Python per il networking

Perché Python per le reti?

Se studi Sistemi e Reti, Python è il tuo migliore amico. Perché? Perché ti permette di fare in **10 righe di codice** quello che in altri linguaggi richiede 100. Vuoi fare un ping? Creare un server? Analizzare pacchetti? Python ha una libreria per tutto.

In questa sezione vedremo come usare Python per le operazioni di rete più comuni: dai socket alla comunicazione client-server, passando per le richieste HTTP e l'analisi degli indirizzi IP.

Indirizzi IP e hostname

Prima di tutto, impariamo a lavorare con gli **indirizzi IP**. Il modulo `socket` è il coltellino svizzero del networking in Python:

```
import socket

# Il tuo hostname (il nome del tuo computer sulla rete)
hostname = socket.gethostname()
print(f"Hostname: {hostname}")

# Risolvere un hostname in indirizzo IP (DNS lookup)
try:
    ip = socket.gethostbyname("www.google.com")
    print(f"IP di www.google.com: {ip}")
except socket.gaierror:
    print("Impossibile risolvere l'hostname")
```

Validare indirizzi IP

```
import ipaddress

# Validare un indirizzo IPv4
try:
    ip = ipaddress.ip_address("192.168.1.1")
    print(f"{ip} è un indirizzo {ip.version} valido")
    print(f"È privato? {ip.is_private}")
    print(f"È di loopback? {ip.is_loopback}")
except ValueError:
    print("Indirizzo IP non valido!")

# Lavorare con le reti (subnet)
rete = ipaddress.ip_network("192.168.1.0/24")
print(f"
Rete: {rete}")
print(f"Netmask: {rete.netmask}")
print(f"Broadcast: {rete.broadcast_address}")
print(f"Numero di host: {rete.num_addresses - 2}")  # Tolgo rete e broad
print(f"Primi 5 host: {list(rete.hosts())[:5]}")
```

Calcoli sulle sottoreti

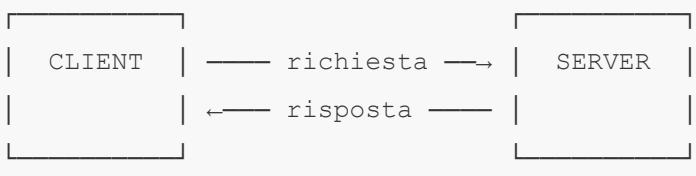
Roba che serve DAVVERO a Sistemi e Reti! Calcolare subnet, netmask, broadcast... tutto con Python:

```
import ipaddress

# Dato un IP con prefisso, calcola tutto
indirizzo = ipaddress.ip_interface("192.168.10.50/24")

print(f"Indirizzo: {indirizzo.ip}")
print(f"Rete: {indirizzo.network}")
print(f"Netmask: {indirizzo.netmask}")
print(f"Broadcast: {indirizzo.network.broadcast_address}")

# Verificare se un IP appartiene a una rete
rete = ipaddress.ip_network("10.0.0.0/8")
ip_test = ipaddress.ip_address("10.25.3.100")
print(f"
{ip_test} appartiene a {rete}? {ip_test in rete}")

ip_test2 = ipaddress.ip_address("192.168.1.1")
print(f"{ip_test2} appartiene a {rete}? {ip_test2 in rete}")
```

I Socket: la base di tutto

Un **socket** è un "punto di comunicazione" tra due programmi sulla rete. Pensa a una presa telefonica: serve per collegarsi e parlare con qualcuno dall'altra parte. Tutta la comunicazione di rete passa dai socket!

Come funziona la comunicazione



1. Il **server** si mette in ascolto su una porta (tipo un negozio che apre)
2. Il **client** si collega al server (tipo un cliente che entra)
3. Si scambiano messaggi
4. Uno dei due chiude la connessione

Server TCP (il negoziante)

Questo codice crea un server che aspetta connessioni. **Non funziona nel browser** (serve eseguirlo in locale con Python), ma è fondamentale capirlo:

```
import socket

# Crea il socket TCP
server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Associa il socket a un indirizzo e una porta
server.bind(("localhost", 12345))

# Mettiti in ascolto (max 5 connessioni in coda)
server.listen(5)
print("Server in ascolto sulla porta 12345...")

while True:
    # Accetta una connessione (si blocca finché non arriva qualcuno)
    client_socket, indirizzo = server.accept()
    print(f"Connessione da {indirizzo}")

    # Ricevi dati dal client
    dati = client_socket.recv(1024).decode("utf-8")
    print(f"Ricevuto: {dati}")

    # Rispondi al client
    risposta = f"Ciao! Hai detto: {dati}"
    client_socket.send(risposta.encode("utf-8"))

    # Chiudi la connessione con questo client
    client_socket.close()
```

