```
serverName = 'hostname'
serverPort = 12000
```

第一行将变量 serverName 置为字符串“hostname”。这里，我们提供了或者包含服务器的 IP 地址（如“128.138.32.126”）或者包含服务器的主机名（如“cis.poly.edu”）的字符串。如果我们使用主机名，则将自动执行 DNS lookup 从而得到 IP 地址。第二行将整数变量 serverPort 置为 12000。

```
clientSocket = socket(socket.AF_INET, socket.SOCK_DGRAM)
```

该行创建了客户的套接字，称为 clientSocket。第一个参数指示了地址簇；特别是，AF_INET 指示了底层网络使用了 IPv4。（此时不必担心，我们将在第 4 章中讨论 IPv4。）第二个参数指示了该套接字是 SOCK_DGRAM 类型的，这意味着它是一个 UDP 套接字（而不是一个 TCP 套接字）。值得注意的是，当创建套接字时，我们并没有指定客户套接字的端口号；相反，我们让操作系统为我们做这件事。既然客户进程的门已经创建，我们将要生成通过该门发送的报文。

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

raw_input()是 Python 中的内置功能。当执行这条命令时，在客户上的用户将以单词“Input lowercase sentence:”进行提示，用户使用她的键盘来输入一行，这被放入变量 message 中。既然我们有了一个套接字和一条报文，我们将要通过该套接字向目的主机发送报文。

```
clientSocket.sendto(message,(serverName, serverPort))
```

在上述这行中，方法 sendto()为报文附上目的地址（serverName，serverPort）并且向进程的套接字 clientSocket 发送结果分组。（如前面所述，源地址也附到分组上，尽管这是自动完成的，而不是显式地由代码完成的。）经一个 UDP 套接字发送一个客户到服务器的报文非常简单！在发送分组之后，客户等待接收来自服务器的数据。

```
modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
```

对于上述这行，当一个来自因特网的分组到达该客户套接字时，该分组的数据被放置到变量 modifiedMessage 中，其源地址被放置到变量 serverAddress 中。变量 serverAddress 包含了服务器的 IP 地址和服务器的端口号。程序 UDPClient 实际上并不需要服务器的地址信息，因为它从起始就已经知道了该服务器地址；而这行 Python 代码仍然提供了服务器的地址。方法 recvfrom 也取缓存长度 2048 作为输入。（该缓存长度用于多种目的。）

```
print modifiedMessage
```

这行在用户显示器上打印出 modifiedMessage。它应当是变用户键入的原始行，现在只是变为大写的了。

```
clientSocket.close()
```

该行关闭了套接字。然后关闭了该进程。

2. UDPServer.py

现在来看看这个应用程序的服务器端：

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