TABLE OF CONTENTS

SOCKET

1.1 DEFINITION OF SOCKET:

The socket itself is just one of the endpoints in a communication between programs on some network.

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket (node) listens on a particular port at an IP, while other socket reaches out to the other to form a connection. Server forms the listener socket while client reaches out to the server. Sockets may be implemented over a number of different channel types: UNIX domain sockets, TCP, UDP, and so on.

Socket programming is started by importing socket library and making a simple socket.

Example:

```
import socket
s=socket. socket(socket.AF_INET,socket.SOCK_STREAM)
```

1.2 SOCKET MODULES:

- There are different parameters available in python networking to create socket. They are socket family, socket type, and protocol.
- The general syntax to create a socket is given below.

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

1.3 SOCKET FAMILY:

- This is either AF_UNIX or AF_INET. AF_INET is used for IPv4 Internet addressing. IPv4 addresses are made up of four octal values separated by dots (e.g.,10.1.1.5 and 127.0.0.1). These values are more commonly referred to as "IP address." Almost all Internet networking is done using IP version 4 at this time.
- AF_INET6 is used for IPv6 Internet addressing. IPv6 is the "next generation" version of the Internet protocol, and supports 128-bit addresses, traffic shaping, and routing features not available under IPv4. Adoption of IPv6 is still limited, but continues to grow.
- AF_UNIX is the address family for Unix Domain Sockets (UDS), an inter-process communication protocol available on POSIX-compliant systems. The implementation of UDS typically allows the operating system to pass data directly from process to process, without going through the network stack. This is more efficient than using restricted to processes on the same system.

1.4 TCP:

- Sockets can be configured to act as a server and listen for incoming messages, or connect to other applications as a client.
- After both ends of a TCP/IP socket are connected, communication is bi-directional.

1.5 UDP:

- UDP works differently from TCP/IP.
- Where TCP is a stream oriented protocol, ensuring that all of the data is transmitted in the right order, UDP is a message oriented protocol.
- UDP does not require a long-lived connection, so setting up a UDP socket is a little simpler.
- On the other hand, UDP messages must fit within a single packet (for IPv4 that means they can only hold 65,507 bytes because the 65,535 byte packet also includes header information) and delivery is not guaranteed as it is with TCP.

1.6 DIFFERENCE BETWEEN TCP AND UDP:

S.No.	TCP	UDP
1.	TCP/IP stands for Transmission Control Protocol/Internet Protocol.	The full form of UDP is User Datagram Protocol.

2.	import **socket**
	import **sys**
	
	# Create a TCP/IP socket
	sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
		import **socket**
		import **sys**
		
		# Create a TCP/IP socket
		sock =
		socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

1.7 SOCKET TYPES:

Depending on the transport layer protocol used, sockets can either be:

- SOCK_DGRAM for message-oriented datagram transport. These sockets are usually associated with the User Datagram Protocol (UDP) which provides an unreliable delivery of individual messages. Datagram sockets are commonly used when the order of messages is not important, such as when sending out the same data to multiple clients.

- SOCK_STREAM for stream-oriented transport often associated with the Transmission Control Protocol (TCP). TCP provides a reliable and ordered delivery of byte between two hosts, with error handling and control, making it useful for implementing applications that involve transfer of large amounts of data.

1.8 SOCKET FUNCTIONS:

Some of the functions and methods that provide access to network related tasks. They are listed below.

- gethostname()- It returns the a string containing the hostname of the machine where the python interpreter is currently executing. For example, localhost. gethostname is used when the client and server are on the same computer (LAN-localip/WAN-publicip).
- gethostbyname()- It returns the IP address of the host. gethostbyname returns just one IP of the host even though the host could resolve to multiple IPs from a given location.
- gethostbyaddr()- This IP address function socket.gethostbyaddr() returns a tuple containing hostname, alias list of IP address if any and IP address of the host.
- getserverbyname()-This function takes a network service name or the protocol name and returns the port number on which the service is defined by /etc/services. The underlying protocol means is the transport layer protocol like TCP/UDP.

