

Einführung in Python

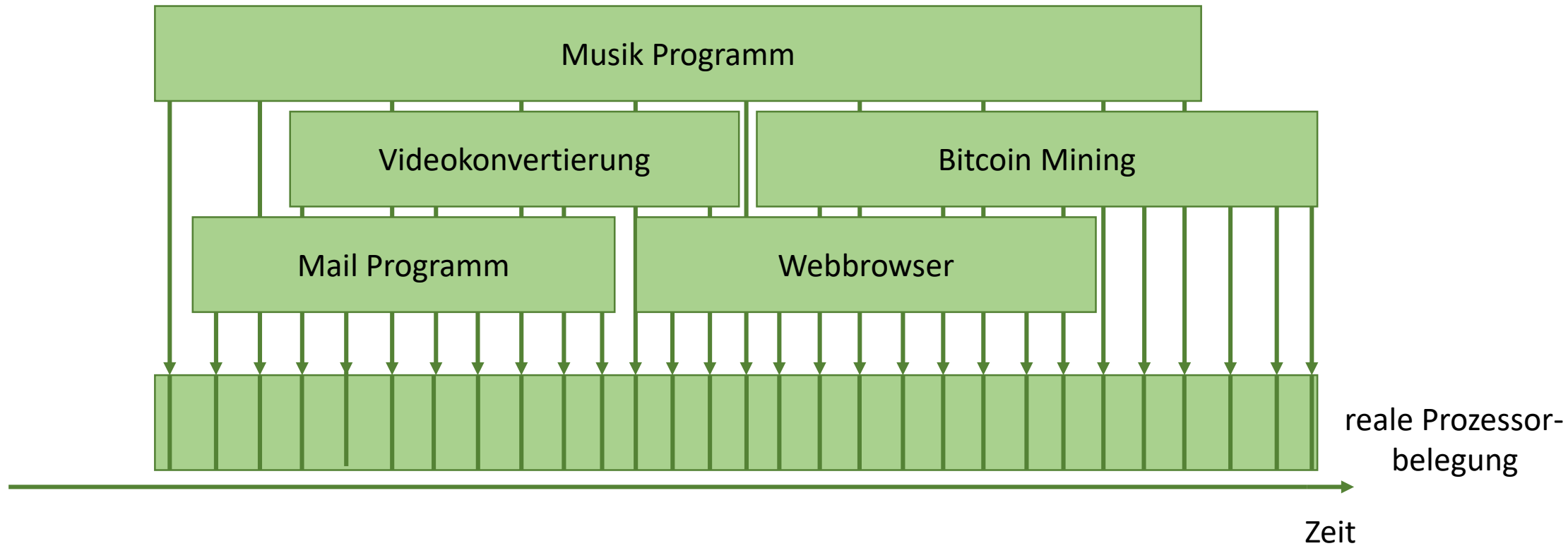
8. Vorlesung



CGI-Skripte und Webprogrammierung

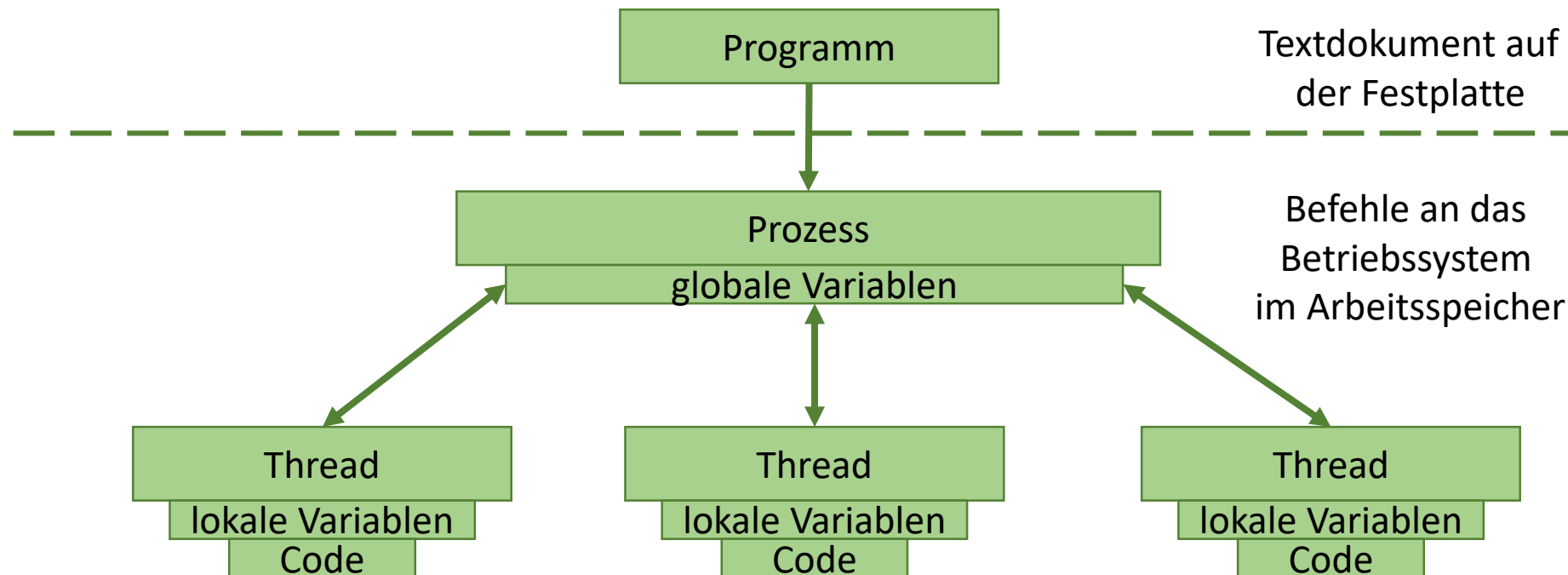


Wiederholung letztes Mal



Wiederholung letztes Mal

- Threads sind (Teil)Prozesse und Prozesse sind Programme in Ausführung
- Threads werden vom Betriebssystem (scheinbar) gleichzeitig ausgeführt



Wiederholung letztes Mal

- In Python gibt es zwei verschiedene Implementierungen für Threads:
 - Das Modul *thread* (Python 2) bzw. *_thread* (Python 3) betrachtet Threads als Funktionen
 - Das Modul *threading* implementiert Threads als eigenständige Objekte und erlaubt komplexere parallele Verarbeitungen



Wiederholung letztes Mal

- Um mehrere Threads zu synchronisieren gibt es verschiedene Objekte:
 - Lock-Objekte schützen globale Variablen vor zeitgleichem Zugriff durch mehrere Threads
 - Event-Objekte lassen Threads auf das Eintreten bestimmter Ereignisse warten
 - Durch Condition-Objekte können sich Threads gezielt ansprechen, die sich gemeinsame Ressourcen teilen
 - Mit Queues kann eine bestimmte Anzahl an Threads gesammelt werden die nach und nach verschiedene Aufgaben abarbeiten

