



POLITECNICO
MILANO 1863



Durante Lab

Fondamenti di Comunicazioni e Internet

**Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci**



POLITECNICO
MILANO 1863



Lab 1: Scripting con Python

Soluzione Post Lab

Esercizio 1.2 (Ora tocca a voi!)

- Scrivere uno script che stampi il nome della pagina col **miglior tempo di risposta medio** tra **6** siti Internet. Per il calcolo del tempo medio, si definisca la funzione *media(list)* che ritorna la media dei valori contenuti in *list*.
- Numero di richieste = 10
- Siti internet:
 1. `http://www.google.com`
 2. `http://www.youtube.com`
 3. `http://www.polimi.it`
 4. `http://www.wikipedia.org`
 5. `http://www.amazon.com`
 6. `http://www.twitter.com`



Soluzione Esercizio 1.2

```
1  import requests
2
3  def media(stat):
4      avg = sum(stat)/len(stat)
5      return avg
6
7  siti = ['http://www.google.com', 'http://www.youtube.com', 'http://www.polimi.it',
8          'http://www.wikipedia.org', 'http://www.amazon.com', 'http://www.twitter.com']
9
10 avg = []
11
12 for url in siti:
13
14     print('Test: ', url)
15     tempi = []
16
17     for _ in range(10):
18         r = requests.get(url)
19         tempi.append(r.elapsed.microseconds/1000)
20
21     tempo_medio = media(tempi)
22     print('Tempo di risposta - AVG: ', tempo_medio, 'ms')
23     avg.append(tempo_medio)
24
25 print("Il sito", siti[avg.index(min(avg))],
26       "ha il tempo di risposta medio più basso:" , min(avg), 'ms')
```



Soluzione Esercizio 1.2

```
1  import requests
2
3  def media(stat):
4      avg = sum(stat)/len(stat)
5      return avg
6
7  siti = ['http://www.google.com', 'http://www.youtube.com', 'http://www.polimi.it',
8          'http://www.wikipedia.org', 'http://www.amazon.com', 'http://www.twitter.com']
9
10 avg = []
11
12 for url in siti:
13
14     print('Test: ', url)
15     tempi = []
16
17     for _ in range(10):
18         r = requests.get(url)
19         tempi.append(r.elapsed.microseconds/1000)
20
21     tempo_medio = media(tempi)
22     print('Tempo di risposta - AVG: ', tempo_medio, 'ms')
23     avg.append(tempo_medio)
24
25 print("Il sito", siti[avg.index(min(avg))],
26       "ha il tempo di risposta medio più basso.", min(avg), 'ms')
```



Soluzione Esercizio 1.2

```
1 import requests
2
3 def media(stat):
4     avg = sum(stat)/len(stat)
5     return avg
6
7 siti = ['http://www.google.com', 'http://www.youtube.com', 'http://www.polimi.it',
8         'http://www.wikipedia.org', 'http://www.amazon.com', 'http://www.twitter.com']
9
10 avg = []
11
12 for url in siti:
13
14     print('Testo:', url)
15     tempi = []
16
17     for _ in range(10):
18         r = requests.get(url)
19         tempo = r.elapsed
20
21         tempo_medio = media(tempi)
22         print('Tempo medio:', tempo_medio, 'ms')
23         avg.append(tempo_medio)
24
25 print("Il sito", siti[avg.index(min(avg))],
26       "ha il tempo di risposta medio più basso:", min(avg), 'ms')
```



Che cosa vuol dire questo codice? Come stiamo estraendo dalla lista il sito mediamente più veloce?

Soluzione Esercizio 1.2

```
1  siti = ['SIT01', 'SIT02', 'SIT03']
2  avg = [150, 90, 200]
3
4  # Per trovare il minimo
5  minimo = min(avg) # = 90
6  # Per ricavare l'indice a esso associato
7  indice_minimo = avg.index(minimo) # = 1
8  # se so già che 90 è il minimo, questa
9  # riga è equivalente alla precedente
10 indice_elemento_90 = avg.index(90) # = 1
11 # inserisco l'indice all'interno della lista
12 # contenente i siti considerati
13 sito_corrispondente = siti[avg.index(min(avg))] # = siti[1] = 'SIT02'
```





