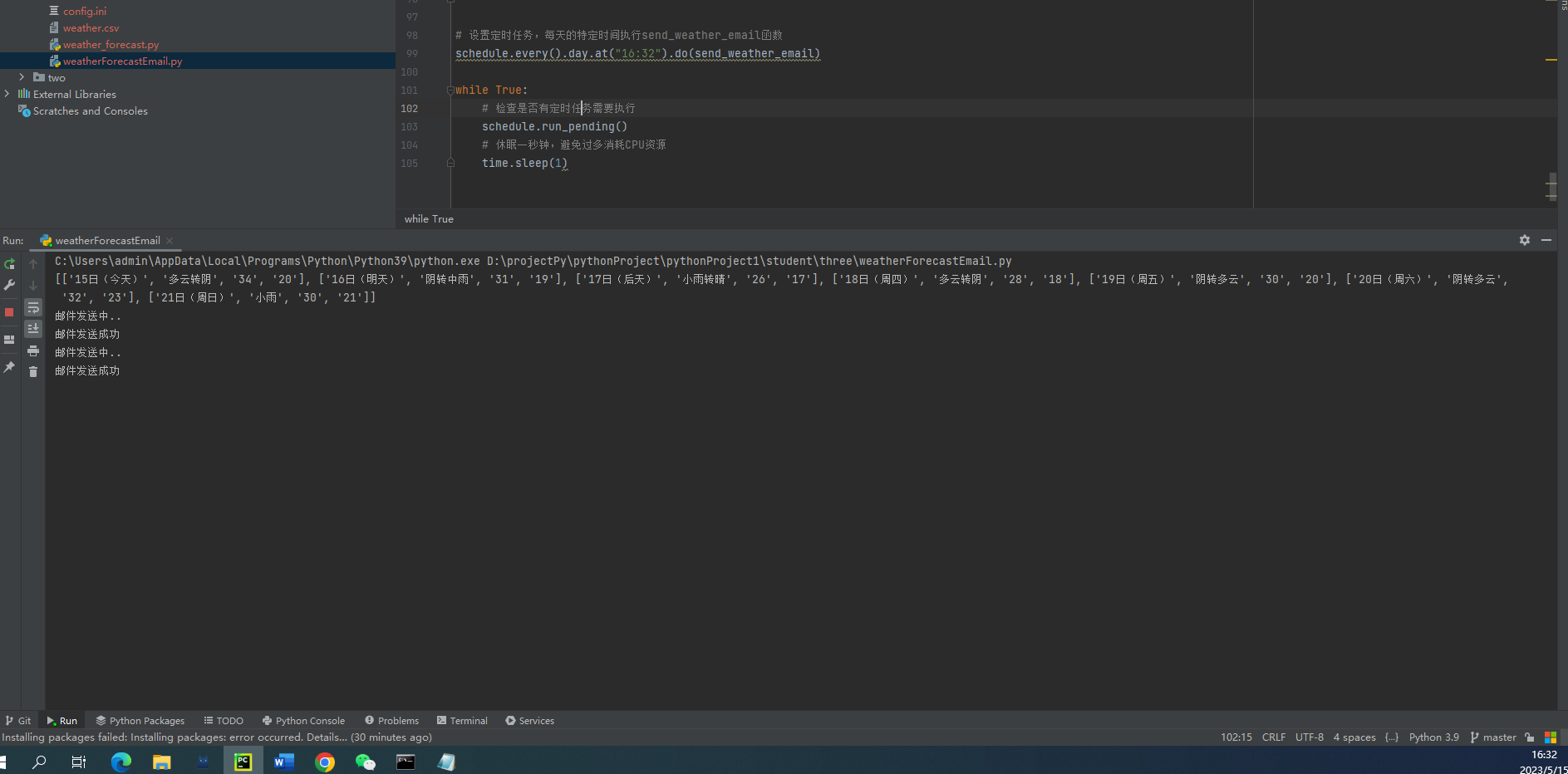
5，定时任务

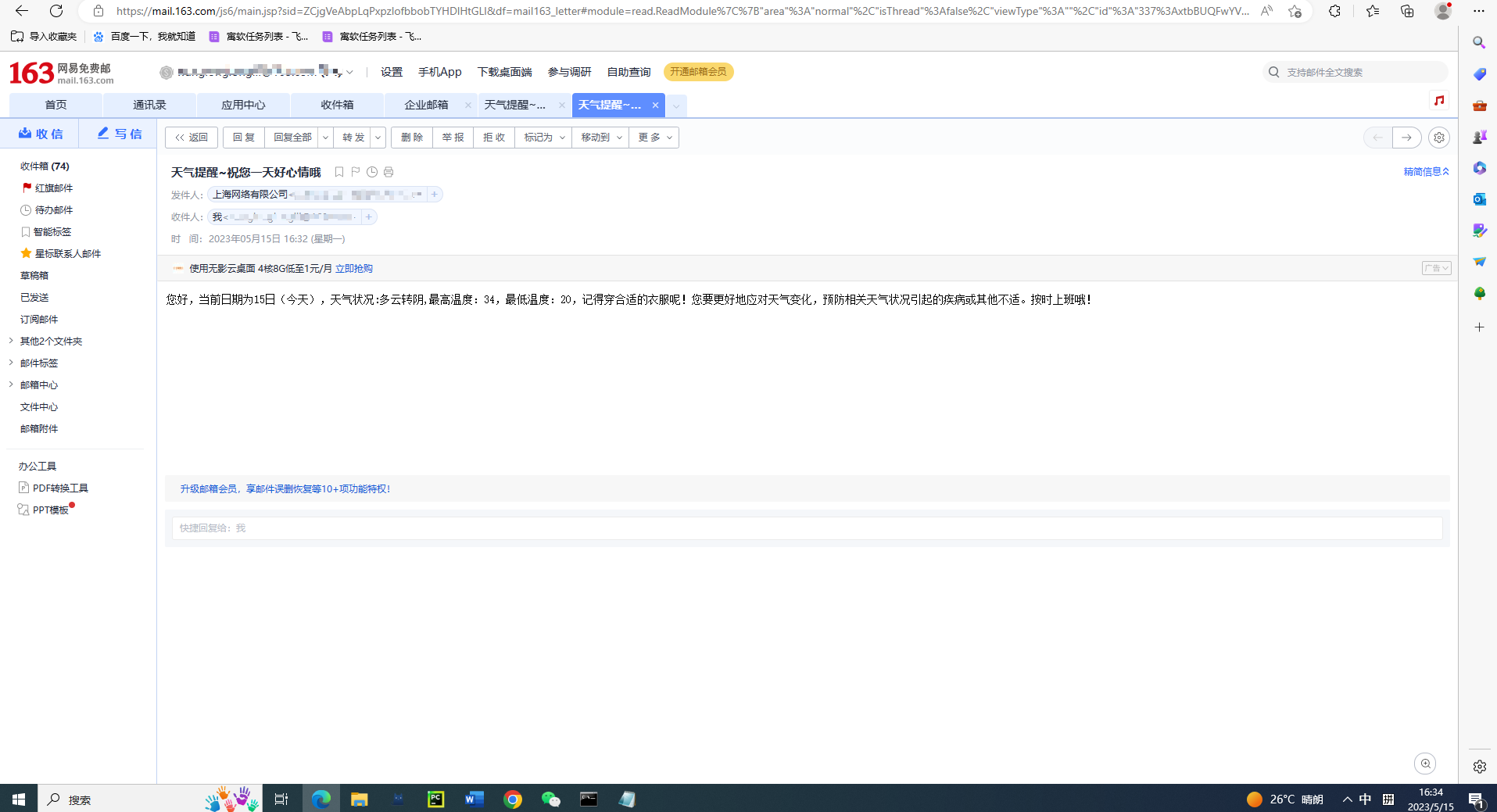
import schedule  
import time

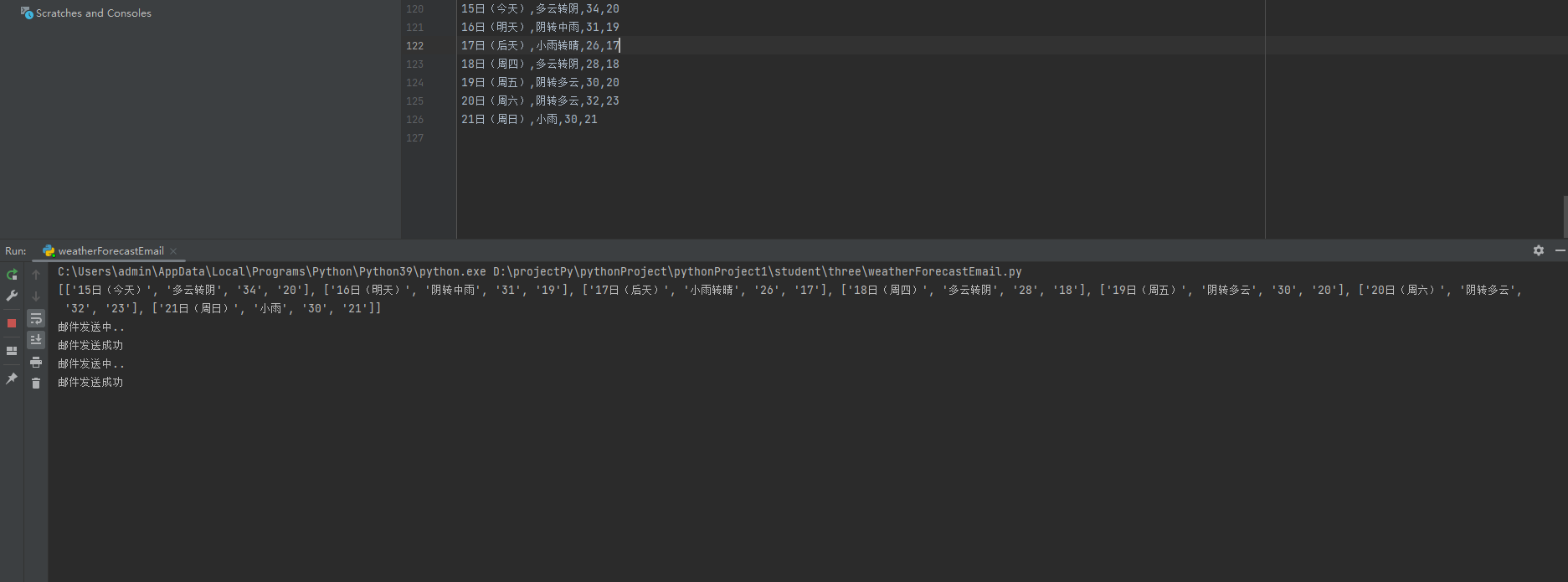
# 设置定时任务，每天的特定时间执行send\_weather\_email函数  
schedule.every().day.at("08:00").do(send\_weather\_email)  
  
while True:  
 # 检查是否有定时任务需要执行  
 schedule.run\_pending()  
