6 Internet und Internetprogrammierung

Vorausgesetzt, unser Rechner verfügt über Internetverbindung, kann Python Emails lesen und versenden, Web-Seiten von entfernten Seiten laden, Dateien über FTP transferieren oder interaktive Web-Seiten konstruieren. Vieles geht relativ einfach, wenn man die Internet-Module verwendet, die in jeder Python-Distribution mitgeliefert werden.

6.1 Einführung in die Funktionsweise des Internet

Bevor wir beginnen, einfache Internetanwendungen zu programmieren bzw. zu verwenden, stellen wir kurz dar, wie das sogenannte Internet entstanden und gewachsen ist und nach welchen Grundprinzipien es funktioniert.

6.1.1 Geschichtliches

Die ursprüngliche Motivation für die Vernetzung von Rechnern war die gemeinsame Nutzung großer Rechenleistung. Rechenleistung war besonders in den frühen Zeiten der Informatik sehr teuer, und nicht jeder Standort eines Unternehmens bzw. einer Universität konnte sich einen eigenen Rechner leisten. Es war sehr viel günstiger, einen großen Rechner durch ein Computernetz mit den Standorten zu verbinden.

Die Organisation, die vor über 40 Jahren am hungrigsten nach Rechenleistung war, war das Verteidigungsministerium der Vereinigten Staaten; und so war die ARPA[1], das Forschungsinstitut des US-Verteidigungsministeriums, der Begründer des heutigen Internets. Das von der ARPA entwickelte Datennetz wurde damals ARPANET genannt. Es verwendete schon in seinen Anfängen eine Technologie, die wir heute noch als *Internetworking* bezeichnen, mit der nicht einzelne Computer, sondern eigentlich Computernetzwerke vernetzt werden können. Abbildung 6.1 zeigt das Wachstum des ARPANET in den 70er- und 80er-Jahren.

Das ARPANET wurde bewusst so angelegt, dass es *dezentral* und möglichst ausfallsicher ist: Es existieren kein Zentralrechner und (möglichst) keine zentralen Internetknoten, an denen alle Verbindungen zusammenlaufen würden.

1 ARPA ist ein Akronym für „Advanced Research Projects Agency“.

Abb. 6.1. Entwicklung des ARPANET zwischen 1969 und 1977.

Aufgabe 6.1

Obwohl man es tendenziell vermeiden möchte, gibt es natürlich bestimmte Internetknotenpunkte. Finden Sie durch eine Web-Recherche heraus, welches die für Deutschland wichtigsten Internetknotenpunkte sind.

Diese beabsichtigte Dezentralität erstreckt sich jedoch nicht vollständig auf die administrative Ebene. Die Organisation für die Pflege und Zuordnung von IP-Adressen auf Domain-Namen ist die ICANN[2], die indirekt durch das US-Wirtschaftsministerium kontrolliert ist. ICANN unterhält Server in vielen Ländern.

6.1.2 Netzwerk-Protokolle

Ein *Protokoll* ist eine bestimmte Vereinbarung, nach der die Kommunikation zweier Parteien abläuft. In einem Protokoll ist beispielsweise festgelegt, wie eine Nachricht beginnt, wie sie endet, wie eine Nachricht formatiert ist, was bei Fehlern oder unerwartetem Verbindungsabbruch passiert, usw. Kommunikationsprotokolle sind *die* Grundlage des Internet. Die beiden für das Internet wichtigsten Kommunikationsprotokolle sind das Internet Protocol (IP) und das Transmission Control Protocol (TCP). Die Bezeichnung TCP/IP steht jedoch meist für die ganze Sammlung der am häufigsten verwendeten Protokolle des Internet – und davon gibt es viele.

6.1.3 Das TCP/IP-Referenzmodell

Um die Kommunikationsaufgaben in Netzwerken übersichtlich darzustellen, werden die entsprechenden Protokolle gemäß funktionaler Ebenen sortiert dargestellt. Das

2 ICANN = Internet Corporation for Assigned Names and Numbers

TCP/IP-Referenzmodell gliedert sich in vier aufeinander aufbauende Schichten: die „unterste“ ist die technischste, die „oberste“ ist die abstrakteste. Tabelle 6.1 zeigt das TCP/IP-Referenzmodell: Ein Grundprinzip ist, dass die Protokolle einer bestimmten

Tab. 6.1. Das TCP/IP-Schichtenmodell

TCP/IP-Schicht	englischer Name	Beispiele
Anwendungsschicht	**Application**	HTTP, FTP, SMTP, POP, Telnet, ...
Transportschicht	**Transport**	TCP, UDP, SCTP, ...
Internetschicht	**Internet**	IP
Bitübertragung	**Physical**	Ethernet, Token Bus, Token Ring, ...

Schicht nie auf die Funktionalitäten der Protokolle einer höheren Schicht zugreifen können. Ein Zugriff von „oben“ nach „unten“ ist jedoch immer möglich: Die Protokolle einer bestimmten Schicht müssen immer in der Lage sein, auf Funktionalitäten der Protokolle tiefer liegender Schichten zuzugreifen. Beispielsweise muss ein Programm der Anwendungsschicht auf Dienste und Funktionen der Transportschicht zugreifen. Dieser Zugriff geschieht über Sockets bzw. Ports.

Auf der *Anwendungsschicht* befinden sich Prozesse mit direkter Benutzerinteraktion, wie Web-Browser, Email-Clients oder FTP-Clients. Programmiert man Internetanwendungen (wie wir das später in diesem Abschnitt tun werden), so verwendet der Programmierer meistens die Anwendungsschicht zur Kommunikation. Hierbei kommen Ports bzw. Sockets zum Einsatz (siehe unten). Im Allgemeinen existiert für jedes Protokoll eine bestimmte Port-Nummer, die verwendet wird.

