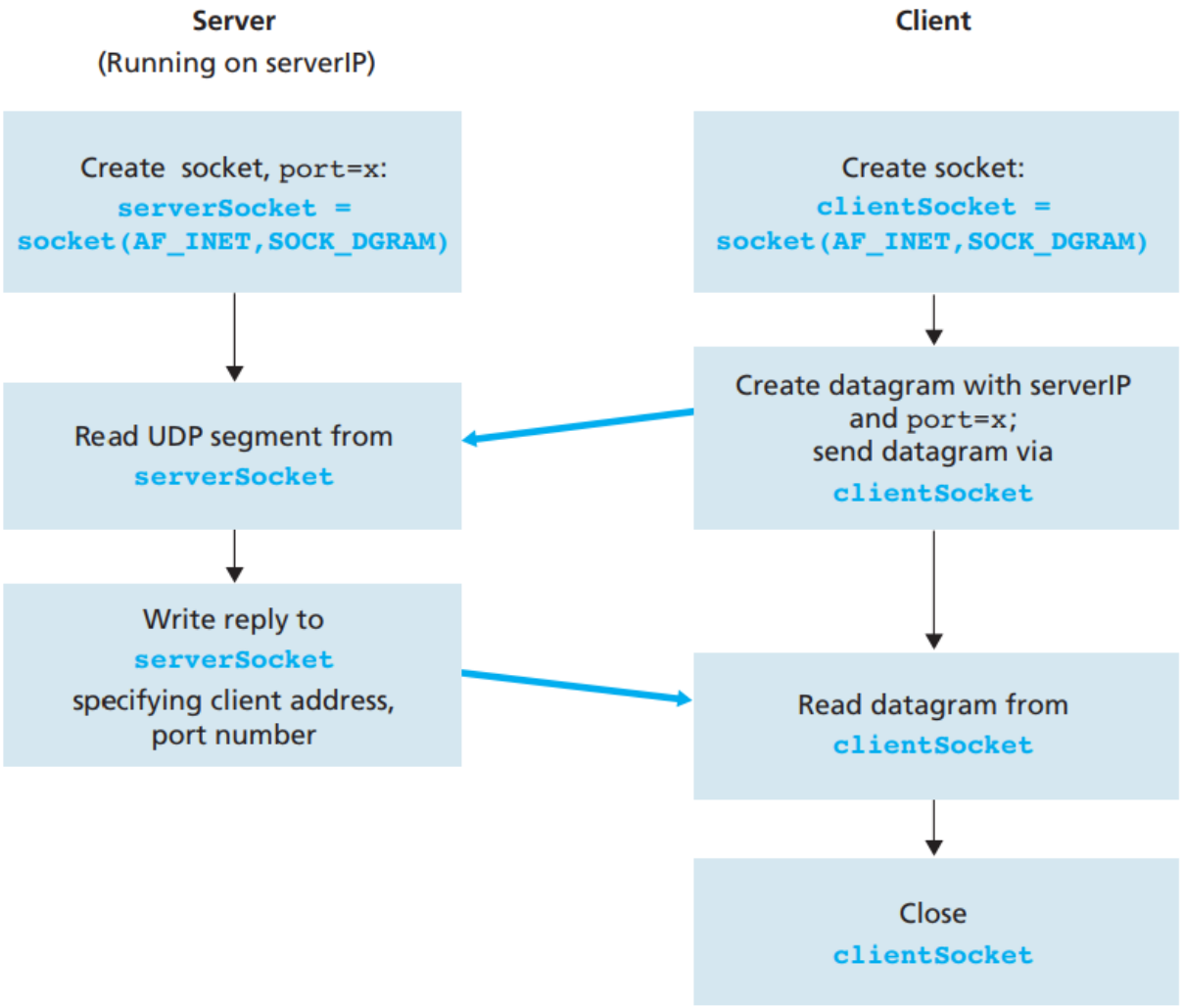
1. PreReqs:
    a. A network application requires TWO programs:
        i. Client
        ii. Server
    b. When the Client and Server programs are executed, their respective processes are created.
        i. These processes communicate with each other by:
            1. Reading from sockets
            2. Writing to sockets
    c. Two types of network apps:
        i. Open source
        ii. Proprietary
    d. Open source-
        i. The implementation is specified in a protocol standard (RFC)
        ii. Because it is known to all, the developers carefully conform to the rules
        iii. Different developers can create the processes and the processes can communicate with each other, independent of the platform
            1. Ex- Firefox browser talking to Apache webserver.
    e. Proprietary-
        i. Not open source, so therefore one developer/team must create both the client and server program.
    f. Your task is to create both a client and server program
    g. You will work with socket programming using two protocols:
        i. UDP
        ii. TCP
2. Your task:
    a. Client side should allow user to input a line of characters from the keyboard and sends that data to the server
    b. The server gets the data and converts the characters to uppercase.
    c. The server sends the modified data to the client.
    d. The client receives the modified data and displays the line on its screen.
3. Socket Programming using UDP:
    a. Processes communicate by sending messages into sockets.
        i. Each process= house
        ii. Process socket= door
        iii. Application is on one side of the door, transport layer is on the other side.
    b. Application developer- can control the application side, but not the transport side!
4. UDP:
    a. Before sending out a packet from the socket, the process must first attach a destination address to the packet.
        i. The other layers will need this to send the packet to the correct receiving process.
    b. Address=

i. IP address
ii. Socket number
    1. Because a host may be running many network application processes, with one or more sockets.