 # 休眠一秒钟，避免过多消耗CPU资源  
 time.sleep(1)

1. 运行截图

爬取天气预报中的数据

定时执行发送天气日报邮件消息

查看邮箱是否接收到消息

天气日报临时存储文件

1. **结果分析**

导入必要的模块，包括smtplib用于发送邮件，MIMEText和MIMEMultipart用于创建邮件内容，Header和formataddr用于设置邮件头部信息，weather\_forecast用于获取天气信息，schedule用于定时任务，time用于控制程序的运行时间。

定义一个sendEmail类，用于发送邮件。在初始化方法中，指定了发送者邮箱、授权码、服务器地址以及接收者邮箱。

定义了一个静态方法message，用于创建邮件内容。根据传入的参数，可以设置邮件标题、正文、文本格式（默认为纯文本），发送者、接收者、自定义邮件名称和附件路径。

定义了send方法，用于实际发送邮件。该方法首先建立与SMTP服务器的连接，然后使用发送者的账号和授权码进行登录。接着设置邮件内容，调用message方法创建消息对象，并将其转换为字符串发送。最后，关闭连接。

定义了send\_weather\_email函数，用于发送天气提醒的邮件。该函数首先实例化两个sendEmail对象，分别指定接收者的邮箱地址。然后设置邮件标题和正文，其中正文包含了从final变量中获取的天气信息。最后，调用send方法发送邮件。