POLITECNICO
MILANO 1863

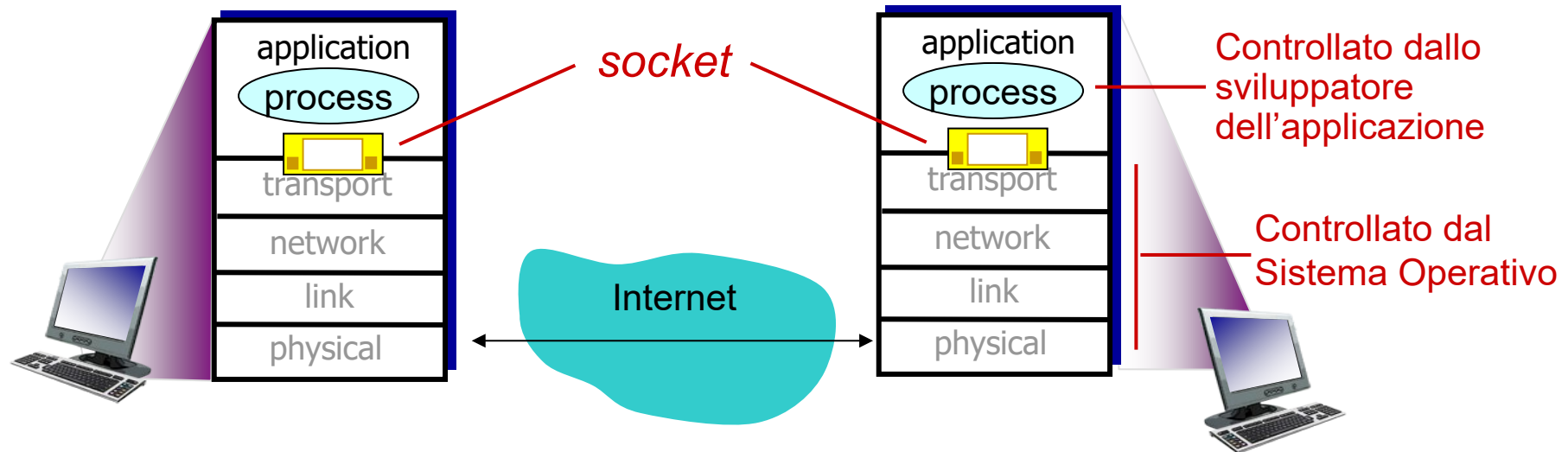


Lab 2: UDP Socket Programming

Antonio Capone, Matteo Cesana,
Guido Maier, Francesco Musumeci

Programmazione Socket

- **Obiettivo:** imparare a sviluppare applicazioni client/server che comunicano utilizzando i socket
- **Socket:** interfaccia 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

SERVER

- Il server deve essere **in esecuzione** prima che il client possa inviare dati ad esso (daemon)
- Il server deve avere un **socket** (porta) attraverso il quale riceve ed invia segmenti

CLIENT

- Allo stesso modo anche il client necessita 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

- No handshake
- Il mittente inserisce esplicitamente indirizzo IP e porta destinazione ad ogni segment
- Il S.O. 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

POV: Applicazione

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

Nota: il termine corretto per **pacchetto UDP** sarebbe **datagramma**, comunemente si usano i termini *segmento*, *pacchetto* e *datagramma* **indistintamente**.



Interazione tra socket Client/Server: UDP

SERVER (running su `hostid`)

Crea socket sulla
porta `srv_port`

serverSocket =
`socket(AF_INET, SOCK_DGRAM)`

↓
Legge datagram da
serverSocket

↓
Scrive una risposta su
serverSocket
specificando
indirizzo di host e numero
di porta del client

CLIENT

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

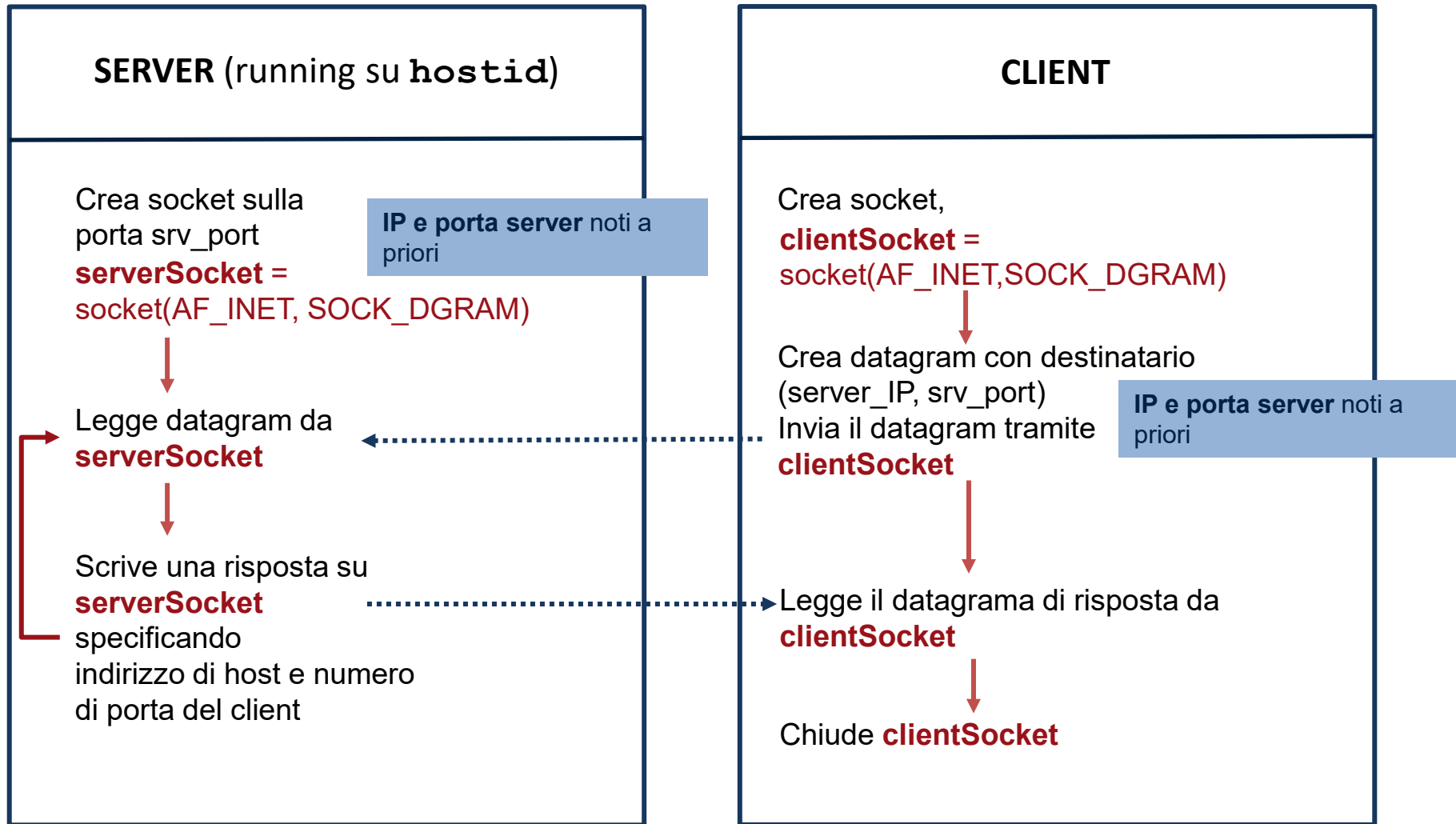
↓
Crea datagram con destinatario
(`server_IP`, `srv_port`)
Invia il datagram tramite
clientSocket

