PROJECT DOCUMENTATION

CSE 482: Internet and Web Technology

Mohammed Ashrafuzzaman Khan

Email Verification Module

Language: Python (3.6.7)

Group Members -

Serial#	Name	ID
1	Muhtadi Rahman Akif	1420119042
2	Mostafa Didar Mahdi	1510871042
3	Abir Roy	1410076042
4	Asif Haider Khan	1421341042

Author:

Asif Haider Khan

Mostafa Didar Mahdi

Table of Contents

Abstract

The project is based on a fully software-based module that checks whether a given list of emails are Valid or not. The whole project is written in Python language (version – 3.6.7). The given list of emails are placed on a .txt file and are fed into the program through as the overall atomic ‘Input’ (non-dynamic). The program takes the emails as input and iterates through each email, checking whether the given emails are valid or not. The program uses various Python Libraries to match the emails, open Sockets and SMTP conversations to pull out MX records and finally give the output on the given list of emails as Valid or Invalid depending on some circumstances. The program runs on the terminal and also delivers the ‘OUTPUT’ at the end of the Terminal.

Introduction

E-mail (or Electronic Mail) is a communication service that is provided by some well known Internet Service Providers (ISPs). The service started as sending textual information through a protocol better known to be as the Simple Mail Transfer Protocol (SMTP), an internet standard for email transmission (Wikipedia, 2018). As the service evolved, it became a widely used service, used throughout the world for information transmission. Nowadays e-mail involves sending and receiving textual information as well as simple file transfers, using SMTP in one form or another. According to Wikipedia[1], "Although proprietary systems such as Microsoft Exchange and IBM Notes and webmail systems such as Outlook.com, Gmail and Yahoo! Mail may use their own non-standard protocols internally, all use SMTP when sending or receiving email from outside their own systems". Apart from being such a useful service in action, the service opened the door for many other ill-uses of the service – Spoofing, Spamming, Scamming and Phishing.

Background

Email Validation is an important task for most corporations or groups. **Simple Mail Transfer Protocol** (SMTP) allows any computer to send any sort of email to any sort of sender, but has no feature to authenticate and/or validate a sender/receiver addresses, which can lead to several circumstances of fraudulent or erroneous situations. Email authentication/validation is a collection of techniques which facilitates the confirmation about the origins of an e-mail address (Wikipedia[2], 2018). Several such frameworks, methods and techniques have been constructed to do the task of validation (SPF, DKIM,etc). This project aims at a quite similar task of Validating a given number of emails, through a series of defined tasks and functionalities, provided by the Libraries of the Python programming language.

Problem Analysis

The problem presented in this project sector is to mainly validate any given (number of) email address(es) in any simple programmable method, free from the constraints of a programming language. The inputs to be taken in from a file containing the list of email addresses and the outputs to be printed on the console. [*]

ANALYSIS

The emails listed in the file (**.txt** file as instructed) are to be taken in through some **input method** and then each of the emails listed in the file are to be **checked** on the **validation criteria.** Afterwards, the respective **Results** would be shown to the user who is running the program.

INPUT METHOD

The input method involves scanning the file for the emails (if the file is not empty). The emails can be taken into as an array (list) of 'Strings' and then each email checked recursively.

CHECKING THE VALIDATION CRITERIA

The Validation criteria can be divided into various other conditions, which may include whether the address to verify itself is a valid regular expression or not.

RESULTS

The results are to be printed on the console respective of each email address read from the input list of emails taken, either to be valid or invalid.

*The problem statement is confirmed according to the expectation of our respected instructor Mohammed Ashraduzzaman Khan.

Project Requirements

Core Software Requirements

The project primarily requires resources which can interact with mail services and their respective domains. Open Source programming languages such as Java and Python have numerous Libraries, functionalities and other similar resources that can be used to read and/or write to files or the console, manipulate strings and similar data types and data structures, match regular expressions, open and utilize sockets, get domain information on mailing services. We also require resources that can utilize some operations of the Simple Mail Transfer Protocol. Once all of the resources are accommodated, the project implementation can be initiated.

Physical Dependencies

The physical dependencies or requirements include the availability of a functional Computer that has a favorable Operating System and an environment that can run programs of such nature. Additional physical Dependencies include an active internet connection.

Resourcing similar projects

For ease and reliability of our implementation, we also need to resource out similar projects or works done on email verification/validation on other languages or platforms, usually open-source. An example code (undeveloped) has been resourced from an open source blog from **www.emailhippo.com.**

Implementation

Programming Language used in the implementation of the project upon a majority of opinion was chosen to be Python (version 3.6.7). Python is a high-level, general purpose programming language, that provides numerous libraries among which a few of them are of great importance to this project.

