

**MAIL MERGE**

A Project Report Submitted in Partial Fulfillment of the Requirements

**AISSCE – 2021**

In

**COMPUTER SCIENCE**

By:

|  |  |  |  |
| --- | --- | --- | --- |
| **S. No.** | **Student Name** | **Class** | **Admit Card No.** |
| 1 | Abey Jacob John | XII-B | 20661546 |
| 2 | Abishek R. | XII-B | 20661547 |
| 3 | Ayan Datta | XII-B | 20661553 |
| 4 | Mudunuri Sai Abhinav Verma | XII-B | 20661571 |

**Certificate**

Certified that the work contained in the project titled “Mail Merge” by “Abey Jacob John, Abishek R. , Ayan Datta , and Mudunuri Sai Abhinav Verma”, has been carried out under my supervision as prescribed by CBSE AISSCE – 2021.

Internal Examiner External Examiner Date:

Institution Stamp:

**Acknowledgement**

I would like to express my gratitude to my Computer Teacher Mr. Mallikarjun for his guidance, support and encouragement through the project and the school lab assistant.

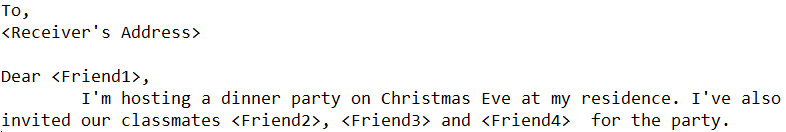
I would further like to thank my parents and friends for helping me with the research required for this project.

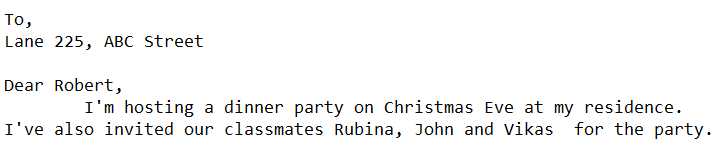
**Contents**

|  |  |  |
| --- | --- | --- |
| 1. | Introduction | 5 |
| 2. | System Requirements | 6 |
| 3. | Python Source Code | 7 |
| 4. | Output | 18 |
| 5. | References | 24 |

**Introduction**

Mail Merge allows the user to create a batch of documents which are personalized for each recipient. It is a powerful tool for writing a personalized letter or e-mail to many people at the same time. It imports data from another source such as a spreadsheet and then uses that to replace placeholders throughout the message with the relevant information for each individual that is being messaged.





The names of each recipient is given by the user and is stored in the default spreadsheet of the system. This is then sent to a database in MySQL, from where the names are acquired and added in the message.

A major advantage of mail merge is that compared to the process of preparing individual letters to convey one set of information to many people, mail merge saves time and effort, producing mass mailings complete in the most efficient manner.

Simple concepts of CSV file and Python-MYSQL connection were used in the successful execution of this project.

**System requirements of the project**

Recommended System Requirements :

Processors:Intel® Core™ i3 processor 4300M at 2.60 GHz. Disk space: 2 to 4 GB.

Operating systems: Windows® 10, MACOS, UBUNTU. Python Versions: 3.X.X or Higher.

Minimum System Requirements

Processors:Intel Atom® processor or Intel® Core™ i3 processor. Disk space: 1 GB.

Operating systems: Windows 7 or later, MACOS, and UBUNTU. Python Versions: 2.7.X, 3.6.X.

Prerequisites before installing MySQL Connector Python

You need root or administrator privileges to perform the installation process.

Python must be installed on your machine.

Note: MySQL Connector Python requires python to be in the system’s PATH. Installation fails if it doesn’t find Python.

On Windows, If Python doesn’t exist in the system’s PATH, please manually add the directory containing python.exe yourself.

**Python Source Code**

main.py

*"""main.py : Consists of the main program."""*import csv\_handler  
import detector  
import input\_func  
import interface  
import sql\_func  
import substituter  
import time  
import traceback  
  
  
def main():  
 *"""Runs the main Mail Merge Program."""* interface.print\_header()  
 time.sleep(1)  
 interface.print\_instructions()  
  
 template\_location = input\_func.inp\_template()  
 if template\_location is None:  
 interface.quit\_program()  
 var\_list = detector.detect\_var(template\_location)  
 var\_list.sort()  
 print(**"Successfully loaded template."**)  
 time.sleep(0.5)  
  
 sql\_access = sql\_func.sql\_access\_prompt()  
  
 if sql\_access is not None:  
 *# Use Data In MySQL Database* table\_name, sql\_connection = sql\_access  
 if not sql\_func.verify\_data(sql\_connection, var\_list, table\_name):  
 interface.user\_error(**"The Table does not contain all the variables used in your template."**)  
  
 data = sql\_func.data\_getter(sql\_connection, table\_name, var\_list)  
 print(**"Successfully Loaded Data"**)  
 time.sleep(0.5)  
  
 else:  
 *# Get User Data* csv\_location = csv\_handler.csv\_namer(template\_location)  
 csv\_handler.csv\_writer(var\_list, csv\_location)  
 input\_func.user\_csv\_prompter(csv\_location)  
 data = csv\_handler.csv\_reader(var\_list, csv\_location)  
 print(**"Successfully Loaded Data"**)  
 time.sleep(0.5)  
  