使用schedule模块设置定时任务，每天的08:00执行send\_weather\_email函数。

进入主循环，不断检查是否有定时任务需要执行，如果有则执行相应的任务。每次循环后休眠1秒钟，以避免过多消耗CPU资源。

总体来说，该程序通过定时任务实现每天定时发送天气提醒的电子邮件。

1. **系统涵盖的主要知识点及完整源代码**

系统涵盖的主要知识点

SMTP（Simple Mail Transfer Protocol）：使用smtplib库与SMTP服务器建立连接，并通过该协议发送电子邮件。

MIME（Multipurpose Internet Mail Extensions）：使用email.mime模块创建和处理邮件内容。程序中使用了MIMEText和MIMEMultipart类来创建邮件正文和支持附件的邮件对象。

邮件头部信息：使用email.header模块的Header类设置邮件的标题和发件人信息。

定时任务：使用schedule库设置定时任务，以实现每天的特定时间执行指定的函数。

异常处理：使用try-except块捕获SMTPException异常，以处理邮件发送过程中可能出现的异常情况。

类的定义与使用：通过定义sendEmail类，封装了发送邮件的功能，并通过类的实例化和方法调用来实现邮件的发送。

字符串格式化：使用f-string格式化字符串，将天气信息嵌入到邮件正文中。

导入自定义模块：从weather\_forecast模块中导入名为final的变量，用于获取天气信息。

无限循环和休眠：使用while循环和time.sleep()函数创建一个持续运行的程序，每秒钟检查定时任务并休眠一秒钟。

总体而言，该程序结合了电子邮件的发送、定时任务的设置和异常处理等多个知识点，展示了如何通过Python编写一个自动发送天气提醒邮件的程序。

1. 源代码

**weatherForecastEmail.py**

import smtplib  
# 导入smplip  
from email.mime.text import MIMEText  
from email.header import Header  
from email.utils import formataddr  
from email.mime.multipart import MIMEMultipart  
from weather\_forecast import final  
import schedule  
import time  
  
class sendEmail:  
 def \_\_init\_\_(self, recipient: str, sender: str = "wanglonglongiii@163.com", password: str = "IPNXYLLGGCSBPTKH",  
 serverAddress: str = 'smtp.163.com'):  
 *"""  
 recipient: 接收者邮箱  
 sender: 发送者邮箱  
 password: 发送者授权码  
 serverAddress: 服务器地址  
 """* self.sender = sender # 发送者邮箱  
 self.password = password # 授权码  
 self.recipient = recipient # 接收者邮箱  
 self.serverAddress = serverAddress # 服务器地址  
  
 @staticmethod  
 def message(title: str, text: str, sender: str, recipient: str, types: str = "plain", name: str = None,  
 file: str = None):  
 *"""  
 title: 邮件标题  
 text: 邮件正文  
 types: 文本格式 plain(纯文本)/html(html代码)  
 sender: 发送者邮箱  
 recipient： 接收者邮箱  
 name: 自定义邮件名称  
 file: 附件路径  
 """* if file:  
 message = MIMEMultipart()  
 # 这里的三个参数：第一个为文本内容，第二个文本格式 plain(纯文本)/html(html代码)，第三个 utf-8 设置编码  
 message.attach(MIMEText(text, types, 'utf-8'))  
 att1 = MIMEText(open(file, 'rb').read(), 'base64', 'utf-8')  
 att1["Content-Type"] = 'application/octet-stream'  
 att1["Content-Disposition"] = f'attachment; filename="{file}"' # 设置文件名称  
 message.attach(att1)  
 else:  
 # 这里的三个参数：第一个为文本内容，第二个文本格式 plain(纯文本)/html(html代码)，第三个 utf-8 设置编码  
 message = MIMEText(text, types, 'utf-8')  
 message['To'] = Header(recipient) # 接收者  
 message['From'] = Header(formataddr((name, sender), "utf-8")) # 接收者  
 message['Subject'] = Header(title, 'utf-8') # 设置标题  
 return message  
  
 def send(self, title: str, text: str, types: str = "plain", name: str = None, file: str = None, ):  
 *"""  
 title: 邮件标题  
 text: 邮件正文  
 types: 文本格式 plain(纯文本)/html(html代码)  
 name : 自定义邮件名称  
 file: 附件路径  
 """* try:  
 print("邮件发送中..")  
 mtpObj = smtplib.SMTP\_SSL(self.serverAddress)  
 # 建立连接  
 mtpObj.connect(self.serverAddress, 587)  
 # 登录--发送者账号和口令  
 mtpObj.login(self.sender, self.password)  
 # 设置发送信息  
 message = self.message(title, text, self.sender, self.recipient, types=types, name=name,  
 file=file, )  
 # 发送邮件  
 mtpObj.sendmail(self.sender, self.recipient, message.as\_string())  
 print("邮件发送成功")  
 mtpObj.quit()  
 except smtplib.SMTPException:  
 print("无法发送邮件")  
  
