SLIDES DURANTE IL LABORATORIO

Fondamenti di Comunicaizoni e Internet

Antonio Capone, Matteo Cesana, Guido Maier, Francesco Musumeci

Programmazione Socket

Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci

Attività di laboratorio: Versioni software

Gli esempi mostrati a lezione usano:

- Python versione 3
- Pycharm IDE education edition
    - include Python 3

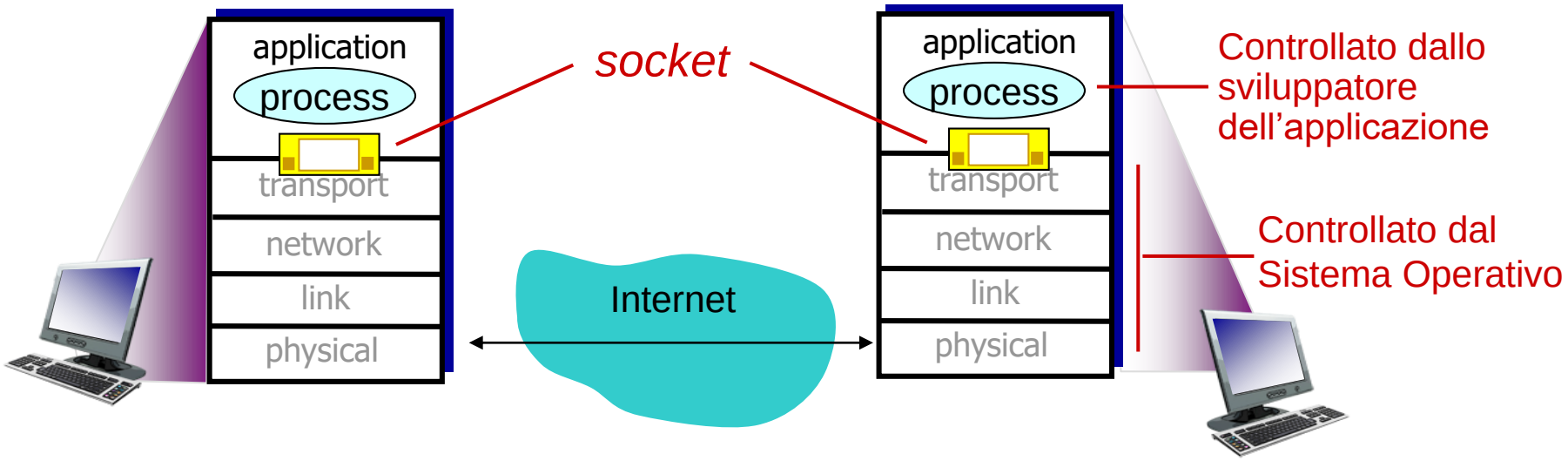
Tutti gli esempi sono testati nella macchina locale

- usiamo i socket per far comunicare processi in esecuzione sulla stessa macchina
- gli esempi funzionano altrettanto bene se i processi sono in esecuzione in due macchine distinte

Programmazione Socket

- **Obiettivo**: imparare a sviluppare applicazioni client/server che comunicano utilizzando i sockets

- **Socket**: porta tra il processo applicativo e il protocollo di trasporto end-to-end



Programmazione Socket

API = Application Programming Interface

Socket API

Introdotto in BSD4.1 UNIX, 1981

Creati, utilizzati e rilasciati esplicitamente dalle applicazioni

Paradigma client/server

Socket API offre due tipi di servizio di trasporto:

- UDP
- TCP

BSD = Berkeley Software Distribution

socket

È un'interfaccia (porta) *creata dall'applicazione e controllata dal SO* attraverso la quale un processo applicativo può *inviare e ricevere* messaggi a/da un altro processo applicativo

Programmazione Socket - Basi

- Il server deve essere in esecuzione prima che il client possa inviare dati ad esso (deamon)
- Il server deve avere un socket (porta) attraverso il quale riceve ed invia segmenti
- Allo stesso modo anche il client necessità di un socket
- Il client deve conoscere l'indirizzo IP del server e il numero di porta del processo server

Programmazione Socket *con UDP*

UDP: **non c'è "connessione" tra client and server**

- Non c'è handshaking
- Il mittente inserisce esplicitamente indirizzo IP e porta destinazione ad ogni segment
- Il SO inserisce l'indirizzo IP e la porta del socket origine ad ogni segmento
- Il server può ricavare indirizzo IP e porta del mittente dai segmenti ricevuti

Punto di vista dell'applicazione

UDP fornisce trasporto non affidabile di gruppi di bytes all'interno di datagrammi scambiati tra client e server

Nota: il termine corretto per "pacchetto UDP" sarebbe "datagramma", ma in questa lezione useremo indistamente i termini "segment", "pacchetto" e "datagramma" UDP.

Esempio

Client:

- L'utente inserisce una riga di testo
- L'applicazione client invia la riga al server

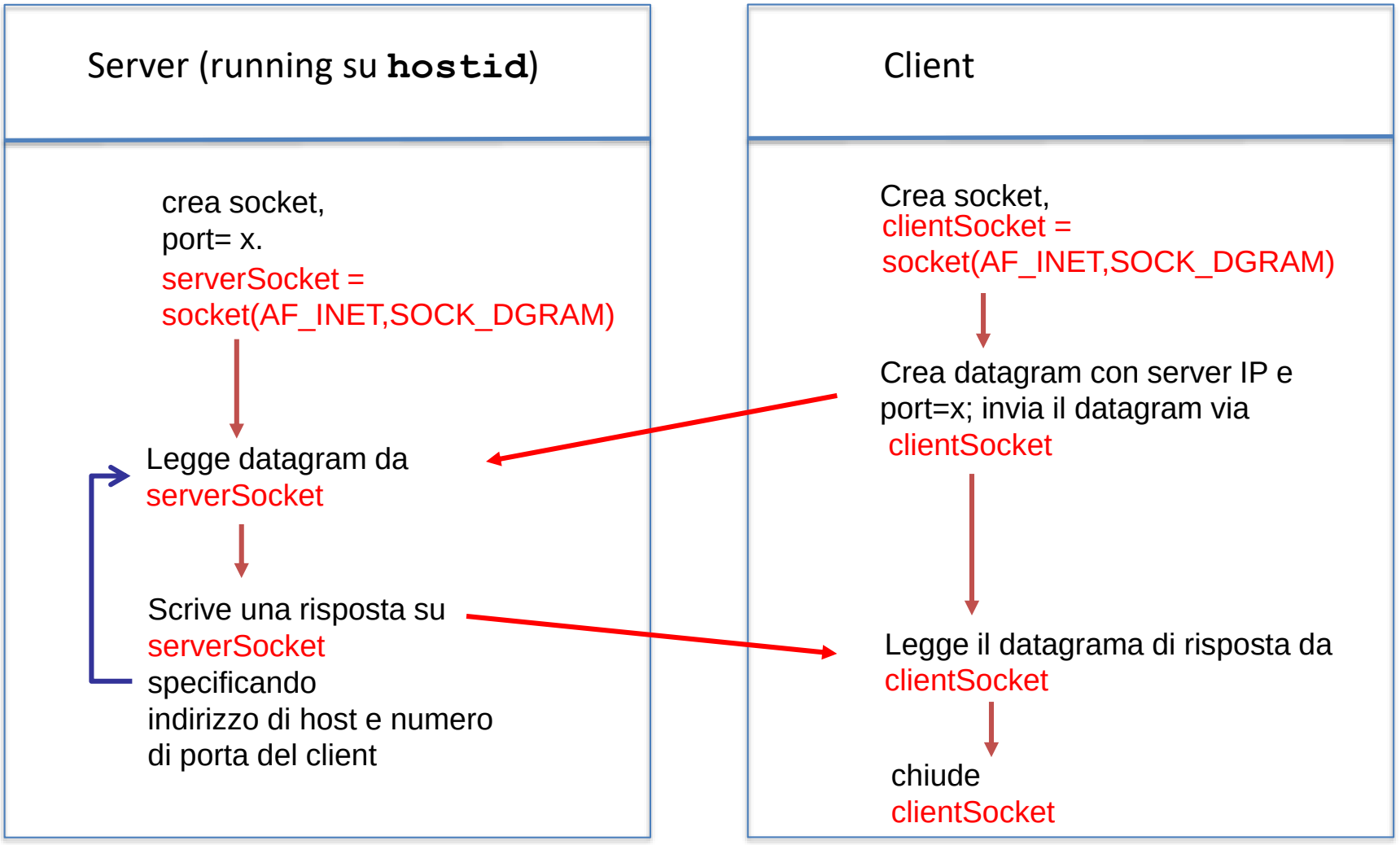
Server:

- Il server riceve la riga di testo
- Rende maiuscole tutte le lettere
- Invia la riga modificata al client

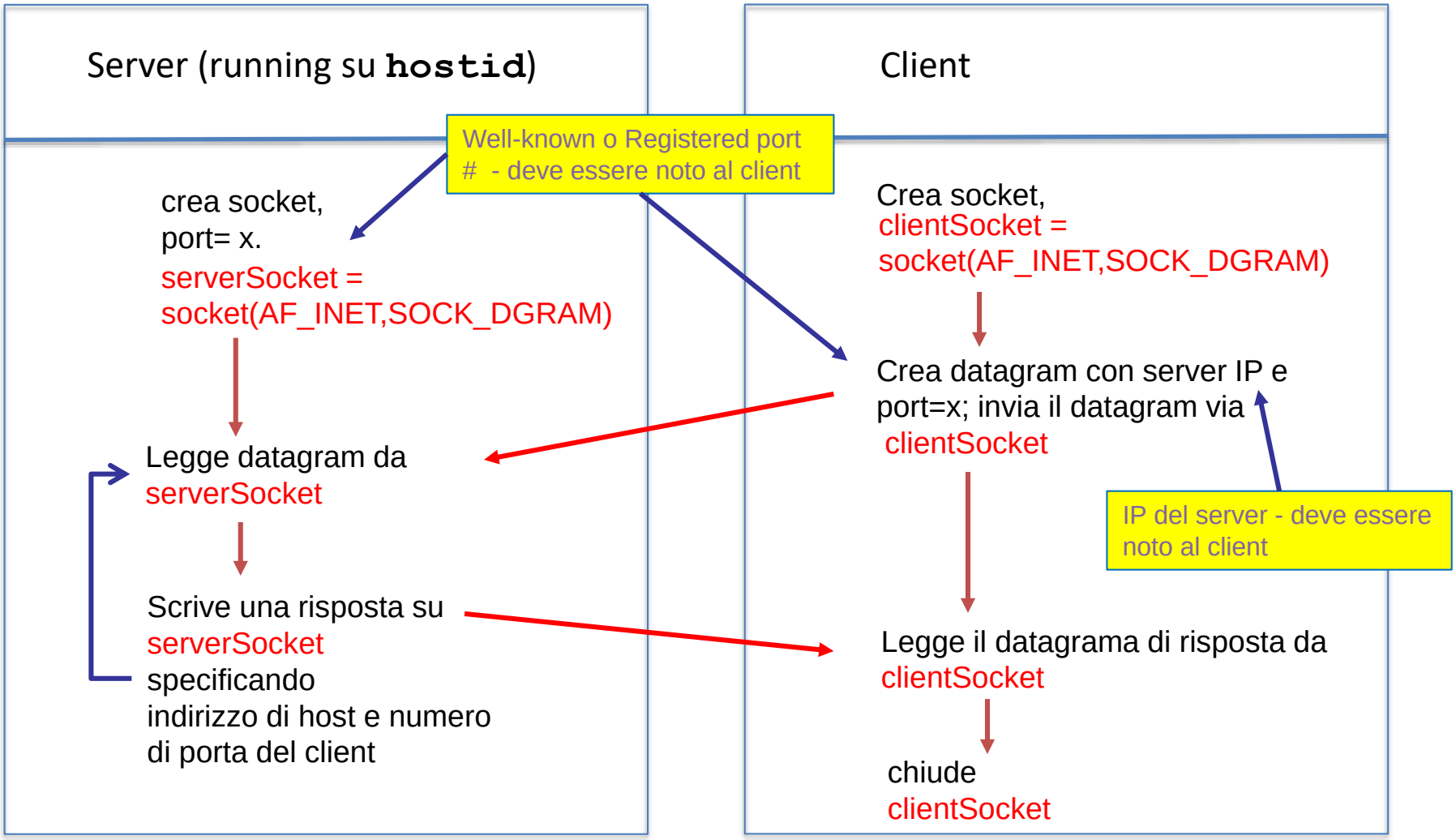
Client:

- Riceve la riga di testo
- La visualizza

Interazione tra socket Client/server: UDP



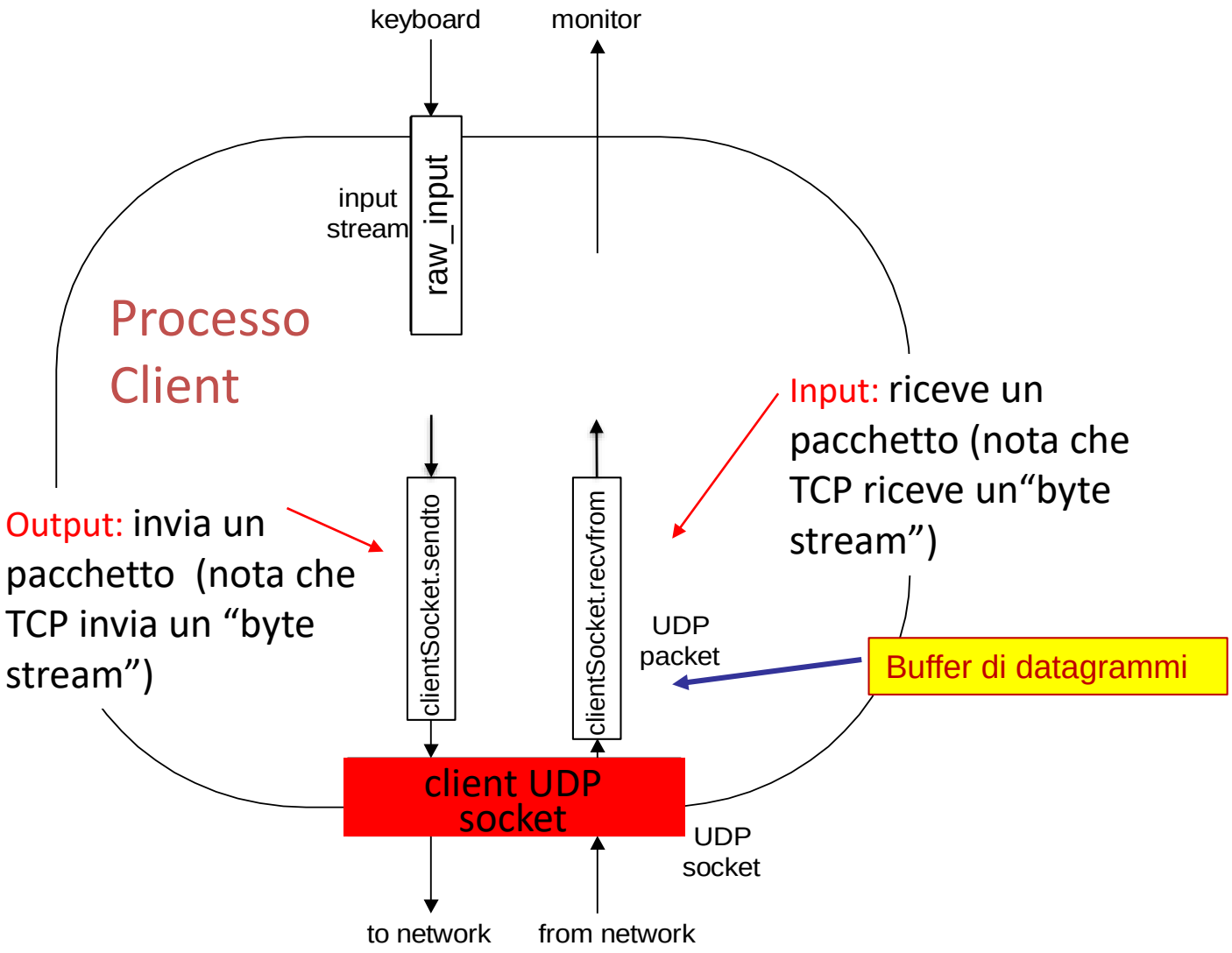
Interazione tra socket Client/server: UDP



serverSocket = socket(AF_INET,SOCK_DGRAM)

Nome oggetto (metodo) Nome classe

Esempio: Python client (UDP)



Applicazione esempio: *UDP client*

Include la libreria `Socket` di Python

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET,
                      SOCK_DGRAM)
message = input(Inserisci lettere:')
clientSocket.sendto(message.encode('utf-8',(serverName, serverPort))
modifiedMessage, serverAddress =
                      clientSocket.recvfrom(2048)
modifiedMessage = modifiedMessage.decode('utf-8')
Print(modifiedMessage)
clientSocket.close()
```

Nome simbolico del server

12000: # porta processo server

Crea un socket UDP per il server

SOCK_DGRAM → UDP

Legge l'input da tastiera

Aggiunge il nome del server e la porta al messaggio; lo invia nel socket

Il socket si pone in ascolto (l'esecuzione procede quando arriva un segmento UDP)

Legge i caratteri della risposta del server dal socket e li memorizza in una stringa

Dimensione del buffer

Mostra la stringa ricevuta e chiude il socket

La chiamata al DNS per traduzione serverName (hostname) → IP server è fatta dal sistema operativo

Applicazione esempio: *UDP server*

Crea un socket UDP

Assegna il socket all porta locale 12000

loop infinito

Legge dal socket UDP message, memorizzando l'indirizzo del client (IP e porta)

Risponde al client con la stringa tutta in maiuscolo

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))
Print("Il server è pronto a ricevere")
while 1:
    message, clientAddress =
        serverSocket.recvfrom(2048)
    print('Datagramma da: ", clientAddress)
    message = message.decode('utf-8')
    modifiedMessage = message.upper()
    serverSocket.sendto
    (modifiedMessage.encode('utf-8'), clientAddress)
```

12000: registered port #

Il socket si pone in ascolto (l'esecuzione procede quando arriva un segmento UDP)

Dimensione del buffer

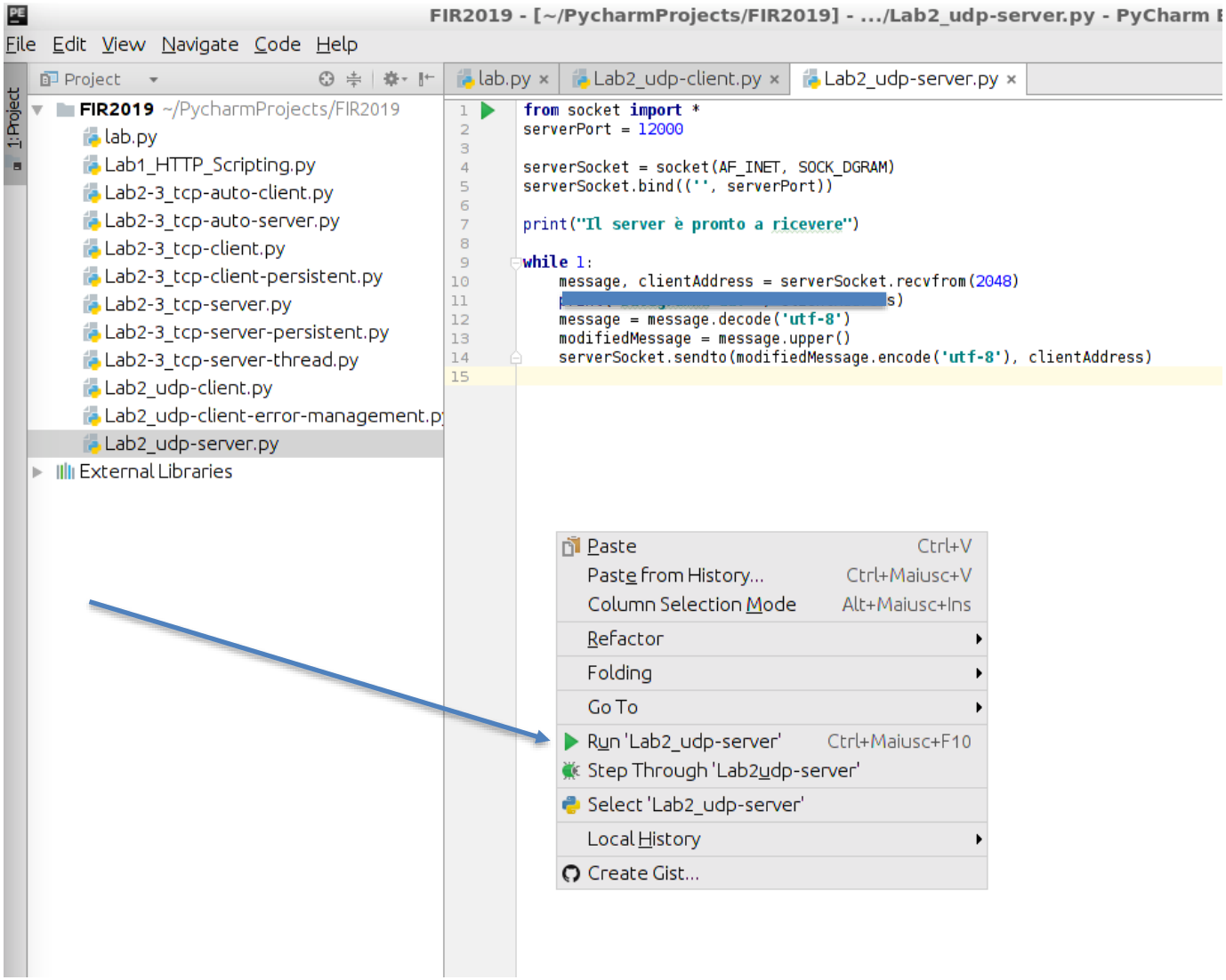
I dati per costruire la risposta sono esplicitamente ricavati dal pacchetto UDP ricevuto