Client TCP (il cliente)

```
import socket

# Crea il socket
client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)

# Connettiti al server
client.connect(("localhost", 12345))

# Invia un messaggio
messaggio = "Ciao server!"
client.send(messaggio.encode("utf-8"))

# Ricevi la risposta
risposta = client.recv(1024).decode("utf-8")
print(f"Risposta dal server: {risposta}")

# Chiudi
client.close()
```

!!! tip "Per provare"

```
Apri **due terminali**: nel primo avvia il server, nel secondo avvia il
```

TCP vs UDP

Due protocolli di trasporto, due filosofie diverse:

Caratteristica	TCP	UDP
Connessione	Sì (handshake)	No
Affidabilità	Garantita (ritrasmette i pacchetti persi)	Non garantita
Ordine	I dati arrivano in ordine	Possono arrivare fuori ordine
Velocità	Più lento	Più veloce
Uso tipico	Web, email, file transfer	Gaming, streaming, DNS

Server UDP (il postino veloce)

```
import socket

# Socket UDP (SOCK_DGRAM invece di SOCK_STREAM)
server = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
server.bind(("localhost", 12345))

print("Server UDP in ascolto sulla porta 12345...")

while True:
    # Ricevi dati (con UDP non serve accept!)
    dati, indirizzo = server.recvfrom(1024)
    messaggio = dati.decode("utf-8")
    print(f"Da {indirizzo}: {messaggio}")

    # Rispondi
    risposta = f"Ricevuto: {messaggio}"
    server.sendto(risposta.encode("utf-8"), indirizzo)
```

Client UDP

```
import socket

client = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)

messaggio = "Ciao via UDP!"
client.sendto(messaggio.encode("utf-8"), ("localhost", 12345))

risposta, server = client.recvfrom(1024)
print(f"Risposta: {risposta.decode('utf-8')}")

client.close()
```

Richieste HTTP con `requests`

Il modulo `requests` è il modo più semplice per fare richieste HTTP (quelle che fa il browser quando visiti un sito). Devi installarlo con `pip install requests`:

```
import requests

# GET — Scaricare una pagina o dei dati
risposta = requests.get("https://api.github.com")

print(f"Status code: {risposta.status_code}")  # 200 = tutto ok!
print(f"Content-Type: {risposta.headers['Content-Type']}")

# Il contenuto come JSON (dizionario Python!)
dati = risposta.json()
print(f"URL corrente: {dati['current_user_url']}")
```

Le API REST

Le **API** sono tipo i camerieri dei servizi web: tu fai una richiesta, loro ti portano i dati. Ecco i metodi principali:

Metodo	Azione	Esempio
`GET`	Leggere dati	Ottieni la lista utenti
`POST`	Creare dati	Crea un nuovo utente
`PUT`	Aggiornare dati	Modifica un utente
`DELETE`	Cancellare dati	Elimina un utente

```
import requests

# GET — leggere dati
risposta = requests.get("https://jsonplaceholder.typicode.com/posts/1")
post = risposta.json()
print(f"Titolo: {post['title']}")

# POST — inviare dati
nuovo_post = {
    "title": "Il mio post",
    "body": "Ciao dal mio programma Python!",
    "userId": 1
}
risposta = requests.post(
    "https://jsonplaceholder.typicode.com/posts",
    json=nuovo_post
)
print(f"Creato con status: {risposta.status_code}")
print(f"ID assegnato: {risposta.json()['id']}")
```

Scansione delle porte

Un classico del networking! Controllare quali porte sono aperte su un host. Attenzione: fallo **solo sui tuoi dispositivi** o con autorizzazione!

```
import socket

def scansione_porte(host, porte):
    print(f"Scansione di {host}...")
    aperte = []

    for porta in porte:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(1)  # Timeout di 1 secondo

        risultato = sock.connect_ex((host, porta))
        if risultato == 0:
            print(f"  Porta {porta}: APERTA")
            aperte.append(porta)

        sock.close()

    return aperte

# Scansiona le porte più comuni
porte_comuni = [21, 22, 25, 53, 80, 110, 143, 443, 3306, 8080]
aperte = scansione_porte("localhost", porte_comuni)
print(f"
Porte aperte trovate: {aperte}")
```

!!! danger "Aspetti etici e legali!"

```
La scansione delle porte su computer altrui senza permesso è **illegale*

- Testare la TUA rete
- Esercitazioni in laboratorio
- CTF (Capture The Flag) e ambienti di test
```

Chat client-server

Mettiamo tutto insieme e creiamo una **mini chat**! Il server gestisce i messaggi, il client li invia e riceve.

