Python Network Programming

- Python provides the access of two levels to network services.
- At a low level, you can access the basic socket support in the underlying operating system, which allows you to implement clients and servers for both connection-oriented and connectionless protocols.
- Python also has libraries that provide higher-level access to specific application-level network protocols, such as FTP, HTTP, and so on.

What is Sockets?

- Sockets are the endpoints of a bidirectional communications channel.
- Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.
- Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on.
- The ***socket library*** provides specific classes for handling the common transports as well as a generic interface for handling the rest.

Term	Description
domain	The family of protocols that will be used as the transport mechanism. These values are constants such as AF_INET, PF_INET, PF_UNIX, PF_X25, and so on.
type	The type of communications between the two endpoints, typically SOCK_STREAM for connection-oriented protocols and SOCK_DGRAM for connectionless protocols.
protocol	Typically zero, this may be used to identify a variant of a protocol within a domain and type.
hostname	The identifier of a network interface: ▫ A string, which can be a host name, a dotted-quad address, or an IPV6 address in colon (and possibly dot) notation ▫ A string "<broadcast>", which specifies an INADDR_BROADCAST address. ▫ A zero-length string, which specifies INADDR_ANY, or ▫ An Integer, interpreted as a binary address in host byte order.
port	Each server listens for clients calling on one or more ports. A port may be a Fixnum port number, a string containing a port number, or the name of a service.

The *socket* Module

- To create a socket, you must use the *socket.socket()* function available in *socket* module, which has the general syntax:

  s = socket.socket (socket_family, socket_type, protocol=0)

- Here is the description of the parameters:
  - **socket_family:** This is either AF_UNIX or AF_INET, as explained earlier.
  - **socket_type:** This is either SOCK_STREAM or SOCK_DGRAM.
  - **protocol:** This is usually left out, defaulting to 0.
- Once you have *socket* object, then you can use required functions to create your client or server program.

Server /Client Socket Methods

Server

Method	Description
s.bind()	This method binds address (hostname, port number pair) to socket.
s.listen()	This method sets up and start TCP listener.
s.accept()	This passively accept TCP client connection, waiting until connection arrives (blocking).

Client

Method	Description
s.connect()	This method actively initiates TCP server connection.

General Socket Methods

Method	Description
s.recv()	This method receives TCP message
s.send()	This method transmits TCP message
s.recvfrom()	This method receives UDP message
s.sendto()	This method transmits UDP message
s.close()	This method closes socket
socket.gethostname()	Returns the hostname.

A Simple Server

- We use the **socket** function available in socket module to create a socket object.
  - A socket object is used to call other functions to setup a socket server.
- Now call **bind(hostname, port)** function to specify a ***port*** for your service on the given host.
- Next, call the *accept* method of the returned object.
  - This method waits until a client connects to the port you specified, and then returns *a connection object* that represents the connection to that client.

Server

```
# server.py
import socket
import time

# create a socket object
serversocket = socket.socket(
                socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999

# bind to the port
serversocket.bind((host, port))

# queue up to 5 requests
serversocket.listen(5)

while True:
    # establish a connection
    clientsocket,addr = serversocket.accept()

    print("Got a connection from %s" % str(addr))
    currentTime = time.ctime(time.time()) + "\r\n"
    clientsocket.send(currentTime.encode('ascii'))
    clientsocket.close()
```