 sql\_prompt = input\_func.sql\_prompt()  
 if sql\_prompt is not None:  
 *# Save User Data* username, password = sql\_prompt  
 sql\_connection = sql\_func.sql\_connect(username, password)  
 sql\_func.database\_creator(sql\_connection)  
 table\_name = sql\_func.table\_name\_prompter(sql\_connection)  
 sql\_func.table\_creator(sql\_connection, table\_name, var\_list, data)  
 print(**"Successfully saved Data"**)  
 time.sleep(0.5)  
  
 save\_location = input\_func.inp\_save\_folder()  
 if save\_location is None:  
 interface.quit\_program()  
  
 substituter.substitute(var\_list, data, template\_location, save\_location)  
  
 print(**"Mail Merge Completed"**)  
 time.sleep(0.5)  
  
 *# Open Saved Files in Explorer.* interface.open\_file(save\_location)  
 interface.quit\_program()  
  
  
if \_\_name\_\_ == **"\_\_main\_\_"**:  
 try:  
 main()  
 except SystemExit:  
 *# This is triggered when exit function is called in any of the functions.* pass  
 except:  
 *# traceback.format\_exc gets the full error message.* interface.program\_error(traceback.format\_exc())

csv\_handler.py

*"""  
csv\_handler.py : Handles Reading and Writing of .csv files. Used for Getting Data from User.  
"""*import csv  
from interface import user\_error  
  
  
def csv\_namer(template\_location):  
 *""" Changes the extension of template file from ".txt" to ".csv" """* csv\_location = template\_location.rstrip(**"txt"**) + **"csv"** return csv\_location  
  
  
def csv\_writer(var\_list, csv\_location):  
 *"""Writes Column names in .csv file"""* with open(csv\_location, mode=**"w"**) as csv\_file: *# Opening the csv file* csv\_file\_writer = csv.writer(csv\_file) *# Creating a writer object* csv\_file\_writer.writerow(var\_list) *# Writing a row in the csv file*def csv\_reader(var\_list, csv\_location):  
 *""" To read the user provided values from the .csv file"""* row\_values = []  
 with open(csv\_location) as csv\_file:  
 reader = csv.reader(csv\_file)  
 for row in reader:  
 row\_values.append(row)  
 if row\_values[0] == var\_list:  
 row\_number = 1  
 for row in row\_values[1:]:  
 if len(row) != len(var\_list):  
 user\_error(  
 **f"Data not filled completely in row number** {row\_number} **(Row number excluding column headers.)"**)  
 return  
 row\_number += 1  
 return row\_values[1:] *# Returns all the row values except the column headers.* else:  
 user\_error(**"ERROR! You Changed The Column Headers."**)

detector.py

*"""detector.py : Handles detection of variables used in the template."""*from interface import user\_error  
  
opening = **"<"**closing = **">"**def detect\_var(file\_path):  
 *"""Scans the text file and finds variables enclosed in <> (Angular Brackets).   
 Returns a list of all distinct variable names"""* with open(file\_path) as template\_file:  
 text = template\_file.read()  
  
 looking\_for\_closing = False  
 last\_opening\_index = None  
 var\_list = []  
  
 for letter\_index in range(len(text)):  
 letter = text[letter\_index]  
  
 if letter == opening:  
 *# Opening tag of variable found.* if looking\_for\_closing:  
 *# If previous opening tag is not closed then raise error* user\_error(**"ERROR IN TEMPLATE : (Two Continuous Openings)"**)  
  
 looking\_for\_closing = True  
 last\_opening\_index = letter\_index  
  
 elif letter == closing:  
 *# Closing tag of variable found.* if not looking\_for\_closing:  
 *# No tag was opened so raise error.* user\_error(**"ERROR IN TEMPLATE : (Randomly used closing)"**)  
  
 var\_name = text[last\_opening\_index + 1:letter\_index]  
  
 if var\_name not in var\_list:  
 var\_list.append(var\_name)  
  
 looking\_for\_closing = False  
 last\_opening\_index = None  
  
 if looking\_for\_closing:  
 user\_error(**"ERROR IN TEMPLATE : (Did not close opening tag)"**)  
  
 return var\_list

input\_func.py

*"""input\_func.py: Handles most of User Input"""*import tkinter as tk  
from tkinter import filedialog  
from interface import open\_file  
  
  
def inp\_template():  
 *"""Asks user for template file location."""  
  