↓
Legge il datagrama di risposta da
clientSocket

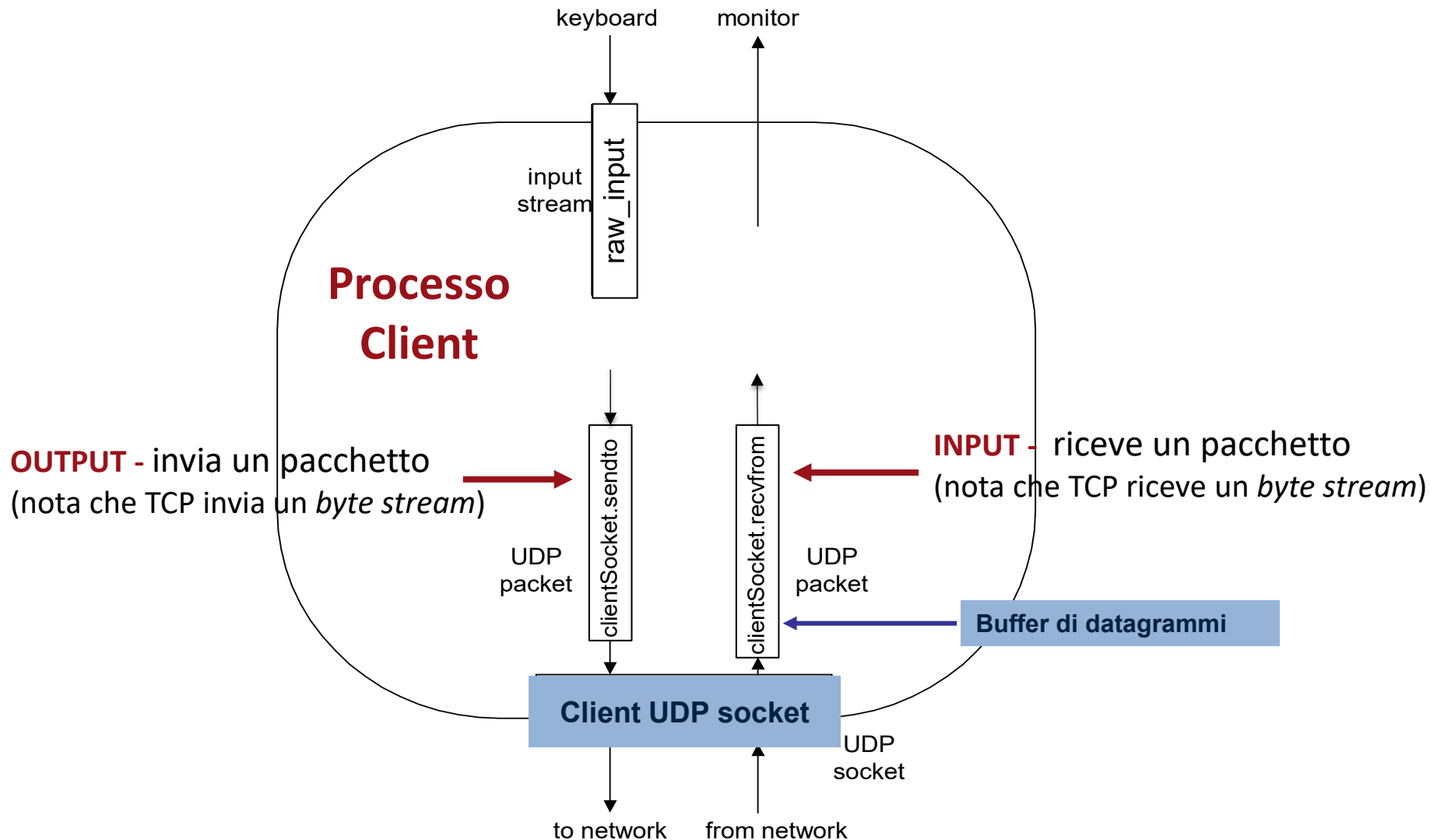
↓
Chiude **clientSocket**



Interazione tra socket Client/Server: UDP



Esempio: Python client (UDP)