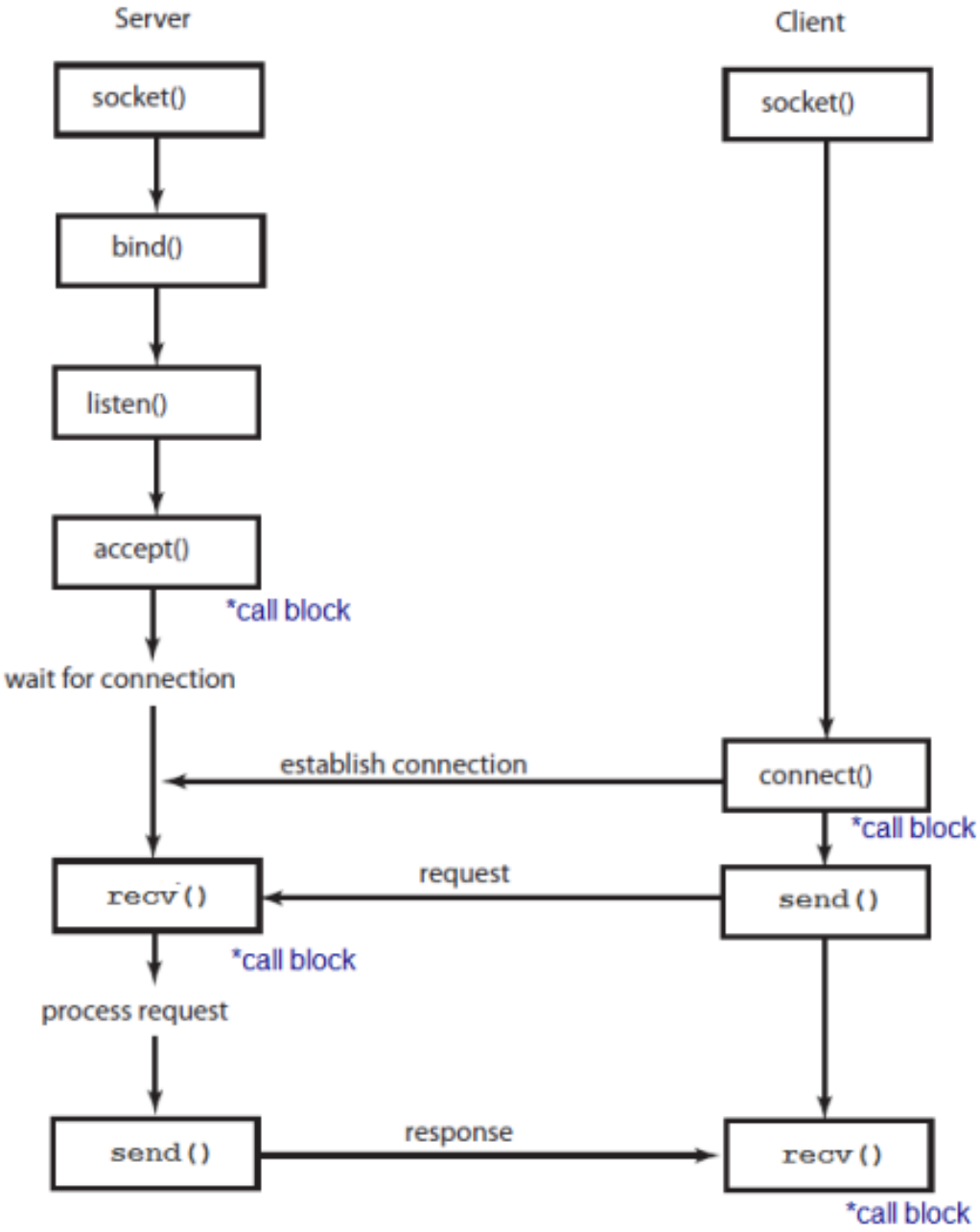
A Simple Client

- We write a very simple client program which will open a connection to a given *port 9999* and *given host*.
- This is very simple to create a socket client using Python's *socket* module function.
- The **socket.connect(hosname, port)** opens a TCP connection to *hostname* on the *port*.
  - Once you have a socket open, you can read from it like any I/O object.
  - When done, remember to close it, as you would close a file.
- The following code is a very simple client that connects to a given host and port, reads any available data from the socket, and then exits:

```python
# client.py
import socket

# create a socket object
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# get local machine name
host = socket.gethostname()

port = 9999

# connection to hostname on the port.
s.connect((host, port))

# Receive no more than 1024 bytes
tm = s.recv(1024)

s.close()

print("The time got from the server is %s" % tm.decode('ascii'))
```

```
C:\Python27>python.exe server.py &
Got a connection from ('192.168.0.11', 59519)
```

```
...
========== RESTART: C:\Python27\client.py ==========
The time got from the server is Mon Apr 03 19:59:40 2017
```


Server
Client
socket()
socket()
bind()
listen()
accept()
*call block
wait for connection
establish connection
connect()
*call block
request
recv()
send()
*call block
process request
send()
response
recv()
*call block

Echo Server

- This is an echo server: the server that echoes back all data it receives to a client that sent it.

```
# echo_server.py
import socket

host = ''         # Symbolic name meaning all available interfaces
port = 12345      # Arbitrary non-privileged port
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.bind((host, port))
s.listen(1)
conn, addr = s.accept()
print('Connected by', addr)
while True:
    data = conn.recv(1024)
    if not data: break
    conn.sendall(data)
conn.close()
```

