imaplib - Simple Guide to Manage Mailboxes using Python¶

Internet Message Access Protocol is an Internet protocol that lets us access email messages from mailboxes. It even let us move mails between mailboxes, create directories within mailboxes, mark emails as read/unread, marks emails as important, etc. It's the protocol used by all the email clients to access the mailbox of the user. The current version of IMAP is IMAP4rev1 generally referred to as IMAP4. Python provides us with the library named **imaplib** which lets us access our mailbox and read emails over IMAP.

As a part of this tutorial, we'll explain with simple examples how we can use **imaplib** to access mailboxes, read emails, copy mails between directories, create directories, etc. We'll be logging in to our Gmail and Yahoo mail servers for this. Gmail and Yahoo mail provides access to IMAP4 over port number 993. Gmail and Yahoo mail let us access the mailbox over SSL only. The below link has information about hostname and port details about Gmail and Yahoo mail that we'll be using for our examples.

- Gmail Server and Port Details
- Yahoo mail Server and Port Details

Please make a **NOTE** that Gmail and yahoo mail nowadays does not let us login to the server using a password nowadays if we are trying to access their account through some other applications/scripts. It requires us to create an app password that will be used to authenticate that application/script.

Below we have given steps to create an app whose password we'll use to access Gmail and Yahoo mail mailboxes.

Steps to create an app password for Gmail

- Login to Gmail.
- Click on **Manage your Google Account** by clicking on the profile icon on the top right.
- Click on **Security** from the left list.
- Click on **App passwords** on the security page (**Signing into Google** Section). It'll ask for a google password to log in again.
- There will be two dropdowns to select (**Select app** and **Select device**). Select **Other** option for both dropdowns. It'll ask you to give the name of the app. Give the app the name that you like (Ex - SMTP Tutorial). If you had previously created some app then they will be shown in the list along with the last date when you last used it. You can delete old apps from here if you are not using them anymore.
- Click on **Generate** and it'll open a window where a temporary password (16 characters) will be displayed.
- Save this password to use it for the tutorial. It'll display the password only once hence save it in a safe place otherwise you will need to create a new app if the password is lost.

Steps to create an app password for yahoo mail

- Login to Yahoo Mail.
- Click on **Account Info** by clicking on the profile icon from the top right.
- Click on **Account Security** from the left list.
- Click on **Manage app passwords** at last on that page.
- On the newly opened window select **Other app** from the dropdown list. This page will also display a list of already created apps with their last use time. You can delete olds apps from here if you are not using them anymore.
- It'll show a text box to enter the name of the app. Give the name to the app (Ex - SMTP Tutorial).
- Click on **Generate** button to generate an app password. It'll open a new window with a password (16 characters).
- Save this password to use for the future. It'll display the password only once hence save it in a safe place otherwise you will need to create a new app if the password is lost.

Please **DELETE** the app if you are no longer using that app for your own safety.

Example 1: Establish Connection to Mail Server¶

As a part of our first example, we are explaining how we can make connections to the mail server over IMAP4. The **imaplib** provides us with a class named **IMPA4** and **IMAP4_SSL** which can be used to make a connection to the server.

- **IMAP4(host='',port=IMAP4_PORT,timeout=None)** - This constructor creates and returns an instance of **IMAP4** by establishing connection with the host over the specified port. This instance can be used to communicate with the server. If the **timeout** parameter is specified then the process will timeout after trying for that many seconds if the connection is still not established.
- **IMAP4_SSL(host='',port=IMAP4_SSL_PORT,timeout=None)** - This constructor creates and returns an instance of **IMAP4_SSL** by establishing connection with host over specified port. It uses SSL connection which is secure. It works exactly like **IMAP4** but provides connection over SSL.

Our code for this example has three-part. The first part tries connection with the Gmail mail server over port 993 using **IMAP4** class. The second part tries the same but with the timeout set to 3 seconds using **IMAP4** class. The third part tries to connect with the Gmail server using **IMAP4_SSL** class. Our first two parts of the code fail to connect to the server because Gmail and Yahoo mail allows connection over SSL only. The third part of our code is successfully able to connect to the mail server.

In [1]:

```
import imaplib
import time

################### IMAP ########################
start = time.time()
try:
    imap = imaplib.IMAP4(host="imap.gmail.com", port=993)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap = None

print("Connection Object : {}".format(imap))
print("Total Time Taken  : {:,.2f} Seconds\n".format(time.time() - start))

############### IMAP with Timeout ######################
start = time.time()
try:
    imap = imaplib.IMAP4(host="imap.gmail.com", port=993, timeout=3)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap = None
```

```
print("Connection Object : {}".format(imap))
print("Total Time Taken  : {:,.2f} Seconds\n".format(time.time() - start))

################ IMAP SSL ##############################
start = time.time()
try:
    imap_ssl = imaplib.IMAP4_SSL(host="imap.gmail.com", port=993)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap_ssl = None

print("Connection Object : {}".format(imap_ssl))
print("Total Time Taken  : {:,.2f} Seconds".format(time.time() - start))
```

```
ErrorType : abort, Error : socket error: EOF
Connection Object : None
Total Time Taken  : 10.68 Seconds

ErrorType : timeout, Error : timed out
Connection Object : None
Total Time Taken  : 3.15 Seconds

Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c64c550>
Total Time Taken  : 1.00 Seconds
```

Example 2: Log in, Log Out¶

As a part of our second example, we are demonstrating how we can log in to the server using credentials and log out once the connection is established using **IMAP4** instance.

