Coздайте два файла — один для сервера, а другой для клиента.  
  
В Python для работы с сокетами используется модуль socket: import socket  
Прежде всего нам необходимо создать сокет: sock = socket.socket()

Здесь ничего особенного нет и данная часть является общей и для клиентских и для серверных сокетов. Дальше мы будем писать код для сервера.

Теперь нам нужно определится с хостом и портом для нашего сервера. Насчет хоста — мы оставим строку пустой, чтобы наш сервер был доступен для всех интерфейсов. А порт возьмем любой от нуля до 65535. Я выбрал порт 9090. Теперь свяжем наш сокет с данными хостом и портом с помощью метода bind, которому передается кортеж, первый элемент (или нулевой, если считать от нуля) которого — хост, а второй — порт:

sock.bind(('', 9090))  
Теперь у нас все готово, чтобы принимать соединения. С помощью метода listen мы запустим для данного сокета режим прослушивания. Метод принимает один аргумент — максимальное количество подключений в очереди. Установим его в единицу:

sock.listen(1)

Ну вот, наконец-то, мы можем принять подключение с помощью метода accept, который возвращает кортеж с двумя элементами: новый сокет и адрес клиента. Именно этот сокет и будет использоваться для приема и посылке клиенту данных.  
conn, addr = sock.accept()  
Вот и все. Теперь мы установили с клиентом связь и можем с ним «общаться». Т.к. мы не можем точно знать, что и в каких объемах клиент нам пошлет, то мы будем получать данные от него небольшими порциями. Чтобы получить данные нужно воспользоваться методом recv, который в качестве аргумента принимает количество байт для чтения. Мы будем читать порциями по 1024 байт (или 1 кб):

while True:

data = conn.recv(1024)

if not data:

break

conn.send(data.upper())

Как мы и говорили для общения с клиентом мы используем сокет, который получили в результате выполнения метода accept. Мы в бесконечном цикле принимаем 1024 байт данных с помощью метода recv. Если данных больше нет, то этот метод ничего не возвращает. Таким образом мы можем получать от клиента любое количество данных.  
  
Дальше в нашем примере для наглядности мы что-то сделаем с полученными данными и отправим их обратно клиенту. Например, с помощью метода upper у строк вернем клиенту строку в верхнем регистре.  
  
Теперь можно и закрыть соединение:

conn.close()

Собственно сервер готов. Он принимает соединение, принимает от клиента данные, возвращает их в виде строки в верхнем регистре и закрывает соединение.