Client

```
# echo_client.py
import socket

host = socket.gethostname()
port = 12345                    # The same port as used by the server
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((host, port))
s.sendall(b'Hello, world')
data = s.recv(1024)
s.close()
print('Received', repr(data))
```

```
C:\Python27>python.exe echo_server.py &
('Connected by', ('192.168.0.11', 59633))
```

```
========== RESTART: C:/Python27/echo_client.py ==========
('Received', "'Hello, world'")
```

Python Internet modules

Protocol	Common function	Port No	Python module
HTTP	Web pages	80	httplib, urllib, xmlrpclib
NNTP	Usenet news	119	nntplib
FTP	File transfers	20	ftplib, urllib
SMTP	Sending email	25	smtplib
POP3	Fetching email	110	poplib
IMAP4	Fetching email	143	imaplib
Telnet	Command lines	23	telnetlib
Gopher	Document transfers	70	gopherlib, urllib

FILE TRANSFER

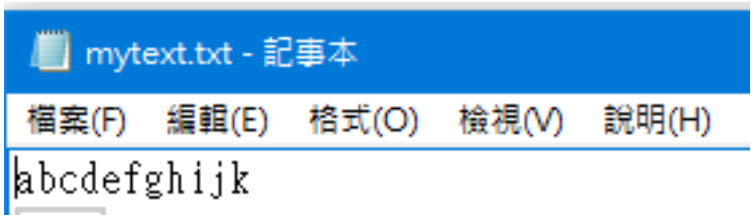
- Here is the code to send a file from a local server to a local client.

```
# server.py

import socket                   # Import socket module

port = 60000                    # Reserve a port for your service.
s = socket.socket()             # Create a socket object
host = socket.gethostname()     # Get local machine name
s.bind((host, port))            # Bind to the port
s.listen(5)                     # Now wait for client connection.

print 'Server listening....'

while True:
    conn, addr = s.accept()     # Establish connection with client.
    print 'Got connection from', addr
    data = conn.recv(1024)
    print('Server received', repr(data))

    filename='mytext.txt'
    f = open(filename,'rb')
    l = f.read(1024)
    while (l):
       conn.send(l)
       print('Sent ',repr(l))
       l = f.read(1024)
    f.close()
```

```
# client.py

import socket                       # Import socket module

s = socket.socket()                 # Create a socket object
host = socket.gethostname()         # Get local machine name
port = 60000                        # Reserve a port for your service.

s.connect((host, port))
s.send("Hello server!")

with open('received_file', 'wb') as f:
    print 'file opened'
    while True:
        print('receiving data...')
        data = s.recv(1024)
        print('data=%s', (data))
        if not data:
            break
        # write data to a file
        f.write(data)

f.close()
print('Successfully get the file')
s.close()
print('connection closed')
```

Server

```
C:\Python27>python.exe file_server.py
Server listening....
Got connection from ('192.168.0.11', 60135)
('Server received', "'Hello server!'")
('Sent ', "'abcdefghijk'")
Done sending
```



Client

```
==================== RESTART: C:/Python27/file_client.py ====================
file opened
receiving data...
('data=%s', 'abcdefghijkThank you for Connecting')
receiving data...
('data=%s', '')
Successfully get the file
Connection closed
```

CHAT SERVER & CLIENT

- The server is like a middle man among clients.
  - It can queue up to 10 clients.
- The server broadcasts any messages from a client to the other participants. So, the server provides a sort of chatting room.
- The server is handling the sockets in non-blocking mode using **select.select()** method:

```
ready_to_read, ready_to_write, in_error = \
               select.select(
                  potential_readers,
                  potential_writers,
                  potential_errs,
                  timeout)
```

- We pass **select()** three lists:
  - the first contains all sockets that we might want to try reading
  - the second all the sockets we might want to try writing to
  - the last (normally left empty) those that we want to check for errors

CHAT SERVER & CLIENT

- Though the **select()** itself is a blocking call (it's waiting for I/O completion), we can give it a timeout.
  - we set **time_out = 0**, and it will poll and never block.
- Actually, the **select()** function monitors all the client sockets and the server socket for readable activity.
- If any of the client socket is readable then it means that one of the chat client has send a message.
- When the select function returns, the **ready_to_read** will be filled with an array consisting of all socket descriptors that are readable.