Esempio di interazione client-server

- **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



Applicazione esempio: client UDP

```
1  from socket import * # include la libreria Socket di Python
2
3  # nome del server (indirizzo IP)
4  server_name = 'localhost'
5
6  # (1 <-> 1023 'Well-known ports')
7  # (1024 <-> 49151 'Registered ports')
8  server_port = 12000 # porta del processo server
9
10 # crea un socket UDP (il nostro client)
11 client_socket = socket(AF_INET, SOCK_DGRAM)
12
13 # TUPLA che include il "nome" del server e la sua porta
14 server_address = (server_name, server_port)
15
16 # legge l'input da tastiera
17 message = input("Inserisci il messaggio per il server:")
18
19 print("Invio il seguente messaggio: ", message)
20
21 # aggiunge nome e porta del server al messaggio
22 client_socket.sendto(message.encode("utf-8"), server_address)
23 buffer_size = 2048 # la dimensione del buffer
24
25 # si mette in ascolto per la risposta...
26 server_response, address = client_socket.recvfrom(buffer_size)
27
28 print("Risposta del server: ", server_response.decode("utf-8"))
29
30 # chiudo il socket
31 client_socket.close()
```

SOCK_DGRAM → UDP

input() legge da tastiera

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

Applicazione esempio: server UDP

```
1  from socket import *
2
3  # nome del server (indirizzo IP)
4  server_name = '127.0.0.1'
5
6  # (1 <-> 1023 'Well-known ports')
7  # (1024 <-> 49151 'Registered ports')
8  server_port = 12000 # numero di porta del server
9
10 # crea un socket UDP (il nostro server)
11 server_socket = socket(AF_INET, SOCK_DGRAM)
12
13 # TUPLA che include il "nome" del server e la sua porta
14 server_address = (server_name, server_port)
15
16 # lega il socket a un indirizzo fisso
17 # rendendolo un server
18 server_socket.bind(server_address)
19
20 print("Il server è funzionante!")
21 buffer_size = 2048 # la dimensione del buffer
22
23 # loop infinito per esporre sempre il servizio
24 while 1:
25     # messaggio ricevuto e client memorizzato (IP e porta)
26     message, client_address = server_socket.recvfrom(buffer_size)
27     message = message.decode("utf-8") # per decodificare il messaggio
28
29     print("Messaggio ricevuto dal client: ", message)
30     modified_message = message.upper() # converte il messaggio in MAIUSCOLO
31
32     #reply with the message upper case
33     server_socket.sendto(modified_message.encode("utf-8"), client_address)
34
35     print("Ho risposto al client con: ", modified_message)
```

Elaborazione strettamente seriale dei pacchetti

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

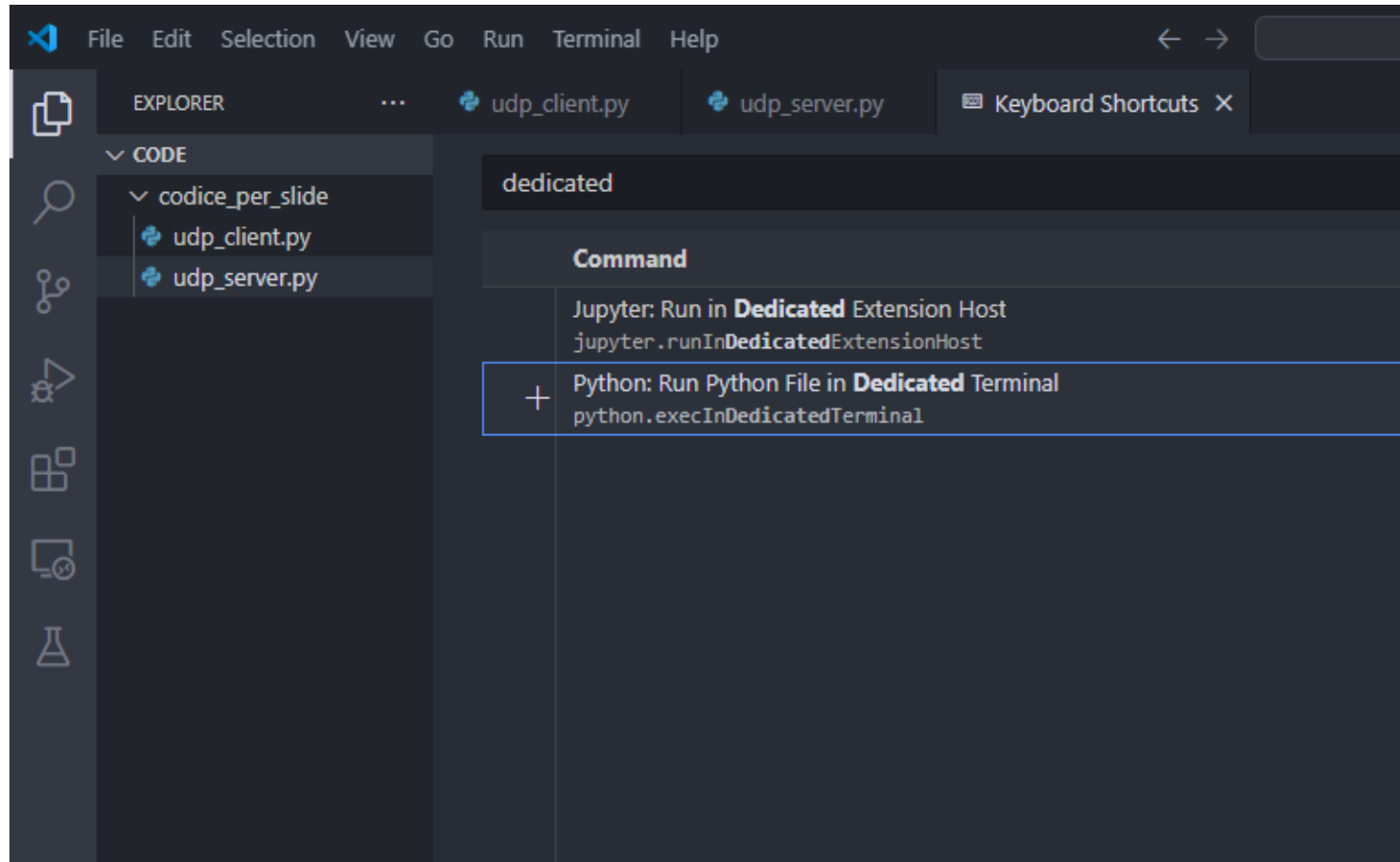
I dati per costruire la risposta sono esplicitamente ricavati dal pacchetto UDP ricevuto

