

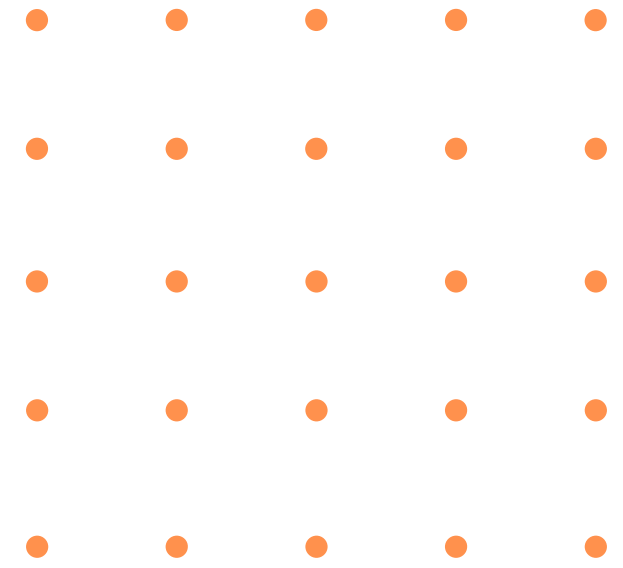
Build Week – 1

REQUEST HTTP - PORT SCANNING

Indice

1 REQUEST HTTP

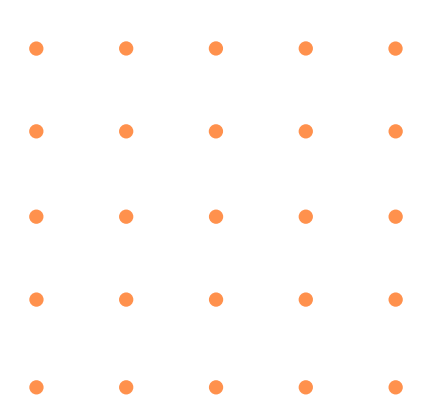
2 PORT SCANNER





COS' È UNA ENUMERAZIONE DI METODI HTTP?

Un'enumerazione di metodi HTTP rappresenta un elenco standardizzato di azioni che possono essere eseguite su risorse web. I principali metodi includono GET (richiede dati), POST (invia dati al server), PUT (aggiorna una risorsa), DELETE (elimina una risorsa) e altri. Ogni metodo ha uno scopo specifico nell'interazione tra client e server su Internet.



```

import http.client

host = input("Metti l'IP: ")
port = input("Metti la porta: ")

if port == "":
    port = 80

try:
    connection = http.client.HTTPConnection(host, port, timeout=5)
    metodi = ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"]
    metodi_SUPP = []

    for metodo in metodi:
        try:
            connection.request(metodo, "/")
            response = connection.getresponse()
            if response.status == 200:
                metodi_SUPP.append(metodo)
        except ConnectionRefusedError:
            print(f"Connessione fallita per il metodo {metodo}")
        except http.client.HTTPException as e:
            print(f"Errore HTTP durante la richiesta {metodo}: {e}")

    connection.close()

except ConnectionRefusedError:
    print("Connessione fallita")
except Exception as e:
    print(f"Errore generico: {e}")
print("Metodi supportati:", metodi_SUPP)

```

METODI HTTP blocco 1

Inizialmente, si importa la libreria **http.client** per agevolare la comunicazione con il server.

Successivamente, si implementano due input per specificare l'**HOST** (indirizzo IP) e la **PORT** (numero di porta) desiderati.

Nel caso in cui l'input per la porta rimanga vuoto, un'istruzione condizionale (**if**) assegnerà automaticamente la porta 80 come valore predefinito.