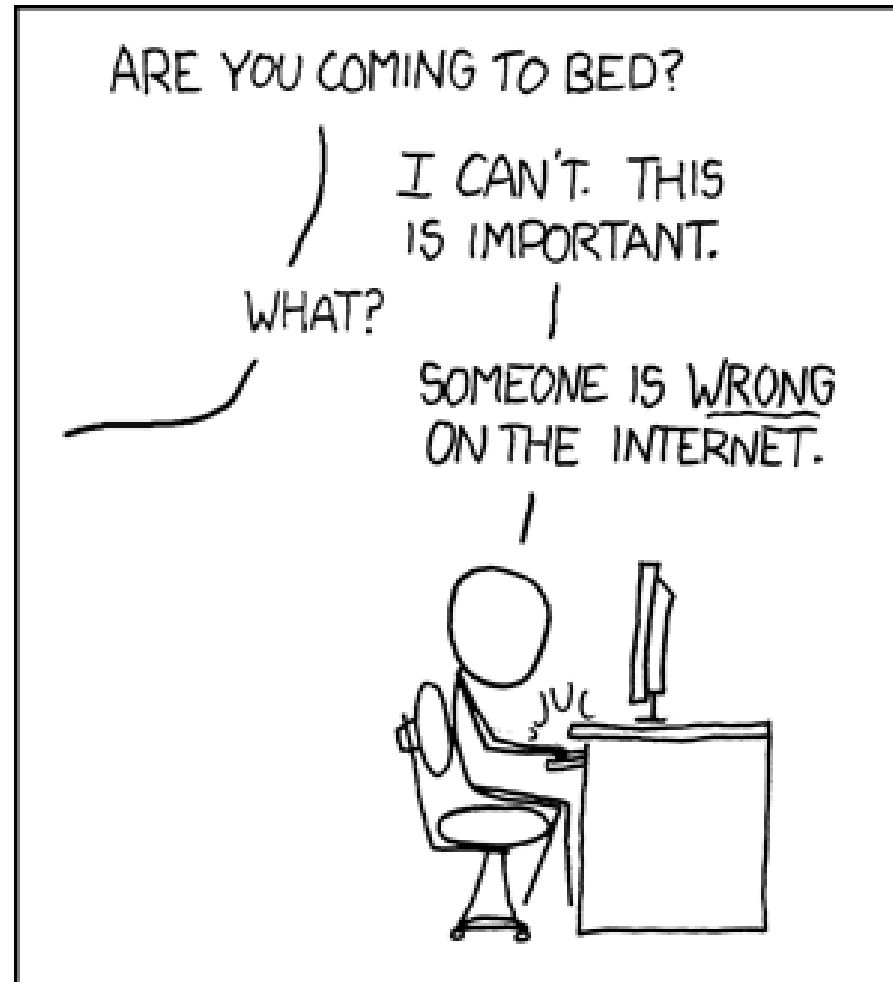


Wiederholung letztes Mal

- Echte Prozesse werden automatisch vom Betriebssystem auf alle vorhandenen Prozessoren gleichmäßig verteilt
- Kinderprozesse teilen sich keinen gemeinsamen globalen Namensraum mit ihrem Elternprozess und *können* nach seiner Beendigung weiterlaufen
- Das Modul *multiprocessing* ist fast exakt wie *threading* aufgebaut, aber statt Threads erzeugt es echte Prozesse