  
def send\_weather\_email():  
 # 拉取当天的天气状况  
 # 实例化对象，传入接收者邮箱参数recipient  
 p1 = sendEmail(recipient="wanglonglongiii@163.com")  
 p2 = sendEmail(recipient="1367446518@qq.com")  
 title = "天气提醒~祝您一天好心情哦"  
 text = f"您好，当前日期为{final[0][0]}，天气状况:{final[0][1]},最高温度：{final[0][2]}，最低温度：{final[0][3]}，记得穿合适的衣服呢！" \  
 f"您要更好地应对天气变化，预防相关天气状况引起的疾病或其他不适。按时上班哦！"  
 p1.send(title=title, text=text, name="上海网络有限公司")  
 p2.send(title=title, text=text, name="北京网络有限公司")  
 """  
 send参数说明:  
 title: 必传 邮件标题  
 text: 必传 邮件正文  
 types: 选传 文本格式 plain(纯文本)/html(html代码) 默认为plain  
 name : 选传 自定义邮件名称  
 file: 选传 附件路径  
 """  
  
# 设置定时任务，每天的特定时间执行send\_weather\_email函数  
schedule.every().day.at("08:00").do(send\_weather\_email)  
  
while True:  
 # 检查是否有定时任务需要执行  
 schedule.run\_pending()  
 # 休眠一秒钟，避免过多消耗CPU资源  
 time.sleep(1)

**weather\_forecast.py**

import csv  
import urllib.request  
from bs4 import BeautifulSoup  
  
url = "http://www.weather.com.cn/weather/101220101.shtml"  
header = ("User-Agent",  
 "Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/69.0.3497.100 Safari/537.36") # 设置头部信息  
opener = urllib.request.build\_opener() # 修改头部信息  
opener.addheaders = [header] # 修改头部信息  
request = urllib.request.Request(url) # 制作请求  
response = urllib.request.urlopen(request) # 得到请求的应答包  
html = response.read() # 将应答包里面的内容读取出来  
html = html.decode('utf-8') # 使用utf-8进行编码，不重新编码就会成乱码  
  
final = [] # 初始化一个空的list，我们为将最终的的数据保存到list  
bs = BeautifulSoup(html, "html.parser") # 创建BeautifulSoup对象  
body = bs.body # 获取body部分  
data = body.find('div', {'id': '7d'}) # 找到id为7d的div  
ul = data.find('ul') # 获取ul部分  
li = ul.find\_all('li') # 获取所有的li  
# print (li)  
  
i = 0  
for day in li: # 对每个li标签中的内容进行遍历  
 if i < 7:  
 temp = []  
 date = day.find('h1').string # 找到日期  
 # print (date)  
 temp.append(date) # 添加到temp中  
 # print (temp)  
 inf = day.find\_all('p') # 找到li中的所有p标签  
 # print(inf)  
 # print (inf[0])  
 temp.append(inf[0].string) # 第一个p标签中的内容（天气状况）加到temp中  
 if inf[1].find('span') is None:  
 temperature\_highest = None # 天气预报可能没有当天的最高气温（到了傍晚，就是这样），需要加个判断语句,来输出最低气温  
 else:  
 temperature\_highest = inf[1].find('span').string # 找到最高温度  
 temperature\_highest = temperature\_highest.replace('℃', '') # 到了晚上网站会变，最高温度后面也有个℃  
 temperature\_lowest = inf[1].find('i').string # 找到最低温度  
 temperature\_lowest = temperature\_lowest.replace('℃', '') # # 最低温度后面有个℃，去掉这个符号  
 temp.append(temperature\_highest)  
 temp.append(temperature\_lowest)  
 final.append(temp)  
 i = i + 1  
  
print(final)  
  
with open('weather.csv', 'a', errors='ignore', newline='') as f:  
 f\_csv = csv.writer(f)  
 f\_csv.writerows(final)