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 i client invieranno i pacchetti allo stesso server socket:
 - La separazione dei dati da/a diversi client è un compito che spetta interamente al server



Eseguire il server



In VS Code il server si può lanciare dal menu contestuale (tasto play in alto a destra) oppure scegliere una sequenza di tasti dal menu shortcut chiamabile con: Ctrl+K+S



Messaggi del server (sul Terminale)

The screenshot shows the Visual Studio Code (VS Code) interface. The Explorer panel on the left shows a project structure with a folder named 'codice_per_slide' containing two files: 'udp_client.py' and 'udp_server.py'. The Code panel displays the contents of 'udp_server.py', which is a Python script for a UDP server. The script includes comments in Italian and code for binding a socket to a specific address and port, printing a message, and entering an infinite loop to handle incoming data.

```

13  # TUPLA che include il "nome" del server e la sua porta
14  server_address = (server_name, server_port)
15
16  # Lega il socket a un indirizzo fisso
17  # rendendolo un server
18  server_socket.bind(server_address)
19
20  print("Il server è funzionante!")
21  buffer_size = 2048 # la dimensione del buffer
22
23  # Loop infinito per esporre sempre il servizio
24  while True:

```

The Terminal panel at the bottom shows the command prompt output. The command executed is:
 `PS C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\FCI_laboratorio\24_25_che_faccio_io\Lab 2\code> & C:/Users/lucaj/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/lucaj/OneDrive - Politecnico di Milano/PhD/lezioni_che_faccio_io/FCI_laboratorio/24_25_che_faccio_io/Lab 2/code/codice_per_slide/udp_server.py"`
 The output shows the message 'Il server è funzionante!' (The server is working!), which is highlighted with a red rectangle in the original image.

The status bar at the bottom indicates the current file is 'Ln 38, Col 1', the encoding is 'UTF-8', the line ending is 'CRLF', the language is 'Python', and the version is '3.13.2 64-bit'.

Verifica nell'elenco dei processi: netstat



```
Windows PowerShell
PS C:\Users\lucaj> netstat -an | FINDSTR 12000
UDP 127.0.0.1:12000 *:*
PS C:\Users\lucaj>
```

Il server è in esecuzione

Comando:
netstat -an

FINDSTR è l'equivalente di grep in Linux - *trova una regular expression*.

Quando concatenato (utilizzando la pipe |) a un comando agisce sull'output del comando precedente.

Con **netstat**: vogliamo cercare il numero di porta utilizzato dal nostro server UDP.