Ciò fornisce una flessibilità all'utente, consentendo di specificare l'indirizzo IP del server e, se necessario, di personalizzare la porta di connessione. L'utilizzo di input interattivi agevola la configurazione dinamica della connessione al server.

```

import http.client

host = input("Mettili l'IP: ")
port = input("Mettili la porta: ")

if port == "":
    port = 80

try:
    connection = http.client.HTTPConnection(host, port, timeout=5)
    metodi = ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"]
    metodi_SUPP = []

    for metodo in metodi:
        try:
            connection.request(metodo, "/")
            response = connection.getresponse()
            if response.status == 200:
                metodi_SUPP.append(metodo)
        except ConnectionRefusedError:
            print(f"Connessione fallita per il metodo {metodo}")
        except http.client.HTTPException as e:
            print(f"Errore HTTP durante la richiesta {metodo}: {e}")

    connection.close()

except ConnectionRefusedError:
    print("Connessione fallita")
except Exception as e:
    print(f"Errore generico: {e}")
print("Metodi supportati:", metodi_SUPP)

```

METODI HTTP blocco 2

Vengono create tre variabili significative per gestire la comunicazione HTTP:

1. **connection**: Questa variabile è utilizzata per definire il target della connessione, accettando come parametri l'**HOST**, la **PORT** e specificando un **timeout** di risposta pari a 5 secondi. Essenzialmente, stabilisce la connessione con il server specificato, fornendo un limite di tempo entro il quale attenderà una risposta.
2. **metodi**: All'interno di questa variabile, vengono definiti tutti i verbi HTTP che possono essere utilizzati nelle richieste. Include i metodi standard come GET, POST, PUT, PATCH, DELETE, OPTIONS, HEAD, e altri. Questa variabile fornisce un set completo di azioni HTTP che possono essere eseguite durante la comunicazione con il server.
3. **metodi_SUPP**: In questa variabile vengono memorizzati i metodi HTTP che sono accettati dal server. In pratica, durante la comunicazione con il server, questa variabile conterrà l'elenco dei metodi supportati dal server, consentendo una gestione dinamica e informativa delle capacità del server rispetto alle richieste HTTP.

L'uso di queste variabili facilita la gestione e la comprensione del codice, contribuendo a una struttura chiara e modulare durante la comunicazione con il server attraverso il protocollo HTTP.

METODI HTTP blocco 3

Utilizzando un ciclo for, si itera attraverso tutti i verbi HTTP contenuti nella variabile metodi. All'interno di un blocco try, si prova ad eseguire il codice corrispondente a ciascun verbo. Se il server risponde con un codice di stato HTTP 200, il verbo viene aggiunto al contenitore metodi_SUPP, indicando che è supportato dal server. Il ciclo continua per verificare gli altri verbi nell'elenco.

Nel caso in cui il metodo non sia supportato, viene gestito un blocco except, che avvisa che il metodo non è supportato. Inoltre, è possibile gestire eventuali errori durante la richiesta, fornendo informazioni dettagliate sull'errore effettivo. Questo consente di ottenere un feedback completo sulle capacità del server e di gestire eventuali problemi che potrebbero sorgere durante le richieste.

Alla fine del processo, la connessione viene chiusa.

```
import http.client

host = input("Metti l'IP: ")
port = input("Metti la porta: ")

if port == "":
    port = 80

try:
    connection = http.client.HTTPConnection(host, port, timeout=5)
    metodi = ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"]
    metodi_SUPP = []

    for metodo in metodi:
        try:
            connection.request(metodo, "/")
            response = connection.getresponse()
            if response.status == 200:
                metodi_SUPP.append(metodo)
        except ConnectionRefusedError:
            print(f"Connessione fallita per il metodo {metodo}")
        except http.client.HTTPException as e:
            print(f"Errore HTTP durante la richiesta {metodo}: {e}")

    connection.close()

except ConnectionRefusedError:
    print("Connessione fallita")
except Exception as e:
    print(f"Errore generico: {e}")
print("Metodi supportati:", metodi_SUPP)
```

```

import http.client

host = input("Metti l'IP: ")
port = input("Metti la porta: ")

if port == "":
    port = 80

try:
    connection = http.client.HTTPConnection(host, port, timeout=5)
    metodi = ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"]
    metodi_SUPP = []

    for metodo in metodi:
        try:
            connection.request(metodo, "/")
            response = connection.getresponse()
            if response.status == 200:
                metodi_SUPP.append(metodo)
        except ConnectionRefusedError:
            print(f"Connessione fallita per il metodo {metodo}")
        except http.client.HTTPException as e:
            print(f"Errore HTTP durante la richiesta {metodo}: {e}")

    connection.close()