 # Get access to the root window created by tkinter* root = tk.Tk()  
  
 *# Makes Sure the file dialog draws on top of all windows.* root.attributes(**"-topmost"**, True)  
 root.lift()  
  
 *# Hides the root window* root.withdraw()  
  
 *# Creates a file dialog box.* location = filedialog.askopenfilename(title=**"Select Your Template File"**,  
 filetypes=((**"Text Files"**, **"\*.txt"**),))  
 if location:  
 return location  
 return  
  
  
def sql\_prompt():  
 *"""Asks user whether to save data in MySQL"""* choice = input(**'Do you want to save the data in a MySQL Database? (Y/N)**\n**>'**)  
 if choice.lower() == **'y'**:  
 username = input(**'Enter MySQL username:'**)  
 password = input(**'Enter MySQL password:'**)  
 return username, password  
 else:  
 return  
  
  
def user\_csv\_prompter(location):  
 *"""Asks user to fill details in the .csv file """* print(**'Fill in the details in the file:'**, location)  
  
 *# Opens .csv file in default editor.* open\_file(location)  
  
 input(**"Press any key to continue."**)  
  
  
def inp\_save\_folder():  
 *"""Asks user for output save location"""  
 # Get access to the root window created by tkinter* root = tk.Tk()  
  
 *# Makes Sure the file dialog draws on top of all windows.* root.attributes(**"-topmost"**, True)  
 root.lift()  
  
 *# Hides the root window* root.withdraw()  
  
 *# Creates a folder dialog box.* location = filedialog.askdirectory(parent=root, title=**"Where To Save Output?"**)  
 if location:  
 return location  
 return

interface.py

*"""interface.py: Handles most of the interaction with user."""*import subprocess  
import sys  
import time  
import os  
  
  
def print\_header():  
 *"""Prints the name of the program."""* print(**"..Mail Merge.."**)  
  
  
def print\_instructions():  
 *"""Displays instructions."""* with open(**"instructions.txt"**) as instructions\_file:  
 print(instructions\_file.read())  
 input(**"Press Any Key to Continue..."**)  
  
  
def user\_error(error):  
 *"""Handles User's Errors"""  
  
 # print the User's error in the error stream.* sys.stderr.write(error)  
  
 *# Give a time gap to make sure error is printed before the quit message.* time.sleep(0.5)  
  
 *# Makes sure quit message is printed in a new line.* print()  
 quit\_program()  
  
  
def program\_error(traceback):  
 *"""Handles program errors."""  
  
 # Gets Current date and time in a format that can be used in a filename.* current\_time = time.strftime(**"%Y%m%d-%H%M%S"**)  
  
 *# Generate filename and filepath.* file\_name = **"traceback\_"** + current\_time + **".txt"** if not os.path.exists(**"Tracebacks"**):  
 os.mkdir(**"Tracebacks"**)  
 file\_path = os.path.join(os.getcwd(), **"Tracebacks"**, file\_name)  
  
 *# Save error in traceback file.* with open(file\_path, **"w"**) as error\_file:  
 error\_file.write(traceback)  
  
 print(**"Something Went Wrong.."**)  
 print(**f"See** {file\_path} **for details."**)  
 quit\_program()  
  
  
def quit\_program():  
 *"""Quits the program."""* input(**"Press any key to quit..."**)  
 exit()  
  
  
def open\_file(file\_path):  
 *"""Opens file in default program. Opens explorer in case a folder path is provided."""* if sys.platform == **"win32"**:  
 *# For Windows.* os.startfile(file\_path)  
 else:  
 *# Other OS like MacOS and Unix-like systems (Linux, FreeBSD, Solaris...)* opener = **"open"** if sys.platform == **"darwin"** else **"xdg-open"** subprocess.call([opener, file\_path])