Zu den Aufgaben der *Transportschicht* zählen der Aufbau einer Verbindung, die Segmentierung von Datenpaketen (ein Teil einer Nachricht kann evtl. andere Wege gehen als ein anderer Teil derselben Nachricht), das Sicherstellen eines verlustfreien Datentransfers (es kann unterwegs immer zu Datenverlusten kommen und es ist wichtig, dass die Kommunikationspartner das merken und entsprechend handeln können) und Mechanismen zur Vermeidung von Daten-Staus. Der Verbindungsaufbau mit TCP erfolgt folgendermaßen (und zwar ganz ähnlich zum herkömmlichen Telefon): Ein Anrufer sendet einen sogenannten *Request* (entspricht dem Klingel-Zeichen beim Telefonieren) an einen Kommunikationspartner. Dieser sendet eine Antwort (engl: *Reply*) an den Anrufer; dies entspricht dem Abheben des Telefonhörers und dem Melden. Der Datenaustausch kann beginnen. Eine Verbindung wird solange aufrecht erhalten, bis sie explizit von einem Kommunikationspartner beendet wird; dies entspricht dem Auflegen des Telefonhörers. Eine TCP-Übertragung fordert eine relativ hohe Netzbelastung (Request-Reply-...). Benötigt ein Dienst keine gesicherte Verbindung zwischen zwei Kommunikationspartnern, so kann das einfachere User Datagram Protocol (UDP) verwendet werden.

Aufgabe 6.2

Informieren Sie sich genauer: Warum ist UDP so viel unsicherer als TCP? Wann genau wird üblicherweise UDP eingesetzt?

6.1.4 Internetworking

Das Ziel des Internetworking ist die Kommunikation über Netzwerkgrenzen hinweg. Für den Zusammenschluss zweier (oder mehrerer) Netzwerke werden *Router* eingesetzt. Üblicherweise verbindet ein Router zwei unterschiedliche Netzwerke. Ein Router hat eine getrennte Schnittstelle für jeden Netzanschluss. Abbildung 6.2 zeigt ein aus vier beliebigen Netzwerken bestehendes Internet. Ein Netzwerk ist als Wolke dargestellt, ein Router als kleines Rechteck zwischen den Wolken.

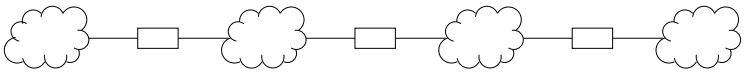


Abb. 6.2. Ein kleines Internet, bestehend aus vier Netzwerken, jeweils dargestellt als Wolken. Diese sind durch Router, dargestellt als kleine Rechtecke, miteinander verbunden.

Meist verbindet ein Router nur zwei physikalisch unterschiedliche Netzwerke, obwohl die meisten kommerziellen Router auch mehr als zwei Netzwerke miteinander verbinden könnten. Der Grund ist ein sehr pragmatischer: Routing ist sehr anspruchsvoll was die erforderliche Rechenleistung betrifft. Die Rechenkapazität eines Routers reicht oft nicht aus, um den Datenverkehr zwischen mehr als zwei Netzwerken zu vermitteln. Außerdem wird durch mehr Redundanz das Internet zuverlässiger, oder anders ausgedrückt: Je mehr zentrale Netzknoten (und ein Router, der viele Netze verbindet, wäre ein solcher) das Internet beinhaltet, desto unzuverlässiger wird es.

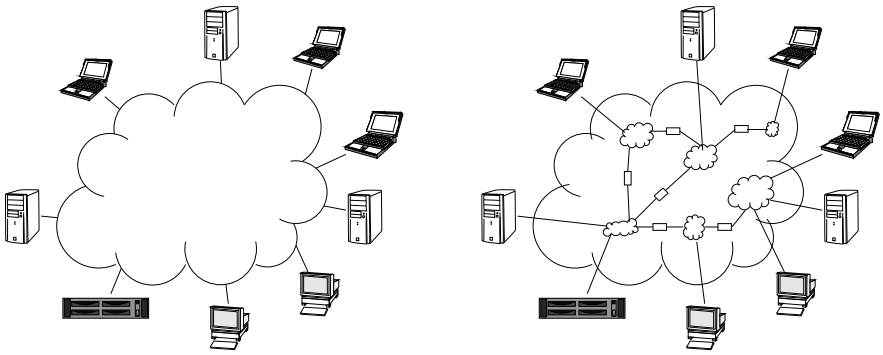
Aufgabe 6.3

Erklären Sie diesen Sachverhalt genau: Warum ist es – was die Zuverlässigkeit betrifft – ungünstig, viele zentrale Netzknoten zu haben?

Aufgabe 6.4

Auf welcher Ebene des TCP-IP-Referenzmodells arbeitet ein Router?

Internetworking gibt dem Benutzer die Illusion eines einzigen, zusammenhängenden Netzwerkes, obwohl es eigentlich aus vielen sehr unterschiedlichen physikalischen Netzwerken besteht. Abbildung 6.3 zeigt diesen Sachverhalt. Der linke Teil veranschaulicht die Illusion des Benutzers, sich in einem homogenen Netzwerk zu befinden; der rechte Teil veranschaulicht die eigentlich zugrundeliegende Struktur: Die Rechner sind in Wirklichkeit an verschiedene physikalische Netzwerke angeschlossen; diese Netzwerke sind über Router verbunden.



Ein virtuelles Netzwerk ... ...bestehend aus vielen Netzwerken

Abb. 6.3. Linker Teil der Abbildung: Veranschaulichung eines virtuellen Netzwerks, das dem Internetworking zugrunde liegt. Rechter Teil der Abbildung: Veranschaulichung der eigentlich zugrundeliegenden physikalischen Struktur: Das virtuelle Netzwerk besteht aus vielen unterschiedlichen physikalischen Netzwerken.

6.1.5 Sockets

Sockets sind Schnittstellen zwischen Netzwerkverbindungen verschiedener Rechner, und die Schnittstellen der Anwendungsschicht zu den darunterliegenden Schichten. Jede Kommunikation über die gängigen Anwendungsprotokolle (wie etwa FTP, HTTP usw.) basiert auf der Kommunikation über Sockets.

Obwohl Sockets sehr oft für Netzwerk-Kommunikation eingesetzt werden, können sie auch dazu verwendet werden, die Kommunikation zwischen Programmen zu ermöglichen, die auf dem selben Rechner laufen – diese Art der Kommunikation wird manchmal auch *Inter-Process-Communication* (Kurz: IPC) genannt.