- In the code, we're dealing with two cases:
  - If the master socket is readable, the server would accept the new connection.
  - If any of the client socket is readable, the server would read the message, and broadcast it back to all clients except the one who send the message.

```
import sys
import socket
import select

HOST = ''
SOCKET_LIST = []
RECV_BUFFER = 4096
PORT = 9009

def chat_server():

    server_socket = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((HOST, PORT))
    server_socket.listen(10)

    # add server socket object to the list of readable connections
    SOCKET_LIST.append(server_socket)

    print "Chat server started on port " + str(PORT)
```

```
while 1:

    # get the list sockets which are ready to be read through select
    # 4th arg, time_out  = 0 : poll and never block
    ready_to_read,ready_to_write,in_error = select.select(SOCKET_LIST,[],[],0)

    for sock in ready_to_read:
        # a new connection request recieved
        if sock == server_socket:
            sockfd, addr = server_socket.accept()
            SOCKET_LIST.append(sockfd)
            print "Client (%s, %s) connected" % addr

            broadcast(server_socket, sockfd, "[%s:%s] entered our chatting room\n" % addr)

        # a message from a client, not a new connection
        else:
            # process data recieved from client,
            try:
                # receiving data from the socket.
                data = sock.recv(RECV_BUFFER)
                if data:
                    # there is something in the socket
                    broadcast(server_socket, sock, "\r" + '[' + str(sock.getpeername()) + '] ' + data)
                else:
                    # remove the socket that's broken
                    if sock in SOCKET_LIST:
                        SOCKET_LIST.remove(sock)

                    # at this stage, no data means probably the connection has been broken
                    broadcast(server_socket, sock, "Client (%s, %s) is offline\n" % addr)

            # exception
            except:
                broadcast(server_socket, sock, "Client (%s, %s) is offline\n" % addr)
                continue

server_socket.close()
```

```
# broadcast chat messages to all connected clients
def broadcast (server_socket, sock, message):
    for socket in SOCKET_LIST:
        # send the message only to peer
        if socket != server_socket and socket != sock :
            try :
                socket.send(message)
            except :
                # broken socket connection
                socket.close()
                # broken socket, remove it
                if socket in SOCKET_LIST:
                    SOCKET_LIST.remove(socket)

if __name__ == "__main__":

    sys.exit(chat_server())
```

On recv() & disconnection

- When a **recv()** returns 0 bytes, it means the other side has closed (or is in the process of closing) the connection. You will not receive any more data on this connection. Ever, you may be able to send data successfully.
- A protocol like HTTP uses a socket for only one transfer. The client sends a request, then reads a reply.
- The socket is discarded. This means that a client can detect the end of the reply by receiving 0 bytes.
- But if you plan to reuse your socket for further transfers, you need to realize that there is no EOT on a socket. I repeat: if a socket send or **recv()** returns after handling 0 bytes, the connection has been broken.
- If the connection has not been broken, you may wait on a **recv()** forever, because the socket will not tell you that there's nothing more to read (for now)."

Client Code

```
import sys, socket, select

def chat_client():
    if(len(sys.argv) < 3) :
        print 'Usage : python chat_client.py hostname port'
        sys.exit()

    host = sys.argv[1]
    port = int(sys.argv[2])

    s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
    s.settimeout(2)

    # connect to remote host
    try :
        s.connect((host, port))
    except :
        print 'Unable to connect'
        sys.exit()

    print 'Connected to remote host. You can start sending messages'
    sys.stdout.write('[Me] '); sys.stdout.flush()
```

```
        while 1:
            socket_list = [sys.stdin, s]

            # Get the list sockets which are readable
            read_sockets, write_sockets, error_sockets = select.select(socket_list , [], [])

            for sock in read_sockets:
                if sock == s:
                    # incoming message from remote server, s
                    data = sock.recv(4096)
                    if not data :
                        print '\nDisconnected from chat server'
                        sys.exit()
                    else :
                        #print data
                        sys.stdout.write(data)
                        sys.stdout.write('[Me] '); sys.stdout.flush()

                else :
                    # user entered a message
                    msg = sys.stdin.readline()
                    s.send(msg)
                    sys.stdout.write('[Me] '); sys.stdout.flush()

if __name__ == "__main__":

    sys.exit(chat_client())
```

We should run the server first:

```
$ python chat_server.py
Chat server started on port 9009
```