```

```

except ConnectionRefusedError:
    print("Connessione fallita")
except Exception as e:
    print(f"Errore generico: {e}")
print("Metodi supportati:", metodi_SUPP)

```

METODI HTTP blocco 4

Per affrontare eventuali fallimenti della connessione, vengono aggiunti due blocchi **except** per gestire i casi in cui si verifica un errore durante la connessione. Questi blocchi forniscono informazioni dettagliate sul fallimento della connessione.

Alla fine del processo, viene stampato in modo ordinato l'elenco dei metodi supportati. Questa informazione è utile per avere un resoconto chiaro delle capacità del server in termini di metodi HTTP supportati.

La struttura del codice assicura che, anche in presenza di errori di connessione, l'esecuzione del programma continui a fornire informazioni utili e che la chiusura della connessione sia gestita adeguatamente.

```

import http.client

host = input("Metti l'IP: ")
port = input("Metti la porta: ")

if port == "":
    port = 80

try:
    connection = http.client.HTTPConnection(host, port, timeout=5)
    metodi = ["GET", "POST", "PUT", "PATCH", "DELETE", "OPTIONS", "HEAD"]
    metodi_SUPP = []

    for metodo in metodi:
        try:
            connection.request(metodo, "/")
            response = connection.getresponse()
            if response.status == 200:
                metodi_SUPP.append(metodo)
        except ConnectionRefusedError:
            print(f"Connessione fallita per il metodo {metodo}")
        except http.client.HTTPException as e:
            print(f"Errore HTTP durante la richiesta {metodo}: {e}")

    connection.close()

except ConnectionRefusedError:
    print("Connessione fallita")
except Exception as e:
    print(f"Errore generico: {e}")
print("Metodi supportati:", metodi_SUPP)

```

METODI HTTP finale

L'intero blocco di codice è racchiuso in un blocco **try** per garantire che tutte le condizioni siano verificate prima dell'esecuzione del codice principale. Questo approccio contribuisce a conferire maggiore robustezza al programma, consentendo di gestire potenziali problemi o errori fin dall'inizio.

In caso di condizioni non soddisfatte, il blocco **except** fornisce dettagliate informazioni sull'errore riscontrato, offrendo un resoconto accurato degli eventuali problemi e facilitando la risoluzione di eventuali criticità.

L'utilizzo di questo costrutto **try-except** migliora la sicurezza e l'affidabilità del programma, in quanto fornisce una gestione preventiva degli errori, contribuendo a mantenere una robusta esecuzione del codice anche in presenza di situazioni impreviste.

OUTPUT

```
Metti l'IP: 80.211.132.161
```

```
Metti la porta: 80
```

```
Errore HTTP durante la richiesta POST: Request-sent
```

```
Errore HTTP durante la richiesta PUT: Request-sent
```

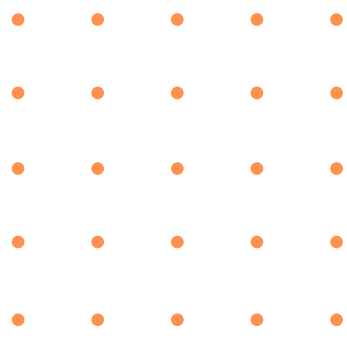
```
Errore HTTP durante la richiesta PATCH: Request-sent
```

```
Errore HTTP durante la richiesta DELETE: Request-sent
```

```
Errore HTTP durante la richiesta OPTIONS: Request-sent
```