- Python **DNS Resolver** Library:

  DNS Resolver is an API of Python that enables a program to collect resolved records about a specific domain. The domain of a e-mail address is parsed out of the string and then is used to retrieve the MX records of that specific domain.

- Python Simple Mail Transfer Protocol Library:

  Python **smtplib** is a module that allows the instantiation of a SMTP Client Session object that can be used to send or receive emails to any machine that has a SMTP listener daemon [6]. Here, the module is used to create an object 'Server' that does a 'SMTP Conversation', connecting and retrieving Mail Exchange (MX) records, matching and in turn receiving a code that implies whether an address is valid or not.

- Python Socket Library:

  The Socket module allows to instantiate a socket object, providing access to a socket interface. Here, the socket module is used to retrieve the host name required for the validation to take place.

- Python Regular Expressions Library:

  A python module that is used to match string and byte patterns; here the module is used match the given e-mail (input) to check if the address is a regular expression.

- Python Random Library:

  A python module that generates random characters and numbers. The random module is required to generate a random e-mail address that is sent SMTP object before receiving a validation code.

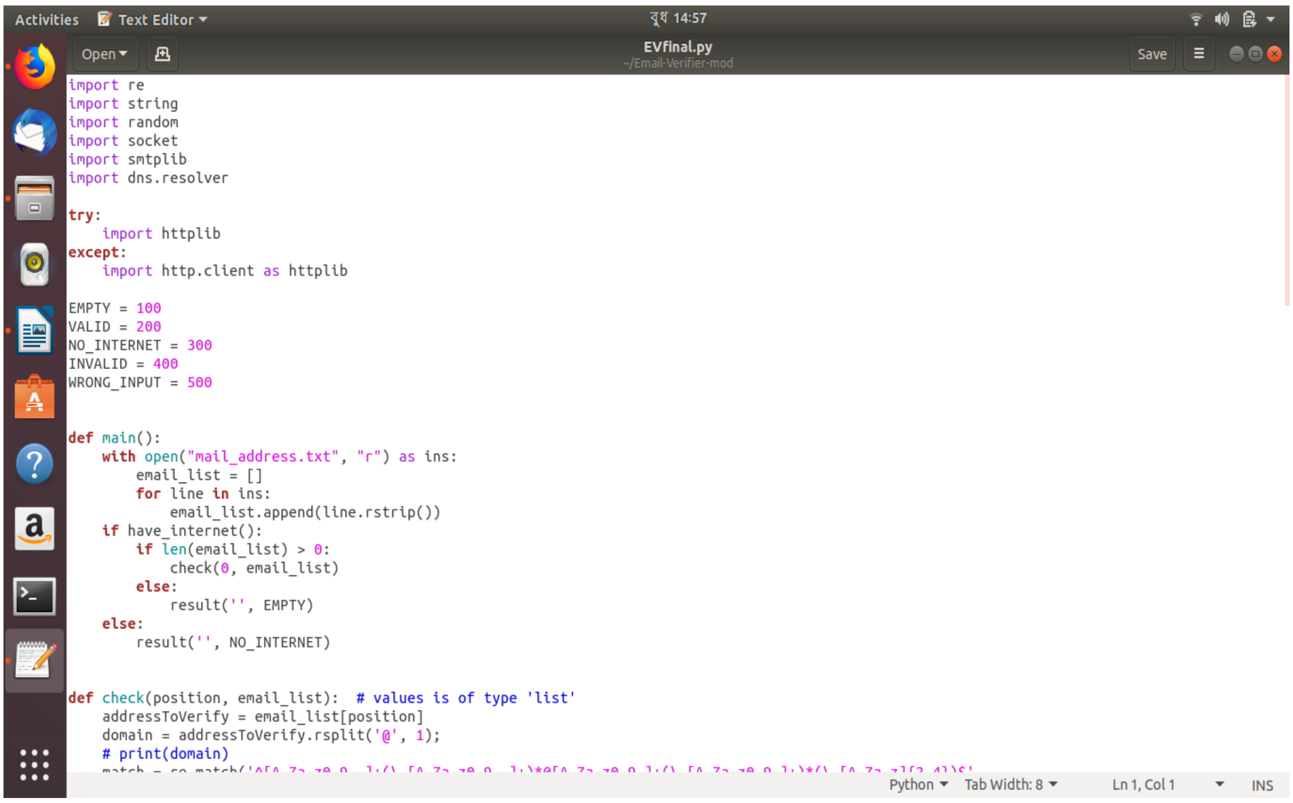
- Python HTTP Client Library:

  A module of python that allows to instantiate a HTTP or HTTPS Client side object. For this project, the module is only used to check if the internet connection is active or not prior to any kind of checking.