Important Methods of IMAP4_SSL Instance¶

- **login(user, password) -** It accepts user name and password as input and returns a tuple of two values. The first value in the tuple is a response code and the second value is a list with response messages.
- **logout() -** This method logs the user out of the server and returns a tuple of two values where the first value is the response code and the second value is a list with a response message.

Please make a **NOTE** that majority of methods of **IMAP4/IMAP4_SSL** returns a tuple of two values of the form **(reponse_code, [reponse_message,...])**. The response code is either **OK** or **NONE**. The response message is a list with a single binary string or a list of binary strings based on the method call. If the method is a simple command like login/logout then the second value of the tuple is a list with one value (binary message) specifying the response of the server after the execution of that method.

Our code for this example first creates an instance of **IMAP4_SSL** and establishes a connection with the server. It then calls the **login()** method with user name and password to log in a user to the server. At last, it calls **logout()** to log the user out. All our method calls are wrapped inside of the try-except block to catch any errors that can happen. We are also printing response codes and messages.

Please make a **NOTE** that we are calling **decode()** methods on response messages to convert them from bytes to string format.

In [2]:

```
import imaplib
import time

################ IMAP SSL ##############################
start = time.time()
try:
    imap_ssl = imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap_ssl = None

print("Connection Object : {}".format(imap_ssl))
print("Total Time Taken  : {:,.2f} Seconds\n".format(time.time() - start))

################ Login to Mailbox ######################
print("Logging into mailbox...")
try:
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}\n".format(response[0].decode()))

################ Logout of Mailbox ######################
print("\nLogging Out....")
try:
    resp_code, response = imap_ssl.logout()
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}".format(response[0].decode()))
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c64c460>
Total Time Taken  : 0.87 Seconds

Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)


Logging Out....
Response Code : BYE
Response      : LOGOUT Requested
```

Example 3: List Mailboxes within Given Directory¶

As a part of our third example, we'll explain how we can list down directories present within the mailbox using **list()** method of **IMAP4/IMAP4_SSL** instance.

Important Methods of IMAP4_SSL Instance¶

- **list(directory,pattern) -** This method returns list of mailboxes present in the **directory** that matches given **pattern** as input. It returns a tuple of two values where the second value is a list of byte strings specifying directory names.

Our code for this example tarts like our previous examples by establishing a connection with the server and logging in to it. It then calls **list()** method without any parameters to list down all mailboxes present. We are then looping through all directories and printing them. We have then called **list()** method with parameter **directory** set to **[Gmail]** so that it returns directory which are subdirectories of it. Our code then logs out of the server.

In [3]:

```
import imaplib
import time

################ IMAP SSL ##############################
start = time.time()
try:
    imap_ssl = imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap_ssl = None

print("Connection Object : {}".format(imap_ssl))
print("Total Time Taken  : {:,.2f} Seconds\n".format(time.time() - start))

############### Login to Mailbox ######################
print("Logging into mailbox...")
try:
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}\n".format(response[0].decode()))

################### List Directores ####################
try:
    resp_code, directories = imap_ssl.list()
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, directories = None, None

print("Response Code : {}".format(resp_code))
print("========= List of Directories ================\n")
for directory in directories:
    print(directory.decode())

################### List Directores ####################
try:
    resp_code, directories = imap_ssl.list(directory="[Gmail]")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, directories = None, None

print("Response Code : {}".format(resp_code))
print("\n========= List of Directories ================\n")
for directory in directories:
    print(directory.decode())

############### Logout of Mailbox ######################
print("\nLogging Out....")
try:
    resp_code, response = imap_ssl.logout()
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}".format(response[0].decode()))
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c665880>
Total Time Taken  : 1.18 Seconds

Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories ================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Myntra"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"
```

```
Response Code : OK

========= List of Directories ================

(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"

Logging Out....
Response Code : BYE
Response      : LOGOUT Requested
```

Example 4: List Mailboxes Matching given Pattern¶

As a part of our fourth example, we are again explaining the usage of **list()** method to list down mailboxes present in the directory but this time we are explaining **pattern** parameter. The **pattern** parameter accepts a pattern to match the list of directories that matches that pattern.

Our code for this example starts by establishing a connection with the server and logs in to it like our previous examples. It then calls **list()** method with a pattern that will match all directories which has **Coursera** word in them. We print all subdirectories which match this pattern. We have then called **list()** method again with a pattern that matches for word **Coursera/Algo** at the beginning of the directory name. We again print a list of directories that matches this pattern. At last, we log out of the server.

