



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.

```
To,  
<Receiver's Address>  
  
Dear <Friend1>,  
    I'm hosting a dinner party on Christmas Eve at my residence. I've also  
invited our classmates <Friend2>, <Friend3> and <Friend4> for the party.
```



```
To,  
Lane 225, ABC Street  
  
Dear Robert,  
    I'm hosting a dinner party on Christmas Eve at my residence.  
I've also invited our classmates Rubina, John and Vikas for the party.
```

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

```
..Mail Merge..
Mail Merge uses Template files to perform word
substitutions.

A template file must be made according to the following
rules:
    -> A variable may be used to indicate where a word
must be substituted or not
    -> A variable name may be declared by enclosing a
variable name in Angular Brackets <> .
    -> A variable name must only contain alphanumeric
characters(a-z, A-Z, 0-9) or _ and must span across a
single line.

Mail Merge first asks for the template file.

Then it asks whether you want to use data from a MySQL
Table.
If not then it will ask you to fill up data in a .csv
file.

After filling the data in the .csv file, Mail Merge
will ask you if you want to save the data in a MySQL
Table.

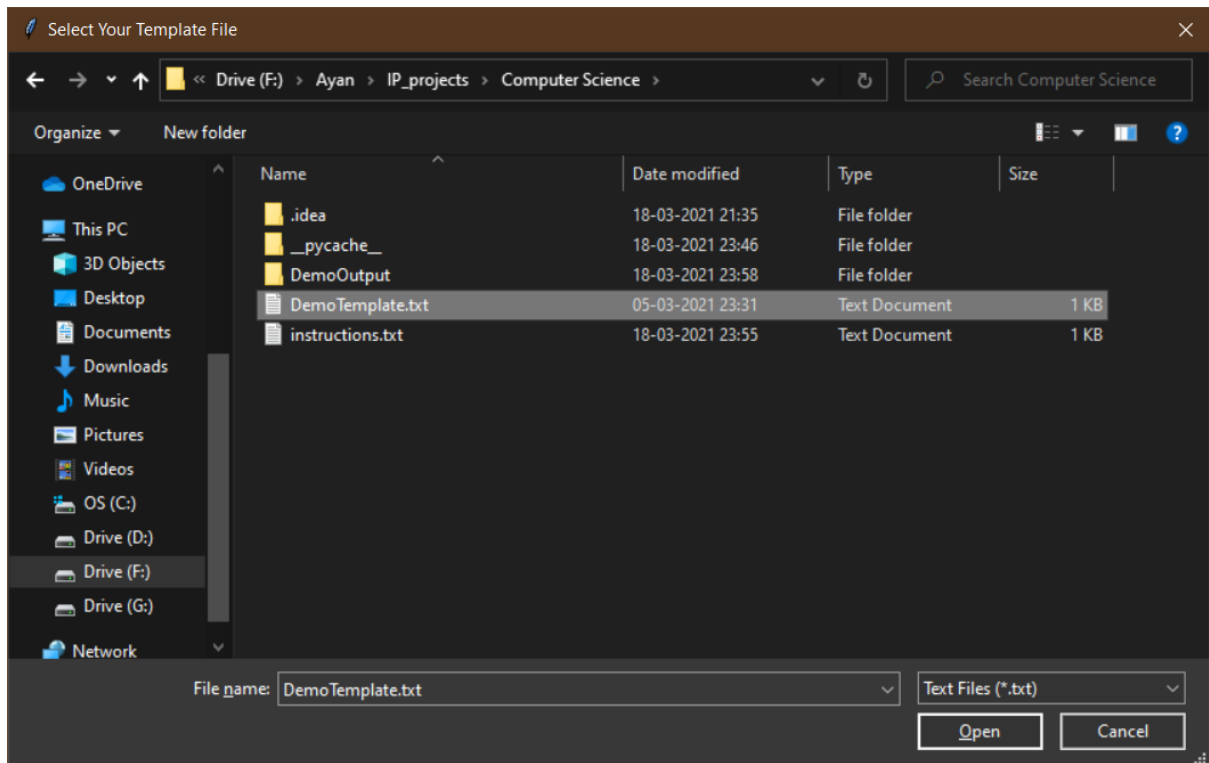
Mail Merge will substitute the values given in the data
in place of variable names used in the template file.

Mail Merge will ask for save location.

The output will be saved in the provided output file.

NOTE: Usage of MySQL databases requires an installation
of MySQL running in localhost.

Press Any Key to Continue...
```