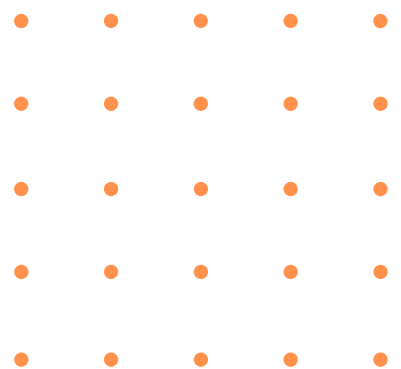
```
Errore HTTP durante la richiesta HEAD: Request-sent
```

```
Metodi supportati: ['GET']
```



COS'È UN PORT SCANNER

Un port scanner è uno strumento informatico utilizzato per individuare e analizzare lo stato delle porte di un dispositivo o di una rete. Esplora le porte di un sistema per identificare quelle aperte, chiuse o in ascolto. Questa analisi è cruciale per valutare la sicurezza di un sistema, individuare possibili vulnerabilità e comprendere quali servizi siano in esecuzione su un server.



PORT SCANNER blocco 1

Vengono importate due librerie:

Socket = facilita la creazione di una connessione

concurrent.futures = esegue più azioni in parallelo

1. **creazione del socket:** Questo codice Python utilizza la libreria socket per creare un socket TCP IPv4. La struttura "with" garantisce la corretta gestione delle risorse, assicurando la chiusura automatica del socket al termine del blocco.
2. **s.settimeout(0.5):** Viene impostato un timeout di 0.5 secondi sulla connessione.
3. **status:** Si utilizza per tentare di stabilire una connessione sulla porta specificata del target specificato.
4. **Return:** La funzione restituisce una tupla contenente il numero di porta (port) e lo stato della connessione (status).

```
import socket
import concurrent.futures

def scanPort(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(0.5)
        status = s.connect_ex((target, port))
        return port, status

target = input("metti un ip: ")
portRange = input("metti un range di porte (ES 1-100): ")
lowPort = int(portRange.split("-")[0])
highPort = int(portRange.split("-")[1])

print("Scanning host: ", target, "da porta", lowPort, "alla", highPort)
print("Questa operazione potrebbe richiedere alcuni minuti")
with concurrent.futures.ThreadPoolExecutor(max_workers=30) as esecutore:
    result = list(esecutore.map(scanPort, range(lowPort, highPort + 1)))

PorteAperte = []
PorteChiuse = []
PorteServScono = []

for port, status in result:
    try:
        if status == 0:
            nomeServ = socket.getservbyport(port)
            PorteAperte.append((port, nomeServ))
        else:
            PorteChiuse.append(port)
    except OSError:
        PorteServScono.append(port)

print("-----CHIUSE-----")
print(PorteChiuse)
print("-----PORTE NOME?-----")
print(PorteServScono)
print("-----APERTE-----")
for porta, servizio in PorteAperte:
    print(f"Porta: {porta} -- Servizio: {servizio}")
```

PORT SCANNER

blocco 2

1. **target:** Qui viene richiesto all'utente di inserire un indirizzo IP.
2. **portRange:** Qui viene richiesto all'utente di inserire un intervallo di porte. L'utente deve inserire l'intervallo nel formato "inizio-fine" (ad esempio, "1-100"). Il valore inserito viene memorizzato nella variabile portRange.
3. **lowPort e highPort:** implementa la logica per spartirsi il range di porte