In [4]:

```
import imaplib
import time

################ IMAP SSL ##############################
start = time.time()
try:
    imap_ssl = imaplib.IMAP4_SSL(host="imap.mail.yahoo.com", port=imaplib.IMAP4_SSL_PORT)
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    imap_ssl = None

print("Connection Object : {}".format(imap_ssl))
print("Total Time Taken  : {:,.2f} Seconds\n".format(time.time() - start))

############### Login to Mailbox ######################
print("Logging into mailbox...")
try:
    resp_code, response = imap_ssl.login("sunny.2309@yahoo.in", "app_password")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}\n".format(response[0].decode()))

################### List Directores ####################
try:
    resp_code, directories = imap_ssl.list(pattern="*Coursera*")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, directories = None, None

print("Response Code : {}".format(resp_code))
print("========= List of Directories ================\n")
for directory in directories:
    print(directory.decode())

################### List Directores ####################
try:
    resp_code, directories = imap_ssl.list(pattern="Coursera/Algo*")
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, directories = None, None

print("Response Code : {}".format(resp_code))
print("\n========= List of Directories ================\n")
for directory in directories:
    print(directory.decode())

############### Logout of Mailbox ######################
print("\nLogging Out....")
try:
    resp_code, response = imap_ssl.logout()
except Exception as e:
    print("ErrorType : {}, Error : {}".format(type(e).__name__, e))
    resp_code, response = None, None

print("Response Code : {}".format(resp_code))
print("Response      : {}".format(response[0].decode()))
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c67a520>
Total Time Taken  : 1.55 Seconds

Logging into mailbox...
Response Code : OK
Response      : LOGIN completed

Response Code : OK
========= List of Directories ================

(\HasChildren) "/" "Coursera"
(\HasNoChildren) "/" "Coursera/Algorithms - 1"
(\HasNoChildren) "/" "Coursera/Algorithms 1 - Standford"
```

```
(\HasNoChildren) "/" "Coursera/Analysis of algorithms"
(\HasNoChildren) "/" "Coursera/Cloud Android"
(\HasNoChildren) "/" "Coursera/Cryptography"
(\HasNoChildren) "/" "Coursera/Enhance Your Career"
(\HasNoChildren) "/" "Coursera/Machine Learning"
(\HasNoChildren) "/" "Coursera/POSA"
(\HasNoChildren) "/" "Coursera/Programming Mobile Application"
Response Code : OK

========= List of Directories =================

(\HasNoChildren) "/" "Coursera/Algorithms - 1"
(\HasNoChildren) "/" "Coursera/Algorithms 1 - Standford"

Logging Out....
Response Code : BYE
Response      : IMAP4rev1 Server logging out
```

Example 5: Use IMAP4_SSL as a Context Manager¶

Our code for this example is almost the same as our code for the third example with the only change that we are using **IMAP4_SSL** instance as a context manager. As we are using **IMAP4_SSL** instance as a context manager, we don't need to call **logout()** method to log out of the server. When we are exiting of context, the **logout()** method will be called by the context manager.

In [5]:

```
import imaplib
import time

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ################### List Directores ####################
    resp_code, directories = imap_ssl.list()

    print("Response Code : {}".format(resp_code))
    print("========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())

    ################### List Directores ####################
    resp_code, directories = imap_ssl.list(directory="[Gmail]")

    print("Response Code : {}".format(resp_code))
    print("\n========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c6650a0>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories =================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Myntra"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"
Response Code : OK

========= List of Directories =================

(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"
```

Example 6: Display Message Count Per Directory¶

As a part of our sixth example, we are demonstrating how we can display the count of mail messages present per-directory using **select()** method of **IMAP4_SSL** instance.

Important Methods of IMAP4_SSL Instance¶

- **select(mailbox='INBOX',readonly=False) -** This method select as mailbox given by parameter **mailbox** so that all the operations that we'll perform after this method will be performed on this selected mailbox. It's set to **INBOX** by default. It returns the tuple of two values where the second value has a count of the mails for a given mailbox.
    - The **readonly** parameter accepts a boolean value. If we set this parameter to **True** then it'll not let us modify mails inside of the mailbox. If set to **False** then it'll let us modify messages. If we set this parameter to **False** and retrieve mails from the mailbox then it'll be marked as read if it's not read yet.
- **close() -** It closes currently selected mailbox. The deleted messages are permanently removed from the mailbox. We should call this command before logging out of the mailbox.

Our code for this example starts by creating an instance of **IMAP4_SSL** and logging in to the mail server. It then lists down a list of directories present in the mailbox using **list()** method. We are then looping through the list of directories and retrieving mail count per-directory using **select()** method. We are accessing the mailbox in read-only mode.

In [6]:

```
import imaplib
import time

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ################### List Directores ####################
    resp_code, directories = imap_ssl.list()

    print("Response Code : {}".format(resp_code))
    print("========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())

    ############### Number of Messages per Directory ############
    print("\n========== Mail Count Per Directory ==============\n")
    for directory in directories:
        directory_name = directory.decode().split('"')[-2]
        directory_name = '"' + directory_name + '"'
        try:
            resp_code, mail_count = imap_ssl.select(mailbox=directory_name, readonly=True)
        except Exception as e:
            print("{} - ErrorType : {}, Error : {}".format(directory_name, type(e).__name__, e))
            resp_code, mail_count = None, None

    ############# Close Selected Mailbox ######################
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c67a190>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories =================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Myntra"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"

========== Mail Count Per Directory ===============

"ELITMUS" - 32
"INBOX" - 3072
"Myntra" - 0
"Personal" - 4
"Receipts" - 0
"Work" - 0
"[Gmail]/All Mail" - 3670
"[Gmail]/Drafts" - 1
"[Gmail]/Important" - 1302
"[Gmail]/Sent Mail" - 612
"[Gmail]/Spam" - 439
"[Gmail]/Starred" - 44
"[Gmail]/Trash" - 4
```

Example 7: Read Mails from Mailbox¶

As a part of our seventh example, we are demonstrating how we can retrieve mails from mailboxes using **search()** method.