Elaborazione strettamente seriale dei pacchetti

UDP: osservazioni e domande

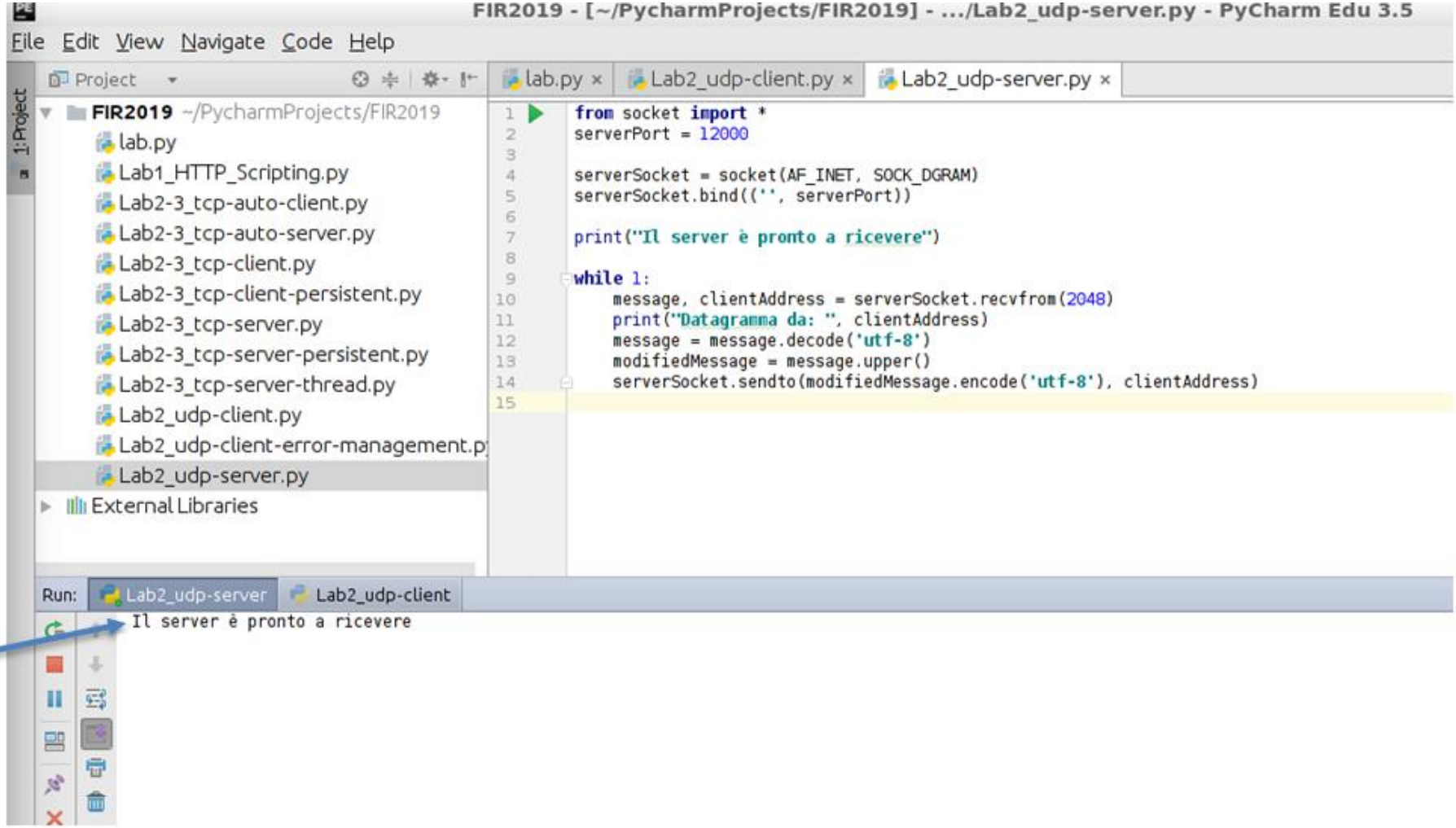
- Client e server utilizzando entrambi DatagramSocket
- IP destinazione e porta sono esplicitamente inseriti nel segmento
- Il client non può inviare un segmento al server senza conoscere l'indirizzo IP e la porta del server
- Il server può essere utilizzato da più di un client, ma tutti invieranno i pacchetti tramite lo stesso socket
  - La separazione dei dati da/a diversi client è un compito che spetta interamente al server