In sintesi, questa parte di codice permette all'utente di specificare un indirizzo IP e un intervallo di porte.

```
import socket
import concurrent.futures

def scanPort(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(0.5)
        status = s.connect_ex((target, port))
    return port, status

target = input("metti un ip: ")
portRange = input("metti un range di porte (ES 1-100): ")
lowPort = int(portRange.split("-")[0])
highPort = int(portRange.split("-")[1])

print("Scanning host: ", target, "da porta", lowPort, "alla", highPort)
print("Questa operazione potrebbe richiedere alcuni minuti")
with concurrent.futures.ThreadPoolExecutor(max_workers=30) as esecutore:
    result = list(esecutore.map(scanPort, range(lowPort, highPort + 1)))

PorteAperte = []
PorteChiuse = []
PorteServScono = []

for port, status in result:
    try:
        if status == 0:
            nomeServ = socket.getservbyport(port)
            PorteAperte.append((port, nomeServ))
        else:
            PorteChiuse.append(port)
    except OSError:
        PorteServScono.append(port)

print("-----CHIUSE-----")
print(PorteChiuse)
print("-----PORTE NOME?-----")
print(PorteServScono)
print("-----APERTE-----")
for porta, servizio in PorteAperte:
    print(f"Porta: {porta} -- Servizio: {servizio}")
```

PORT SCANNER blocco 3

1. **creazione di ThreadPoolExecutor:** Viene creato un oggetto dal modulo **concurrent.futures**, il quale gestirà l'esecuzione di funzioni in parallelo.
2. **result:** Utilizza la funzione map per applicare scanPort a ciascuna porta nell'intervallo specificato, I risultati delle scansioni vengono raccolti in una lista

```
import socket
import concurrent.futures

def scanPort(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(0.5)
        status = s.connect_ex((target, port))
    return port, status

target = input("metti un ip: ")
portRange = input("metti un range di porte (ES 1-100): ")
lowPort = int(portRange.split("-")[0])
highPort = int(portRange.split("-")[1])

print("Scanning host: ", target, "da porta", lowPort, "alla", highPort)
print("Questa operazione potrebbe richiedere alcuni minuti")
with concurrent.futures.ThreadPoolExecutor(max_workers=30) as esecutore:
    result = list(esecutore.map(scanPort, range(lowPort, highPort + 1)))

PorteAperte = []
PorteChiuse = []
PorteServScono = []

for port, status in result:
    try:
        if status == 0:
            nomeServ = socket.getservbyport(port)
            PorteAperte.append((port, nomeServ))
        else:
            PorteChiuse.append(port)
    except OSError:
        PorteServScono.append(port)

print("-----CHIUSE-----")
print(PorteChiuse)
print("-----PORTE NOME?-----")
print(PorteServScono)
print("-----APERTE-----")
for porta, servizio in PorteAperte:
    print(f"Porta: {porta} -- Servizio: {servizio}")
```

PORT SCANNER blocco 4

Inizialmente, vengono create tre array vuoti: PorteAperte, PorteChiuse e PorteServScono, pronte per ospitare i risultati della scansione delle porte.

Successivamente, attraverso un ciclo che itera sui risultati ottenuti dalla scansione delle porte, il codice esamina ogni tupla composta da un numero di porta (port) e uno stato (status). Lo stato 0 indica che la porta è aperta.

Se la porta è aperta (status == 0), il codice utilizza la funzione **getservbyport** del modulo socket per ottenere il nome del servizio associato a quella porta. Questo nome del servizio viene quindi aggiunto alla lista **PorteAperte** insieme al numero di porta.

Se la porta non è aperta (status != 0), il numero di porta viene aggiunto alla lista **PorteChiuse**.

