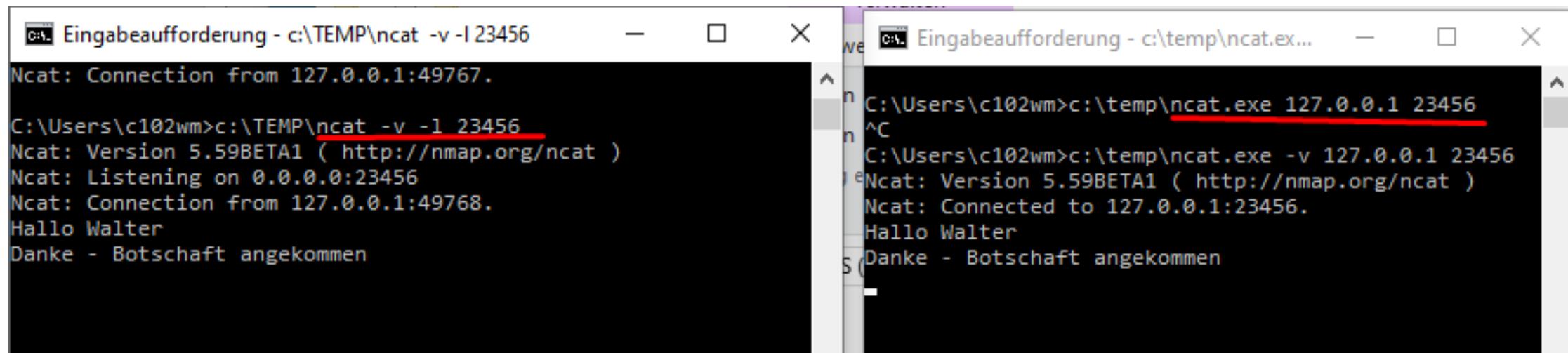


# Python sockets/networking

# IP-Networking zwischen Klienten und Servern



The image shows two terminal windows side-by-side, illustrating an IP networking session between two hosts using the Ncat tool.

**Left Terminal (Client):**

```
Eingabeaufforderung - c:\TEMP\ncat -v -l 23456
Ncat: Connection from 127.0.0.1:49767.
C:\Users\c102wm>c:\TEMP\ncat -v -l 23456
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Listening on 0.0.0.0:23456
Ncat: Connection from 127.0.0.1:49768.
Hallo Walter
Danke - Botschaft angekommen
```

**Right Terminal (Server):**

```
Eingabeaufforderung - c:\temp\ncat.exe 127.0.0.1 23456
^C
C:\Users\c102wm>c:\temp\ncat.exe -v 127.0.0.1 23456
Ncat: Version 5.59BETA1 ( http://nmap.org/ncat )
Ncat: Connected to 127.0.0.1:23456.
Hallo Walter
Danke - Botschaft angekommen
```

In the left terminal, the command `c:\TEMP\ncat -v -l 23456` is run, followed by a connection from the client host at port 49767. The client then sends the message "Hallo Walter" and "Danke - Botschaft angekommen". In the right terminal, the command `c:\temp\ncat.exe 127.0.0.1 23456` is run, followed by a connection from the client host at port 23456. The server then receives the message "Hallo Walter" and "Danke - Botschaft angekommen". The `^C` character in the server's terminal indicates it was interrupted.

# Python – socket Modul

- Sockets erlauben die Kommunikation
  - zwischen Prozessen auf einem Rechner aber auch
  - zwischen Anwendungen auf verschiedenen Rechnern via TCP/IP
- Doku:  
<https://docs.python.org/3/library/socket.html>
- Vorgangsweise:
  - socket als Objekt anlegen
  - dann Verbindung aufbauen, oder am
  - socket lauschen und Verbindung annehmen

# Python – networking client

```
1 #!/usr/bin/python
2 # networking client see https://docs.python.org/3/library/socket.html
3
4 import socket          # Import socket module
5
6 host = "127.0.0.1"
7 port = 12345
8
9 # s = socket.socket()      # Create a socket object
10 # s.connect((host, port))    # connect to remote TCP service
11
12 # or more modern variant tries IPv4 and IPv6, multiple addresses
13 s=socket.create_connection(("localhost",port))
14
15 s.send("Hallo Walter".encode("UTF8"))  # write data
16 bytes=s.recv(1024)                  # receive data
17 print("Response from server:",bytes.decode("UTF8"))    # decode and output received data
18 s.close()                          # Close the socket when done
```

# Python – networking server

```
8 import sys
9 import socket # Import socket module
10
11 s = socket.socket() # Create a socket object
12 host = '' # unspecified ip - all interfaces on host
13 port = 12345 # Reserve a port for your service.
14 s.bind((host, port)) # Bind to the port
15
16 s.listen(1) # Now wait for client connection.
17 connection, client_address = s.accept() # Establish / get one connection with client.
18
19 try:
20     print('Got connection from %s' % str(client_address), file=sys.stderr)
21
22     # Receive the data in small chunks and retransmit it
23     while True:
24         byte_data = connection.recv(30)      # read maximum of 30 bytes from connection
25         data = byte_data.decode("latin1")    # convert byte data to string
26         print('received "%s"' % data, file=sys.stderr)
27         print('sending data back to the client', file=sys.stderr)
28         connection.sendall(("server got: %s" % data).encode("latin1")) # send reply to connection
29     except:
30         print("Unexpected error:", sys.exc_info()[0])
31     finally:
32         connection.close() # clean up the connection
33         s.close() # clean up listening socket
```

# Python und Threads

```
1  # Threads and Python
2  # see https://docs.python.org/3/library/threading.html#thread-objects
3
4  import threading
5  from time import sleep
6
7
8  def count_to(thread_name="default", count_to=10, sleep_time=1.0):
9      for i in range(count_to):
10          print("Thread %s: %d" % (thread_name, i))
11          sleep(sleep_time)
12
13
14 for thread_number in range(3):
15     t = threading.Thread(target=count_to, args=(str(thread_number), 10)) # a thread is an object
16     t.start() # start this thread
17     sleep(2) # wait before starting the next one
18
```

# Python – multithreaded server

- Für jede hereinkommende Verbindung  
also erfolgreiches accept
- Wird  
ein connection handler als eigenständiger thread gestartet
- Das Programm kann dann mehrere Klienten gleichzeitig bedienen.