Eseguire il server

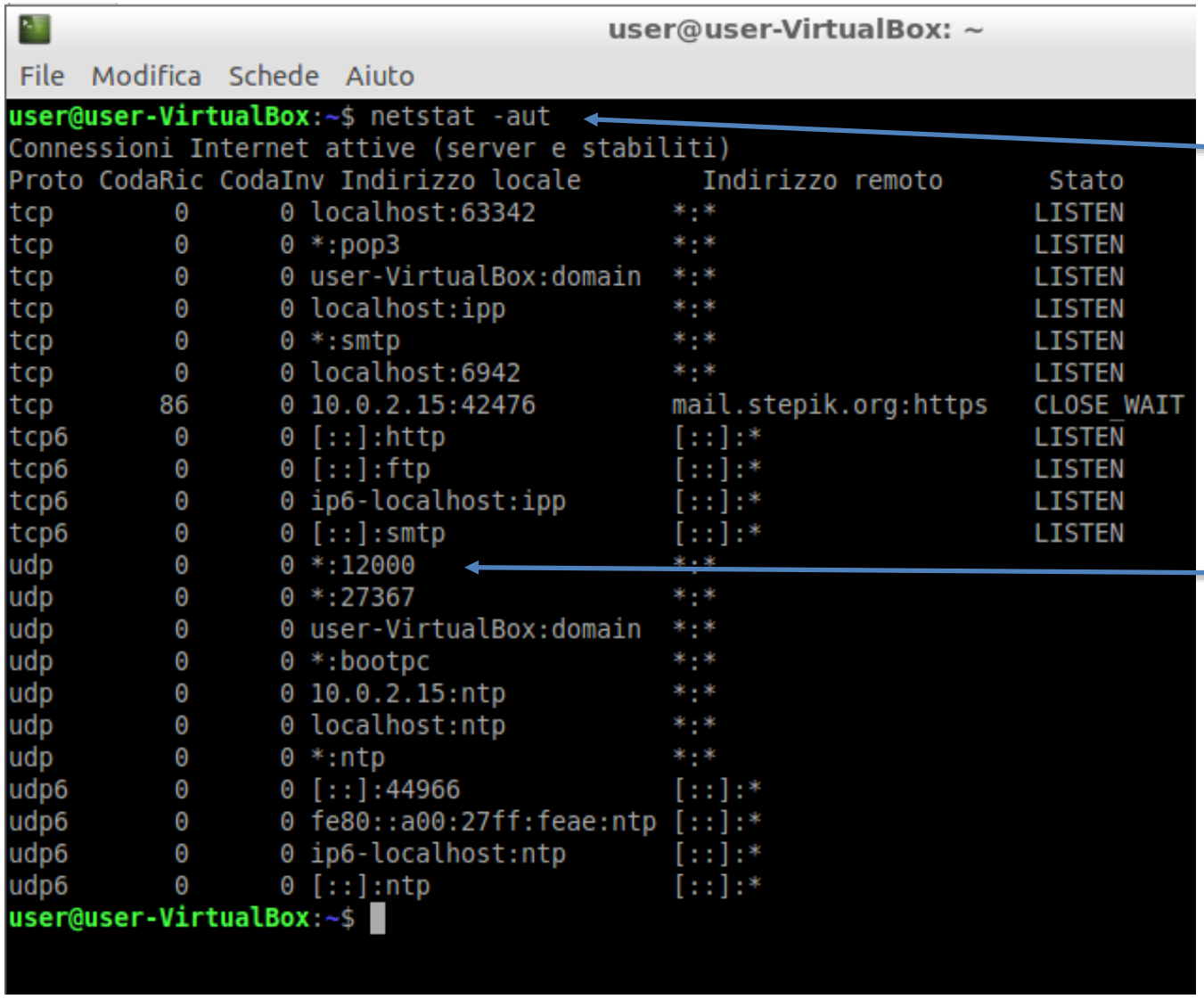


In pycharm il server si può lanciare dal menu contestuale (tasto destro sul nome dello script) oppure con la sequenza di tasti Ctrl+Maiusc+F10

Messaggi del server



Verifica nell'elenco dei processi: netstat

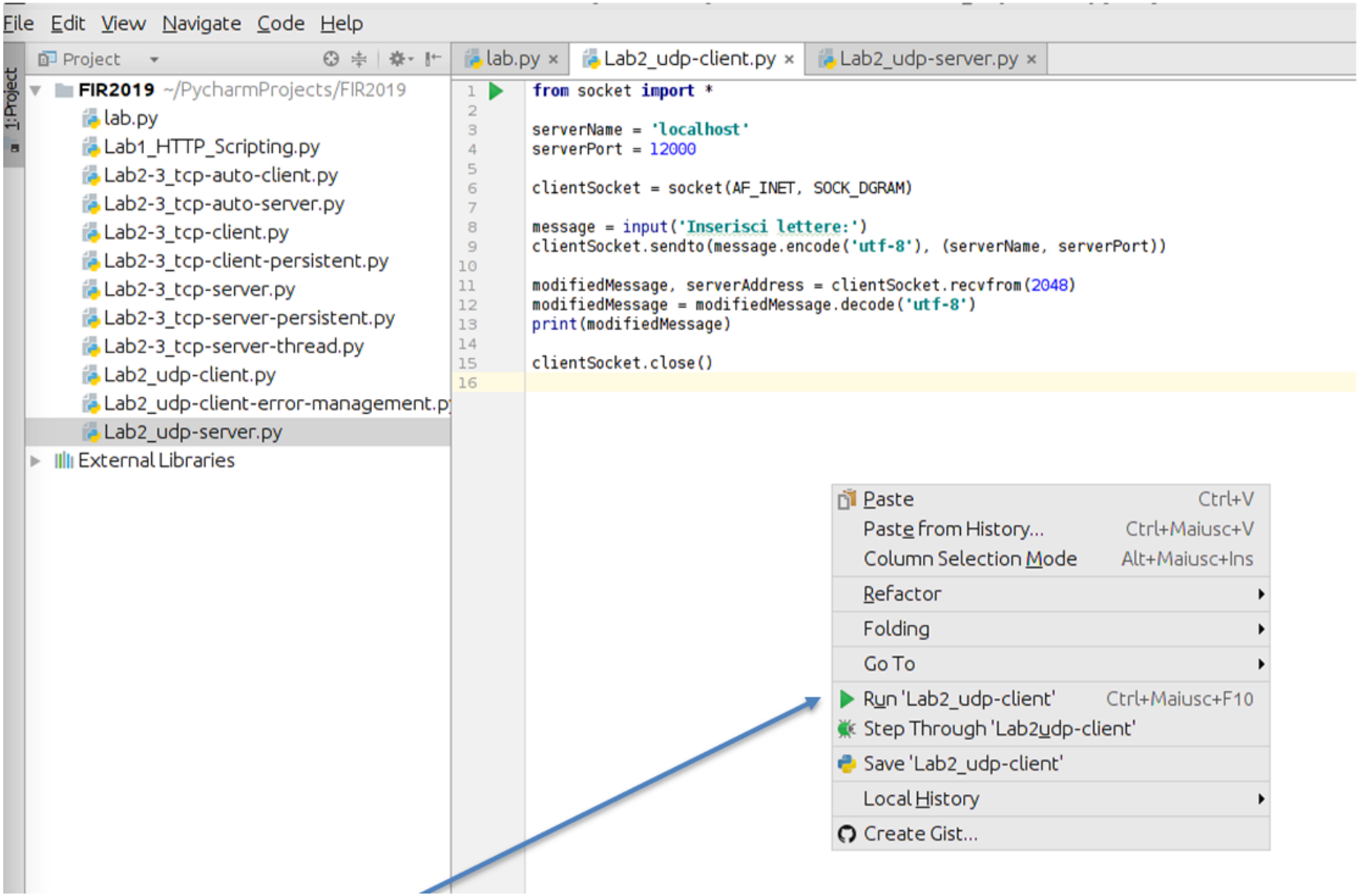


Comando:

`netstat -aut`

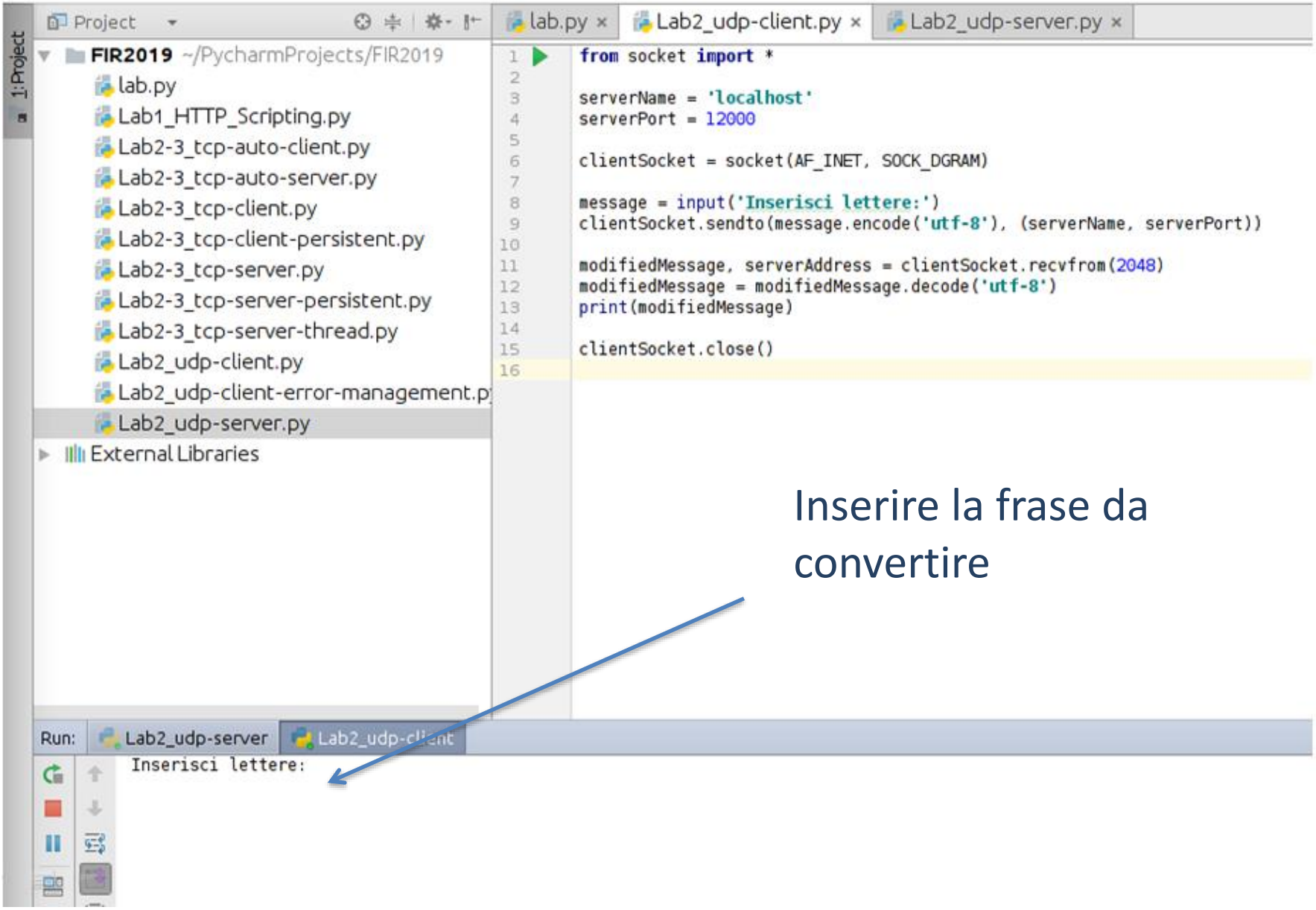
Il server è in esecuzione

Eseguire il client



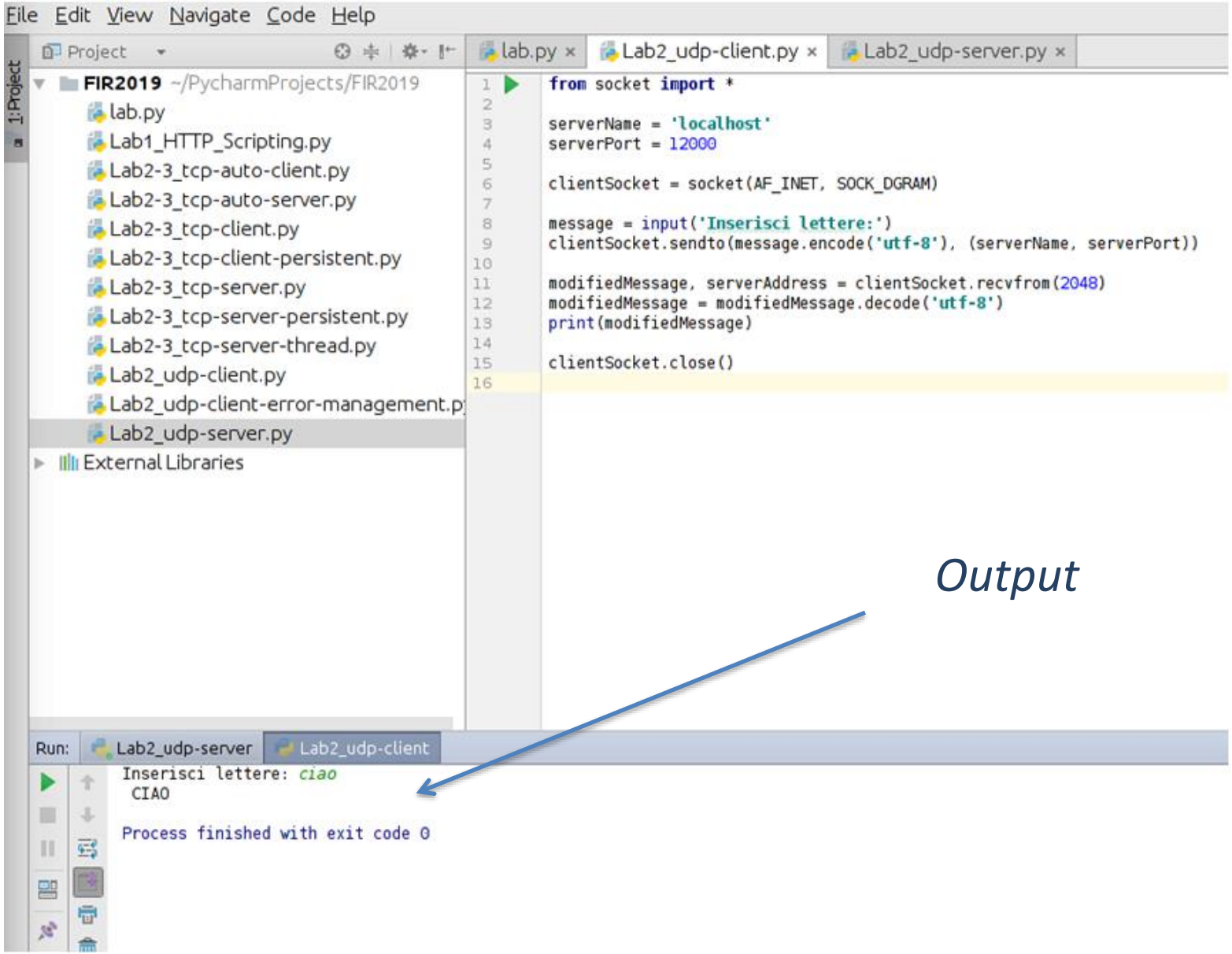
Analogamente al server anche il client si può lanciare dal menu contestuale (tasto destro sul nome dello script).