In caso di un'eccezione di tipo **OSError**, che può verificarsi se non è possibile ottenere il nome del servizio associato alla porta, il numero di porta viene aggiunto alla lista **PorteServScono**.

```
import socket
import concurrent.futures

def scanPort(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(0.5)
        status = s.connect_ex((target, port))
    return port, status

target = input("metti un ip: ")
portRange = input("metti un range di porte (ES 1-100): ")
lowPort = int(portRange.split("-")[0])
highPort = int(portRange.split("-")[1])

print("Scanning host: ", target, "da porta", lowPort, "alla", highPort)
print("Questa operazione potrebbe richiedere alcuni minuti")
with concurrent.futures.ThreadPoolExecutor(max_workers=30) as esecutore:
    result = list(esecutore.map(scanPort, range(lowPort, highPort + 1)))
```

```
PorteAperte = []
PorteChiuse = []
PorteServScono = []

for port, status in result:
    try:
        if status == 0:
            nomeServ = socket.getservbyport(port)
            PorteAperte.append((port, nomeServ))
        else:
            PorteChiuse.append(port)
    except OSError:
        PorteServScono.append(port)
```

```
print("-----CHIUSE-----")
print(PorteChiuse)
print("-----PORTE NOME?-----")
print(PorteServScono)
print("-----APERTE-----")
for porta, servizio in PorteAperte:
    print(f"Porta: {porta} -- Servizio: {servizio}")
```

PORT SCANNER blocco 5

Stampa i risultati dividendoli in diverse sezioni: una dedicata alla lista di porte chiuse e quindi non disponibili, un'altra che indica le porte aperte e i servizi forniti da ognuna di essa.

```
import socket
import concurrent.futures

def scanPort(port):
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as s:
        s.settimeout(0.5)
        status = s.connect_ex((target, port))
    return port, status

target = input("metti un ip: ")
portRange = input("metti un range di porte (ES 1-100): ")
lowPort = int(portRange.split("-")[0])
highPort = int(portRange.split("-")[1])

print("Scanning host: ", target, "da porta", lowPort, "alla", highPort)
print("Questa operazione potrebbe richiedere alcuni minuti")
with concurrent.futures.ThreadPoolExecutor(max_workers=30) as esecutore:
    result = list(esecutore.map(scanPort, range(lowPort, highPort + 1)))

PorteAperte = []
PorteChiuse = []
PorteServScono = []

for port, status in result:
    try:
        if status == 0:
            nomeServ = socket.getservbyport(port)
            PorteAperte.append((port, nomeServ))
        else:
            PorteChiuse.append(port)
    except OSError:
        PorteServScono.append(port)

print("-----CHIUSE-----")
print(PorteChiuse)
print("-----PORTE NOME?-----")
print(PorteServScono)
print("-----APERTE-----")
for porta, servizio in PorteAperte:
    print(f"Porta: {porta} -- Servizio: {servizio}")
```