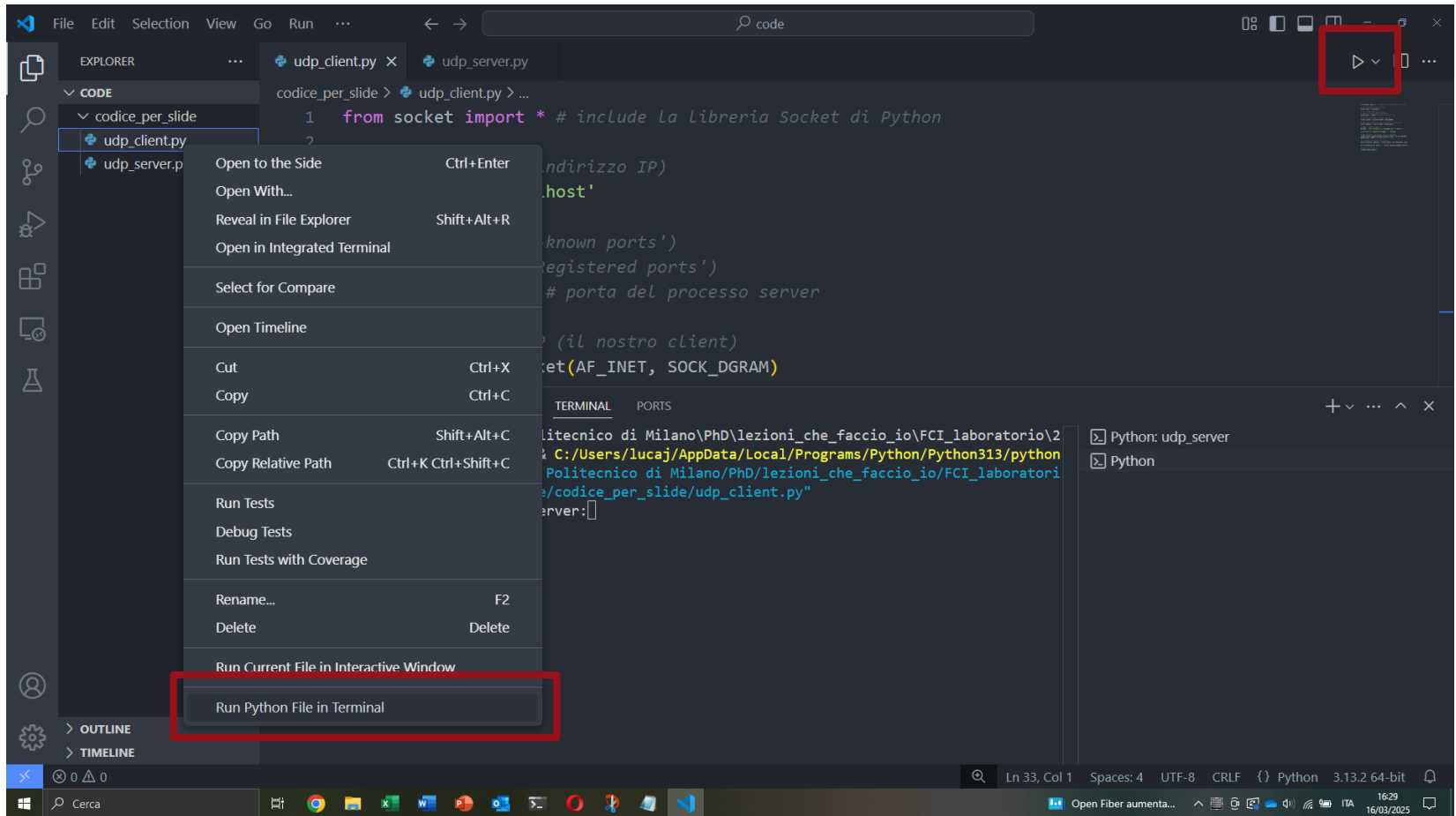


Extra: alcune flag di netstat interessanti

- -a
Visualizza tutte le connessioni e le porte di ascolto
- -b
Visualizza l'eseguibile coinvolto nella creazione di ogni connessione o porta di ascolto (**richiede privilegi admin**)
- -n
Visualizza indirizzi e numeri di porta in formato numerico
- -p <protocollo>
Mostra le connessioni per il protocollo proprietario specificato da <protocollo>: IP, IPv6, ICMP, ICMPv6, TCP, UDP, TCPv6 o UDPv6



Eseguire il client



Anche il client si può lanciare dal bottone play in alto a destra, da un terminale dedicato, o dal menu contestuale (tasto destro sul nome dello script).



Inserimento Input

The image shows a Visual Studio Code editor with a Python file named `udp_client.py` open. The code is as follows:

```
15
16 # Legge l'input da tastiera
17 message = input("Inserisci il messaggio per il server:")
18
19 print("Invio il seguente messaggio: ", message)
20
21 # aggiunge nome e porta del server al messaggio
22 client_socket.sendto(message.encode("utf-8"), server_address)
23 buffer_size = 2048 # la dimensione del buffer
24
25 # si mette in ascolto per la risposta...
```

Below the code editor is a terminal window showing the command prompt and the execution of the program. The prompt is `Inserisci il messaggio per il server:`, and a red arrow points to it from a callout box that says "Inserire la frase da convertire".