1.9 ECHO SERVER:

To set up a server it must perform the sequence of methods socket(), bind(), listen(), and accept().

- **socket()-** It creates a new socket given the address family and a socket type.
- **bind()-** It binds our socket object to a particular address composed of a host and a port number.
- **listen()-** It allows the server to start accepting connections and takes in an argument, backlog, which is the number of unaccepted connections that system can allow before refusing all new connections.
- **accept()-** It accepts the incoming connections and returns a tuple of (conn, address) where conn is a new socket that can be used to send and receive messages from the connection and address is the address bound to the socket on the other end of the connection.

1.10 SOCKET METHODS:

Socket methods are of two types. They are server socket methods and client socket methods.

- **SERVER SOCKET METHODS:**
  - **s.bind()-** This method binds the address to the socket.
  - **s.listen()**- This method sets up and start TCP listener.
  - **s.accept()-** This method passively accept TCP client connection, waiting until connection arrives (blocking).

- **CLIENT SOCKET METHODS:**
  - **s.connect()-** This method actively initiates TCP server connection
    .

1.11 COMPARISON OF TCP AND UDP SOCKET METHODS:

S.NO.	**TCP**	**UDP**
1.	s.recv()- This method recives TCP message.	s.recvfrom()- This method receives UDP message.
2.	s.send()- This method send TCP message.	s.sendfrom()- This method sends UDP messade.
3.	s.close()- This method closes the socket.	s.close()- This method closes the socket.

1.12 A SIMPLE SERVER PROGRAM:

We use socket function available in the socket module to create a socket object, to write internet servers. A server has a bind() method which binds it to a specific IP and port so that it can listen to incoming requests. A server has listen() method which puts the server into listen mode. This allows the server to listen the incoming connection. And last a server has an accept() and close() method.

```
import socket

s=socket.socket()

host=socket.gethostname()

port=12345

s.bind((host,port))

s.listen(5)

while True:

    c,addr=s.accept()

    print(“got connection from “,addr)
```

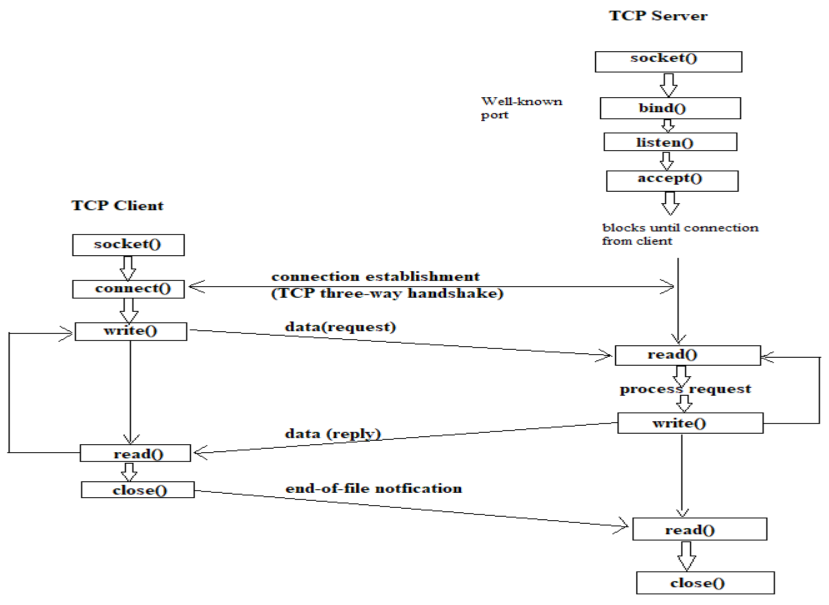
```
c.send(“Thank you for connecting”)

c.close()
```

1.13 A SIMPLE CLIENT PROGRAM:

This program opens a connection to a given port 12345 and given host. It is very simple to create a socket client using python’s socket module function. The socket.connect(hostname,port) opens a TCP connection to hostname on the port.

```
import socket

s=socket.socket()

host=socket.gethostname()

port=12345

s.connect((host,port))

print(s.recv(1024))

 s.close()
```

1.14 FLOW CHART OF CLIENT SERVER MODEL:



2.SMTP

2.1 DEFINITION OF 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.

Here is a simple syntax to create one SMTP object.