Important Methods of IMAP4_SSL Instance¶

- **search(charset, criterion) -** This method searches mailbox based on criteria provided using **criterion** parameter. The **criterion** parameter accepts string specifying which mails to retrieve. This method will return a tuple of two values where the second value will have a byte string having a list of mail ids separated by space.

- **fetch(message_set,message_parts) -** This method accepts string specifying list of message ids and string specifying message parts to retrieve for all messages.

    - The **message_set** accepts strings like **1:4** (retrieve mails from id 1 to 4), **1:4,6:10** (retrieve mails from id 1 to 4 and 6 to 10), **5** (retrieve mail with id 5), **5:*** (retrieve mail using id 5 till the end), **1,2,3** (retrieve mails with id 1,2 and 3), etc.
    - The **message_parts** parameter accepts string values like **RFC822** (returns whole message), **UID** (unique identifier of the message), **BODY** (body of message), etc. We can provide more than one strings separated by space.

Below link has detailed information about strings which can be used as **criterion** parameter of **search()** method and **message_parts** parameter of **fetch()** method.

- IMAP Commands

Our code for this example starts by connecting to the server by creating an instance of **IMAP4_SSL**. It then logs in to the server and then selects **ELITMUS** as a mailbox. It then retrieves mail ids for all mails inside of **ELITMUS** mailbox using **select()** method. The mail ids are an integer from 1 to a number of mails in the box. The number 1 represents the oldest mail and the maximum number represents the latest mail id. We then take the last two mail ids and retrieve mail contents using **fetch()** method. We are creating an instance of **EmailMessage** which is a wrapper class provided by **email** module which is used to represent mails. We can generate an instance of **EmailMessage** by calling **message_from_bytes()** method of **email** module. The method accepts a message represented as byte string as input and returns **EmailMessage** instance.

We are then printing various parts of the mail like **from, to, bcc, data, subject, and body**. We are calling **walk()** method instance of **EmailMessage** to retrieve all parts of the message and printing only part which is text.

If you are interested in learning about how emails are represented in Python then please feel free to check our tutorial on module **email** which is used for it.

- email - How to Represent an Email Message in Python

In [2]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="ELITMUS", readonly=True)

    ############### Retrieve Mail IDs for given Directory #############
    resp_code, mails = imap_ssl.search(None, "ALL")
    print("Mail IDs : {}\n".format(mails[0].decode().split()))

    ############### Display Few Messages for given Directory #############
    for mail_id in mails[0].decode().split()[-2:]:
        print("================= Start of Mail [{}] ====================".format(mail_id))
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("From       : {}".format(message.get("From")))
        print("To         : {}".format(message.get("To")))
        print("Bcc        : {}".format(message.get("Bcc")))
        print("Date       : {}".format(message.get("Date")))
        print("Subject    : {}".format(message.get("Subject")))

        print("Body : ")
        for part in message.walk():
            if part.get_content_type() == "text/plain":
                body_lines = part.as_string().split("\n")
                print("\n".join(body_lines[:12])) ### Print first 12 lines of message

        print("================= End of Mail [{}] ====================\n".format(mail_id))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f935814b190>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Mail IDs : ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25

================== Start of Mail [31] ====================
From       : eLitmus DoNotReply <elitmus.donotreply.fresher1@gmail.com>
To         : undisclosed-recipients:;
Bcc        : elitmus-jobs@googlegroups.com
Date       : Mon, 1 Feb 2021 19:14:41 +0530
Subject    : iQuanti | 4.5-5 LPA | Analyst | 2020 | all branches
Body :
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

Hi Friends,

iQuanti - a *digital marketing* and *analytics* company is hiring freshers
from all branches.

They enable organizations to use digital data to expand business. With 90%
of their clients being fortune 500 companies, they are growing at 200% year
on year.

================== End of Mail [31] ====================

================== Start of Mail [32] ====================
From       : eLitmus DoNotReply <elitmus.donotreply.fresher1@gmail.com>
To         : undisclosed-recipients:;
Bcc        : elitmus-jobs@googlegroups.com
Date       : Mon, 1 Mar 2021 09:59:02 +0530
Subject    : Girl Power Talk | Digital Marketing Analyst | 2021 | Mohali | upto
 3.8 LPA
Body :
Content-Type: text/plain; charset="UTF-8"
Content-Transfer-Encoding: quoted-printable

Hi Friends,

Girl Power Talk, a digital marketing company, is hiring freshers from the
2021 batch.