CGI-Programmierung und Webprogrammierung

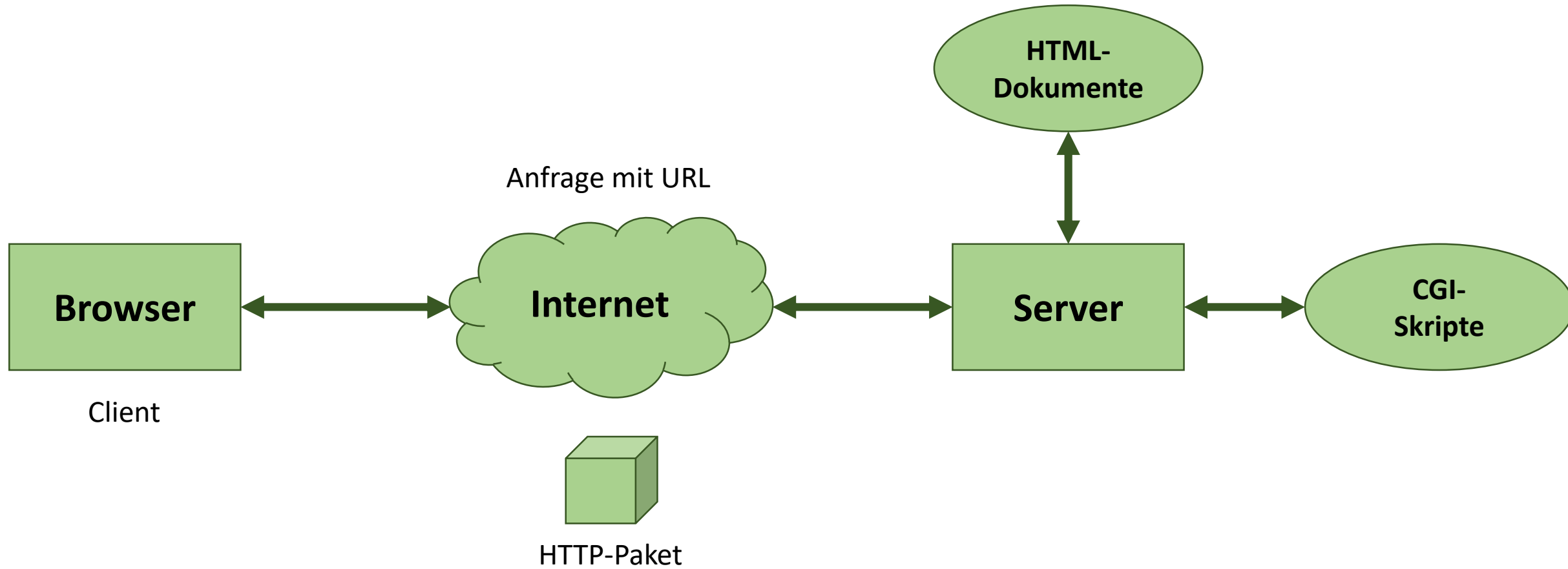


CGI-Programmierung

- CGI (Common Gateway Interface) ist ein Standard um ein Programm auf einem Server zu starten und Daten von diesem mit dem Client auszutauschen
- Ein Server mit CGI-Unterstützung stellt eine Laufzeitumgebung mit Umgebungsvariablen und IO-Kanälen bereit, in welcher dann bestimmte Programme (z. B. Pythonskripte) laufen können
- Wird unter Anderem genutzt für:
 - Dynamische Erzeugung von Webseiten
 - Verarbeitung von Daten die über ein HTML-Formular eingegeben wurden
 - Zugriff auf Onlinedatenbanken (Suchmaschinen, Wetterdienste, etc.)
 - Kommunikationsplattformen (Foren, Chats)



CGI-Programmierung



CGI-Programmierung

- Wird ein CGI-Skript anstelle einer HTML-Seite angefordert, so wird dieses Programm auf dem Server gestartet und ist dann für die Zusammenstellung der zurückgesendeten HTML-Pakete verantwortlich
- CGI ist ein einheitlicher und plattformunabhängiger Standard, deswegen können CGI-Skripte in einer beliebigen Sprache geschrieben sein
- Meist befinden sich CGI-Skripte auf dem Server unter cgi-bin/
- Nachteile: CGI-Ausführung ist relativ langsam, da für jeden Aufruf ein eigener Prozess gestartet wird → Wird auf hochfrequentierten Seiten nicht mehr genutzt



Aufbau eines einfachen CGI-Skripts

- Besteht aus mindestens zwei Teilen:
 - Spezifizierung des Interpreters

```
#!/usr/bin/python3
```

- *print()*-Anweisungen mit denen ein HTTP-Paket über die Standardausgabe *sys.stdout* ausgegeben und erzeugt wird

```
from time import localtime
print('Content-Type: text/html; char-set-utf-8\n')
print('<html> <body>\n')
print('<h2>Die aktuelle Uhrzeit</h2>\n')
print(f'Es ist {localtime()[3]} Uhr und {localtime()[4]} Minuten.\n')
print('</body> </html>\n')
```



HTTP-Server

- Zum Testen eines CGI-Skripts braucht man einen eigenen Webserver
- Es gibt freie und sehr populäre Webserver für Unix-Systeme (Apache) und auch für Windows-Systeme (Xitami/X5)
- Man kann auch in Python relativ einfach einen eigene HTTP-Server schreiben und zum Testen lokal auf dem eigenen Rechner starten