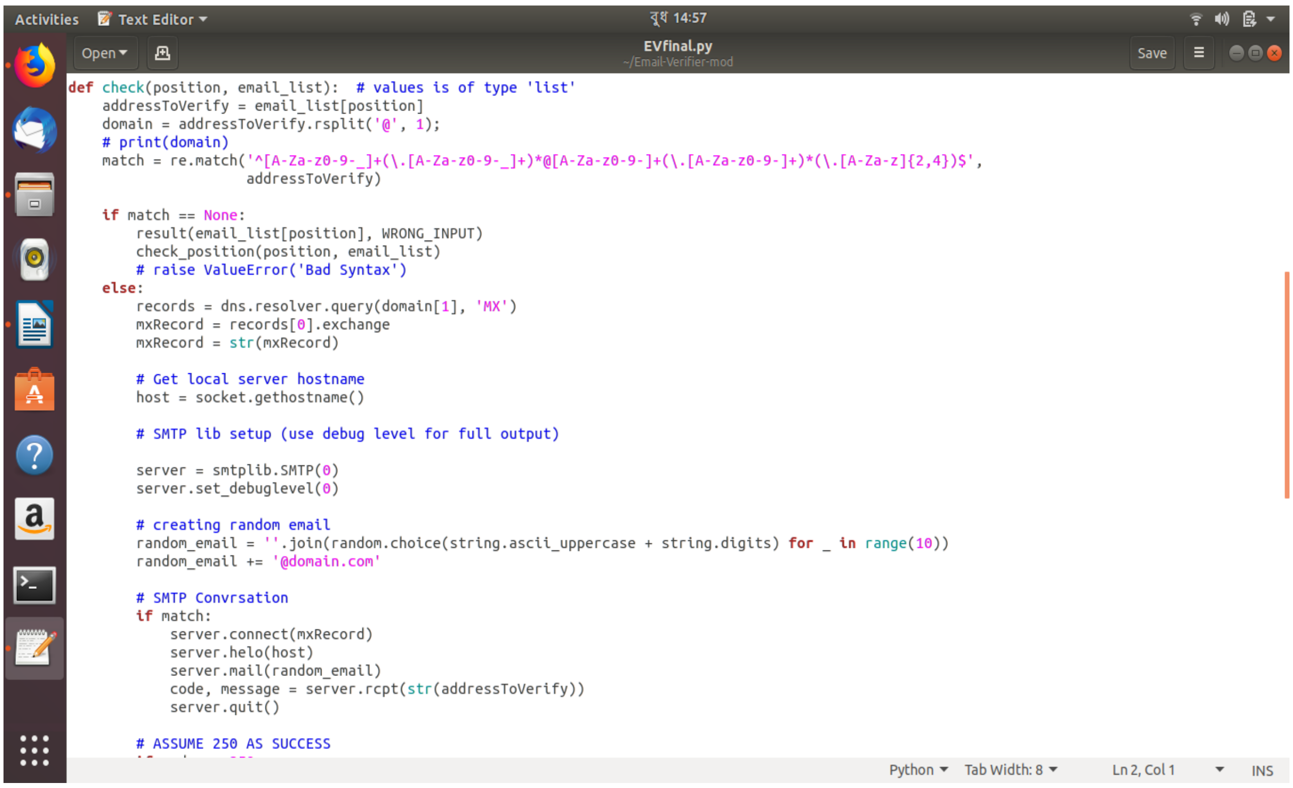
Source Code implementation -

The implementation of the whole project has been compiled to a single Python file. On run-time, the program takes inputs from another file (**mail_address.txt**) which contains the list of the e-mails to be validated. Upon being read and stored into a 'List', each e-mail is then **checked** through function calls recursively. After each successful check has been done, the results (val_response) are then printed on the console alongside each e-mail.