Girl Power Talk, in partnership with its sister company Blue Ocean Global
Technology proactively builds stellar reputations. They create and promote
top digital assets that accelerate the growth of brand equity and provide
comprehensive reputation management services, which include
================== End of Mail [32] ====================


Closing selected mailbox....
```

Example 8: Create, Delete and Rename Mailboxes¶

As a part of our eighth example, we are demonstrating how we can create, delete and rename mailboxes using **create()**, **delete()** and **rename()** methods of **IMAP4_SSL** instance.

Important Methods of IMAP4_SSL Instance¶

- **create(mailbox)** - This method creates a mailbox.
- **delete(mailbox)** - This method deletes a mailbox.
- **rename(oldmailbox,newmailbox)** - This method renames a mailbox.
- **expunge()** - This method permanently remove deleted items from the mailbox.

Our code for this example starts by establishing a connection to the server by creating an instance of **IAP4_SSL**. We are then logging in to the mail server. We are then selecting mailbox as **INBOX** using **select()** method. We are then creating a mailbox named **ELITMUS_2** using **create()** method. Once the mailbox is created, we are listing down directories using **list()** method to check whether the mailbox was created or not. We are then deleting mailbox using **delete()** method. Once the mailbox is deleted, we are again listing down mailboxes to check whether the mailbox was deleted or not. The last part of our code rename mailbox **ELITMUS_2** to **ELITMUS_3** using **rename()** method. We are then again listing down directories to check whether the mailbox was renamed or not.

Once all operations are performed, we are calling **expunge()** method to make the change permanent and close the mailbox using **close()** method.

In [9]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="INBOX", readonly=False)

    ############### Create Mailbox #############
    print("Creating Mailbox : ELITMUS_2")
    resp_code, mail_count = imap_ssl.create(mailbox="ELITMUS_2")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))
```

```
    ##################### List Directores ####################
    resp_code, directories = imap_ssl.list()
    print("\nResponse Code : {}".format(resp_code))
    print("========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())

    ############### Delete Mailbox #############
    print("\nDeleting Mailbox : Myntra")
    resp_code, mail_count = imap_ssl.delete(mailbox="Myntra")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))

    ##################### List Directores ####################
    resp_code, directories = imap_ssl.list()

    print("\nResponse Code : {}".format(resp_code))
    print("========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())

    ############### Rename Mailbox #############
    print("\nRenaming Mailbox : ELITMUS_2")
    resp_code, mail_count = imap_ssl.rename(oldmailbox="ELITMUS_2", newmailbox="ELITMUS_3")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))

    ##################### List Directores ####################
    resp_code, directories = imap_ssl.list()
    print("\nResponse Code : {}".format(resp_code))
    print("========= List of Directories =================\n")
    for directory in directories:
        print(directory.decode())

    ################### Expunge ######################
    print("\nFinalizing changes...")
    resp_code, directories = imap_ssl.expunge()
    print("\nResponse Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()

Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c64c550>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Creating Mailbox : ELITMUS_2
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories =================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "ELITMUS_2"
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Myntra"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"

Deleting Mailbox : Myntra
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories =================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "ELITMUS_2"
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"

Renaming Mailbox : ELITMUS_2
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
========= List of Directories =================

(\HasNoChildren) "/" "ELITMUS"
(\HasNoChildren) "/" "ELITMUS_3"
```

```
(\HasNoChildren) "/" "INBOX"
(\HasNoChildren) "/" "Personal"
(\HasNoChildren) "/" "Receipts"
(\HasNoChildren) "/" "Work"
(\HasChildren \Noselect) "/" "[Gmail]"
(\All \HasNoChildren) "/" "[Gmail]/All Mail"
(\Drafts \HasNoChildren) "/" "[Gmail]/Drafts"
(\HasNoChildren \Important) "/" "[Gmail]/Important"
(\HasNoChildren \Sent) "/" "[Gmail]/Sent Mail"
(\HasNoChildren \Junk) "/" "[Gmail]/Spam"
(\Flagged \HasNoChildren) "/" "[Gmail]/Starred"
(\HasNoChildren \Trash) "/" "[Gmail]/Trash"

Finalizing changes...

Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Closing selected mailbox....
```

Example 9: Copy Mails to Different Directory¶

As a part of the ninth example, we'll explain how we can copy mails from one directory to another using **copy()** method of **IMAP4_SSL** instance.

Important Methods of IMAP4/IMAP4_SSL Instance¶

- **copy(message_set,new_mailbox) -** It accepts **message_set** specifying ids of mails from selected mailbox and copies those mails to **new_mailbox** mailbox. The **message_set** parameter works exactly same as it works with **fetch()** method.

Our code for this example starts by creating an instance of **IMPA4_SSL** to establish a connection to the server. It then logs in to the server and selects the mailbox as **ELITMUS**. It then searches the mailbox using **search()** method to retrieve all mail ids. It then takes the last two mail ids and copies them using **copy()** method to **Personal** mailbox. It passes both mail ids together to be copied. It then copies the first two emails from the **ELITMUS** mailbox to **Personal** mailbox. This time it loops through ids and copies individual mail by calling **copy()** method more than once. Once all operations are done, it closes mailbox using **close()** method.