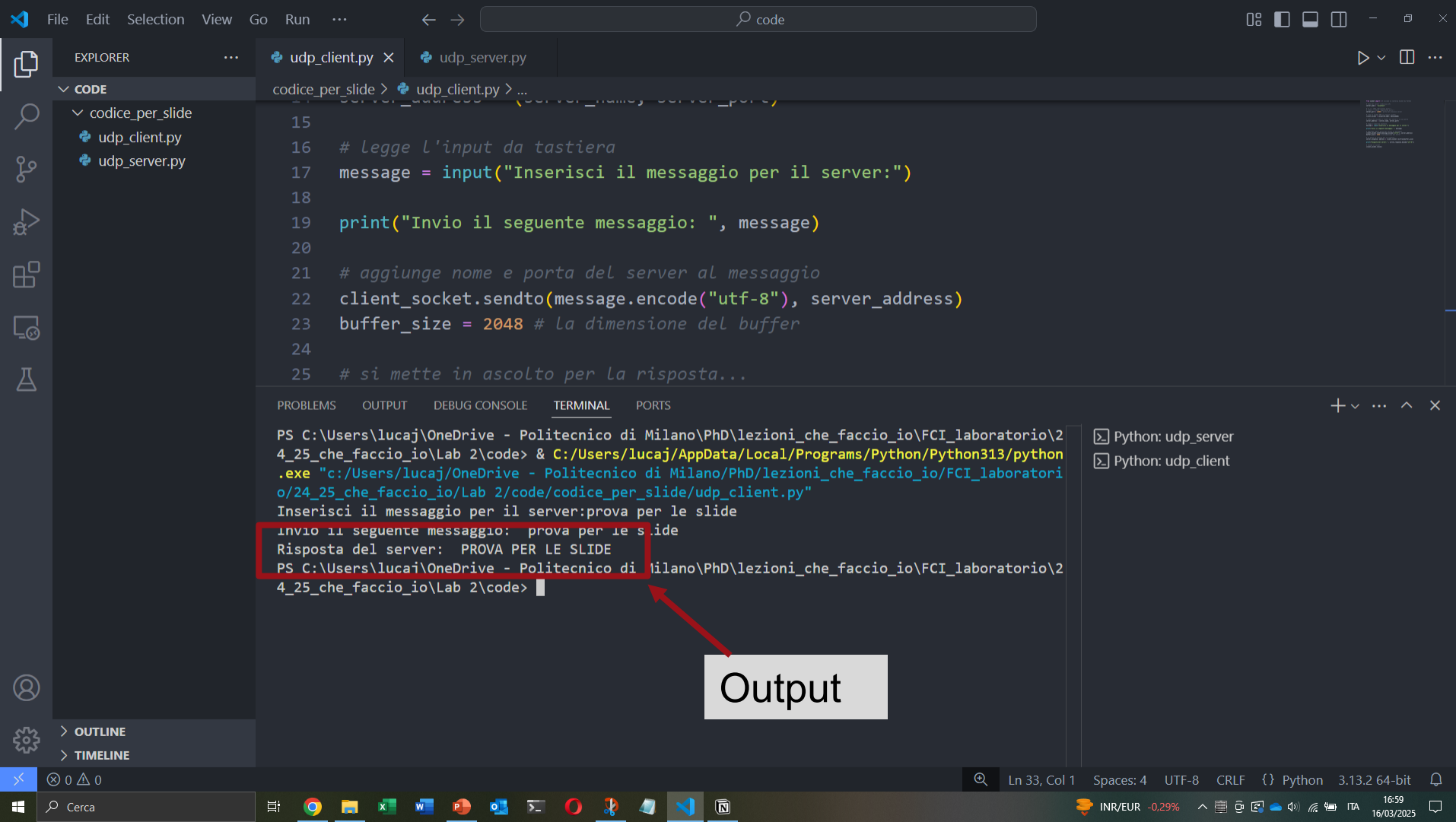
Terminal output:

```
PS C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\FCI_laboratorio\24_25_che_faccio_io\Lab 2\code> & C:/Users/lucaj/AppData/Local/Programs/Python/Python313/python.exe "c:/Users/lucaj/OneDrive - Politecnico di Milano/PhD/lezioni_che_faccio_io/FCI_laboratorio/24_25_che_faccio_io/Lab 2/code/codice_per_slide/udp_client.py"
Inserisci il messaggio per il server:
```

Callout box text: Inserire la frase da convertire



Ricezione e visualizzazione output



The image shows a Visual Studio Code editor window with a dark theme. The Explorer sidebar on the left shows a project named 'codice_per_slide' containing two files: 'udp_client.py' and 'udp_server.py'. The main editor area displays the code for 'udp_client.py'. The code is as follows:

```
15
16 # Legge l'input da tastiera
17 message = input("Inserisci il messaggio per il server:")
18
19 print("Invio il seguente messaggio: ", message)
20
21 # aggiunge nome e porta del server al messaggio
22 client_socket.sendto(message.encode("utf-8"), server_address)
23 buffer_size = 2048 # la dimensione del buffer
24
25 # si mette in ascolto per la risposta...
```

Below the code editor is the TERMINAL panel. It shows the command prompt for a Windows PowerShell session. The user has navigated to the directory 'C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\FCI_laboratorio\24_25_che_faccio_io\Lab 2\code' and executed the command 'python .\udp_client.py'. The terminal output is as follows:

```
PS C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\FCI_laboratorio\24_25_che_faccio_io\Lab 2\code> & C:/Users/lucaj/AppData/Local/Programs/Python/Python313/python.exe ".\udp_client.py"
Inserisci il messaggio per il server:prova per le slide
Invio il seguente messaggio: prova per le slide
Risposta del server: PROVA PER LE SLIDE
PS C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\FCI_laboratorio\24_25_che_faccio_io\Lab 2\code>
```

A red box highlights the output 'Risposta del server: PROVA PER LE SLIDE'. A red arrow points from a white box labeled 'Output' to this highlighted text. On the right side of the terminal panel, there is a list of running processes: 'Python: udp_server' and 'Python: udp_client'.