Example:

```
Import smtplib

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

2.2 SMTP PARAMETES:

- **host** − This is the host running your SMTP server. You can specify IP address of the host or a domain name. This is optional argument.
- **port** − If you are providing host argument, then you need to specify a port, where SMTP server is listening. Usually 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.

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:

```
smtpObj.sendmail(sender, receiver, message)
```

2.3 SENDING EMAIL FROM GMAIL:

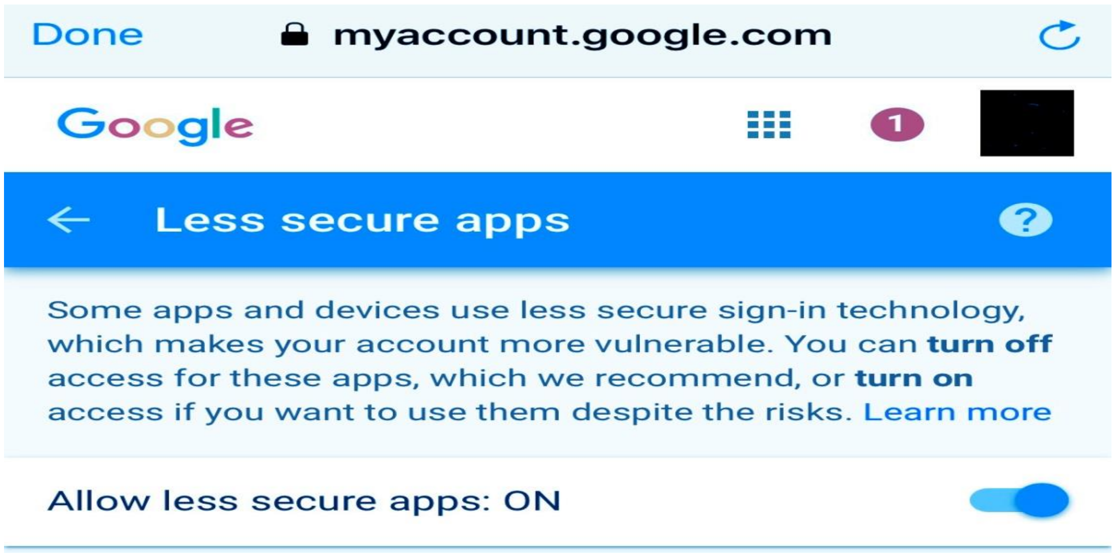
There are cases where the emails are sent using the Gmail SMTP server. In this case, we can pass Gmail as the SMTP server instead of using the local host with the port 587.

Use the following syntax.

smtpObj = smtplib.SMTP("gmail.com", 587)

Here, we need to login to the Gmail account using Gmail user name and password. For this purpose, the smtplib provide the login() method, which accepts the username and password of the sender.

This may make your Gmail ask you for access to less secure apps if you're using Gmail. You will need to turn this ON temporarily for this to work.



Some apps and devices use less secure sign-in technology, which makes your account more vulnerable. You can **turn off** access for these apps, which we recommend, or **turn on** access if you want to use them despite the risks. Learn more

Allow less secure apps: ON

We can also create an app password for email instead of making it less secure apps.

2.4 SMTP OBJECTS:

SMTP.**login**(user, password, *, initial_response_ok=True)

Log in on an SMTP server that requires authentication. The arguments are the username and the password to authenticate with. If there has been no previous EHLO or HELO command this session, this method tries ESMTP EHLO first. This method will return normally if the authentication was successful, or may raise the following exceptions.

SMTP.**connect**(host='localhost', port=0)

Connect to a host on a given port. The defaults are to connect to the local host at the standard SMTP port (25). If the hostname ends with a colon (':') followed by a number, that suffix will be stripped off and the number interpreted as the port number to use. This method is automatically invoked by the constructor if a host is specified during instantiation. Returns a 2-tuple of the response code and message sent by the server in its connection response.

SMTP.**ehlo**(name='')

Identify yourself to an ESMTP server using EHLO. The hostname argument defaults to the fully qualified domain name of the local host. Examine the response for ESMTP option and store them for use by has_extn(). Also sets several informational attributes: the message returned by the server is stored as the ehlo_resp attribute, does_esmtp is set to true or false depending on whether the server supports ESMTP, and esmtp_features will be a dictionary containing the names of the SMTP service extensions this server supports, and their parameters (if any).