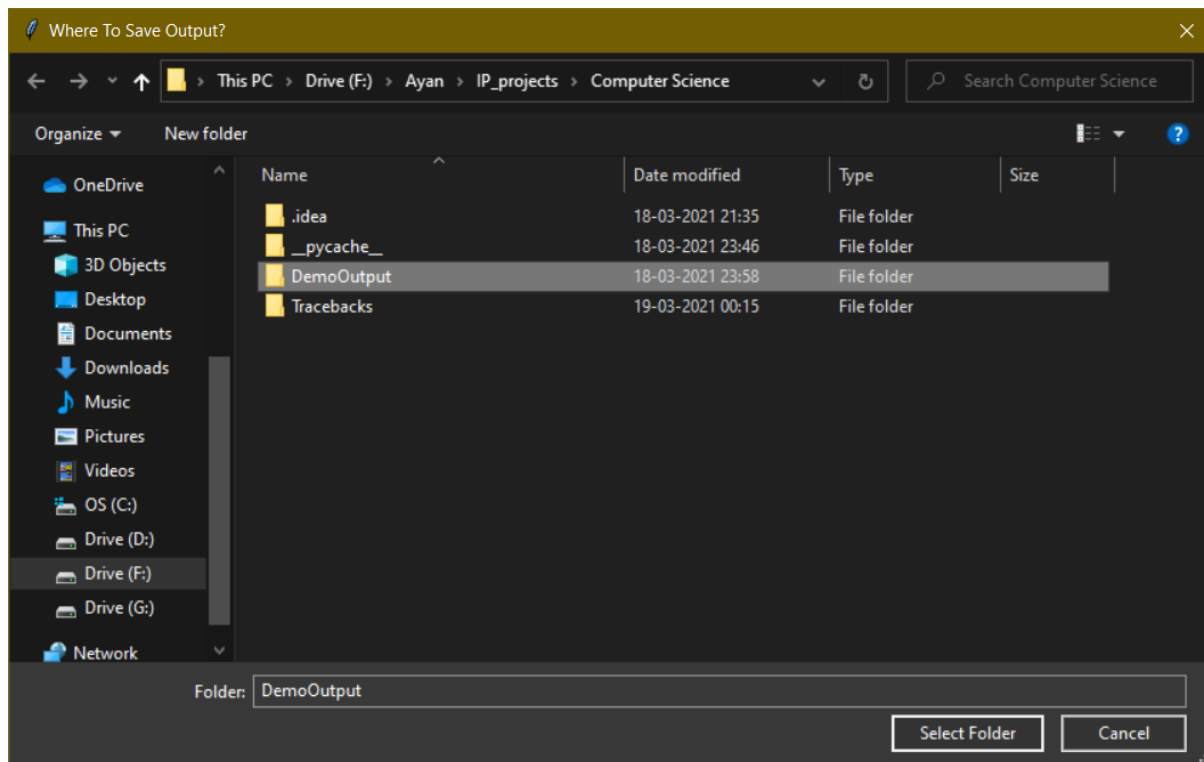
Successfully loaded template.
 Do you want to use data saved in a MySQL Table (Y/N)
 >n
 Fill in the details in the file: F:/Ayan/IP_projects/
 Computer Science/DemoTemplate.csv

	A	B	C	D	E	F	G	H
1	Friend1	Friend2	Friend3	Friend4	Name	Personal_Message	Receiver's Address	Sender's Address
2								
3								
4								