Stop del Server

The screenshot shows the Visual Studio Code interface with a Python file named `udp_server.py` open. The code defines a UDP server that binds to a specific address and port, prints a message, and enters an infinite loop to receive and respond to client messages. The terminal window shows the execution of the script, which has successfully started and is currently receiving a message from a client.

Per fermare il server bisogna interrompere il processo (terminale)

The terminal output shows the following commands and results:

```
PS C:\Users\lucaj\OneDrive - Politecnico di Milano\PhD\lezioni_che_faccio_io\ECT_laboratorio\2_4_25_che_faccio_io\Lab 2\code> & C:/Users/lucaj/AppData/Local/Programs/Python/Python313/Python.exe "c:/Users/lucaj/OneDrive - Politecnico di Milano/PhD/lezioni_che_faccio_io/24_25_che_faccio_io/Lab 2/code/codice_per_slide/udp_server.py"
```

Il server è funzionante!
Messaggio ricevuto dal client: gfh
Ho risposto al client con: GFH
Messaggio ricevuto dal client: prova per le slide

The context menu for the terminal is open, showing options to split, move, or change the terminal. The **Kill Terminal** option is highlighted with a red box and a red arrow pointing to it from the text box.



Esercizio 2.1 (5 Min)

Modificare il server in modo che scriva l'indirizzo IP e numero di porta del client

- Lasciamo il server in esecuzione...
- Eseguiamo il client più volte

➤ **Cosa succede ai numeri di porta?**



Gestione degli errori: Problematiche

- Cosa accade se inviamo dati (e aspettiamo dati) a un server inesistente?
1. **CLIENT**: modifichiamo la porta destinazione in 13000 (*nessun processo è in ascolto su tale porta*).
 2. **Il server** dovrebbe inviare un errore, o non rispondere affatto
 3. L'errore potrebbe perdersi o arrivare quando il client è bloccato sulla `socket.recvfrom`

Come gestire gli errori?



Gestione degli errori: Soluzioni

Due possibili risultati dal server:

1. Il messaggio di errore dal server diventa **un'eccezione Python**
2. Il client aspetta all'infinito un messaggio dal server

Gestione degli errori

1. Impostare un timeout alle operazioni sui socket
2. Catturare le eccezioni Python
 - a) Scadenza timeout
 - b) Messaggi di errore



Impostare un timeout

- Si usa il comando:

```
clientSocket.setTimeout(<timeout>)
```

- Può essere inserito ovunque nel codice del processo client **prima** della chiamata a `recvfrom`.
- `<timeout>` è un valore decimale espresso in secondi.



Catturare le eccezioni: Try - Except

- Gli errori (es. *destinazione non raggiungibile e timeout scaduti*) lanciano delle eccezioni.
- Per catturare l'eccezione racchiudere il codice che può lanciare l'eccezione in un blocco ***try ... except***

```
try:
    mod_message, server_address = client_socket.recvfrom(2048)
    mod_message = mod_message.decode('utf-8')
    print(mod_message)
except:
    print("Timeout scaduto! Server irraggiungibile")
finally:
    client_socket.close()
```



Extra: differenziare le eccezioni

```
28  try:
29      # si mette in ascolto per la risposta...
30      server_response, address = client_socket.recvfrom(buffer_size)
31      print("Risposta del server: ", server_response.decode("utf-8"))
32
33  except ConnectionResetError:
34      print("Server non raggiungibile!")
35  except TimeoutError:
36      print("Timeout scaduto!")
37  except Exception as ex:
38      template = "Si è verificata un'eccezione del tipo {0}. Gli argomenti sono:\n{1!r}"
39      messaggio_errore = template.format(type(ex).__name__, ex.args)
40      print(messaggio_errore)
41  finally: # il ramo finally verrà SEMPRE ESEGUITO!
42      # chiudo il socket
43      client_socket.close()
```



Esercizio 2.2 (5 min)

Scrivere un nuovo client UDP che gestisca l'errore nel caso di server inesistente.



Esercizio 2.3 (15 min)

Scrivere un'applicazione client-server UDP per contare il numero di consonanti presenti in una stringa.

- Client chiede all'utente di inserire una stringa, che viene inviata al server.
- 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:

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



Esercizio 2.4 (Dopo Lab)

Scrivere un'applicazione client-server UDP dove il client invia un numero e il server risponde **indicando se è primo o no**

Tutto bello ma cos'è un **numero primo**?

https://it.wikipedia.org/wiki/Numero_primo

<https://www.youmath.it/lezioni/algebra-elementare/lezioni-di-algebra-e-aritmetica-per-scuole-medie/1770-numeri-primi.html>

- Il client chiede all'utente il numero da inviare al server
- Usare indirizzi IPv4
- Timeout in ricezione (lato client): **2 secondi**
- Lunghezza buffer ricezione: **2048 byte**