The client code:

```
$ python chat_client.py localhost 9009
Connected to remote host. You can start sending messages
```

```
// server terminal
$ python chat_server.py
Chat server started on port 9009
Client (127.0.0.1, 48952) connected
Client (127.0.0.1, 48953) connected
Client (127.0.0.1, 48954) connected


// client 1 terminal
$ python chat_client.py localhost 9009
Connected to remote host. You can start sending messages
[Me] [127.0.0.1:48953] entered our chatting room
[Me] [127.0.0.1:48954] entered our chatting room
[Me] client 1
[('127.0.0.1', 48953)] client 2
[('127.0.0.1', 48954)] client 3
[Me] Client (127.0.0.1, 48954) is offline
[Me]
```

```
// client 2 terminal
$ python chat_client.py localhost 9009
Connected to remote host. You can start sending messages
[Me] [127.0.0.1:48953] entered our chatting room
[Me] [127.0.0.1:48954] entered our chatting room
[Me] client 1
[('127.0.0.1', 48953)] client 2
[('127.0.0.1', 48954)] client 3
[Me] Client (127.0.0.1, 48954) is offline
[Me]

// client 3 terminal
$ python chat_client.py localhost 9009
Connected to remote host. You can start sending messages
[('127.0.0.1', 48952)] client 1
[('127.0.0.1', 48953)] client 2
[Me] client 3
[Me] ^CTraceback (most recent call last):
  File "chat_client.py", line 52, in
    sys.exit(chat_client())
  File "chat_client.py", line 30, in chat_client
    read_sockets, write_sockets, error_sockets = select.select(socket_list
KeyboardInterrupt
```

Note that the client #3 did go off the line at the end by typing ^C

Python Sending Email using SMTP

- Simple Mail Transfer Protocol (SMTP) is a protocol, which handles sending e-mail and routing e-mail between mail servers.
- Python provides **smtplib** module, which defines an SMTP client session object that can be used to send mail to any Internet machine with an SMTP or ESMTP listener daemon.

```
import smtplib

smtpObj = smtplib.SMTP( [host [, port [, local_hostname]]] )
```

Parameters

- **host:** This is the host running your SMTP server.
  - You can specifiy IP address of the host or a domain name like nchu.edu.tw.
  - This is optional argument.
- **port:** If you are providing *host* argument, then you need to specify a port, where SMTP server is listening.
  - This port would be 25.
- **local_hostname**: If your SMTP server is running on your local machine, then you can specify just *localhost* as of this option.

Python Sending Email using SMTP

- An SMTP object has an instance method called **sendmail**, which is typically used to do the work of mailing a message.
- It takes three parameters −
  - The *sender* - A string with the address of the sender.
  - The *receivers* - A list of strings, one for each recipient.
  - The *message* - A message as a string formatted as specified in the various RFCs.

Example

```
import smtplib

to = 'hwtseng@nchu.edu.tw'
gmail_user = 'hwtseng@cs.nchu.edu.tw'
gmail_pwd = 'xxxxxxxx'
smtpserver = smtplib.SMTP("smtp.gmail.com",587)
smtpserver.ehlo()
smtpserver.starttls()
smtpserver.ehlo
smtpserver.login(gmail_user, gmail_pwd)
header = 'To:' + to + '\n' + 'From: ' + gmail_user + '\n' + 'Subject:testing \n'
print header
msg = header + '\n this is test msg from hsteng \n\n'
smtpserver.sendmail(gmail_user, to, msg)
print 'done!'
smtpserver.close()
```

```
====================== RESTART: C:/Python27/smtp3.py ======================
To:hwtseng@nchu.edu.tw
From: hwtseng@cs.nchu.edu.tw
Subject:testing

done!
```

Google

查看遭拒的登入嘗試

您會收到這封郵件是因為 hwtseng@nchu.edu.tw 是 hwtseng@cs.nchu.edu.tw 的備援
如果 hwtseng@cs.nchu.edu.tw 不是您的 Google 帳戶，請按這裡取消連結該帳戶，並
的電子郵件。

學文您好：

Google 剛剛已禁止某人透過可能會危害你帳戶的應用程式登入你的 Google 帳戶
hwtseng@cs.nchu.edu.tw。

低安全性應用程式

2017 年 4 月 4 日 星期二 下午 2:00 (台灣時間)