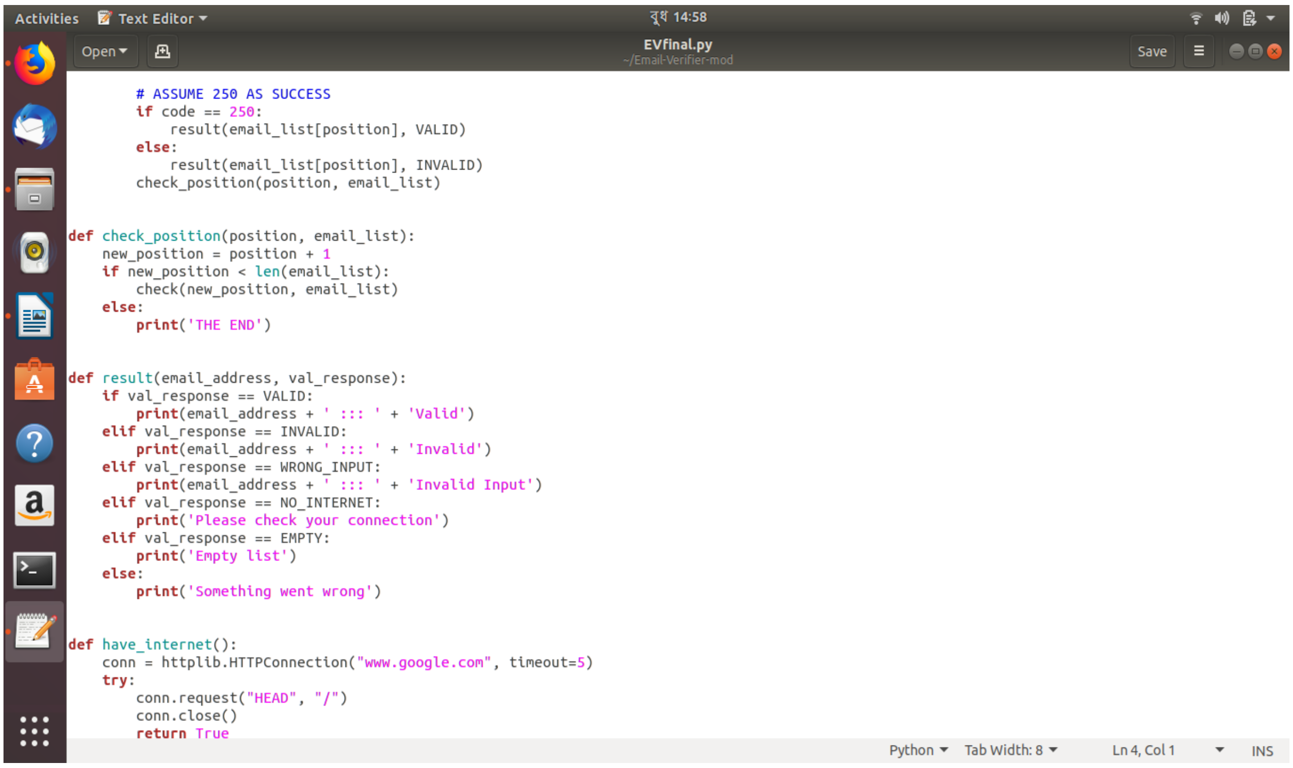
Source Code -



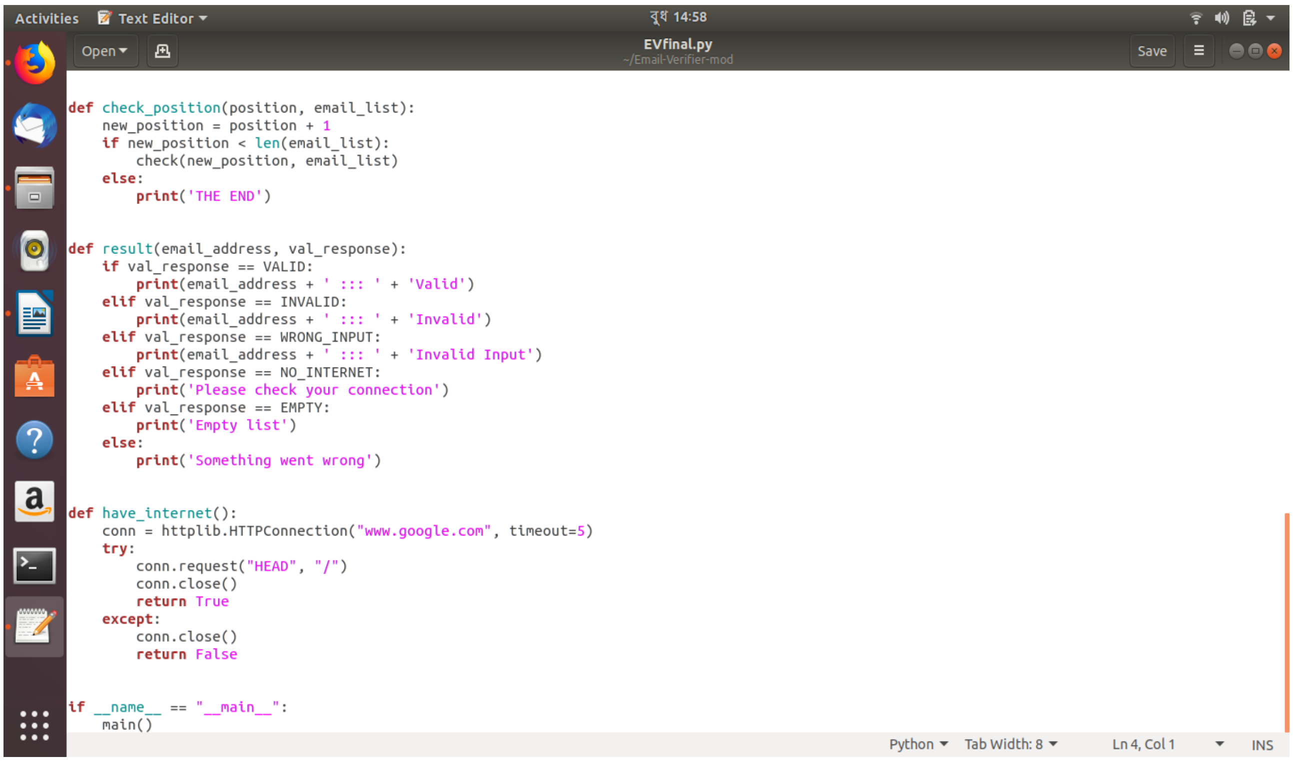
```python
import re
import string
import random
import socket
import smtplib
import dns.resolver

try:
    import httplib
except:
    import http.client as httplib

EMPTY = 100
VALID = 200
NO_INTERNET = 300
INVALID = 400
WRONG_INPUT = 500


def main():
    with open("mail_address.txt", "r") as ins:
        email_list = []
        for line in ins:
            email_list.append(line.rstrip())
    if have_internet():
        if len(email_list) > 0:
            check(0, email_list)
        else:
            result('', EMPTY)
    else:
        result('', NO_INTERNET)


def check(position, email_list):  # values is of type 'list'
    addressToVerify = email_list[position]
    domain = addressToVerify.rsplit('@', 1);
    # print(domain)
```



```python
def check(position, email_list):  # values is of type 'list'
    addressToVerify = email_list[position]
    domain = addressToVerify.rsplit('@', 1);
    # print(domain)
    match = re.match('^[A-Za-z0-9-_]+(\.[A-Za-z0-9-_]+)*@[A-Za-z0-9-]+(\.[A-Za-z0-9-]+)*(\.[A-Za-z]{2,4})$',
                     addressToVerify)

    if match == None:
        result(email_list[position], WRONG_INPUT)
        check_position(position, email_list)
        # raise ValueError('Bad Syntax')
    else:
        records = dns.resolver.query(domain[1], 'MX')
        mxRecord = records[0].exchange
        mxRecord = str(mxRecord)

        # Get local server hostname
        host = socket.gethostname()

        # SMTP lib setup (use debug level for full output)

        server = smtplib.SMTP(0)
        server.set_debuglevel(0)

        # creating random email
        random_email = ''.join(random.choice(string.ascii_uppercase + string.digits) for _ in range(10))
        random_email += '@domain.com'

        # SMTP Convrsation
        if match:
            server.connect(mxRecord)
            server.helo(host)
            server.mail(random_email)
            code, message = server.rcpt(str(addressToVerify))
            server.quit()

        # ASSUME 250 AS SUCCESS
```