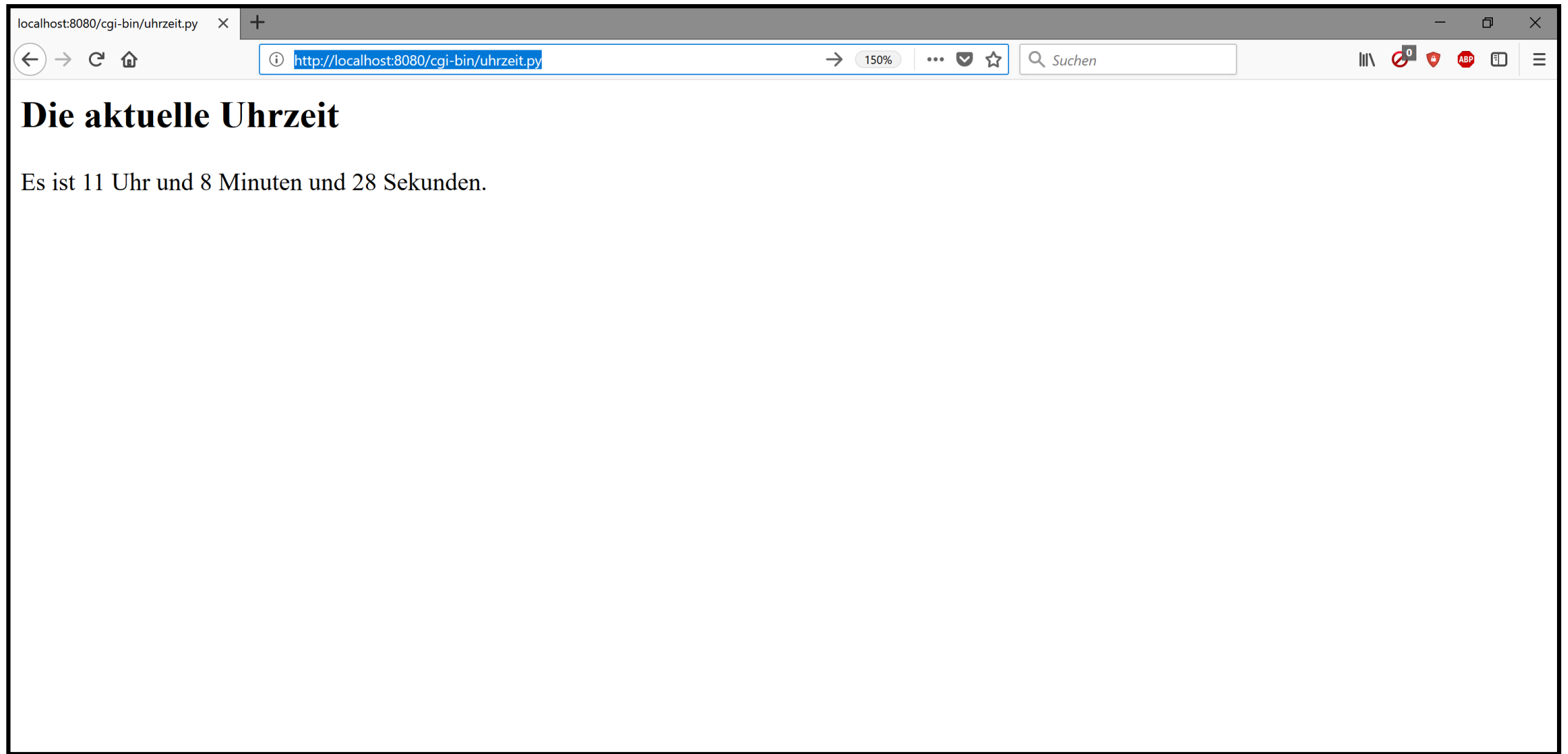
Eigener Python HTTP-Server

- Zuerst einen eigenen Ordner anlegen z. B.
c:\python_server\ mit c:\python_server\cgi-bin\
bzw.
/home/user/python_server/ mit /home/user/python_server/cgi-bin/
- Das Serverskript anlegen und starten, z.B. c:\python_server\httpd.py :