Note: There are lot more objects available in SMTP the above are some examples.

2.5 EXCEPTIONS OF SMTP:

SMTPHeloError
The server didn't reply properly to the HELO greeting.

SMTPAuthenticationError
The server didn't accept the username/password combination.

SMTPNotSupportedError
The AUTH command is not supported by the server.

SMTPException
No suitable authentication method was found.

SMTPRecipientsRefused
All recipients were refused. Nobody got the mail. The recipients attribute of the exception object is a dictionary with information about the refused recipients (like the one returned when at least one recipient was accepted).

SMTPHeloError
The server didn't reply properly to the HELO greeting.

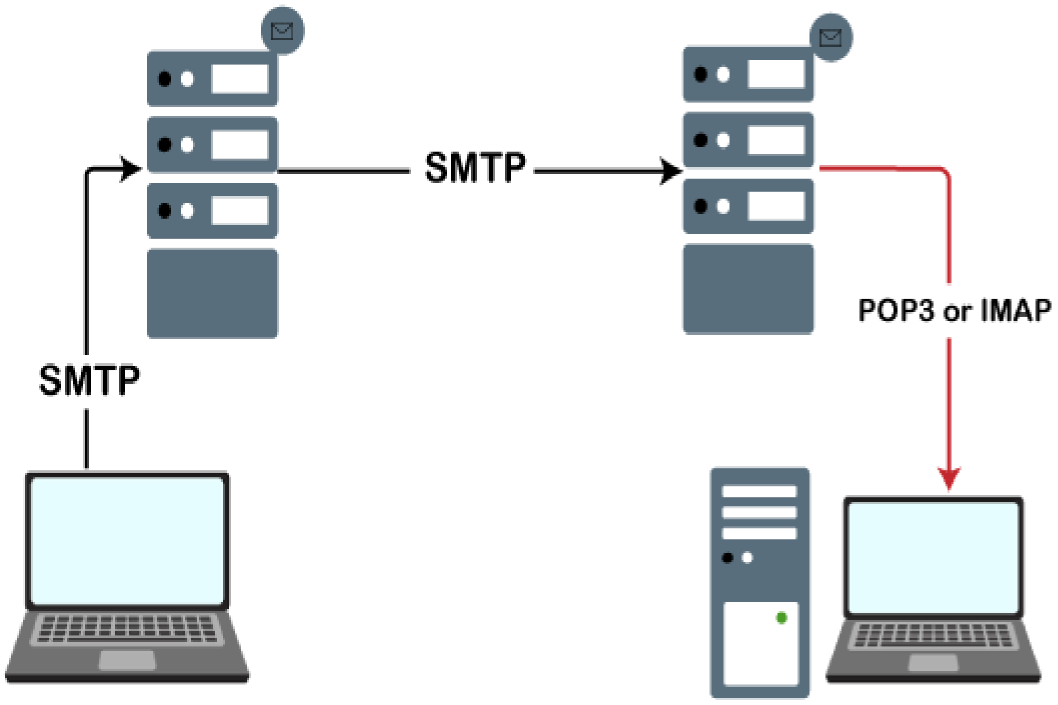
SMTPSenderRefused
The server didn't accept the from_addr.

SMTPDataError
The server replied with an unexpected error code (other than a refusal of a recipient).

SMTPNotSupportedError

SMTPUTF8 was given in the mail_options but is not supported by the server.

2.6 FLOW CHART OF SMTP :



3.FTP

3.1 DEFINITION OF FTP:

FTP or File Transfer Protocol is a well-known network protocol used to transfer files between computers in a network. It is created on client server architecture and can be used along with user authentication. It can also be used without authentication but that will be less secure. FTP connection which maintains a current working directory and other flags, and each transfer requires a secondary connection through which the data is transferred. Most common web browsers can retrieve files hosted on FTP servers.

3.2 METHODS IN FTP CLASS:

Method	Description
pwd()	Current working directory.
cwd()	Change current working directory to path.
dir([path[,...[,cb]])	Displays directory listing of path. Optional call-back cb passed to retrlines().
storlines(cmd, f)	Uploads text file using given FTP cmd - for example, STOR file name.
delete(path)	Deletes remote file located at path.
mkd(directory)	Creates remote directory.
connect(host[, port[, timeout]])	Connects to the given host and port. The default port number is 21, as specified by the FTP protocol
login(user='anonymous', passwd='', acct='')	Log in as the given user. The passwd and acct parameters are optional and default to the empty string. If no user is specified, it defaults to 'anonymous'. If user is 'anonymous', the default passwd is 'anonymous@'.
quit()	Closes connection and quits.

3.3 EXCEPTIONS OF FTP:

exception ftplib.**error_reply**

Exception raised when an unexpected reply is received from the server.

exception ftplib.**error_temp**

Exception raised when an error code signifying a temporary error (response codes in the range 400–499) is received.

exception ftplib.**error_perm**

Exception raised when an error code signifying a permanent error (response codes in the range 500–599) is received.