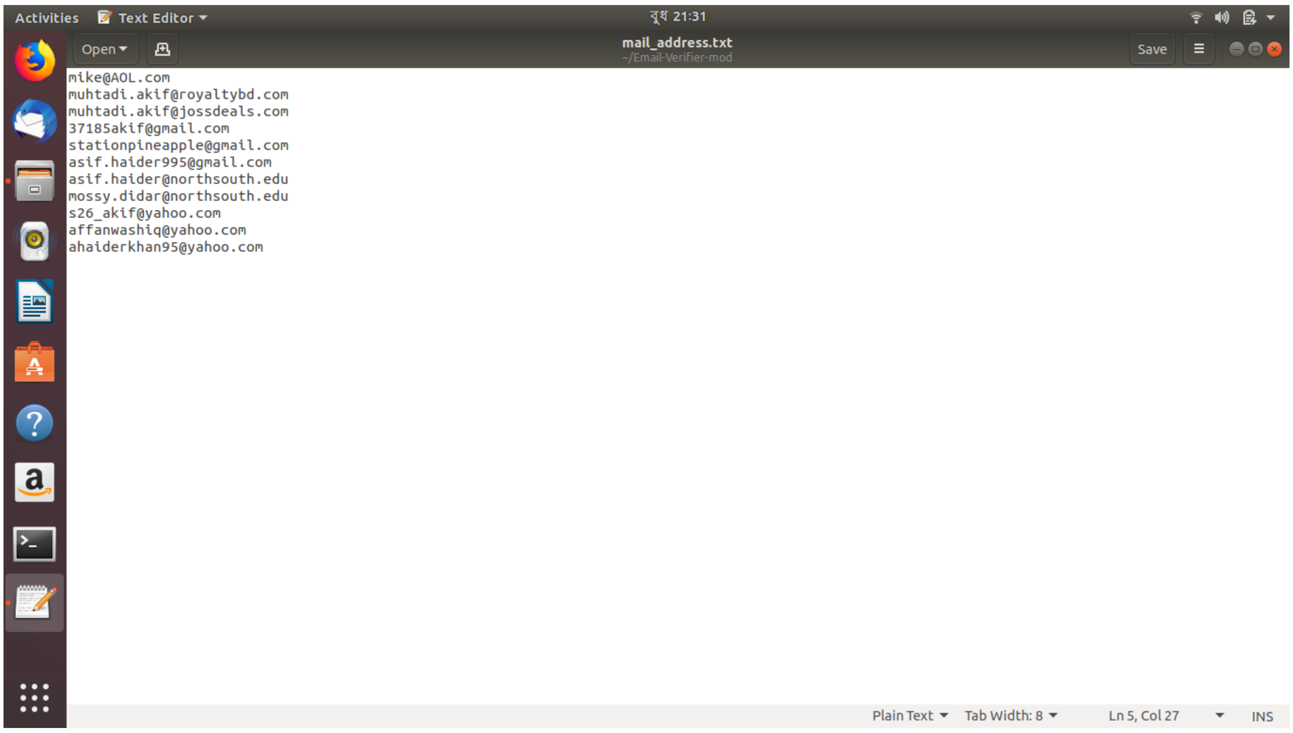
```
        # ASSUME 250 AS SUCCESS
        if code == 250:
            result(email_list[position], VALID)
        else:
            result(email_list[position], INVALID)
        check_position(position, email_list)

def check_position(position, email_list):
    new_position = position + 1
    if new_position < len(email_list):
        check(new_position, email_list)
    else:
        print('THE END')

def result(email_address, val_response):
    if val_response == VALID:
        print(email_address + ' ::: ' + 'Valid')
    elif val_response == INVALID:
        print(email_address + ' ::: ' + 'Invalid')
    elif val_response == WRONG_INPUT:
        print(email_address + ' ::: ' + 'Invalid Input')
    elif val_response == NO_INTERNET:
        print('Please check your connection')
    elif val_response == EMPTY:
        print('Empty list')
    else:
        print('Something went wrong')

def have_internet():
    conn = httplib.HTTPConnection("www.google.com", timeout=5)
    try:
        conn.request("HEAD", "/")
        conn.close()
        return True
```



```
def check_position(position, email_list):
    new_position = position + 1
    if new_position < len(email_list):
        check(new_position, email_list)
    else:
        print('THE END')

def result(email_address, val_response):
    if val_response == VALID:
        print(email_address + ' ::: ' + 'Valid')
    elif val_response == INVALID:
        print(email_address + ' ::: ' + 'Invalid')
    elif val_response == WRONG_INPUT:
        print(email_address + ' ::: ' + 'Invalid Input')
    elif val_response == NO_INTERNET:
        print('Please check your connection')
    elif val_response == EMPTY:
        print('Empty list')
    else:
        print('Something went wrong')

def have_internet():
    conn = httplib.HTTPConnection("www.google.com", timeout=5)
    try:
        conn.request("HEAD", "/")
        conn.close()
        return True
    except:
        conn.close()
        return False

if __name__ == "__main__":
    main()
```