In [10]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="ELITMUS", readonly=False)

    ############### Retrieve Mail IDs for given Directory #############
    resp_code, mails = imap_ssl.search(None, "ALL")
    print("Mail IDs : {}\n".format(mails[0].decode().split()))

    ################## Copy Messages to new mailbox ##################
    print("\nCopying few mails of ELITMUS to different directory....")
    mail_ids =  mails[0].decode().split()[-2:]
    mail_ids = ":".join(mail_ids)
    print(mail_ids)

    resp_code, response = imap_ssl.copy(mail_ids, "Personal")
    print("Response Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))

    ################## Copy Messages to new mailbox ##################
    print("\nCopying few mails of ELITMUS to different directory....")
    mail_ids =  mails[0].decode().split()[:2]

    for mail_id in mail_ids:
        resp_code, response = imap_ssl.copy(mail_id, "Personal")
        print("Response Code : {}".format(resp_code))
        print("Response      : {}".format(response[0].decode()))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c1e4ee0>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Mail IDs : ['1', '2', '3', '4', '5', '6', '7', '8', '9', '10', '11', '12', '13', '14', '15', '16', '17', '18', '19', '20', '21', '22', '23', '24', '25


Copying few mails of ELITMUS to different directory....
31:32
Response Code : OK
Response      : [COPYUID 626162797 1591,1598 4,8] (Success)

Copying few mails of ELITMUS to different directory....
Response Code : OK
Response      : [COPYUID 626162797 1480 6] (Success)
Response Code : OK
Response      : [COPYUID 626162797 1525 7] (Success)
```

```
Closing selected mailbox....
```

Example 10: Search Mailbox using Different Patterns¶

As a part of our tenth example, we are demonstrating how we can search mailbox using different patterns passed to **criterion** parameter of the **search()** method.

Our code for this example like our previous example starts by establishing a connection with the server and logging in to the server. It then selects a mailbox named **INBOX**. It then gives a pattern that searches for mails from a particular sender. It retrieves all mail ids matching that pattern using **search()** method. It then prints the contents of the last two emails from that sender. The second search call searches for a mail from a particular sender and subject line containing a particular string. It retrieves all mail ids matching that pattern and then uses that mail ids to print contents of the latest two emails matching that pattern.

In [11]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="INBOX", readonly=True)

    ############### Search mails in a given Directory #############
    resp_code, mails = imap_ssl.search(None, '(FROM "Neil Patel")')
    mail_ids = mails[0].decode().split()
    print("Total Mail IDs : {}\n".format(len(mail_ids)))

    for mail_id in mail_ids[-2:]:
        print("================= Start of Mail [{}] ====================".format(mail_id))
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("From       : {}".format(message.get("From")))
        print("To         : {}".format(message.get("To")))
        print("Bcc        : {}".format(message.get("Bcc")))
        print("Date       : {}".format(message.get("Date")))
        print("Subject    : {}".format(message.get("Subject")))
        print("Body : ")
        for part in message.walk():
            if part.get_content_type() == "text/plain":
                body_lines = part.as_string().split("\n")
                print("\n".join(body_lines[:6])) ### Print first few lines of message
        print("================= End of Mail [{}] ====================\n".format(mail_id))


    ############### Search mails in a given Directory #############
    resp_code, mails = imap_ssl.search(None, 'FROM "Medium Daily Digest" Subject "Jupyter"')
    mail_ids = mails[0].decode().split()
    print("Total Mail IDs : {}\n".format(len(mail_ids)))
    for mail_id in mail_ids[-2:]:
        print("================= Start of Mail [{}] ====================".format(mail_id))
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("From       : {}".format(message.get("From")))
        print("To         : {}".format(message.get("To")))
        print("Bcc        : {}".format(message.get("Bcc")))
        print("Date       : {}".format(message.get("Date")))
        print("Subject    : {}".format(message.get("Subject")))
        print("Body : ")
        for part in message.walk():
            if part.get_content_type() == "text/plain":
                body_lines = part.as_string().split("\n")
                print("\n".join(body_lines[:6])) ### Print first few lines of message
        print("================= End of Mail [{}] ====================\n".format(mail_id))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c61f460>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Total Mail IDs : 40

================= Start of Mail [3010] ====================
From       : "Neil Patel" <np@neilpatel.com>
To         : "Sunny" <mail2sunny.2309@gmail.com>
Bcc        : None
Date       : Tue, 23 Feb 2021 14:13:33 +0000 (UTC)
Subject    : Today's the day I teach you marketing
Body :
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset=UTF-8
Mime-Version: 1.0

I hope you are excited because my marketing summit starts today,
where my team and I teach you marketing.
================= End of Mail [3010] ====================

================= Start of Mail [3033] ====================
From       : "Neil Patel" <np@neilpatel.com>
```

```
To         : "Sunny" <mail2sunny.2309@gmail.com>
Bcc        : None
Date       : Sat, 27 Feb 2021 16:24:04 +0000 (UTC)
Subject    : This is what's changing in marketing
Body :
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset=UTF-8
Mime-Version: 1.0

Advertising is getting more expensive, regulations are more
strict, and marketing is changing.
================== End of Mail [3033] ====================

Total Mail IDs : 13

================== Start of Mail [2829] ====================
From       : Medium Daily Digest <noreply@medium.com>
To         : mail2sunny.2309@gmail.com
Bcc        : None
Date       : Thu, 28 Jan 2021 02:30:00 +0000 (UTC)
Subject    : Reproducible Experiments with Jupyter Notebooks and Guild AI |
 Garrett Smith in Towards Data Science
Body :
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset=utf-8
Mime-Version: 1.0

From your following

================== End of Mail [2829] ====================

================== Start of Mail [3031] ====================
From       : "Medium Daily Digest" <noreply@medium.com>
To         : mail2sunny.2309@gmail.com
Bcc        : None
Date       : Sat, 27 Feb 2021 01:30:00 +0000 (UTC)
Subject    : The new age of Jupyter Widgets | Dimitris Poulopoulos in Towards Data Science
Body :
Content-Transfer-Encoding: quoted-printable
Content-Type: text/plain; charset=UTF-8
Mime-Version: 1.0

From your following

================== End of Mail [3031] ====================


Closing selected mailbox....
```