Kommunikation über Sockets ist immer *bidirektional*, d. h. Daten können sowohl gesendet als auch empfangen werden. Aus der Sicht eines Programmierers bestehen Sockets einfach aus einer Menge von bereitgestellten Funktionen. Jeder Socket bezieht sich auf eine bestimmte Adresse, die sich zusammensetzt aus ...

1. ...der *IP-Adresse des Rechners*.
   Diese ist entweder eine Folge von Zahlen, durch Punkte getrennt (beispielsweise `82.94.164.162`) oder wird als *Domain-Namen*, eine durch Menschen besser lesbarer Form, angegeben (beispielsweise `www.python.org`). Diese Domain-Namen werden automatisch auf die entsprechende Folge von Zahlen abgebildet; dies bewerkstelligt ein sogenannter *Domain Name Server* (DNS). Dieser kann mit einem Telefonbuch verglichen werden, das ebenso wie der DNS Namen auf Zahlen, also Telefonnummern, abbildet.
2. ...der *Portnummer*.
   Jede Kommunikationsart entspricht einer bestimmten Zahl, der sog. Portnummer. Rechner, die mit dem Internet verbunden sind, unterstützen im Allgemeinen eine große Zahl verschiedener Kommunikationsdienste. Zwei Rechner, die über das Internet kommunizieren wollen, müssen beide bestimmte Sockets für die Verbindung reservieren, die beide die gleiche Portnummer enthalten müssen. Wollen die beiden Rechner beispielsweise über das HTTP-Protokoll kommunizieren, müssen beide dafür die Portnummer 80 verwenden.

Die Portnummern von Sockets können theoretisch aus dem Bereich zwischen 0 und 65535 kommen. Standardprotokolle werden jedoch meist auf eine Portnummer aus dem Bereich zwischen 0 und 1023 abgebildet; jedes Protokoll hat eine vordefinierte Portnummer. Abbildung 6.2 zeigt eine Liste der wichtigsten Protokolle zusammen mit den entsprechenden Portnummern.

Tab. 6.2. Einige der wichtigsten Internetprotokolle zusammen mit den zugehörigen Portnummern.

Protokoll	Funktion	Portnr	Pythonmodul
HTTP	Webseiten	80	`httplib`
FTP - Daten	Dateitransfer	20	`ftplib`
FTP - Control	Dateitransfer	21	`ftplib`
SMTP	Emails senden	25	`smtplib`
POP3	Emails empfangen	110	`poplib`
Telnet	Kommandozeile	23	`telnetlib`

6.1.6 Host, Server, Client

Wir beginnen mit der Definition der Begriffe „Host“, „Server“ und „Client“, die im Folgenden immer wieder verwendet werden.

Der Begriff „*Host*“(engl. für Gastgeber) bezeichnet einen Rechner, der in ein Netzwerk eingebunden ist und bestimmte Dienste, wie etwa Datenbankabfragen oder die Durchführung bestimmter Berechnungen, entweder anbietet oder beansprucht.

Als *Server* (engl. für Diener) bezeichnet man einen Rechner, manchmal auch nur ein bestimmtes Programm auf einem Rechner, das bestimmte Dienste in einem Netzwerk bereitstellt. Ein Server wartet dabei *passiv* auf Anfragen von außen. Oft kann ein leistungsfähiger Server parallel mehrere Anfragen gleichzeitig bearbeiten. Meist sind Server sehr leistungsfähige Rechner, die rechen- oder speicheraufwändige Dienste bereitstellen, wie etwa die Suche in großen Datenbanken, die Ausführung komplexer numerischer Berechnungen, die Bereitstellung von Speicher oder den Zugriff auf große Dateisysteme.

Als *Clients* (engl. Kunden) bezeichnet man diejenigen Rechner (manchmal auch: diejenigen Programme), die eine Verbindung zu einem Server herstellen, um dessen Dienste zu nutzen. Hierbei spielt der Client immer die aktive Rolle: er entscheidet, wann er die Dienste des Servers in Anspruch nehmen will.

Das Client-Server-Modell ist besonders im Zeitalter des Internet die vorherrschende Methode, Dienste innerhalb eines Netzwerks zu verteilen. Email-Anwendungen sind beispielsweise sehr oft so organisiert. Der Mail-Server verwaltet die Emails einer Gruppe von Nutzern, d. h. er empfängt, versendet und speichert Emails; er befindet sich meist auf einem leistungsfähigen zentralen Rechner. Der Email-Client ist das Programm, das bei einem bestimmten Nutzer auf dem lokalen Rechner läuft; es realisiert die graphische Benutzeroberfläche und nimmt Verbindung zum Mail-Server auf, um dessen Dienste in Anspruch zu nehmen.

6.2 Socketprogrammierung

Dieser Abschnitt erklärt, wie man mit Python auf Socket-Ebene, d. h. auf der Ebene der Transportschicht, Verbindungen aufbauen und Daten austauschen kann. Alle folgenden Abschnitte zeigen, wie man in Python Anwendungsprotokolle verwenden kann, und zwar am Beispiel von FTP, HTTP, und SMTP – und in der Tat ist das die wohl häufigste Art der Internetprogrammierung.

Um einfache Netzwerkverbindungen zwischen Rechnern aufzubauen, kann das Pythonmodul `socket` verwendet werden. Es stellt eine Schnittstelle zur Netzwerkkommunikation mit Sockets bereit. Die durch dieses Modul bereitgestellten Funktionen arbeiten auf der Transportschicht und sprechen entsprechende Protokolle (TCP/IP, UDP, ...) direkt an. Die meisten Protokoll- und Servermodule (wie etwa `ftplib`, `smtplib`, ...) setzen direkt auf diesem Modul auf. Die Programmierung direkt mit Sockets ist daher sehr viel „technischer“ als die Programmierung von Internetkommunikation auf der Anwendungsschicht.

