Do-it-yourself: Statischer HTTP-Server

Wir verwenden wieder einmal die Programmiersprache Python. Die *Socket-Programmierschnittstelle* haben wir ja schon hier kennengelernt. Deswegen steigen wir direkt in den Code ein:

```
import socket


# Allgemeine Defiitionen:
SERVER_HOST = '0.0.0.0' # d.h. alle Netzwerkschnittstellen des Rechners
SERVER_PORT = 8000 # nur im Beispiel, per Konvention geht http über Port 80

# Wir binden uns an einen Socket
with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as server_socket:

    server_socket.setsockopt(socket.SOL_SOCKET, socket.SO_REUSEADDR, 1)
    server_socket.bind((SERVER_HOST, SERVER_PORT))
    server_socket.listen(1)
    print('Lausche auf Port %s ...' % SERVER_PORT)

    # Abarbeiten aller Client-Anfragen in einer Endlosschleife:
    while True:
        # hier warten wir auf den ersten Client
        client_connection, client_address = server_socket.accept()

        # da hat einer angebissen, jetzt die Daten des Requests:
        request = client_connection.recv(1024).decode()
        print(request)

        # Und unser HTTP Response dazu:
        response = 'HTTP/1.0 200 OK\n\nHallo Welt\n\n'
        client_connection.sendall(response.encode())
        client_connection.close()

# Wenn die Endlosschleife am Ende ist...
server_socket.close()
```

Das ist der allereinfachste statische Webserver. Es wird immer (egal welche Ressource angefordert wurde) Das (Pseudo-)Dokument „Hallo Welt“ zurückgeliefert.

Den Kommunikationsablauf schauen wir uns trotzdem mal in der Konsole (wieder mit Rechtskick+Neues Fenster) an:

Server starten mit

```
cd Schulung
python3 hello_http.py
```

und in einer zweiten Konsole:

```
telnet 127.0.0.1 8000
...
GET /irgendwas HTTP/1.0

...
```

Und dabei nach dem GET nicht die leere Zeile vergessen!

So, das war nur zu Demonstrationszwecken. Dieser Code prüft nicht einmal auf korrekte Syntax des Requests (warum auch ).

Diese Bedienung der Socket-Schnittstelle UND die korrekte Behandlung der Requests gibt es in Python auch fertig verpackt zur leichteren Nutzung:

```
cd Schulung/web
python3 -m http.server 8000
```

Dabei wird die Ressource nach GET wie folgt interpretiert:

- Falls eine Datei in dem aktuellen Verzeichnis existiert, deren Name auf die Ressource passt, dann wird `200 OK` mit dem Inhalt der Datei als Response zurückgeschickt.
- Wenn die Ressource das Zeichen / enthält, dann wird das wie Namen von Unterverzeichnissen behandelt und die passende Datei aus dem Unterverzeichnis gesendet
- Wenn die Ressource genau auf ein Verzeichnis (einschließlich / für das aktuelle), dann wird die Datei mit dem Namen index.html als die eigentlich gewünschte Ressource angenommen.
- Ansonsten wird eine Fehlermeldung in den Response gepackt: `404 Not found`