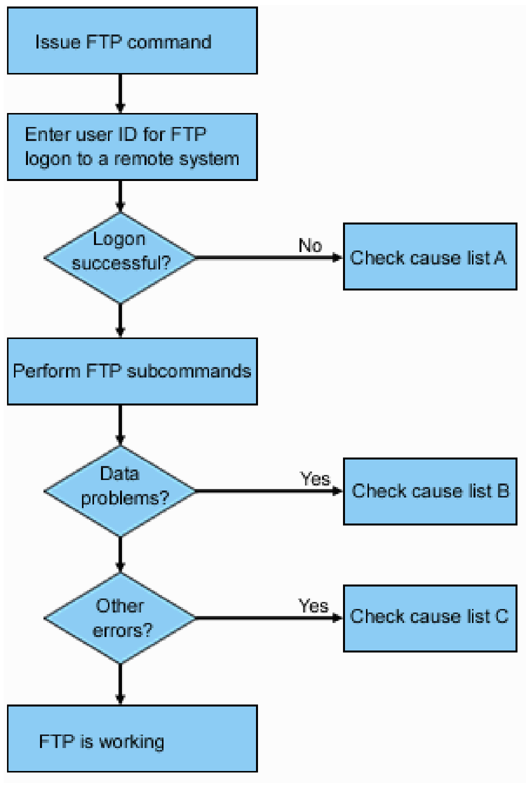
exception ftplib.**error_proto**

Exception raised when a reply is received from the server that does not fit the response specifications of the File Transfer Protocol, i.e. begin with a digit in the range 1–5.

ftplib.**all_errors**

The set of all exceptions (as a tuple) that methods of FTP instances may raise as a result of problems with the FTP connection (as opposed to programming errors made by the caller). This set includes the four exceptions listed above as well as OSError and EOFError.

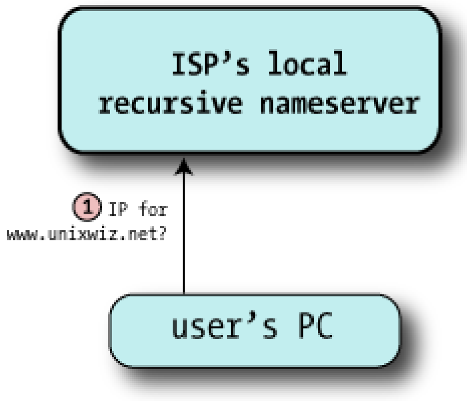
3.4 FLOW CHART OF FTP :



4.DOMAIN NAME SYSTEM

4.1. DEFINITION:

The Domain Name System, or DNS, is one of the Internet's fundamental building blocks. It is the global, hierarchical, and distributed host information database that's responsible for translating names into addresses and vice versa, routing mail to its proper destination, and many other services.



4.2. DNS TERMINOLOGY:

4.2.1. RESOLVER :

Resolver is the client part of the DNS client/server system: it asks the questions about hostnames. The resolver is usually a small library compiled into each program that requires DNS services, and it knows just enough to send questions to a nearby nameserver.

4.2.2.RECURSIVE NAMESERVER:

This is a nameserver that's willing to go out on the internet and find the results for zones it's not authoritative for, as a service to its clients. Not all nameservers are configured to provide recursive service, or are limited to just trusted clients .

4.2.3. NAMESERVER:

This is server software that answers DNS questions.

4.2.4. AUTHORATIVE NAMESERVER:

For every zone, somebody has to maintain a file of the hostnames and IP address associations. This is generally an administrative function performed by a human, and in most cases one machine has this file. It's the zone master.

4.2.5. RESOURCE RECORD:

Though most think of DNS as providing hostname-to-IP mapping, there are actually other kinds of questions we can ask of a nameserver, and this highlights the notion that DNS is really a database of “resource records”.

The most common type is an IP Address (an “A” record), but other records exist too: NS (nameserver), MX (mail exchanger), SOA (Start of Authority), and so on.

4.2.6. DNS PYTHON:

DNS python is a DNS toolkit for Python. It supports almost all record types. It can be used for queries, zone transfers, and dynamic updates.dnspython provides both high and low level access to DNS. The high level classes perform queries for data of a given name, type, and class, and return an answer set. The low level classes allow direct manipulation of DNS zones, messages, names, and records.

5. HTTP (REQUEST LIBRARY)

5.1 DEFINITION:

HTTP stands for the 'HyperText Transfer Protocol,' where communication is possible by request done by the client and the response made by the server.