Example 11: Delete Mails from Mailbox¶

As a part of our eleventh example, we'll demonstrate how we can delete mails from the mailbox using **store()** method by changing flags.

Important Methods of IMAP4/IMAP4_SSL Instance¶

- **store(message_set,command,flag_list)** - This method changes flat details for give set of mail ids.
  - **message_set** - This parameter works exactly like it works for **fetch()** and **copy()** methods.
  - **command** - This method accepts one of the three arguments (**FLAGS, +FLAGS, -FLAGS**). The **FLAGS** indicates to replace existing flags. The **+FLAGS** indicates to add new flags to existing. The **-FLAGS** indicates to remove flags from existing flags.
  - **flag_list** - This parameter accepts flags separated by space that will be replaced, added or removed from mail data based on the value of **command** paramcter. It accepts flags like (**\Seen, \Answered, \Flagged, \Deleted, \Draft, \Recent**)

Below mentioned RFC has very detailed explanations of flags.

- RFC-2060

Our code for this example starts as usual by establishing a connection and logging into the server. It then select mailbox **INBOX** using **select()** method and retrieves emails ids from particular sender using **search()** method. We have set **readonly** parameter of **select()** method to **False** so that we can modify the mailbox otherwise we'll get an error when performing operations that try to modify the mailbox. It then loops through mail ids and sets the flag as **\Deleted** for the first two mail ids using **store()** method. It prints details with mail id, data, and subject informing which mails will be deleted. The **store()** method will set a flag indicating that these mails need to be deleted. Once flags are modified for two mail ids, we are calling **expunge()** method which will permanently delete both mails.

Please make a **NOTE** that we have used two backslashes inside the string of **\\Deleted** flag because a single backslash is used as an escape character and we need to escape it.

In [12]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="INBOX", readonly=False)

    ############### Search and delete mails in a given Directory #############
```

```
    resp_code, mails = imap_ssl.search(None, '(FROM "Neil Patel")')
    mail_ids = mails[0].decode().split()
    print("Total Mail IDs : {}\n".format(len(mail_ids)))

    print("Deleting Mails...")
    for mail_id in mail_ids[:2]:
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("Mail ID : {}, Date : {}, Subject : {}".format(mail_id, message.get("Date"), message.get("Subject")))
        resp_code, response = imap_ssl.store(mail_id, '+FLAGS', '\\Deleted')
        print("Response Code : {}".format(resp_code))
        print("Response      : {}\n".format(response[0].decode()))

    resp_code, response = imap_ssl.expunge()
    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c1e4d00>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Total Mail IDs : 40

Deleting Mails...
Mail ID : 1978, Date : Tue, 07 Jul 2020 13:30:46 +0000 (UTC), Subject : SEO Pack: 21 Worksheets, Templates, and Cheat Sheets
Response Code : OK
Response      : 1978 (FLAGS (\Seen \Deleted))

Mail ID : 2012, Date : Sat, 11 Jul 2020 14:29:35 +0000 (UTC), Subject : How to Market on Amazon
Response Code : OK
Response      : 2012 (FLAGS (\Seen \Deleted))