sql\_func.py

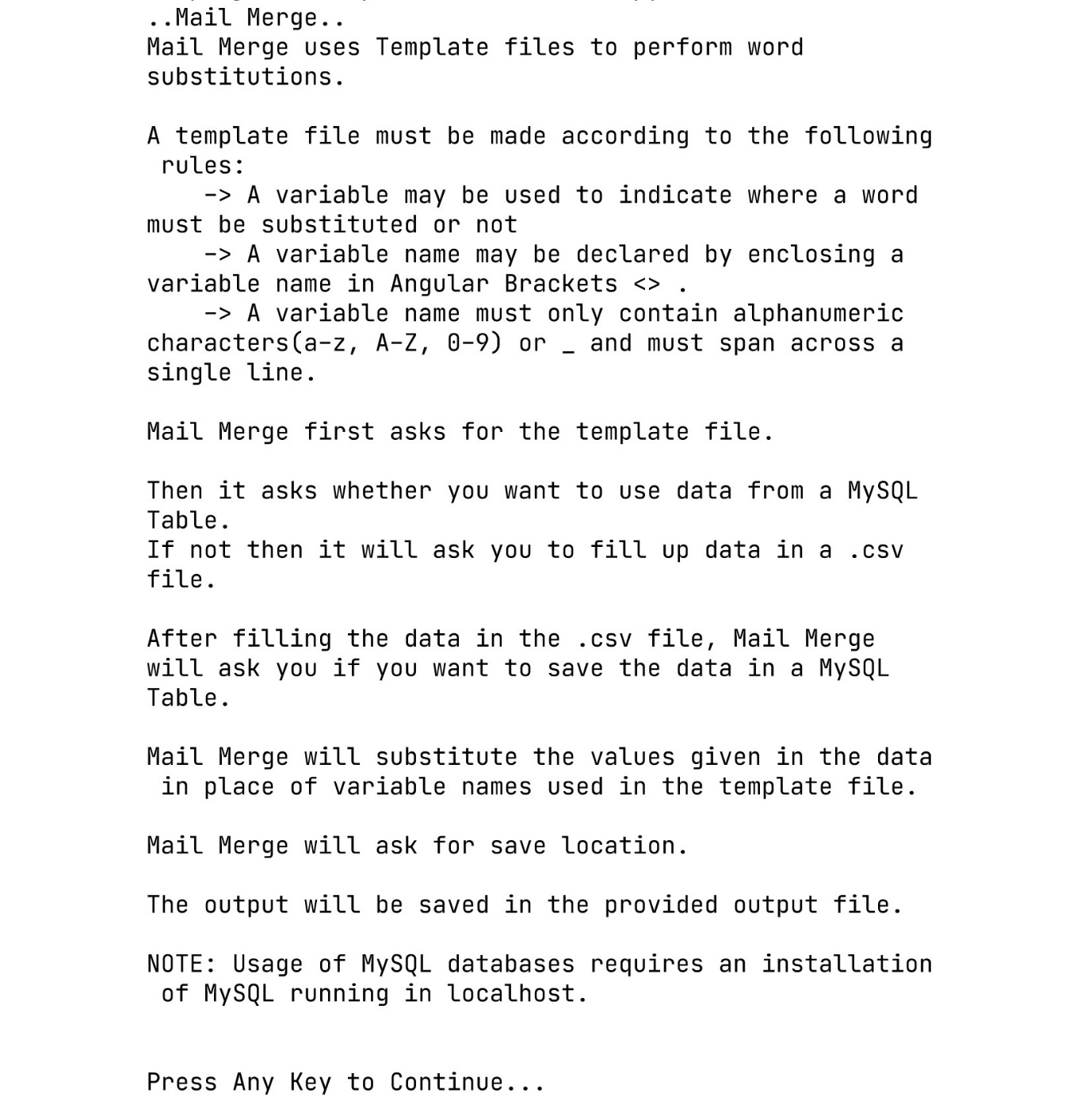
*"""sql\_func.py : Handles MySQL Database operations."""*import mysql.connector  
from interface import user\_error  
import traceback  
  
database\_name = **"MailMerge"**def sql\_connect(username, password):  
 *"""Connect to MySQL Server running in localhost"""* try:  
 sql\_connection = mysql.connector.connect(host=**'localhost'**, user=username, password=password)  
 except:  
 user\_error(**"SQL ERROR :**\n**"** + traceback.format\_exc())  
 else:  
 return sql\_connection  
  
  
def table\_name\_prompter(sql\_connection):  
 *"""Asks User for a new table name."""* while True:  
 table\_name = input(**'Enter a Table Name:'**)  
 cursor = sql\_connection.cursor()  
 cursor.execute(**'SHOW TABLES'**)  
 tables = cursor.fetchall()  
  
 table\_list = []  
 for row in tables:  
 table\_list.append(row[0])  
  
 if table\_name in table\_list:  
 print(table\_name, **'already exists. Choose Again..'**)  
 else:  
 return table\_name  
  
  
def table\_creator(sql\_connection, table\_name, var\_list, data):  
 *"""Creates a new table and inserts data in it."""* table\_creator\_cursor = sql\_connection.cursor()  
  
 *# Generate SQL for creating table containing user's data.* table\_creator\_sql = **f"create table `**{table\_name}**` ("** for \_ in var\_list[:-1]:  
 table\_creator\_sql += **"`%s` longtext not null,"** table\_creator\_sql += **"`%s` longtext not null);"** table\_creator\_cursor.execute(table\_creator\_sql, var\_list)  
  
 *# Generate sql for inserting data into the table.* data\_inserter\_sql = **f"insert into** {table\_name} **values("** for \_ in data[0]:  
 data\_inserter\_sql += **"%s,"** data\_inserter\_sql = data\_inserter\_sql.rstrip(**","**) + **");"** data\_inserter\_cursor = sql\_connection.cursor()  
 data\_inserter\_cursor.executemany(data\_inserter\_sql, data)  
  
 sql\_connection.commit()  
  
  
def database\_creator(sql\_connection: mysql.connector.connection.MySQLConnection):  
 *"""Creates database if it doesnt exist and uses it."""* database\_creator\_cursor = sql\_connection.cursor()  
 database\_creator\_cursor.execute(**f"create database if not exists `**{database\_name}**`"**)  
  
 database\_user\_cursor = sql\_connection.cursor()  
 database\_user\_cursor.execute(**f"use `**{database\_name}**`"**)  
  
  
def sql\_access\_prompt():  
 *"""Asks user whether to use data saved in a MySQL Table."""* choice = input(**'Do you want to use data saved in a MySQL Table (Y/N)**\n**>'**)  
  
 if choice.lower() == **'y'**:  
 username = input(**'Enter the username:'**)  
 password = input(**'Enter the password:'**)  
 else:  
 return  
  
 sql\_connection = sql\_connect(username, password)  
  
 while True:  
 table\_name = input(**f'Enter the Table Name (Must be in a database named** {database\_name}**):'**)  
 database\_creator(sql\_connection)  
 table\_checker\_cursor = sql\_connection.cursor()  
 table\_checker\_cursor.execute(**'SHOW TABLES'**)  
 tables = table\_checker\_cursor.fetchall()  
  
 table\_list = []  
 for row in tables:  
 table\_list.append(row[0])  
  
 if table\_name not in table\_list:  
 print(table\_name, **'does not exist.'**)  
 else:  
 return table\_name, sql\_connection  
  
  
def data\_getter(sql\_connection: mysql.connector.MySQLConnection, table\_name, var\_list):  
 *"""Gets data from the MySQL Table."""* data\_getter\_cursor = sql\_connection.cursor()  
  