Inserimento input

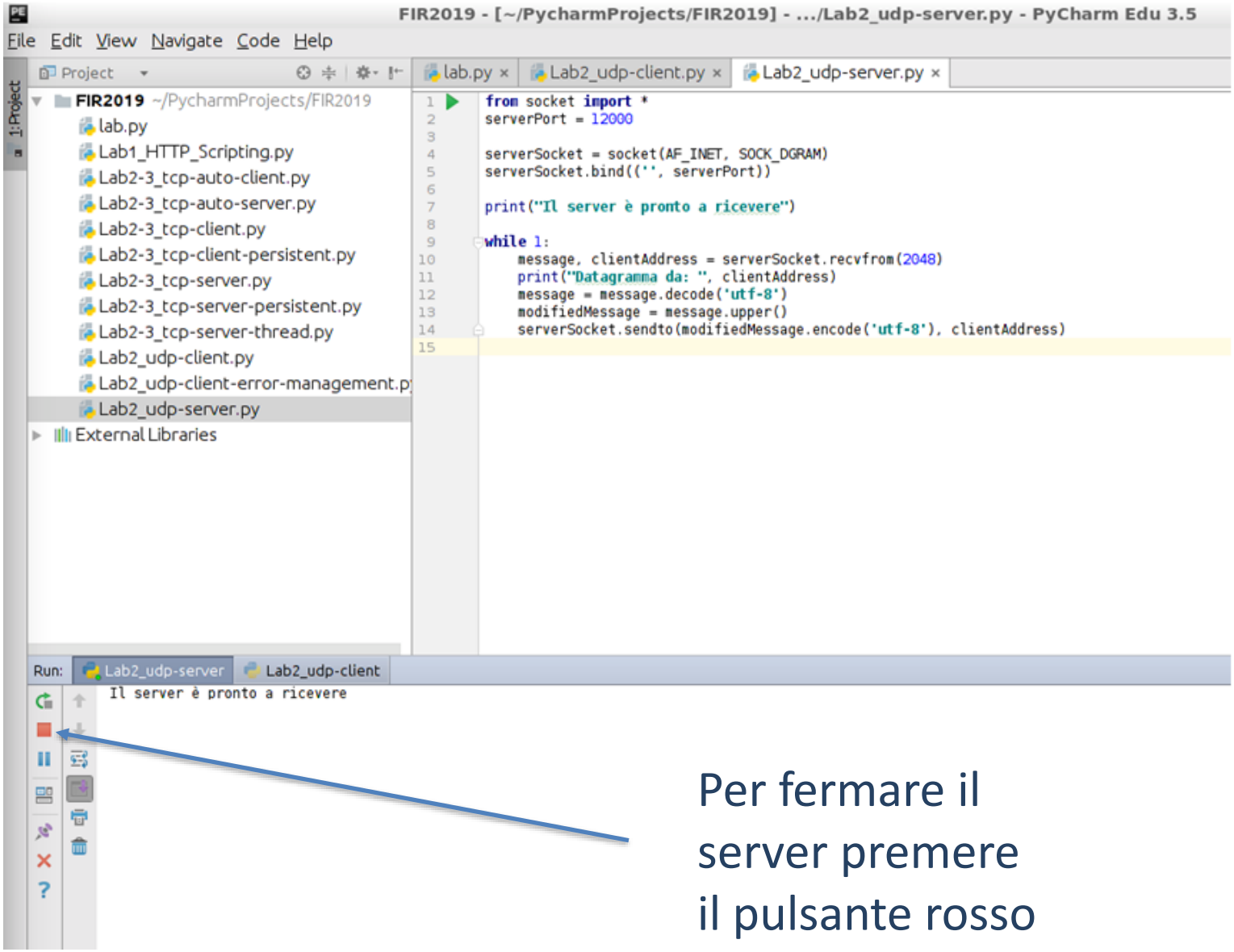


Inserire la frase da convertire

Ricezione e visualizzazione output



Stop del Server



Per fermare il server premere il pulsante rosso

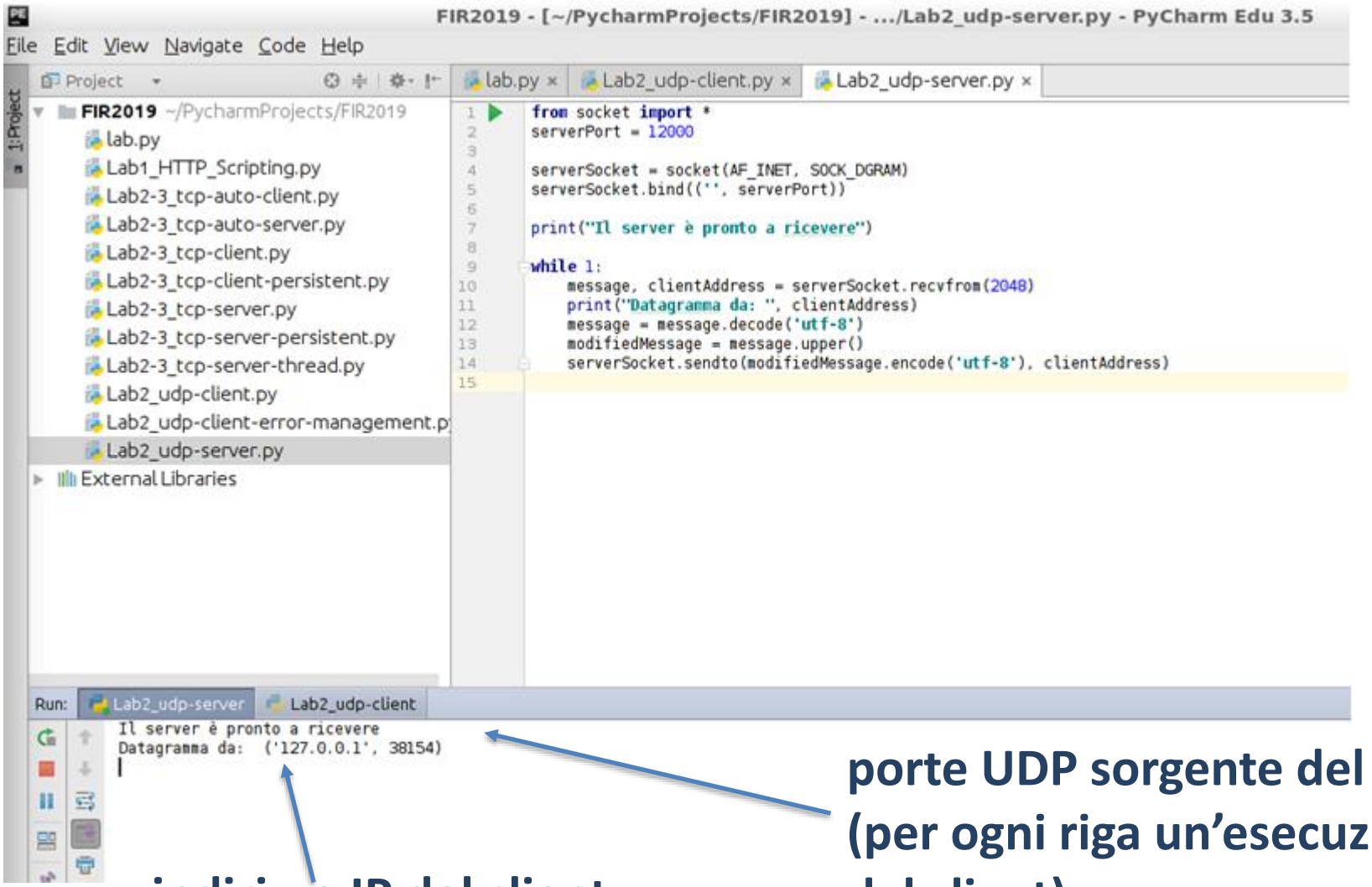
Passi successivi

- Il server è ancora in esecuzione
- Il client può essere eseguito altre volte e il sistema operativo assegna una porta diversa ogni volta.

Esercizio 2.1

a) Modificare il server in modo che scriva l’indirizzo IP e numero di porta del client.

Esercizio 2.1



porte UDP sorgente del client (per ogni riga un'esecuzione del client)

indirizzo IP del client
127.0.0.1 = loopback

Gestione degli errori: Problematiche

- Vediamo cosa accade se inviamo dati (e aspettiamo dati) da un server inesistente.
- Nel client modifichiamo la porta destinazione in 12001 (nessun processo è in ascolto su tale porta).
- Di norma il server risponde con un messaggio di errore, che deve essere gestito dal client.
- Tuttavia il messaggio di errore potrebbe perdersi oppure arrivare quando il client è bloccato sulla `socket.recvfrom`. In tal caso il messaggio viene ignorato.

Gestione degli errori: Soluzioni

Due possibili esiti:

1) il messaggio di errore dal server diventa un'eccezione Python

2) il client aspetta all'infinito un messaggio dal server

Soluzione:

- ✓ impostare un timeout alle operazioni sui socket
- ✓ catturare le eccezioni Python
  - scadenza timeout
  - messaggi di errore

Impostare un timeout

- Si usa il comando:
  - `clientSocket.settimeout(<timeout>)`

- Può essere inserito ovunque prima della chiamata a `recvfrom`.
- Il timeout è un valore con virgola espresso in secondi.

Catturare le eccezioni: Try - Except

- Gli errori di socket (compresi i timeout) lanciano eccezioni python.
- Per catturare l'eccezione racchiudere il codice che può lanciare l'eccezione in un blocco ***try … except***

```
try:
    modifiedMessage, serverAddress =
clientSocket.recvfrom(2048)
    print(modifiedMessage.decode('utf-8))
except:
    print("Timeout scaduti: Server non
raggiungibile")
finally:
    clientSocket.close()
```

Esercizio 2.2

a) Scrivere un nuovo client UDP che non dia errore nel caso di server inesistente.

b) Scrivere un client UDP errato che termina prima di ricevere la risposta dal server. Cosa accade al server? Che modifiche sono necessarie?

Soluzione 2.2

```
from socket import *
serverName = 'localhost'
serverPort = 12001