Response Code : OK
Response      : 1978


Closing selected mailbox....
```

Example 12: Flag Mails as Important (Starred)¶

As a part of our twelfth example, we are demonstrating how we can mark the mail as important (starred) using **store()** method.

Our code for this example starts by establishing a connection with the server and then logs in to the server. It then selects **INBOX** as a mailbox to work with. It then retrieves mail ids of all mails in the mailbox. It then loops through the 5 latest mail ids and sets their flag (**+FLAGS**) as **\Flagged** using **store()** method. This will make mails as important mails. In our Gmail and Yahoo mail mailboxes, these mails will be starred. We are then again looping through mail ids and removing **\Flagged** emails from the latest two mails to mark them as unimportant again (**-FLAGS**). We are also printing details of mails (mail id, date, and subject) which are getting marked as important and unimportant.

In [13]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ############### Set Mailbox #############
    resp_code, mail_count = imap_ssl.select(mailbox="INBOX", readonly=False)

    ############### Mark few latest mails as important in a given Directory #############
    resp_code, mails = imap_ssl.search(None, 'ALL')
    mail_ids = mails[0].decode().split()
    print("Total Mail IDs : {}\n".format(len(mail_ids)))

    print("Flagging Important Mails...")
    for mail_id in mail_ids[-5:]:
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("Mail ID : {}, Date : {}, Subject : {}".format(mail_id, message.get("Date"), message.get("Subject")))
        resp_code, response = imap_ssl.store(mail_id, '+FLAGS', '\\Flagged')
        print("Response Code : {}".format(resp_code))
        print("Response      : {}\n".format(response[0].decode() if response[0] else response[0]))


    ############### Mark few latest mails as unimportant in a given Directory #############
    print("Unflagging few Mails...")
    for mail_id in mail_ids[-2:]:
        resp_code, mail_data = imap_ssl.fetch(mail_id, '(RFC822)') ## Fetch mail data.
        message = email.message_from_bytes(mail_data[0][1]) ## Construct Message from mail data
        print("Mail ID : {}, Date : {}, Subject : {}".format(mail_id, message.get("Date"), message.get("Subject")))
        resp_code, response = imap_ssl.store(mail_id, '-FLAGS', '\\Flagged')
        print("Response Code : {}".format(resp_code))
        print("Response      : {}\n".format(response[0].decode() if response[0] else response[0]))

    ############# Close Selected Mailbox ######################
    print("\nClosing selected mailbox....")
    imap_ssl.close()
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f848c64ce80>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Total Mail IDs : 3070

Flagging Important Mails...
Mail ID : 3066, Date : Mon, 01 Mar 2021 06:50:58 -0800 (PST), Subject : Test Email
Response Code : OK
Response      : 3066 (FLAGS (\Seen \Flagged))

Mail ID : 3067, Date : Mon, 1 Mar 2021 15:47:01 +0000, Subject : =?UTF-8?Q?Sunny_Solanki,_here_are_toda?=
 =?UTF-8?Q?y=E2=80=99s_highlights:_markets_ended?=
 =?UTF-8?Q?_green,_Feb_auto_sales,_gold_bonds_tranche_12_open,_&_more?=
Response Code : OK
Response      : 3067 (FLAGS (\Seen \Flagged))

Mail ID : 3068, Date : Mon, 01 Mar 2021 12:03:51 -0800, Subject : Google Cloud Platform & APIs: Your invoice is available for 012B9E-8BAD8E-73B348
Response Code : OK
Response      : 3068 (FLAGS (\Seen \Flagged))

Mail ID : 3069, Date : Mon, 01 Mar 2021 16:45:47 -0600, Subject : You wrote a ton! Way to go!
Response Code : OK
Response      : 3069 (FLAGS (\Seen \Flagged))

Mail ID : 3070, Date : Tue, 02 Mar 2021 02:30:00 +0000 (UTC), Subject : Improve visualization readability with meaningful text and annotations | Jan M
Response Code : OK
Response      : 3070 (FLAGS (\Seen \Flagged))

Unflagging few Mails...
Mail ID : 3069, Date : Mon, 01 Mar 2021 16:45:47 -0600, Subject : You wrote a ton! Way to go!
Response Code : OK
Response      : 3069 (FLAGS (\Seen))

Mail ID : 3070, Date : Tue, 02 Mar 2021 02:30:00 +0000 (UTC), Subject : Improve visualization readability with meaningful text and annotations | Jan M
Response Code : OK
Response      : 3070 (FLAGS (\Seen))


Closing selected mailbox....
```

Example 13: No Operation to Keep Connection Alive¶

As a part of our thirteenth example, we'll explain how we can send **NOOP** command to the server to indicate no command at all. It can be used to keep the connection alive.

Important Methods of IMAP4/IMAP4_SSL Instance¶

- **noop() -** It sends **NOOP** command to the mail server.

In [1]:

```
import imaplib
import email

################ IMAP SSL ##############################

with imaplib.IMAP4_SSL(host="imap.gmail.com", port=imaplib.IMAP4_SSL_PORT) as imap_ssl:
    print("Connection Object : {}".format(imap_ssl))

    ############### Login to Mailbox ######################
    print("Logging into mailbox...")
    resp_code, response = imap_ssl.login("mail2sunny.2309@gmail.com", "app_password")

    print("Response Code : {}".format(resp_code))
    print("Response      : {}\n".format(response[0].decode()))

    ################## Keeping Connection Alive ###############
    resp_code, response = imap_ssl.noop() ## Keep connection Alive

    print("Response Code : {}".format(resp_code))
    print("Response      : {}".format(response[0].decode()))
```

```
Connection Object : <imaplib.IMAP4_SSL object at 0x7f34e6262550>
Logging into mailbox...
Response Code : OK
Response      : mail2sunny.2309@gmail.com authenticated (Success)

Response Code : OK
Response      : Success
```

This ends our small tutorial explaining how we can use **IMAP4** protocol through **imaplib** Python library to perform various operations with a mailbox. Please feel free to let us know your views in the comments section.

References¶

- Gmail Server and Port Details
- Yahoo mail Server and Port Details
- RFC-3501
- RFC-2060
- IMAP Commands
- smtplib - Simple Guide to Sending Mails using Python
- email - How to Represent an Email Message in Python