 *# Generate SQL for getting data in alphabetical order of column headers.* data\_getter\_sql = **"select"** for \_ in var\_list:  
 data\_getter\_sql += **"`%s`,"** data\_getter\_sql = data\_getter\_sql.rstrip(**","**) + **f"from `**{table\_name}**`;"** data\_getter\_cursor.execute(data\_getter\_sql, var\_list)  
 data = data\_getter\_cursor.fetchall()  
 return data  
  
  
def verify\_data(sql\_connection, var\_list, table\_name):  
 *"""Check whether the table contains the data required."""* cursor = sql\_connection.cursor()  
 cursor.execute(**"DESCRIBE `{}`"**.format(table\_name))  
  
 field\_names = cursor.fetchall()  
  
 table\_headers = []  
 for field in field\_names:  
 *# Remove single quotes from field names that are added while creating tables  
 # and also remove extra MySQL escape characters.* table\_headers.append(field[0].strip(**"**\'**"**).replace(**"**\\**"**, **""**))  
  
 for val in var\_list:  
 if val not in table\_headers:  
 return False  
 return True

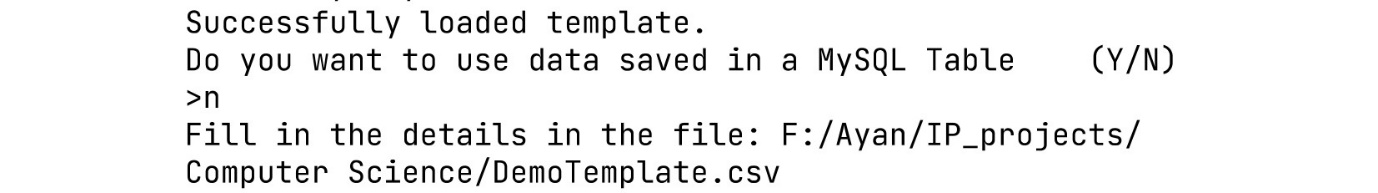
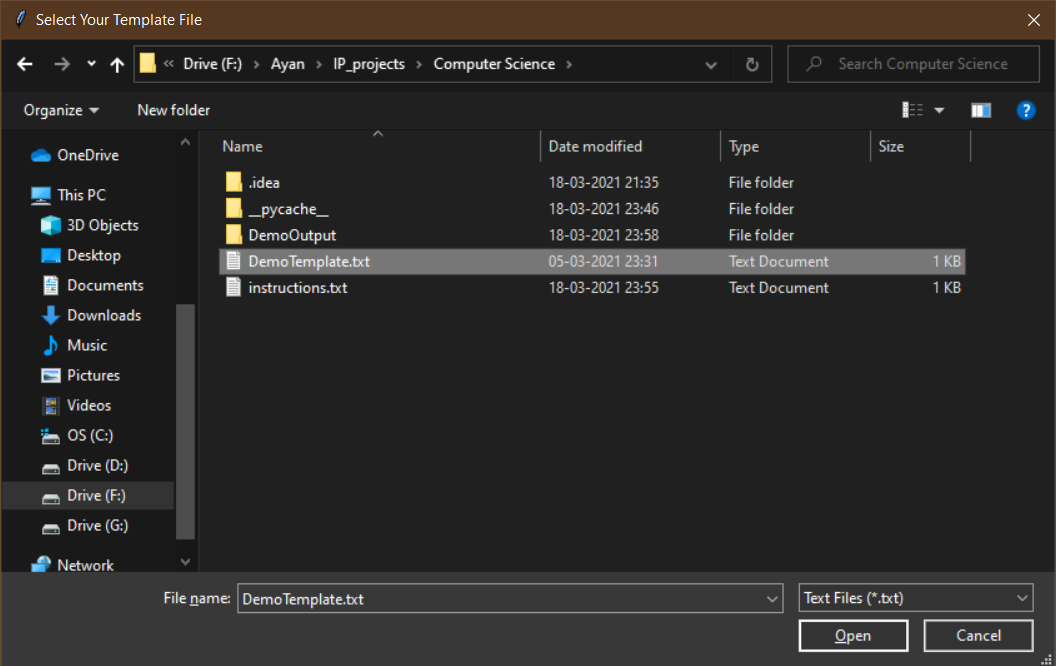
substituter.py

