```
clientSocket.connect((serverName,serverPort))
```

前面讲过在客户能够使用一个 TCP 套接字向服务器发送数据之前（反之亦然），必须在客户与服务器之间创建一个 TCP 连接。上面这行就发起了客户和服务器之间的这条 TCP 连接。connect()方法的参数是这条连接中服务器端的地址。这行代码执行完后，执行三次握手，并在客户和服务器之间创建起一条 TCP 连接。

```
sentence = raw_input('Input lowercase sentence:')
```

如同 UDPClient 一样，上一行从用户获得了一个句子。字符串 sentence 连续收集字符直到用户键入回车以终止该行为止。代码的下一行也与 UDPClient 极为不同：

```
clientSocket.send(sentence)
```

上一行通过该客户的套接字并进入 TCP 连接发送字符串 sentence。值得注意的是，该程序并未显式地创建一个分组并为该分组附上目的地址，而使用 UDP 套接字却要那样做。相反，客户程序只是将字符串 sentence 中的字节放入该 TCP 连接中去。客户然后就等待接收来自服务器的字节。

```
modifiedSentence = clientSocket.recv(2048)
```

当字符到达服务器时，它们被放置在字符串 modifiedSentence 中。字符继续积累在 modifiedSentence 中，直到收到回车符才会结束该行。在打印大写句子后，我们关闭客户的套接字。

```
clientSocket.close()
```

最后一行关闭了套接字，因此关闭了客户和服务器之间的 TCP 连接。它引起客户中的 TCP 向服务器中的 TCP 发送一条 TCP 报文（参见 3.5 节）。

### 2. TCPServer.py

现在我们看一下服务器程序。

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET,SOCK_STREAM)
serverSocket.bind(('',serverPort))
serverSocket.listen(1)
print 'The server is ready to receive'
while 1:
   connectionSocket, addr = serverSocket.accept()
   sentence = connectionSocket.recv(1024)
   capitalizedSentence = sentence.upper()
   connectionSocket.send(capitalizedSentence)
   connectionSocket.close()
```

现在我们来看看上述与 UDPServer 及 TCPClient 有显著不同的代码行。与 TCPClient 相同的是，服务器创建一个 TCP 套接字，执行：

```
serverSocket=socket(AF_INET,SOCK_STREAM)
```

与 UDPServer 类似，我们将服务器的端口号 serverPort 与该套接字关联起来：

```
serverSocket.bind(('',serverPort))
```

但对 TCP 而言，serverSocket 将是我们的欢迎套接字。在创建这扇欢迎之门后，我们将等待并聆听某个客户敲门：

```
serverSocket.listen(1)
```

该行让服务器聆听来自客户的 TCP 连接请求。其中参数定义了请求连接的最大数（至少为 1）。

```
connectionSocket, addr = serverSocket.accept()
```