台灣東區*

對這個活動沒有印象嗎？

如果您最近透過非 Google 應用程式存取 Google 服務 (例如 Gmail) 時，並未收到錯誤
已有其他人取得您的密碼。

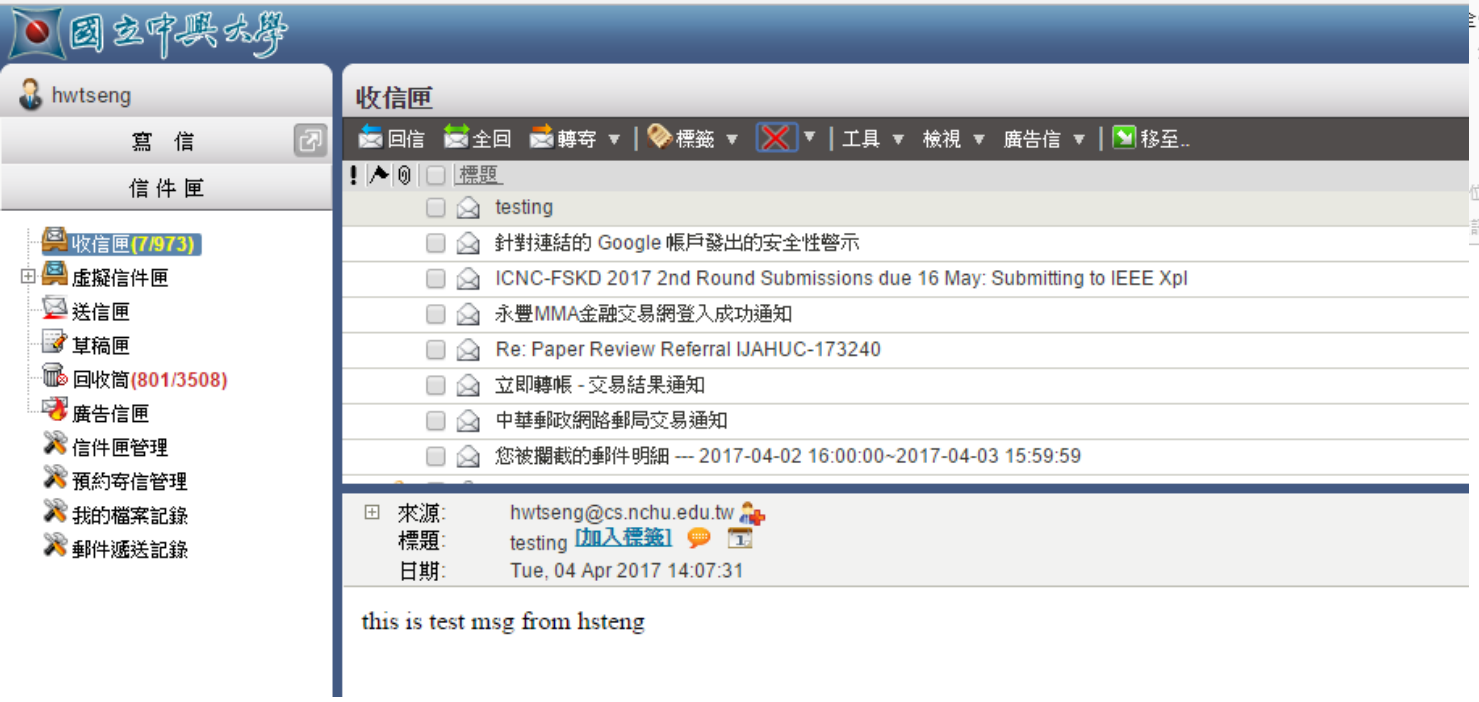
確保您的帳戶安全

嘗試登入者是您本人嗎？

性問題或版本過舊，因此 Google 將繼續禁止該應用
您可以啟用低安全性應用程式的存取權限，但這樣會

位置。

訊，請造訪 Google 帳戶說明中心。



SMTP Objects

- SMTP.helo([*hostname*]): Identify yourself to the SMTP server using HELO.
- SMTP.ehlo([*hostname*]): Identify yourself to an ESMTP server using EHLO.
- SMTP.starttls([*keyfile*[, *certfile*]]): Put the SMTP connection in TLS (Transport Layer Security) mode.
  - All SMTP commands that follow will be encrypted. You should then call ehlo() again.

Sending an HTML e-mail using Python

- When you send a text message using Python, then all the content are treated as simple text.
- Even if you include HTML tags in a text message, it is displayed as simple text and HTML tags will not be formatted according to HTML syntax.
- But Python provides option to send an HTML message as actual HTML message.
- While sending an e-mail message, you can specify a Mime version, content type and character set to send an HTML e-mail.