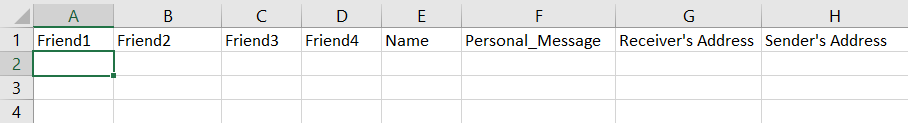
*"""substituter.py : Handles Substitution of variables with their respective values."""*import os.path  
  
  
def substitute(var\_list, data, template\_location, save\_location):  
 *"""Substitutes variables with their respective values."""* with open(template\_location) as template\_file:  
 template\_text = template\_file.read()  
  
 row\_number = 1  
 for row in data:  
 new\_text = template\_text  
 for var, sub in zip(var\_list, row):  
 new\_text = new\_text.replace(**"<"** + var + **">"**, sub)  
 with open(os.path.join(save\_location, **f"**{row\_number}**.txt"**), **"w"**) as out\_file:  
 out\_file.write(new\_text)  
 row\_number += 1

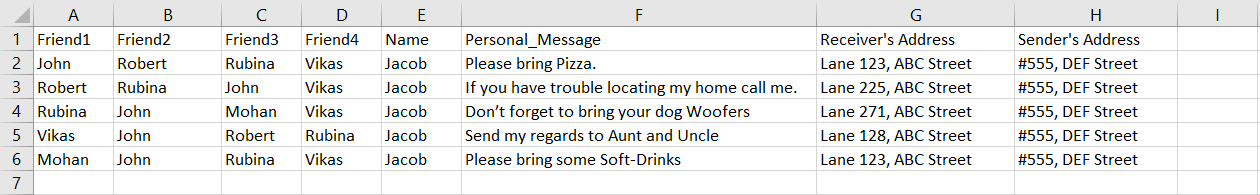
**Output of the Program**

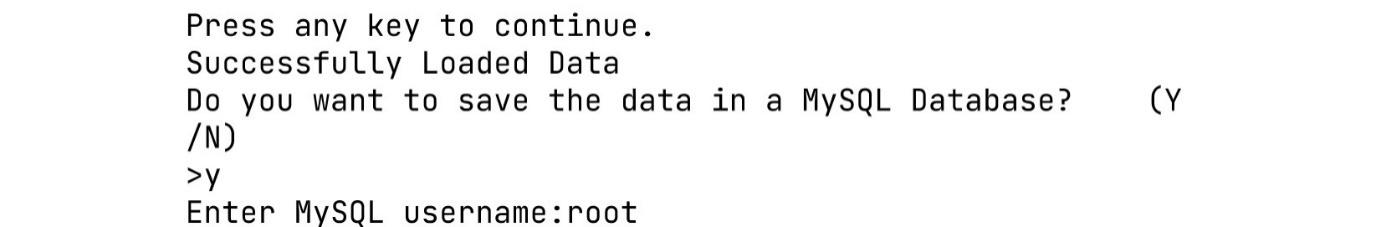
File: main.py