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.settimeout(2)
message = input('Inserisci lettere:')
clientSocket.sendto(message.encode('utf-8'), (serverName, serverPort))
try:
    modifiedMessage, serverAddress = clientSocket.recvfrom(2048)
    # in case of error blocks forever
    print(modifiedMessage.decode('utf-8'))
except:
    print("Timeout scaduto: Server non raggiungibile ")
finally:
    clientSocket.close()
```

Esercizio 2.3

Si vuole scrivere un'applicazione client/server UDP per conteggiare il numero di consonanti presenti in una stringa.

- **Il client chiede all'utente di inserire una stringa**
- **il server risponde indicando il numero di consonanti presenti nella stringa (sia maiuscole che minuscole).**

Hint: y.count(x) conta quante volte appare l'elemento x nella lista y.

Scrivere gli script "UDP client" e "UDP server" date le seguenti specifiche:

- Utilizzare indirizzi IPv4
- Time-out in ricezione (lato client): 5 secondi
- Lunghezza buffer di ricezione: 2048 byte

Soluzione 2.3

UDP server

```
from socket import *
serverPort = 12000
serverSocket = socket(AF_INET, SOCK_DGRAM)
serverSocket.bind(('', serverPort))

print('Server pronto a ricevere!')
vocali = ['A','E','I','O','U’]

while 1:
    messaggio, clientAddress = serverSocket.recvfrom(2048)
    messaggio = messaggio.decode(‘utf-8’)
    num = len(messaggio)
    for voc in vocali:
        num = num – messaggio.count(voc)
    risposta = "Il messaggio contiene "+str(num)+" consonanti."
    serverSocket.sendto(risposta.encode(‘utf-8’), clientAddress)
```

Soluzione 2.3

UDP client

```
from socket import *
serverName = 'localhost'
serverPort = 12000
clientSocket = socket(AF_INET, SOCK_DGRAM)
clientSocket.settimeout(5)

message = input('Inserisci una parola (senza caratteri speciali):')
clientSocket.sendto(message.encode(‘utf-8’), (serverName, serverPort))

try:
    reply, serverAddress = clientSocket.recvfrom(2048)
    print(reply.decode(‘utf-8’))
except:
    print("Il server non ha risposto entro il timeout...")
finally:
    clientSocket.close()
```

Esercizio 2.4 (Soluzione Dopo Il Lab)

Si vuole scrivere un'applicazione client/server UDP

- **Il client chiede all'utente di inserire un numero**
- **Il server risponde indicando se il numero inserito e' un numero primo o no**

Scrivere gli script "UDP client" e "UDP server" date le seguenti specifiche:

- Utilizzare indirizzi IPv4
- Time-out in ricezione (lato client): 2 secondi