Example

```
import smtplib

to = 'hwtseng@nchu.edu.tw'
gmail_user = 'hwtseng@cs.nchu.edu.tw'
gmail_pwd = 'xxxxxxxx'
smtpserver = smtplib.SMTP("smtp.gmail.com",587)
smtpserver.ehlo()
smtpserver.starttls()
smtpserver.ehlo
smtpserver.login(gmail_user, gmail_pwd)
header = 'To:' + to + '\n' + 'From: ' + gmail_user + '\n' + 'Subject:testing \n'
print header
msg = header + """From: hwtseng@cs.nchu.edu.tw
To: hwtseng@cs.nchu.edu.tw
MIME-Version: 1.0
Content-type: text/html
Subject: SMTP HTML e-mail test

This is an e-mail message to be sent in HTML format

<b>This is HTML message.</b>
<h1>This is headline.</h1>
"""
smtpserver.sendmail(gmail_user, to, msg)
print 'done!'
smtpserver.close()
```

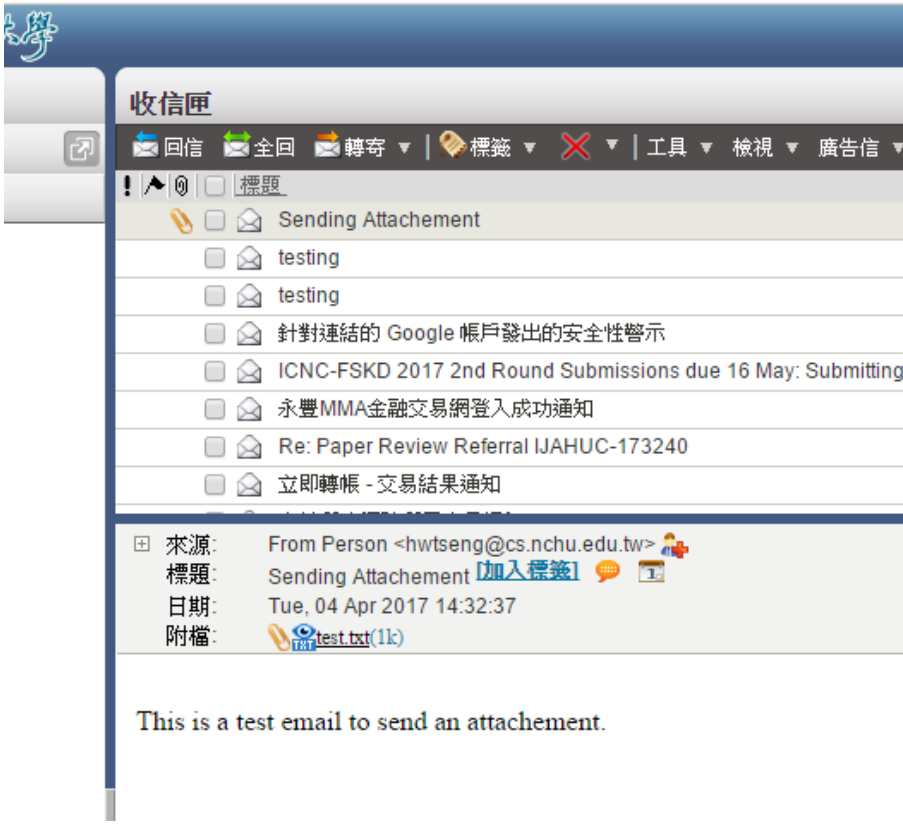
```
========== RESTART: C:/Python27/smtp3.py ==========
To:hwtseng@nchu.edu.tw
From: hwtseng@cs.nchu.edu.tw
Subject:testing

done!
```