c. Creating sockets-
    i. At point of creation, an identifier (port number) is assigned to it.

d. Summary:
    i. Sending process attaches the packet a destination address
        1. Made up of the destination host's ip address
        2. The destination socket's port number
    ii. Sender's own source address
        1. Ip address and port number of the source socket are attached to packet
        2. Keep in mind—attached source address is not done by UDP app code, typically
            a. Done by the underlying operating system.



5. The Source Code for the UDPClient.py
    a. Note** the app arbitrarily uses 12000 for the server port number

```
from socket import *
serverName = 'hostname'
serverPort = 12000
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
message = raw_input('Input lowercase sentence:')
clientSocket.sendto(message,(serverName, serverPort))
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
print modifiedMessage
clientSocket.close()
```

    b. From socket import *
        i. We are importing a socket module that has important code that is foundational to network communication in Python.
        ii. This line allows us to create the socket
    c. serverName='hostname' /n serverPort=12000
        i. the first line sets the string "hostname" to the variable serverName
        ii. you will switch your serverName to be "localhost"
        iii. then we set the serverPort variable to be 12000.
    d. clientSocket=socket(socket.AF_INET, socket.SOCK_DGRAM)
        i. we create the client's socket, called clientSocket by calling the function socket(…)
        ii. we pass the first parameter as socket.AF_INET
            1. This is the address family, which indicates that the underlying network is using IPv4.
                a. More in chapter 4.
    e. Message=raw_input('Input Lower case sentence:')
        i. The raw_input() allows the user to enter data, with the prompt Input Lower….
        ii. The user enters a line, which is saved into message.
    f. clientSocket.sendto(message,(serverName,serverPort))
        i. we call the function sendto() on the client socket.
        ii. The sendto() attaches the destination address (the name and the port) to the message
        iii. Sends the resulting packet to the client socket, clientSocket.
        iv. After the client sends the packet, the client waits to get the data from the server.
    g. ModifiedMessage, serverAddress= clientSocket.recvfrom(2048)
        i. The right side calls the recvfrom() function, to get a buffer size of 2048.
        ii. The packet arrives from the Internet at the client's socket, the data is put into the variable modifiedMessage, and then the source address is put into the variable serverAddress

1. The serverAddress contains both the IP address of the server and the port number.

h. Print(modifiedMessage)
   i. This will print out the modified Message on the user's display.
   ii. It should be the same line from the user, but capitalized.

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
print "The server is ready to receive"
while 1:
        message, clientAddress = serverSocket.recvfrom(2048)
        modifiedMessage = message.upper()
        serverSocket.sendto(modifiedMessage, clientAddress)
```

UDPServer.py

1. From socket import *
   a. Same as in the UDPClient
2. serverPort=12000
   a. Setting the serverPort to 12000, just like in client side.
3. serverSocket=socket(AF_INET, SOCK_DGRAM)
   a. This is setting the socket module again to IPv4
   b. And makes the socket of type SOCK_DGRAM, a UDP socket.
4. serverSocket.bind((‘’, serverPort))
   a. This binds (or assigns) the port number 12000 to the server's socket.
   b. This is important because the server needs to know which port number to listen for connections on!
   c. If anyone sends a packet to port 12000, at the IP address of the server, then the packet will be directed to this socket.
5. While 1
   a. This is the same as saying while(true)
   b. It basically loops forever, because the server should continually listen for connections.
6. message, clientAddress=serverSocket.recvfrom(2048)
   a. when the packet arrives at the server's socket, the packet data is put into the variable message
   b. the source address listed on the packet is put into the variable clientAddress.
      i. This will contain both the IP address and the port number
7. modifiedMessage=message.upper()
   a. this is the code which capitalizes the original message.
8. serverSocket.sendto(modifiedMessage, clientAddress)

a. This attaches the client's address
    i. IP number and port number
b. To the capitalized message, and sends the resulting packet into the server's socket.
c. The server's address is also attached to the packet, but done automatically by the OS.
d. The packet is then delivered to the client side by the internet.
e. After the server sends the packet, it continues in the infinite loop, waiting for new UDP packets to arrive at the socket.

TO TEST:

** This code works best in python2, you can modify it to run with 3, but you will encounter errors that you have to work around, extra credit if you can write it in 3 **

1. Have the UDPServer on your machine, running on idle
2. Start the UDPClient process, on idle as well.
3. Send the messages back and forth between the two processes.