	A	B	C	D	E	F	G	H	I
1	Friend1	Friend2	Friend3	Friend4	Name	Personal_Message	Receiver's Address	Sender's Address	
2	John	Robert	Rubina	Vikas	Jacob	Please bring Pizza.	Lane 123, ABC Street	#555, DEF Street	
3	Robert	Rubina	John	Vikas	Jacob	If you have trouble locating my home call me.	Lane 225, ABC Street	#555, DEF Street	
4	Rubina	John	Mohan	Vikas	Jacob	Don't forget to bring your dog Woofers	Lane 271, ABC Street	#555, DEF Street	
5	Vikas	John	Robert	Rubina	Jacob	Send my regards to Aunt and Uncle	Lane 128, ABC Street	#555, DEF Street	
6	Mohan	John	Rubina	Vikas	Jacob	Please bring some Soft-Drinks	Lane 123, ABC Street	#555, DEF Street	
7									

```
Press any key to continue.  
Successfully Loaded Data  
Do you want to save the data in a MySQL Database? (Y  
/N)  
>y  
Enter MySQL username:root  
  
Enter MySQL password:  
Enter a Table Name:mytable1  
Successfully saved Data
```



```
Mail Merge Completed  
Press any key to quit...
```

1.txt - Notepad

File Edit Format View Help

#555, DEF Street

23 December 2020

To,
Lane 123, ABC Street

Dear John,

I'm hosting a dinner party on Christmas Eve at my residence. I've also invited our classmates Robert, Rubina and Vikas for the party. I would be delighted if you could come and join us to celebrate. We'll end this year with a bash!

Hoping to see you soon.

Jacob

P.S : Please bring Pizza.

Ln 1, Col 1 100% Windows (CRLF) UTF-8

2.txt - Notepad

File Edit Format View Help

#555, DEF Street

23 December 2020

To,
Lane 225, ABC Street

Dear Robert,

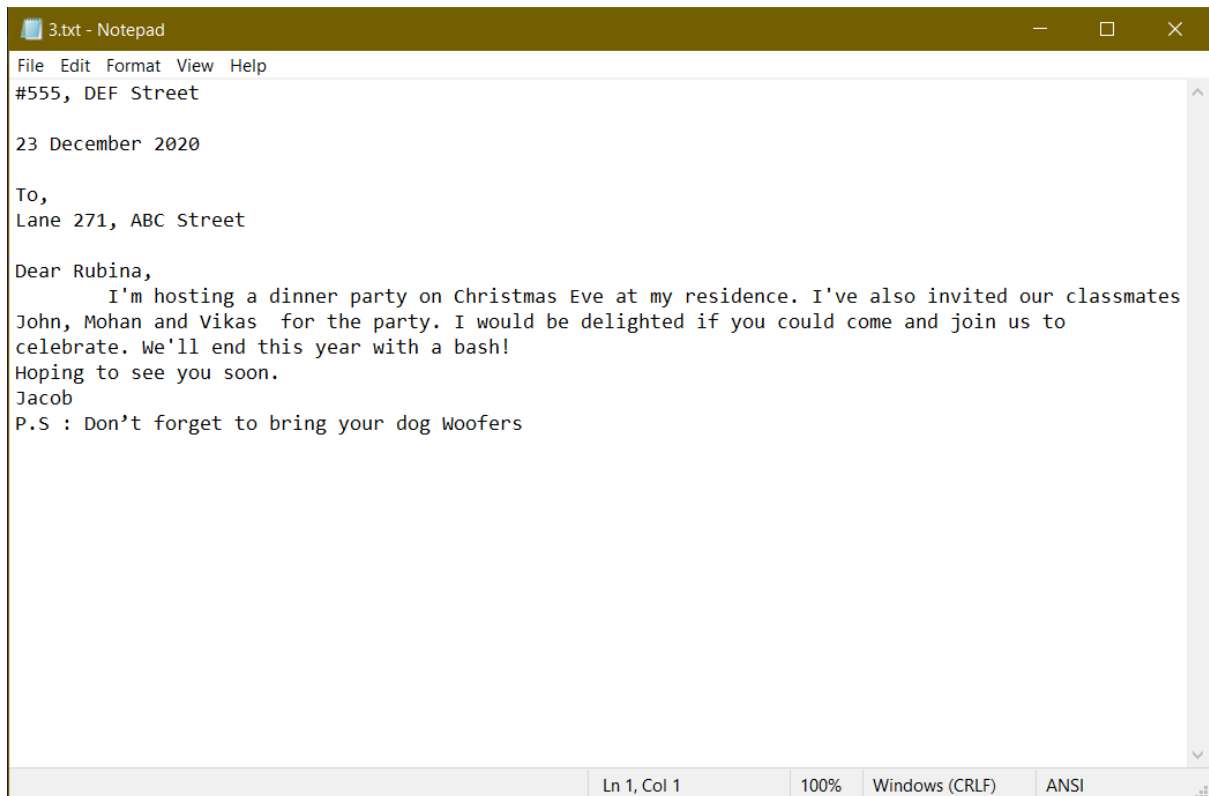
I'm hosting a dinner party on Christmas Eve at my residence. I've also invited our classmates Rubina, John and Vikas for the party. I would be delighted if you could come and join us to celebrate. We'll end this year with a bash!