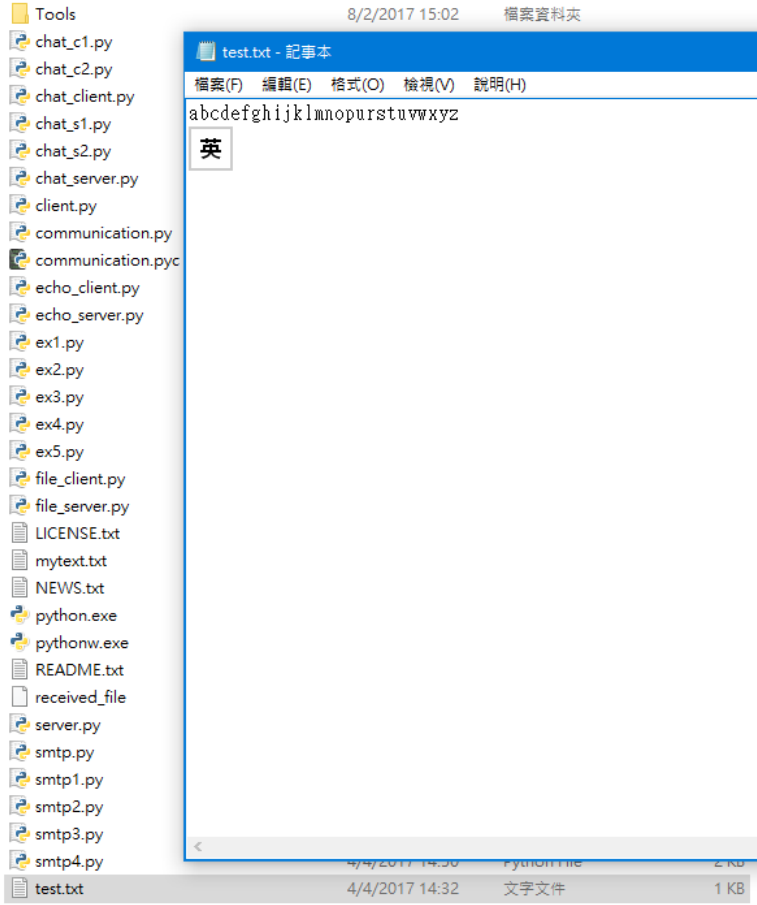
Sending Attachments as an E-mail

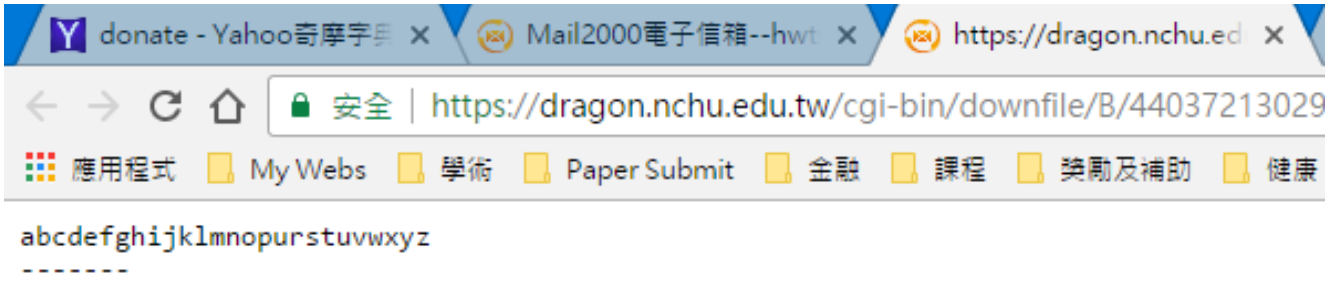
- To send an e-mail with mixed content requires to set **Content-type** header to **multipart/mixed**.
- Then, text and attachment sections can be specified within **boundaries**.
- A boundary is started with two hyphens (--)followed by a unique number, which cannot appear in the message part of the e-mail.
- A final boundary denoting the e-mail's final section must also end with two hyphens.
- Attached files should be encoded with the **pack("m")** function to have base64 encoding before transmission.

Example

```
import smtplib
import base64

filename = "test.txt"
# Read a file and encode it into base64 format
fo = open(filename, "rb")
filecontent = fo.read()
encodedcontent = base64.b64encode(filecontent)  # base64  encodedcontent = unicode(filecontent, 'ascii')

marker = "AUNIQUEMARKER"

body ="""
This is a test email to send an attachement.
"""
# Define the main headers.
part1 = """From: hwtseng@cs.nchu.edu.tw
To: hwtseng@nchu.edu.tw
Subject: Sending Attachement
MIME-Version: 1.0
Content-Type: multipart/mixed; boundary=%s
--%s
""" % (marker, marker)

英 ine the message action
 = """Content-Type: text/plain
Content-Transfer-Encoding:8bit

%s
--%s
""" % (body,marker)

# Define the attachment section
part3 = """Content-Type: multipart/mixed; name=\"%s\"
Content-Transfer-Encoding:base64
Content-Disposition: attachment; filename=%s

%s
--%s--
""" %(filename, filename, encodedcontent, marker)
message = part1 + part2 + part3



to = 'hwtseng@nchu.edu.tw'
gmail_user = 'hwtseng@cs.nchu.edu.tw'
gmail_pwd = '
smtpserver = smtplib.SMTP("smtp.gmail.com",587)
smtpserver.ehlo()
smtpserver.starttls()
smtpserver.ehlo
smtpserver.login(gmail_user, gmail_pwd)
header = 'To:' + to + '\n' + 'From: ' + gmail_user + '\n' + 'Subject:testing \n'
print header

smtpserver.sendmail(gmail_user, to, message)
print 'done!'
smtpserver.close()
```








```
abcdefghijklmnopurstuvwxyz
-------
```