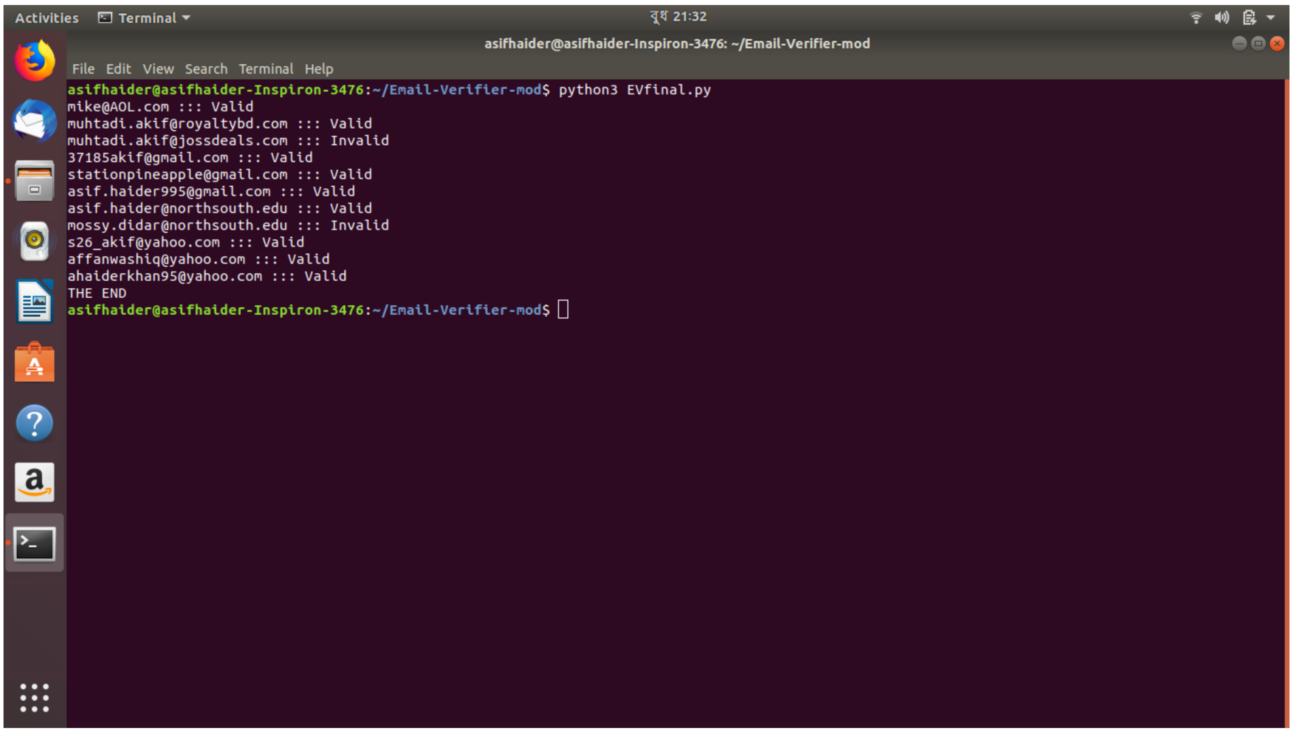
Test Results

Test Data -



```
mike@AOL.com
muhtadi.akif@royaltybd.com
muhtadi.akif@jossdeals.com
37185akif@gmail.com
stationpineapple@gmail.com
asif.haider995@gmail.com
asif.haider@northsouth.edu
mossy.didar@northsouth.edu
s26_akif@yahoo.com
affanwashiq@yahoo.com
ahaiderkhan95@yahoo.com
```

Results -



Activities
Terminal
বুধ 21:32
asifhaider@asifhaider-Inspiron-3476: ~/Email-Verifier-mod
File Edit View Search Terminal Help

```
asifhaider@asifhaider-Inspiron-3476:~/Email-Verifier-mod$ python3 EVfinal.py
mike@AOL.com ::: Valid
muhtadi.akif@royaltybd.com ::: Valid
muhtadi.akif@jossdeals.com ::: Invalid
37185akif@gmail.com ::: Valid
stationpineapple@gmail.com ::: Valid
asif.haider995@gmail.com ::: Valid
asif.haider@northsouth.edu ::: Valid
mossy.didar@northsouth.edu ::: Invalid
s26_akif@yahoo.com ::: Valid
affanwashiq@yahoo.com ::: Valid
ahaiderkhan95@yahoo.com ::: Valid
THE END
asifhaider@asifhaider-Inspiron-3476:~/Email-Verifier-mod$
```

Project Constraints

The project which have been implemented and tested have several constraints which can be categorized as follows -

- Communication constraints – The module requires an active Internet connection in order to work. Since the module employs other modules which requires uninterrupted internet access, the project module does not work without an active internet connection.
- Library constraints – The project module will not run unless the required Library dependencies are met. The python library modules mentioned (smtplib, socket, dns.resolver, re) play a vital role in running the project since most of the tedious work is done by these modules. Without installing the necessary modules the project will not run.
- Domain and Network constraints – Some domains show irregular responses. Domains such as **gmail.com** always show expected results on the emails given as input, however, domains such as **hotmail.com, live.com** show inconsistent results. This could be partially due to the domains or service providers not employing a generic SMTP model. IP blacklisting is also an issue, since some service providers tend to find the activity suspicious and therefore blacklists the IP if the verification module is run with the same provided dynamic IP. This problem can be avoided if the SMTP requests are sent through a proxy network. Generally, the python SMTP library does not provide any sort of proxy support.