Hoping to see you soon.

Jacob

P.S : If you have trouble locating my home call me.

Ln 1, Col 1 100% Windows (CRLF) UTF-8

A screenshot of a Notepad window titled '3.txt - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text content is as follows:

#555, DEF Street

23 December 2020

To,
Lane 271, ABC Street

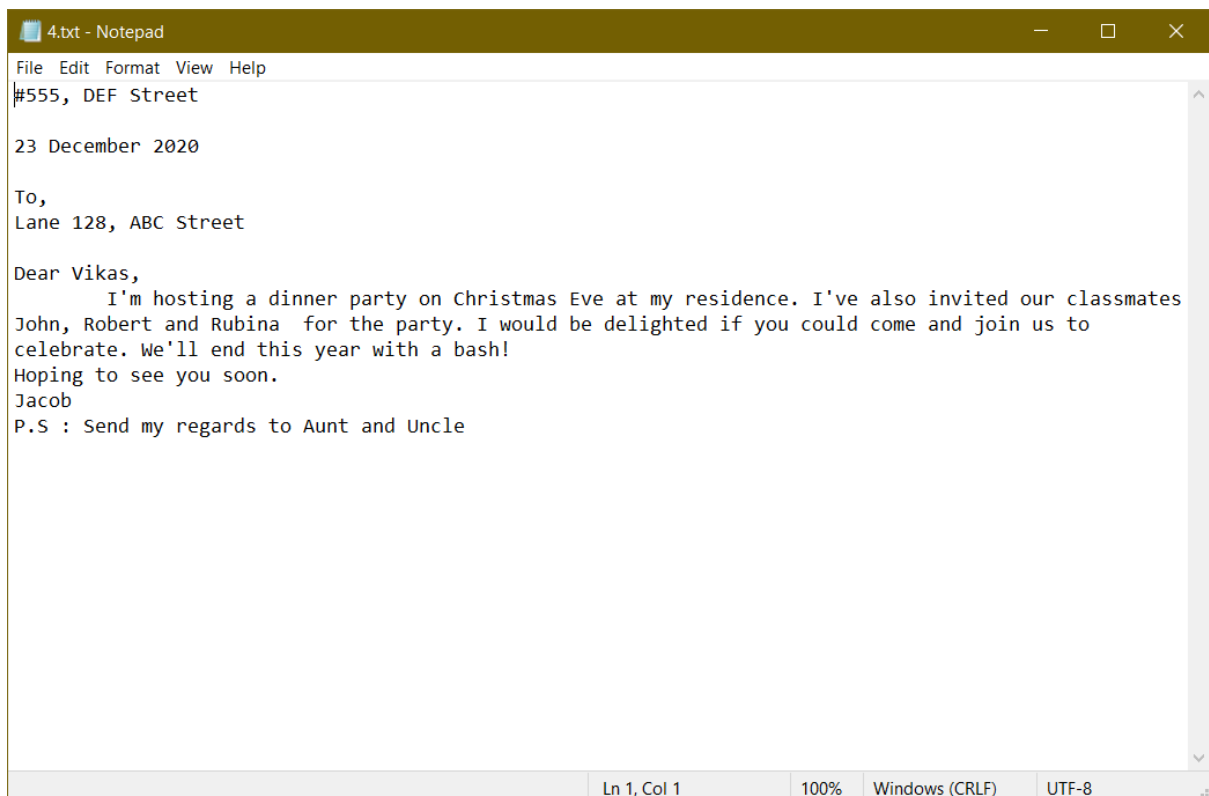
Dear Rubina,

I'm hosting a dinner party on Christmas Eve at my residence. I've also invited our classmates John, Mohan and Vikas for the party. I would be delighted if you could come and join us to celebrate. We'll end this year with a bash!