Server della chat

```
import socket
import threading

server = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
server.bind(("localhost", 55555))
server.listen()

clienti = []
nomi = []

def broadcast(messaggio):
    """Invia un messaggio a tutti i client connessi"""
    for client in clienti:
        client.send(messaggio)

def gestisci_client(client):
    """Gestisce i messaggi di un singolo client"""
    while True:
        try:
            messaggio = client.recv(1024)
            broadcast(messaggio)
        except:
            indice = clienti.index(client)
            clienti.remove(client)
            client.close()
            nome = nomi[indice]
            nomi.remove(nome)
            broadcast(f"{nome} ha lasciato la chat!".encode("utf-8"))
            break

def ricevi_connessioni():
    """Accetta nuove connessioni"""
    while True:
        client, indirizzo = server.accept()
        print(f"Connessione da {indirizzo}")

        client.send("NOME".encode("utf-8"))
```

```
        nome = client.recv(1024).decode("utf-8")
        nomi.append(nome)
        clienti.append(client)

        print(f"Nome: {nome}")
        broadcast(f"{nome} è entrato nella chat!".encode("utf-8"))
        client.send("Connesso al server!".encode("utf-8"))

        thread = threading.Thread(target=gestisci_client, args=(client,)
        thread.start()

print("Server chat in ascolto...")
ricevi_connessioni()
```

Client della chat

```
import socket
import threading

client = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
client.connect(("localhost", 55555))

nome = input("Inserisci il tuo nome: ")

def ricevi():
    """Riceve messaggi dal server"""
    while True:
        try:
            messaggio = client.recv(1024).decode("utf-8")
            if messaggio == "NOME":
                client.send(nome.encode("utf-8"))
            else:
                print(messaggio)
        except:
            print("Errore di connessione!")
            client.close()
            break

def scrivi():
    """Invia messaggi al server"""
    while True:
        messaggio = f"{nome}: {input('')}"
        client.send(messaggio.encode("utf-8"))

# Avvia i thread per ricevere e inviare contemporaneamente
thread_ricevi = threading.Thread(target=ricevi)
thread_ricevi.start()

thread_scrivi = threading.Thread(target=scrivi)
thread_scrivi.start()
```

!!! tip "Threading? Cos'è?"

```
Il `threading` permette di fare **più cose contemporaneamente**. Nella c
```

Protocolli di rete in Python

Ecco un riepilogo dei moduli Python più utili per il networking:

Modulo	Uso	Livello
`socket`	Comunicazione di basso livello (TCP/UDP)	Trasporto
`requests`	Richieste HTTP (client web)	Applicazione
`http.server`	Server HTTP semplice	Applicazione
`ipaddress`	Manipolazione indirizzi IP e reti	Rete
`ssl`	Connessioni sicure (HTTPS)	Trasporto
`smtplib`	Invio email	Applicazione
`ftplib`	Trasferimento file (FTP)	Applicazione

Server HTTP in una riga!

Vuoi condividere file sulla rete locale? Python ha un server HTTP integrato:

```
# Da terminale, nella cartella che vuoi condividere:
# python -m http.server 8080
#
# Poi apri il browser su http://localhost:8080 e vedi i file!
```

Esercizi

Esercizio 1: Calcolatore di subnet

Scrivi un programma che, dato un indirizzo IP con prefisso (es. "192.168.1.0/24"), stampa: indirizzo di rete, netmask, broadcast, numero di host disponibili e il range degli indirizzi.

```
import ipaddress

ip_input = input("Inserisci IP/prefisso (es. 192.168.1.0/24): ")

# Calcola e stampa tutte le informazioni della rete
```

Esercizio 2: Client-Server echo

Crea un server TCP che risponde a ogni messaggio ripetendolo al contrario (echo invertito). Il client invia una frase, il server la restituisce capovolta.

```
# server_echo.py
import socket

# Crea il server che inverte i messaggi ricevuti
```

```
# client_echo.py
import socket

# Connettiti al server e invia messaggi
```

Esercizio 3: Verifica porte

Scrivi un programma che verifica se una lista di servizi comuni (HTTP, HTTPS, SSH, FTP, DNS) è raggiungibile su un host specificato dall'utente. Stampa il risultato in modo ordinato.

```
import socket

servizi = {
    "FTP": 21,
    "SSH": 22,
    "DNS": 53,
    "HTTP": 80,
    "HTTPS": 443
}

host = input("Inserisci l'hostname: ")

# Verifica ogni porta e stampa il risultato
```