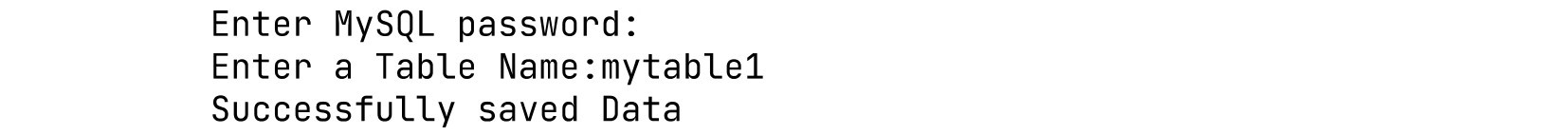


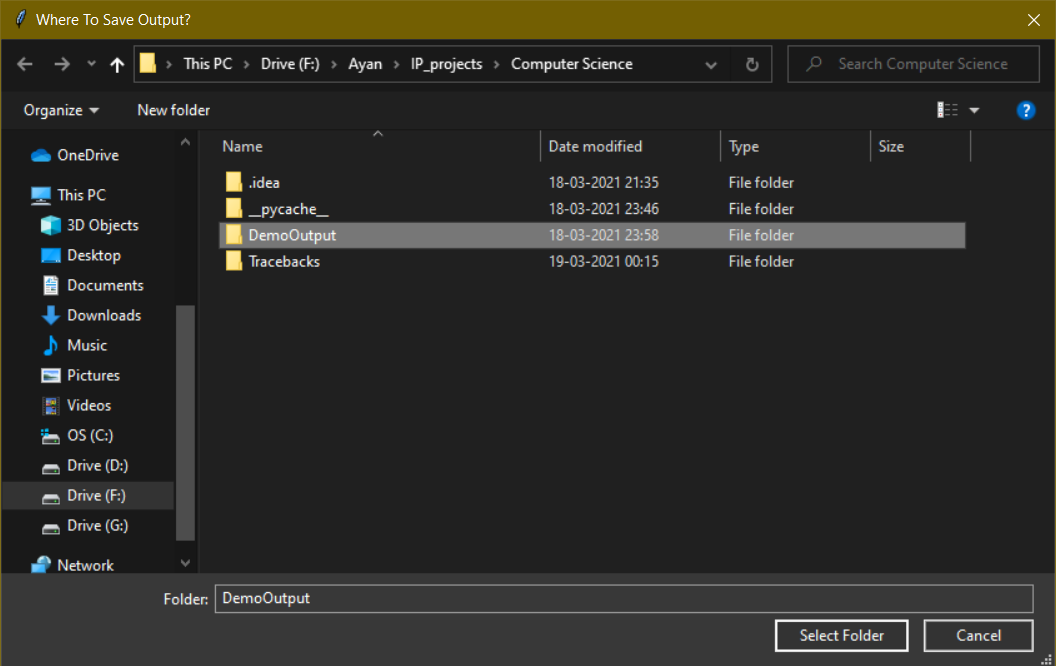
****

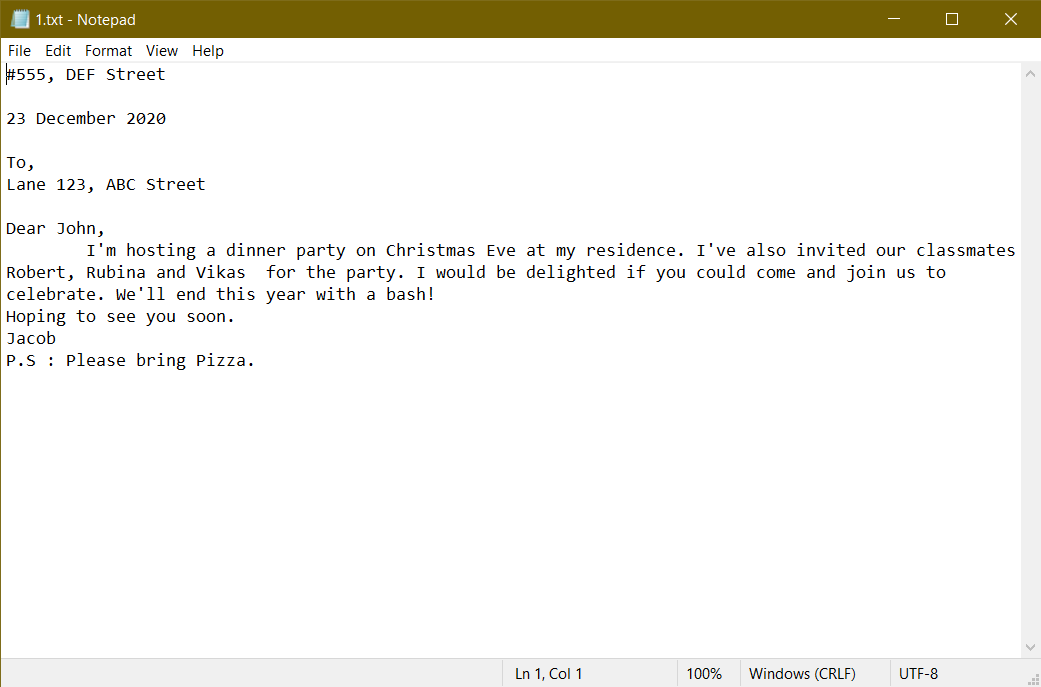
****

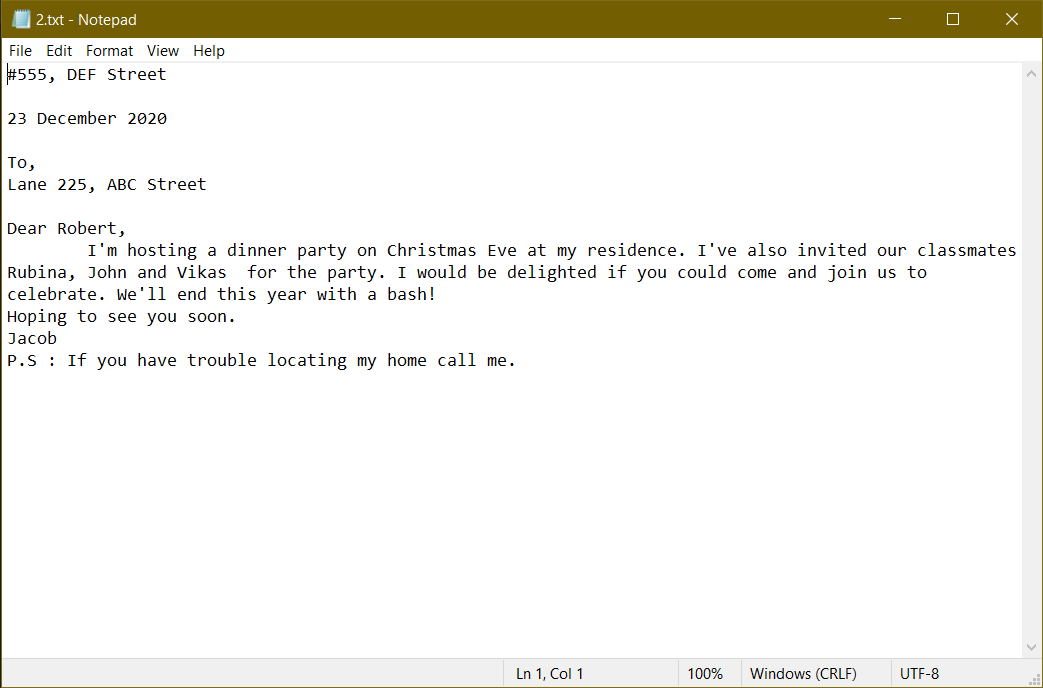
****

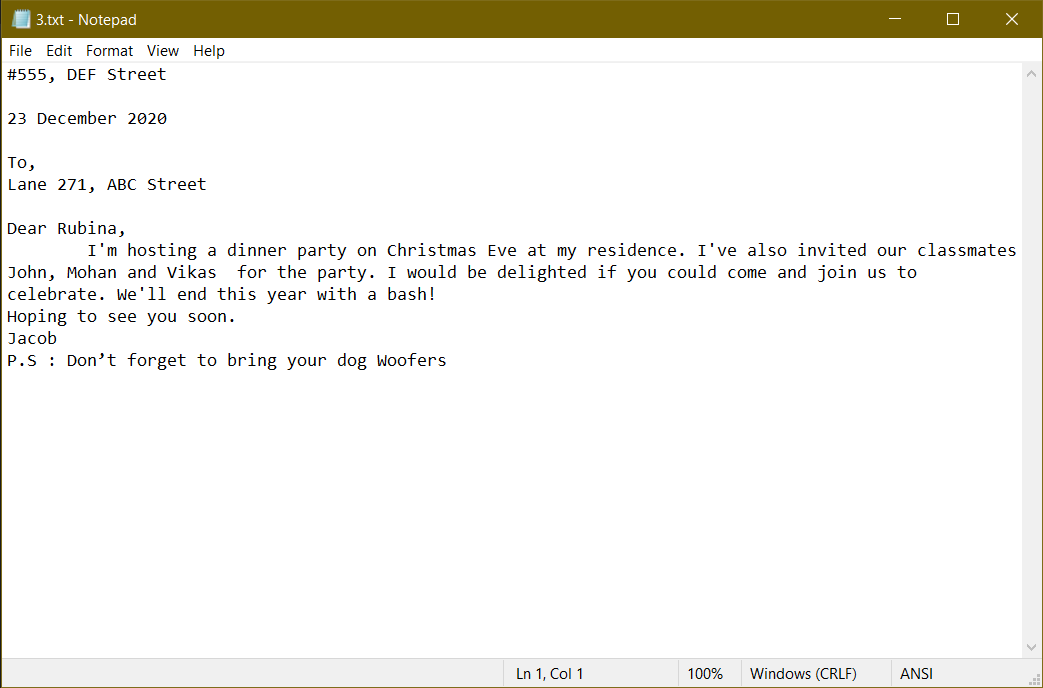




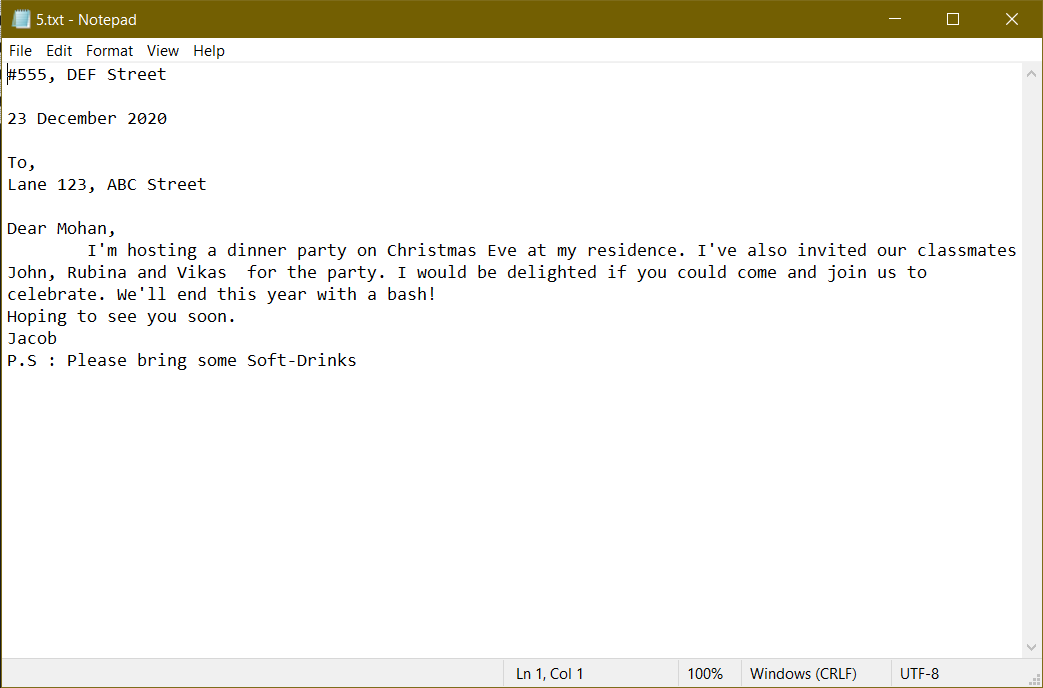
****

****

****

****

****

****

**References**

python.org

codecademy.com

tutorialspoint.com

developers.google.com/edu/python

learnpython.org

stackoverflow.com