For example, you can use the client(browser) to search for a 'dog' image on Google. Then that sends an HTTP request to the server, i.e., a place where a dog image is hosted, and the response from the server is the status code with the requested content. This is a process also known as a request-response cycle.

5.1.1 LIBRARIRES IN PYTHON TO MAKE HTTP REQUEST:

There are many libraries to make an HTTP request in Python, which are httplib, urllib, httplib2 , treq, etc., but requests are the simplest and most well-documented libraries among them all.

5.1.2 USING REQUEST IN PYTHON:

i.Using GET requestr=requests.get('https://xkcd.com/1906/')

ii.Using POST request

r=requests.post('https://httpbin.org/post',data = pload)

1. r.status_code
2. r.text
3. r.headers / r.headers[“Content-Type”]
4. r.url
5. r.json()

5.2 HTTP (http.server)

- This module defines classes for implementing HTTP servers (Web servers).
- One class, HTTPServer, is a socketserver.TCPServer subclass.
- It creates and listens at the HTTP socket, dispatching the requests to a handler.
- Code to create and run the server looks like this.

5.2.1 class http.server.HTTPServer(server_address, RequestHandlerClass)

- This class builds on the TCPServer class by storing the server address as instance variables named server_name and server_port.
- The server is accessible by the handler, typically through the handler’s server instance variable.

5.2.2 class http.server.ThreadingHTTPServer(server_address, RequestHandlerClass)

This class is identical to HTTPServer but uses threads to handle requests by using the ThreadingMixIn. This is useful to handle web browsers pre-opening sockets, on which HTTPServer would wait indefinitely.

The HTTPServer and ThreadingHTTPServer must be given a RequestHandlerClass on instantiation, of which this module provides three different variants:

5.2.3 classhttp.server.BaseHTTPRequestHandler(request, client_address, server)

This class is used to handle the HTTP requests that arrive at the server. By itself, it cannot respond to any actual HTTP requests; it must be subclassed to handle each request method (e.g. GET or POST). BaseHTTPRequestHandler provides a number of class and instance variables, and methods for use by subclasses.

5.3 INSTANCE VARIABLE:

Base HTTP RequestHandler has the following instance variables,

1. client_address
2. server
3. close_connection
4. headers
5. rfile
6. wfile
7. requestline
8. command
9. path
10. request_version

5.4 METHODS:

BaseHttpRequestHandler has following methods in it.

1. handle()
2. handle_one_request()
3. handle_except_100()
4. send_error(code, message=None, explain=None)
5. send_response(code, message=None)
6. send_header(keyword,value)
7. send_response_only(code,message=None)
8. end_headers() …..etc

5.4.1 Class http.server.SimpleHTTPRequestHandler(request, client_address, server, directory=None)

This class serves files from the current directory and below, directly mapping the directory structure to HTTP requests.