Das in Listing 6.1 gezeigte Programm ist ein einfaches Beispiel, das die Verwendung von Socket-Kommunikation veranschaulichen soll. Das Programm wartet dar-

auf, dass ein anderer Rechner eine Verbindung zu einem bestimmten Socket auf Port `50007` nutzt und Daten schickt – ein solches Programm, das passiv darauf wartet, von einem anderen Rechner „angesprochen“ zu werden, um bestimmte Dienste zu leisten, nennt man oft auch *Server*. Der „Dienst“ des Servers in diesem Beispiel besteht einfach darin, dass er ankommende Daten einfach wieder mit dem vorangestellten String `'Echo=>'` zurückschickt. Es ist gut möglich, dass – falls sie das Programm unter Windows ausführen – die Firewall von Windows die Aktionen blockiert. Immerhin geben Sie damit ja einen Socket für Anfragen von außen frei, was potentiell gefährlich sein kann. Um das Programm zu testen, müssen sie dann eventuell der Firewall mitteilen, dass sie es in diesem Fall zulassen soll.

```
1  from socket import *
2  myHost = '' # Lokaler Host
3  myPort = 50007
4  s = socket(AF_INET, SOCK_STREAM)
5  s.bind((myHost,myPort))
6  s.listen(5)
7
8  while True:
9    verb, adr = s.accept()
10   print('I am connected by'+adr)
11   while True:
12     dat = verb.recv(1024)
13     if not dat: break
14     verb.send('Echo=>'+dat)
15   verb.close()
```

Listing 6.1. Ein einfaches Server Programm

Dieses kleine Server-Programm arbeitet im Detail wie folgt:

- Zeile 4: `s = socket(AF_INET, SOCK_STREAM)` erzeugt ein TCP-Socket-Objekt. Sowohl `AF_INET` also auch `SOCK_STREAM` sind im `socket`-Modul vordefinierte Variablen; zusammen verwendet bedeuten sie einfach, dass ein TCP/IP-Socket verwendet wird.
- Zeile 5: `s.bind((myHost, myPort))` bindet das Socket-Objekt an eine Adresse, bestehend aus einer IP-Adresse und einem Port. Handelt es sich um Server-Programme (wie in diesem Fall), so ist der Hostname in aller Regel der leere String `''`, was bedeutet, dass die Adresse des lokalen Rechners verwendet wird.
- Zeile 6: `s.listen(5)` wartet auf einkommende Verbindungen von außen; 5 Verbindungen können gleichzeitig in wartendem Zustand sein. Für den Fall, dass noch mehr Clients gleichzeitig den Host kontaktieren wollen, werden diese dann zurückgewiesen und müssen es später nochmals versuchen.

- Zeile 9: `verb, adr = s.accept()` wartet darauf, dass der nächste Client-Rechner den Host kontaktiert. Wenn dies passiert, dann wird ein neues Socket-Objekt `verb` zurückgeliefert, über das einkommende Daten fließen. Zusätzlich wird auch noch die IP-Adresse `adr` des kontaktierenden Clients zurückgeliefert.
- Zeile 12: `dat = verb.recv(1024)` liest höchstens 1024 Bytes an Daten, die der Client sendet. Wenn der Client fertig ist, wird ein leerer String zurückgeliefert und `dat` entspricht dem Wert `False`.
- Zeile 14: `verb.send('Echo=>'+dat)` sendet den zuletzt erhaltenen Datenblock zusammen mit einem vorangestellten `'Echo=>'` zurück an den Client.

Die in Listing 6.2 gezeigte Funktion `senden(message)` stellt ein zugehöriges Client-Programm dar, das Daten auf den vom Server geöffneten Socket senden kann.

```
1  from socket import *
2  serverPort=50007
3  serverHost='41.87.8.37'
4
5  def senden(txt):
6      s = socket(AF_INET, SOCK_STREAM)
7      s.connect((serverHost, serverPort))
8      for zeile in txt:
9          s.send(zeile)
10         daten = s.recv(1024)
11         print('Client received:' +str(daten))
12     s.close()
```

Listing 6.2. Das zu Listing 6.1 passende Client-Programm.

Client und Server müssen dieselbe IP-Adresse und dieselbe Portnummer verwenden und schon können so ganz einfach Daten ausgetauscht werden – vorausgesetzt die Firewall blockiert den Zugriff nicht. Der verwendete Port `50007` ist eine recht zufällig gewählte Zahl und außerhalb des Bereiches, in dem sich die Port-Nummern für die gängigen Anwendungsprotokolle (wie Email, FTP, usw.) befinden.

Aufgabe 6.5

Wir versuchen, ein einfaches Python-Skript zu schreiben, mit dem man Text-Dateien übertragen kann:

- Schreiben Sie eine Funktion `sendeDatei(datei,ipAdr,port)`, das versucht, dem Rechner mit der Adresse. `ipAdr` auf Port `port` die Datei `datei` Zeile für Zeile zu senden.

– Schreiben Sie ein passendes Server-Programm, das auf die Daten des soeben geschrieben Client-Programms wartet und die gesendeten Daten in eine Datei `tmp.txt` schreibt.

6.3 Dateitransfer mit FTP

FTP ist eines der ältesten Protokolle der Anwendungsschicht des TCP/IP-Referenzmodells. Das Kürzel „FTP“ steht für „File Transfer Protocol“. Es handelt sich um ein Protokoll, das speziell für die Übermittlung von Dateien entworfen wurde.

FTP verfügt über zwei Socketverbindungen. Eine Verbindung, auch *Steuerkanal* genannt, um Befehle (wie etwa „Hole Datei xy“ oder „Öffne Verbindung“) zu einem entfernten Rechner zu senden. Diese Socketverbindung belegt überlicherweise den Port 21. Über die zweite Socketverbindung des FTP-Protokolls, auch *Datenkanal* genannt, werden die die eigentlichen Daten der zu übermittelnden Datei gesendet. Nach dem Öffnen einer FTP-Verbindung muss sich der Benutzer authentifizieren. Ist der FTP-Server öffentlich zugänglich, so gibt man üblicherweise als Benutzername `anonymous` ein und als Passwort seine Email-Adresse.

Alle Details der Implementierung des FTP-Protokolls sind im Python-Modul `ftplib` gekapselt.