Hoping to see you soon.

Jacob

P.S : Don't forget to bring your dog Woofers

The status bar at the bottom shows 'Ln 1, Col 1', '100%', 'Windows (CRLF)', and 'ANSI'.A screenshot of a Notepad window titled '4.txt - Notepad'. The window has a menu bar with 'File', 'Edit', 'Format', 'View', and 'Help'. The text content is as follows:

#555, DEF Street

23 December 2020

To,
Lane 128, ABC Street

Dear Vikas,

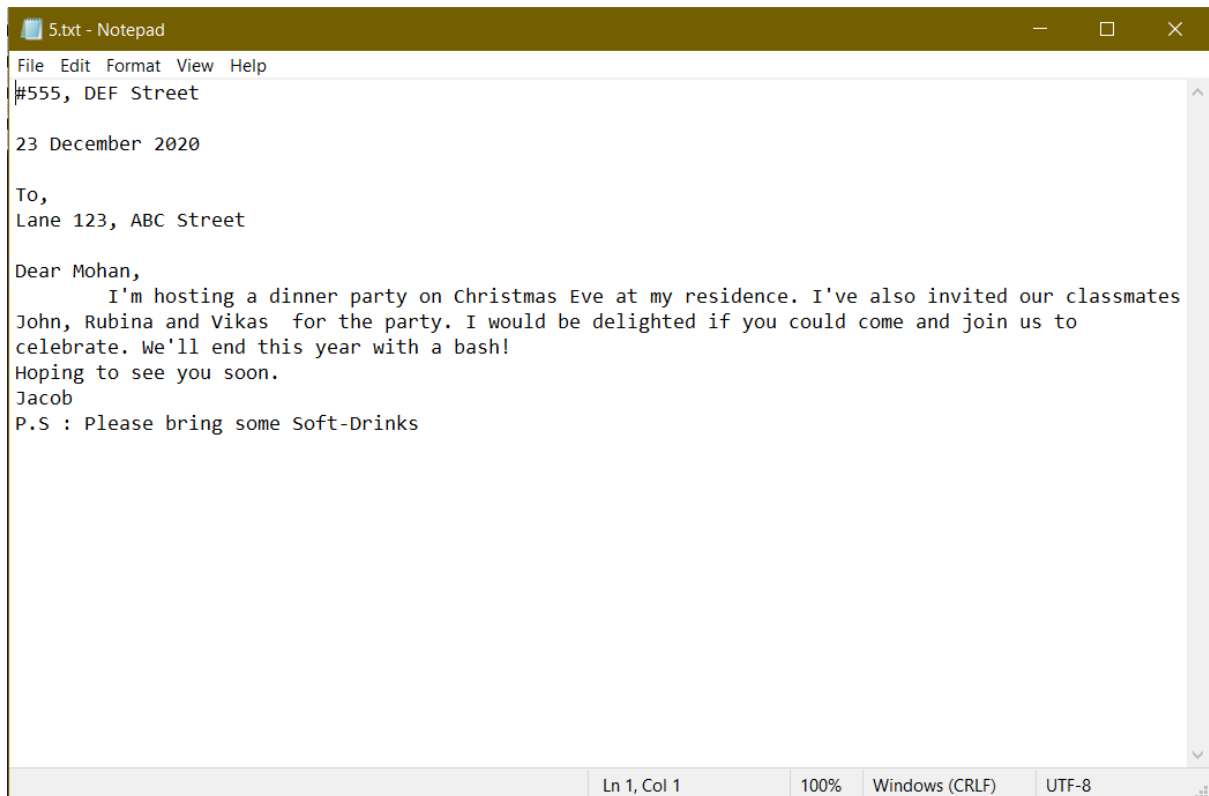
I'm hosting a dinner party on Christmas Eve at my residence. I've also invited our classmates John, Robert and Rubina for the party. I would be delighted if you could come and join us to celebrate. We'll end this year with a bash!

Hoping to see you soon.

Jacob

P.S : Send my regards to Aunt and Uncle

The status bar at the bottom shows 'Ln 1, Col 1', '100%', 'Windows (CRLF)', and 'UTF-8'.



References

python.org

codecademy.com

tutorialspoint.com

developers.google.com/edu/python

learnpython.org

stackoverflow.com