Discussion

While implementing the project we had to take a couple of trial and error routes before coming up with a solution. Firstly we had used an API known as **Clearbit.** The Clearbit API uses the same mail validation technique by pinging the mail servers and accepting a request.

The problem with Clearbit was that after sending a few requests it would start blocking out the IP's from which it send requests. Additionally it had a limit of around 100 requests per day for the free version, which really was a setback. Also we needed a server to host the system and as we were not allowed to have one we discarded the Clearbit API module.

Before implementing the project, we had little familiarity with the different connection error codes and what they meant. With the completion of this project, we have a basic level of understanding.

The following error codes were used:

EMPTY = 100
VALID = 200
NO_INTERNET = 300
INVALID = 400
WRONG_INPUT = 500
BLOCKED = 600
INTERNAL_ERROR = 700
BLACK_LIST = 800

To check which e-mails are valid, a txt file containing all possible email addresses are taken as input using the python file read command. All the email addresses in the text file are looped through and checked for validity.

However there were some exceptions:

Exception 1: Connecting to Microsoft servers such as Hotmail, Live and Outlook was unsuccessful using our IP's. The program seemed to crash every time it tried to connect to MS servers.

Exception 2: A virtual private network was required to connect to the mail servers because most of our IPs have been flagged for testing. To add a VPN to this project we had created a proxy server that acts as an intermediary for requests from our script to the mail servers. Although the proxy server class was added to the project, the problem still persisted and still some of the IP's could not get the required results while some IP's worked perfectly.

Exception 3: The project file is still not made executable for windows. However it can be made executable if instructed by the supervisor.

Exception 4: The requirement for this project is at least having **Python 3.**

References

[1]"Simple Mail Transfer Protocol", *En.wikipedia.org*, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Simple_Mail_Transfer_Protocol. [Accessed: 20- Dec- 2018].

[2]"Email authentication", *En.wikipedia.org*, 2018. [Online]. Available: https://en.wikipedia.org/wiki/Email_authentication. [Accessed: 20- Dec- 2018].

[3]"MX record", *En.wikipedia.org*, 2018. [Online]. Available: https://en.wikipedia.org/wiki/MX_record. [Accessed: 20- Dec- 2018].

[4]"2. THE SMTP MODEL", *Freesoft.org*, 2018. [Online]. Available: https://www.freesoft.org/CIE/RFC/821/2.htm. [Accessed: 21- Dec- 2018].

[5]"3.3. VERIFYING AND EXPANDING", *Freesoft.org*, 2018. [Online]. Available: https://www.freesoft.org/CIE/RFC/821/6.htm. [Accessed: 21- Dec- 2018].

[6]"21.17. smtplib — SMTP protocol client — Python 3.6.8 documentation", *Docs.python.org*, 2018. [Online]. Available: https://docs.python.org/3.6/library/smtplib.html. [Accessed: 18- Dec- 2018].

[7]"dnspython", *Dnspython.org*, 2018. [Online]. Available: http://www.dnspython.org/docs/1.15.0/. [Accessed: 19- Dec- 2018].

[8]"18.1. socket — Low-level networking interface — Python 3.6.8 documentation", *Docs.python.org*, 2018. [Online]. Available: https://docs.python.org/3.6/library/socket.html. [Accessed: 19- Dec- 2018].

[10]"re — Regular expression operations — Python 3.7.2 documentation", *Docs.python.org*, 2018. [Online]. Available: https://docs.python.org/3/library/re.html. [Accessed: 19- Dec- 2018].

[11]"What's New In Python 3.5 — Python 3.6.8 documentation", *Docs.python.org*, 2018. [Online]. Available: https://docs.python.org/3.6/whatsnew/3.5.html#http-client. [Accessed: 19- Dec- 2018].

[12]"Verify emails over SOCKS proxy using python – Arjun Singh Yadav – Medium", *Medium*, 2018. [Online]. Available: https://medium.com/@arjunsinghy96/verify-emails-over-socks-proxy-using-python-5589cb75c405. [Accessed: 26- Dec- 2018].

[13]"RFC 821 - Simple Mail Transfer Protocol", *Tools.ietf.org*, 2018. [Online]. Available: https://tools.ietf.org/html/rfc821.html. [Accessed: 26- Dec- 2018].

[14] “Clearbit API Module” - https://clearbit.com/docs [Accessed: 26- Dec- 2018]

[15] “Proxy Server using Python” - https://null-byte.wonderhowto.com/how-to/sploit-make-proxy-server-python-0161232/ [Accessed: 26- Dec- 2018]