Das in Listing 6.3 gezeigte Python-Skript beispielsweise loggt sich automatisch am Webserver der Hochschule Albstadt-Sigmaringen ein, wechselt ins Verzeichnis `public_html` und transferiert alle als Kommandozeilenargument übergebenen Dateien des lokalen Rechners auf den FTP-Server in eben dieses Verzeichnis.

```
import ftplib, sys
ftp = ftplib.FTP('www.hs-albsig.de')
ftp.login('haeberlein','...')
ftp.cwd('public_html')
for filename in sys.argv[1:]:
  ftp.storbinary('STOR '+filename, open(filename, 'rb'))
```

Listing 6.3. Einfaches Beispiel für die Verwendung des `ftplib`-Moduls

Um eine Verbindung zu einem FTP-Server zu öffnen, muss man mit der Funktion `ftplib.FTP` ein FTP-Objekt erzeugen. Diese Funktion erhält als Argument die IP-Adresse (in Form eines Strings) des FTP-Servers. Dieses Objekt stellt Methoden bereit, mit denen man FTP-Befehle ausführen kann. Mit der `login`-Methode kann man sich mit Benutzernamen und Passwort beim FTP-Server anmelden; mit der `cwd`-Methode kann man das Server-Verzeichnis wechseln. Die `storbinary`-Methode kann Dateien aus dem aktuellen lokalen Verzeichnis auf dem aktuellen Serververzeichnis speichern. Diese Methode erwartet als erstes Argument einen FTP-Kommandostring; der

String `'STOR'+filename` stellt das übliche Format für einen FTP-Kommandostring dar. Das zweite Argument ist eine Funktion oder Methode, die das FTP-Kommando mit Daten versorgt. Will man die umgekehrte Richtung gehen, d. h. Daten vom Server auf den lokalen Rechner laden, so kann man den Befehl

```
ftp.retrbinary('RETR '+dateiname, lokalfile.write)
```

verwenden. Auch in diesem Fall ist das erste Argument ein FTP-Kommandostring und das zweite Argument eine Funktion oder Methode, an die Python die gelesenen Daten weitergibt.

Es scheint etwas inkonsequent, dass die Methoden `retrbinary` und `storbinary` die Strings `'RETR'` bzw. `'STOR'` benötigen, denn diese sind eigentlich Teil des FTP-Protokolls, das ja das `ftplib`-Modul verstecken sollte. Siehe hierzu die folgende Aufgabe.

Aufgabe 6.6

Sie wollen die Details des FTP-Protokolls für das Holen und Speichern von Dateien verstecken und zwei Funktionen schreiben, die Ihnen eine etwas abstraktere (d. h. technische Details versteckende) Schnittstelle auf das Holen und Speichern von Dateien bietet, als dies die Methoden `retrbinary` und `storbinary`, die das Python-Modul `ftplib` zur Verfügung stellt, tun.

(a) Schreiben Sie eine Python-Funktion

```
getfile(file, site, dir, user=())
```

das die im Verzeichnis **`dir`** befindliche Datei **`file`** von der Adresse `site` in dem aktuellen Verzeichnis speichert (unter dem gleichen Namen). Optional kann im Parameter `user` ein Tupel bestehend aus Benutzername und Passwort mit übergeben werden.

(b) Schreiben Sie eine Python-Funktion

```
putfile(file, site, dir, user=())
```

das die im aktuellen Verzeichnis befindliche Datei **`file`** auf den FTP-Server mit Adresse `site` in dessen Verzeichnis **`dir`** speichert. Optional kann wiederum im Parameter `user` ein Tupel bestehend aus Benutzername und Passwort mit übergeben werden.

Manchmal sind FTP-Sites nicht erreichbar. Angenommen, wir haben eine Liste von FTP-Seiten gespeichert, von denen wir Daten holen wollen. Listing 6.4 zeigt ein Python-Skript, das die nicht erreichbaren FTP-Seiten aus dieser Liste ausfiltert.

```
import socket, ftplib
def isFTPSeiteDa(seite):
  try:
    ftplib.FTP(seite).quit()
  except socket.error:
    return False
  else:
    return True
```

Listing 6.4. Ausfiltern nicht erreichbarer FTP-Sites

Die `try`-Anweisung fängt Ausnahmen[3] ab. Jeder `try`-Anweisung kann eine (oder mehrere) `except`-Klauseln folgen, die dann ausgeführt werden, wenn eine Ausnahmen eintritt. Jeder `try`-Anweisung kann (muss aber nicht) eine `else`-Klausel folgen, die dann ausgeführt wird, wenn keine Ausnahme aufgetreten ist.

Die eigentliche gewünschte Filter-Funktionalität können wir beispielsweise durch eine einfache Listenkomprehension implementieren:

```
def filterFTPSeiten(seiten):
  return [for seite in seiten if isFTPSeiteda(seite)]
```

Aufgabe 6.7

Verwenden Sie statt einer Listenkomprehension die `filter`-Funktion, um `filterFTPSeiten` zu implementieren.

6.4 HTML und Datentransfer von URLs

6.4.1 HTML

HTML ist ein Protokoll der Anwendungsschicht und steht für „Hypertext Markup Language“. HTML, die Sprache des World Wide Web, ist eine typische Auszeichnungs-Sprache, im Englischen als Markup-Language bezeichnet. Das bedeutet, man vermeidet in HTML die Angabe des konkreten Formats bestimmter Textteile, sondern gibt bevorzugt Informationen darüber an, wie diese Textteile „ausgezeichnet“ werden. Mit anderen Worten, durch HTML beschreibt man die Art der Daten und nicht deren Formatierung. Das Grundgerüst einer HTML-Datei kann wie folgt aussehen:

3 Ausnahmen (auch Exceptions genannt) werden bestimmte Bedingungen genannt, die den vorgesehenen Kontrollfluss einer Programmausführung ändern; dies können etwa bestimmte Fehler, aber auch von außen kommende Ereignisse sein, wie etwa Signale von der Tastatur oder von anderen externen Geräten.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
       "http://www.w3.org/TR/html4/loose.dtd">
<html>
<head> <title>
       ⟨Beschreibung der Seite⟩
</title> </head>
<body>
       ⟨Inhalt der Seite⟩
</body>
</html>
```

Einige häufig verwendete Textauszeichnungen, oft auch als *Tags* bezeichnet, sind im Folgenden aufgelistet:

⟨*text*⟩	⟨*text*⟩ bildet einen Paragraphen;
⟨*text*⟩	⟨*text*⟩ ist eine Aufzählungsliste;
⟨*text*⟩	⟨*text*⟩ ist betont;
<pre> ⟨*text*⟩ </pre>	⟨*text*⟩ ist präformatiert (wie im Editor eingeben);
 ⟨*text*⟩ 	⟨*text*⟩ ist ein Hyperlink auf die URL ⟨*url*⟩;
⟨*text*⟩	⟨*text*⟩ ist eine eine Überschrift der 1. Ebene;
...	...
⟨*text*⟩	⟨*text*⟩ ist eine eine Überschrift der 4. Ebene.

6.4.2 Datentransfer von URLs

URLs (=Uniform Ressource Locator) identifizieren eine Datenquelle zusammen mit dem zur Übertragung verwendeten Protokoll. Umgangssprachlich nennt man URLs meist einfach „Internetadresse“. Beispielsweise ist die URL des Wikipedia-Eintrags zum Begriff „URL“

```
http://de.wikipedia.org/wiki/Uniform_Ressource_Locator
```

Am Kürzel `http` (= Hypertext Transfer Protocol) erkennt man, dass die URL eine HTML Seite enthält, die mittels des Protokolls HTTP aus der Anwendungsschicht (siehe Tabelle 6.1) übertragen wird.

Um die Details des HTTP-Protokolls braucht sich der Python-Programmierer nicht unbedingt zu kümmern; diese sind im Modul `urllib2`, das intern auf das `socket`-Modul zurückgreift, versteckt. Den Inhalt einer Web-Seite kann man mit Python ganz einfach folgendermaßen auslesen:

```
>>> import urllib2
>>> [zeile for zeile in
...        urllib2.urlopen('http://www.hs-albsig.de/haeberlein')]
```

Die Methode `urlopen` erzeugt ein iterierbares Objekt, das etwa in einer Schleife oder einer Listenkomprehension verwendet werden kann. Die geöffnete Seite ist eine Datei im HTML-Format.

Um eine HTML-Seite nach bestimmten Strings zu durchsuchen, kann man einfach Pythons String-Operationen verwenden. Der folgende Code liest beispielsweise alle Zeilen der Web-Seite des Autors aus, die Überschriften der Ebene 4 oder Ebene 3 besitzen.

```
>>> import urllib2
>>> for zeile in urllib2.urlopen('http://www.hs-albsig.de/haeberlein'):
...     if '<h4>' in zeile or '<h3>' in zeile:
...         print(zeile)
```

Aufgabe 6.8

Verwenden Sie statt einer `for`-Schleife eine Listenkomprehension, um die gleiche Funktionalität zu programmieren, wie obiges Beispielprogramm sie realisiert.

Aufgabe 6.9

(a) Erweitern Sie das obige Programm so, dass Umlaute in einer lesbaren Form ausgegeben werden. Also etwa statt

```
<h4>B&uuml;cher</h4>
```

sollte besser

```
<h4>Buecher</h4>
```

ausgegeben werden.

(b) Erweitern Sie das obige Programm weiter so, dass der Text (und zwar nur der Text, ohne die HTML-Tags `<h4>`, `<h3>`, usw.) *aller* Überschriften in gut lesbarer Form mit ausgegeben wird.

Aufgabe 6.10

Schreiben Sie eine Python-Funktion, die eine URL als Eingabe erhält und die alle Link-Adressen, die die übergebene Seite enthält, als String-Liste zurückliefert. Sie können hierzu entweder das Python-Modul `HTMLParser` verwenden oder versuchen, das Ganze selbst zu programmieren – was aber die schwerere der beiden Varianten ist.

6.5 Dynamische Web-Seiten

CGI ist ein Standard, der festlegt, wie ein Webserver ein bestimmtes Programm (oft einfach CGI-Skript genannt) ausführt und so eine Webseite dynamisch erstellt. Hierbei wird die Standardausgabe `stdout` des CGI-Skripts mit der Antwort des Webservers verknüpft. Das CGI-Skript sollte also gültiges HTML ausgeben, und diese Ausgabe wird dann – statt des Inhaltes einer statischen Web-Seite – an den Client übermittelt. Man kann eigentlich jede beliebige Programmiersprache verwenden, um ein CGI-Skript zu schreiben, vorausgesetzt der Web-Server unterstützt die entsprechende Skriptsprache.

6.5.1 `htmlgen`: Generierung von HTML-Code

CGI-Skripte müssen, wie oben beschrieben, immer gültigen HMTL-Code auf der Standardausgabe `stdout` ausgeben. In Python kann man dies entweder einfach durch `print`-Anweisungen realisieren, die entsprechende HTML-Codezeilen ausgeben; dazu muss der Programmierer sich aber genau in der HTML-Syntax auskennen. Das Erzeugen von HTML-Code kann in Python viel komfortabler mit Hilfe des Moduls `HTMLgen` erfolgen. Das Pythonmodul `HTMLgen` ist eine Klassenbibliothek, mit der man HTML-Texte einfach und ohne genaue Kenntnis der HTML-Syntax erstellen kann. Sie ist nicht im Standardlieferumfang von Python enthalten, kann aber durch

```
$ python -m pip install htmlgen
```

einfach nachinstalliert werden.

```
>>> import htmlgen
>>> doc = htmlgen.Document(title='Hello')
>>> doc.append_body(htmlgen.Heading(1, 'Hello World'))
>>> print(doc)
```

Das mittels `htmlgen.Document` erzeugte Objekt `doc` kann mittels `print(doc)` ausgegeben werden. Das Ergebnis ist folgender HTML-Code, der eine Überschrift der Ebene 1 mit Titel `'Hello World'` enthält:

```
<!DOCTYPE html>
<html lang="en" xml:lang="en" xmlns="http://www.w3.org/1999/xhtml">
<head><title>HELLO</title>
  <meta charset="utf-8"/>
  </head>
  <body>
    <h1>Hello World</h1>
  </body></html>
```

Das Modul `htmlgen` eignet sich hervoragend für die einfache Erstellung von Tabellen und Listen. Man kann beispielsweise eine Tabelle aller Einträge in `/etc/passwd` erstellen, indem man den in Listing 6.5 gezeigten Code an obiges Listing anfügt:

```
tabelle = htmlgen.Table()
kopf  = tabelle.create_simple_header_row("Name", "Ort")
zeile1 = tabelle.create_simple_row("TH", "Blaustein")
zeile2 = tabelle.create_simple_row("ST", "Stuttgart")
tabelle = htmlgen.Table(tabletitle='passwd', border=2,
                        heading=['User', 'Ort'])
doc.append(tabelle)
```

Listing 6.5. Erstellung einer einfachen Tabelle mit `htmlgen`

Abbildung 6.4 zeigt die Darstellung der durch dieses Skript erzeugten Tabelle in einem Webbrowser.

Hello World

User	Ort
TH	Blaustein
ST	Stuttgart

Abb. 6.4. Darstellung einer durch `htmlgen` erzeugten Tabelle in einem Webbrowser (Ausschnitt).

Aufgabe 6.11

Schreiben Sie ein Python-Skript unter Verwendung von `htmlgen`, das eine Multiplikationstabelle der Zahlen 1 bis 5 erstellt.

Das von Ihnen erstellte Python-Skript sollte HTML-Code ausgeben, der in einem Webbrowser folgendermaßen dargestellt wird:

Multiplikationstabelle

	1	2	3	4	5
1	1	2	3	4	5
2	2	4	6	8	10
3	3	6	9	12	15
4	4	8	12	16	20
5	5	10	15	20	25

Aufgabe 6.12

Erstellen Sie mit Hilfe von `htmlgen` ein HTML-Dokument, das in Tabellenform die ersten 1000 Primzahlen enthält.

6.5.2 Ein einfacher Web-Server

Um Code für eigene CGI-Skripte möglichst einfach testen zu können, implementieren wir hier einen eigenen, sehr einfachen lokalen Webserver, der einem Webbrowser entweder HTML-Seiten oder die Ausgabe eines CGI-Skripts zur Verfügung stellt. Dank leistungsfähiger Python-Module, die alle notwendigen Funktionalitäten eines Webservers schon bereitstellen, kann man einen einfachen Webserver mit wenigen Zeilen Python-Code implementieren:

```
import os, sys
from http.server import HTTPServer
from http.server import CGIHTTPRequestHandler

webdir='.'
port = 80
os.chdir(webdir)
svraddr=('',port)

srvobj = HTTPServer(svraddr, CGIHTTPRequestHandler)
srvobj.serve_forever()
```

Listing 6.6. Ein einfacher Web-Server

Ein Hinweis vorab: Wenn Sie dieses Skript unter Windows laufen lassen, so werden sie feststellen, dass der Port 80 häufig durch Dienstprogramme des Betriebssystems belegt ist; in diesem Fall wird eine Exception ausgelöst, und eine Meldung ausgegeben, dass es keine Zugriffsrechte auf den Socket gibt. In diesem Fall empfiehlt es sich, die Variable `port` statt auf den Wert 80 auf einen anderen Wert, etwa 99, zu setzten. Sie müssen dann aber im Folgenden darauf achten, dass die Web-Adressen der erstellten Seiten nicht mehr unter der Adresse `http://localhost/...` ansprechen, sondern stattdessen unter `http://localhost:99/...`.

Die Klasse `HTTPServer` implementiert ein HTTP-Socket und leitet jede Anfrage an einen Handler weiter, in diesem Falle den `CGIHTTPRequestHandler`. Die Klasse `CGIHTTPRequestHandler` versorgt einen Server sowohl mit (HTML-)Dateien als auch, falls dies angefordert wird, mit der von CGI-Skripts produzierten Standardausgabe; d. h. der in Listing 6.6 definierte `CGIHTTPRequestHandler` wird eine Datei ausführen, falls er diese als ein CGI-Skript klassifiziert, und die Ausgabe dieser Datei an den Client weiterleiten.

Starten wir nun das obige Webserver-Skript, erzeugen dann in dem selben Verzeichnis, in dem der Webserver gestartet wurde, eine Datei `webseite.html` mit folgendem Inhalt:

```
<html> <title>Eine HTML-Seite</title>
<body>
  <h1>Eine einfache HTML-Seite</h1>
  <p>Hallo Welt</p>
</body></html>
```

so kann man in einem Webbrowser unter der URL

`http://localhost/webseite.html`

die entsprechende Webseite betrachten.

6.5.3 Ein erstes CGI-Skript

Pythons `CGIHTTPRequestHandler` erwartet, dass CGI-Skripte in das Unterverzeichnis `cgi-bin` gespeichert werden. Ein erstes, sehr einfaches CGI-Skript gibt gültigen HTML-Code auf der Standardausgabe aus – die hier genau dem Inhalt der Datei `webseite.html` entspricht. Wir erstellen dazu folgendes in Listing 6.7 gezeigtes Skript unter `cgi-bin/cgi1.py`:

```
#!/usr/bin/python
print("Content-type: text/html\n")
print("<TITLE>CGI Test</TITLE>")
print("<H1>Ein ganz einfaches CGI-Skript</H1>")
print("<P>Hallo Welt</P>")
```

Listing 6.7. Ein einfaches CGI-Skript

Wird dieses Skript nun durch einen Webbrowser mit der URL

`http://localhost/cgi-bin/cgi1.py`

abgerufen so geschieht, im Gegensatz zum Abrufen der oben beschriebenen HTML-Datei, folgendes: Dieses CGI-Skript wird vom Webserver ausgeführt, und die vom Skript produzierte Ausgabe auf `stdout` wird an den Client, in unserem Fall: den Webbrowser, weitergeleitet. Abbildung 6.5 zeigt die Ausgabe des Webbrowsers:

Abb. 6.5. Von einem einfachen CGI-Skript erzeugte Webseite

Aufgabe 6.13

Verwenden Sie `htmlgen`, um die in Abbildung 6.5 gezeigte Webseite zu erzeugen.

6.5.4 Komplexere CGI-Skripte

Wir stellen hier ein einfaches Beispiel vor, in dem ein CGI-Skript verwendet wird, um eine Benutzereingabe zu verarbeiten. Diese Eingabe wird in HTML üblicherweise mit einem `<FORM>`-Element realisiert. Man könnte zwar diese `<FORM>`-Elemente relativ einfach direkt konstruieren, wenn man sich in HTML auskennt, einfacher geht es jedoch in Python, wenn man das oben vorgestellte Modul `htmlgen` verwendet. Es stellt den Konstruktor `Form` (der gleich lautenden Klasse „`Form`“) zur Verfügung, mit der man HTML-Elemente erzeugen kann, die eine Benutzereingabe erlauben.

Das in Listing 6.8 gezeigte Code-Beispiel erzeugt mit Hilfe des `htmlgen`-Konstrukors `Select` eine Selektionsliste, bestehend aus allen Dateien des aktuellen Verzeichnisses (erzeugt mit `os.listdir()`), aus der der Benutzer eine auswählen kann. `Select` erwartet also zwei Argumente: zum einen eine Liste, die die Werte der Auswahlliste enthält, und zum anderen den Namen des Selektionsfeldes – über diesen Namen kann man später auf den Wert dieses Feldes zugreifen, etwa in einem CGI-Skript. Das über die Form referenzierte CGI-Skript ist genau das in Listing 6.9 (auf Seite 189) gezeigte.