OUTPUT

```
metti un ip: 80.211.132.161
metti un range di porte (ES 1-100): 10-800
Scanning host: 80.211.132.161 da porta 10 alla 800
Questa operazione potrebbe richiedere alcuni minuti
-----CHIUSE-----
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63
, 64, 65, 66, 67, 68, 69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 81, 82, 83, 84, 85, 86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99, 100, 101, 102, 103, 104, 105, 106, 107, 108, 109, 110, 111, 112, 113,
114, 115, 116, 117, 118, 119, 120, 121, 122, 123, 124, 125, 126, 127, 128, 129, 130, 131, 132, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144, 145, 146, 147, 148, 149, 150, 151, 152, 153, 154, 155,
156, 157, 158, 159, 160, 161, 162, 163, 164, 165, 166, 167, 168, 169, 170, 171, 172, 173, 174, 175, 176, 177, 178, 179, 180, 181, 182, 183, 184, 185, 186, 187, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 1
98, 199, 200, 201, 202, 203, 204, 205, 206, 207, 208, 209, 210, 211, 212, 213, 214, 215, 216, 217, 218, 219, 220, 221, 222, 223, 224, 225, 226, 227, 228, 229, 230, 231, 232, 233, 234, 235, 236, 237, 238, 239, 24
0, 241, 242, 243, 244, 245, 246, 247, 248, 249, 250, 251, 252, 253, 254, 255, 256, 257, 258, 259, 260, 261, 262, 263, 264, 265, 266, 267, 268, 269, 270, 271, 272, 273, 274, 275, 276, 277, 278, 279, 280, 281, 282
, 283, 284, 285, 286, 287, 288, 289, 290, 291, 292, 293, 294, 295, 296, 297, 298, 299, 300, 301, 302, 303, 304, 305, 306, 307, 308, 309, 310, 311, 312, 313, 314, 315, 316, 317, 318, 319, 320, 321, 322, 323, 324,
325, 326, 327, 328, 329, 330, 331, 332, 333, 334, 335, 336, 337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349, 350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362, 363, 364, 365, 366,
367, 368, 369, 370, 371, 372, 373, 374, 375, 376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388, 389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400, 401, 402, 403, 404, 405, 406, 407, 408, 4
09, 410, 411, 412, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430, 431, 432, 433, 434, 435, 436, 437, 438, 439, 440, 441, 442, 444, 445, 446, 447, 448, 449, 450, 451, 45
2, 453, 454, 455, 456, 457, 458, 459, 460, 461, 462, 463, 464, 465, 466, 467, 468, 469, 470, 471, 472, 473, 474, 475, 476, 477, 478, 479, 480, 481, 482, 483, 484, 485, 486, 487, 488, 489, 490, 491, 492, 493, 494
, 495, 496, 497, 498, 499, 500, 501, 502, 503, 504, 505, 506, 507, 508, 509, 510, 511, 512, 513, 514, 515, 516, 517, 518, 519, 520, 521, 522, 523, 524, 525, 526, 527, 528, 529, 530, 531, 532, 533, 534, 535, 536,
537, 538, 539, 540, 541, 542, 543, 544, 545, 546, 547, 548, 549, 550, 551, 552, 553, 554, 555, 556, 557, 558, 559, 560, 561, 562, 563, 564, 565, 566, 567, 568, 569, 570, 571, 572, 573, 574, 575, 576, 577, 578,
579, 580, 581, 582, 583, 584, 585, 586, 587, 588, 589, 590, 591, 592, 593, 594, 595, 596, 597, 598, 599, 600, 601, 602, 603, 604, 605, 606, 607, 608, 609, 610, 611, 612, 613, 614, 615, 616, 617, 618, 619, 620, 6
21, 622, 623, 624, 625, 626, 627, 628, 629, 630, 631, 632, 633, 634, 635, 636, 637, 638, 639, 640, 641, 642, 643, 644, 645, 646, 647, 648, 649, 650, 651, 652, 653, 654, 655, 656, 657, 658, 659, 660, 661, 662, 66
3, 664, 665, 666, 667, 668, 669, 670, 671, 672, 673, 674, 675, 676, 677, 678, 679, 680, 681, 682, 683, 684, 685, 686, 687, 688, 689, 690, 691, 692, 693, 694, 695, 696, 697, 698, 699, 700, 701, 702, 703, 704, 705
, 706, 707, 708, 709, 710, 711, 712, 713, 714, 715, 716, 717, 718, 719, 720, 721, 722, 723, 724, 725, 726, 727, 728, 729, 730, 731, 732, 733, 734, 735, 736, 737, 738, 739, 740, 741, 742, 743, 744, 745, 746, 747,
748, 749, 750, 751, 752, 753, 754, 755, 756, 757, 758, 759, 760, 761, 762, 763, 764, 765, 766, 767, 768, 769, 770, 771, 772, 773, 774, 775, 776, 777, 778, 779, 780, 781, 782, 783, 784, 785, 786, 787, 788, 789,
790, 791, 792, 793, 794, 795, 796, 797, 798, 799, 800]
-----PORTE NOME?-----
[]
-----APERTE-----
Porta: 22 -- Servizio: ssh
Porta: 80 -- Servizio: http
Porta: 443 -- Servizio: https
```