5.4.2 Class-level attributes of SimpleHTTPRequestHandler :

1. server_version

2. extension_map
3. directory

5.4.3 The SimpleHTTPRequestHandler class defines the following methods:

1. do_HEAD()
2. do_GET()

4. Misc Networking Libraries (netaddr)

netaddr:A Python library for representing and manipulating network addresses.

5.5 Layer 3 addresses:

- IPv4 and IPv6 addresses, subnets, masks, prefixes.Iterating, slicing, sorting, summarizing and classifying IP networks.
- Dealing with various ranges formats (CIDR, arbitrary ranges and globs, nmap).
- Set based operations (unions, intersections etc) over IP addresses and subnets.
- Parsing a large variety of different formats and notations. Looking up IANA IP block information.
- Generating DNS reverse lookups.Supernetting and subnetting.

5.6 Layer 2 addresses:

- Representation and manipulation MAC addresses and EUI-64 identifiers.
- Looking up IEEE organisational information (OUI, IAB). Generating derived IPv6 addresses.

6.IMAP

6.1 DEFINITON OF IMAP:

- IMAP means Internet Message Access Protocol. It is a way that a laptop or a desktop can connect to larger internet server to view and manipulate the user’s mail.
- It is a email retrieval protocol that does not download the mail instead it just reads them and display the message.

- IMAP allows the client program to manipulate the email message over the server without downloading it in the local machine.
- This library is installed using the command pip install imap.

6.2 IMAP COMMANDS:

There are some commands in the IMAP library to retrieve the content from the mail they are listed below.

- IMAP_LOGIN: T his command opens the connection.
- CAPABILITY: It is used to send request for listing the capabilities that the server supports.
- NOOP: This command is used as a periodic poll for new messages or message status that updates during a period of inactivity.
- SELECT: This command helps to select a mailbox to access the messages.
- EXAMINE: It is same as SELECT command except no change to the mailbox is permitted.
- CREATE: It is same as SELECT command except no change to the mailbox is permitted.
- DELETE: It is used to permanently delete a mailbox with a given name.
- RENAME: It is used to change the name of the mailbox.
- LOGOUT: This command informs the server that client is done with the session. The server must send BYE untagged response before the OK response and then close the network connection.

6.3 A SIMPLE IMAP PROGRAM:

```
import imaplib

import pprint

imap_host = 'imap.gmail.com'

imap_user = 'username@gmail.com'

imap_pass = 'password'

# connect to host using SSL

imap = imaplib.IMAP4_SSL(imap_host)

## login to server

imap.login(imap_user, imap_pass)

imap.select('Inbox')

tmp, data = imap.search(None, 'ALL')
```

```
for num in data[0].split():
        tmp, data = imap.fetch(num, '(RFC822)')
        print('Message: {0}\n'.format(num))
        pprint.pprint(data[0][1])
        break
imap.close()
```