```
import htmlgen, os
doc = htmlgen.Document(title='Selektionsliste')
f   = htmlgen.Form(method='POST', \
                   url='http://localhost/cgi-bin/cgi2.py')
sel = htmlgen.Select(name='datei')
for d in os.listdir('.'): sel.create_option(d)
f.append(sel)
f.append(htmlgen.Division(htmlgen.SubmitButton("Submit")))
doc.append_body(f)
print(doc)
```

Listing 6.8. CGI-Skript, das die FORM-Daten verarbeitet

Die Ausgabe dieses Skripts erzeugt gültigen HTML-Code, der in einem Web-Browser wie in Abbildung 6.6 gezeigt dargestellt wird.

Wie man sieht, erzeugt `HTMLgen` hierbei automatisch einen „Send“-Button[4], mit dem man die eingegebenen Daten an den Webserver schicken kann. In obigem Bei-

4 Als „Buttons“ bezeichnet man die graphischen Metaphern für Schalt-Knöpfe in graphischen Benutzeroberflächen.

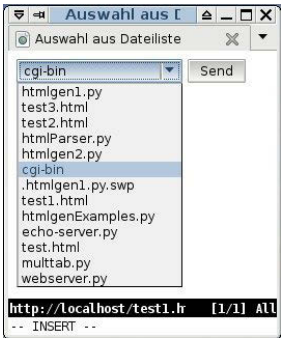


Abb. 6.6. Darstellung der durch das obige Skript erzeugten Ausgabe in einem Web-Browser. Verwendet wurde ein `Select`-Element, umgeben von einem `Form`-Element. `HTMLgen` erzeugt dabei automatisch einen „Send"-Button, mit dem der Benutzer die eingegebenen Werte an den Webserver schicken kann.

spiel werden diese Daten vom Webserver an ein CGI-Skript (das ebenfalls in Python programmiert wurde) weitergeleitet. Die Adresse dieses CGI-Skripts, nämlich

```
http://localhost/cgi-bin/cgi2.py
```

wurde dem `Form`-Konstruktor als Argument mit übergeben. Die meisten Web-Server verlangen übrigens, dass die CGI-Skripte in dem Unterverzeichnis `cgi-bin` gespeichert werden.

Bleibt nur noch die Frage, wie das CGI-Skript `cgi2.py` auf die Benutzereingaben des Selektionsfeldes zugreifen kann. Importiert man das Modul `cgi` in Python, so kann mittels `cgi.FieldStorage()` ein Dictionary-ähnliches Objekt erzeugt werden, das alle durch das `<FORM>`-Element an das CGI-Skript übergebenen Daten[5] enthält. Dieses `FieldStorage`-Objekt kann dann über den Namen der Felder indiziert werden.

Listing 6.9 ist ein einfaches Beispiel für ein CGI-Antwortskript `cgi2.py`, das auf Daten reagiert, die durch obiges `<Form>`-Element gesendet wurden.

```
import cgi, htmlgen
form = cgi.FieldStorage()
doc = htmlgen.Document(title='Benutzereingabe')
dateiname = form['datei'].value
ausgabe = 'Sie haben die Datei ' +dateiname +' gewaehlt'
doc.append_body(htmlgen.Paragraph(ausgabe))
print(doc)
```

Listing 6.9. CGI-Skript, das die FORM-Daten verarbeitet

5 Die übermittelten Daten befinden sich größtenteils in bestimmten Umgebungsvariablen, teilweise aber auch in der Standardeingabe; im CGI-Standard sind entsprechende Details genau festgelegt. Verwendet man das Python-Modul `cgi` hat das den großen Vorteil, dass man sich um diese Details nicht zu kümmern braucht.

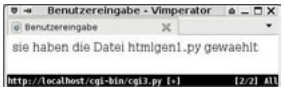


Abb. 6.7. Darstellung der durch das CGI-Skript in Listing 6.9 erzeugten Ausgabe in einem Web-Browser.

Der Selektionsliste im aufrufenden HTML-Dokument wurde der Name `'datei'` gegeben und somit kann über

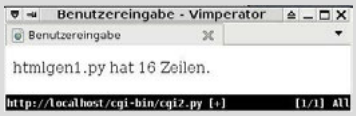
```
form['datei'].value
```

auf den Namen der ausgewählten Datei zugegriffen werden. Abbildung 6.7 zeigt eine Ausgabe dieses Skript bei entsprechender Wahl des Benutzers.

Aufgabe 6.14

Erweitern Sie das eben vorgestellte Skript folgendermaßen:

Dem Benutzer soll die Zeilenlänge der gewählten Datei im Webbrowser ausgegebenen werden; die Ausgabe soll ähnlich wie im rechts gezeigten Screenshot aussehen.



Aufgabe 6.15

Programmieren Sie eine kleine Web-Applikation zur Berechnung des Binomialkoeffizienten. Der Benutzer soll in zwei Textfelder zwei Zahlen n und m eingeben. Nach Drücken des „Send“-Buttons, soll ein cgi-Skript den Wert des Binomialkoeffizienten „n über m“ ausgeben.