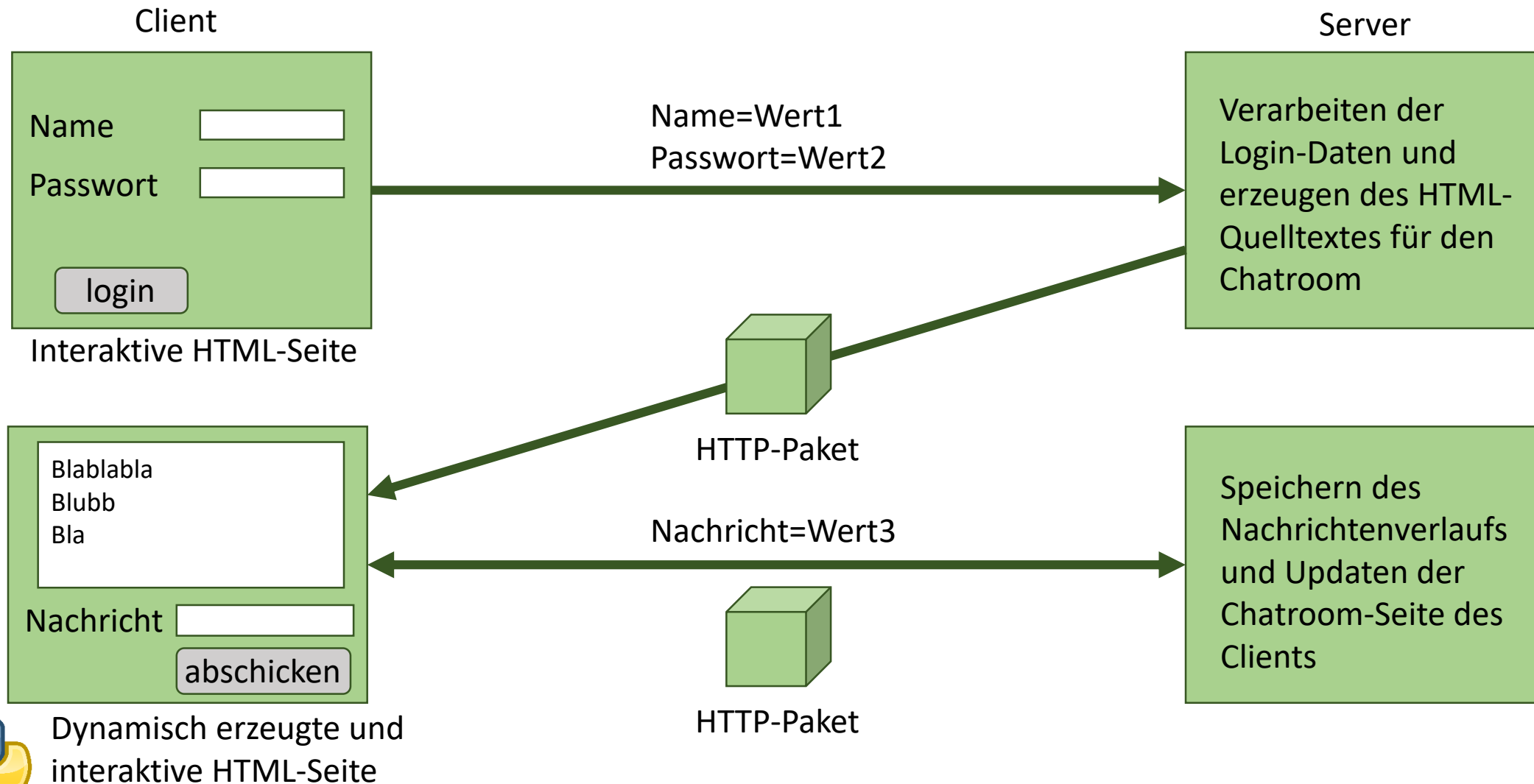
```
from http.server import HTTPServer, CGIHTTPRequestHandler  
  
serveradresse = ('', 8080)  
server = HTTPServer(serveradresse, CGIHTTPRequestHandler)  
  
server.serve_forever()
```



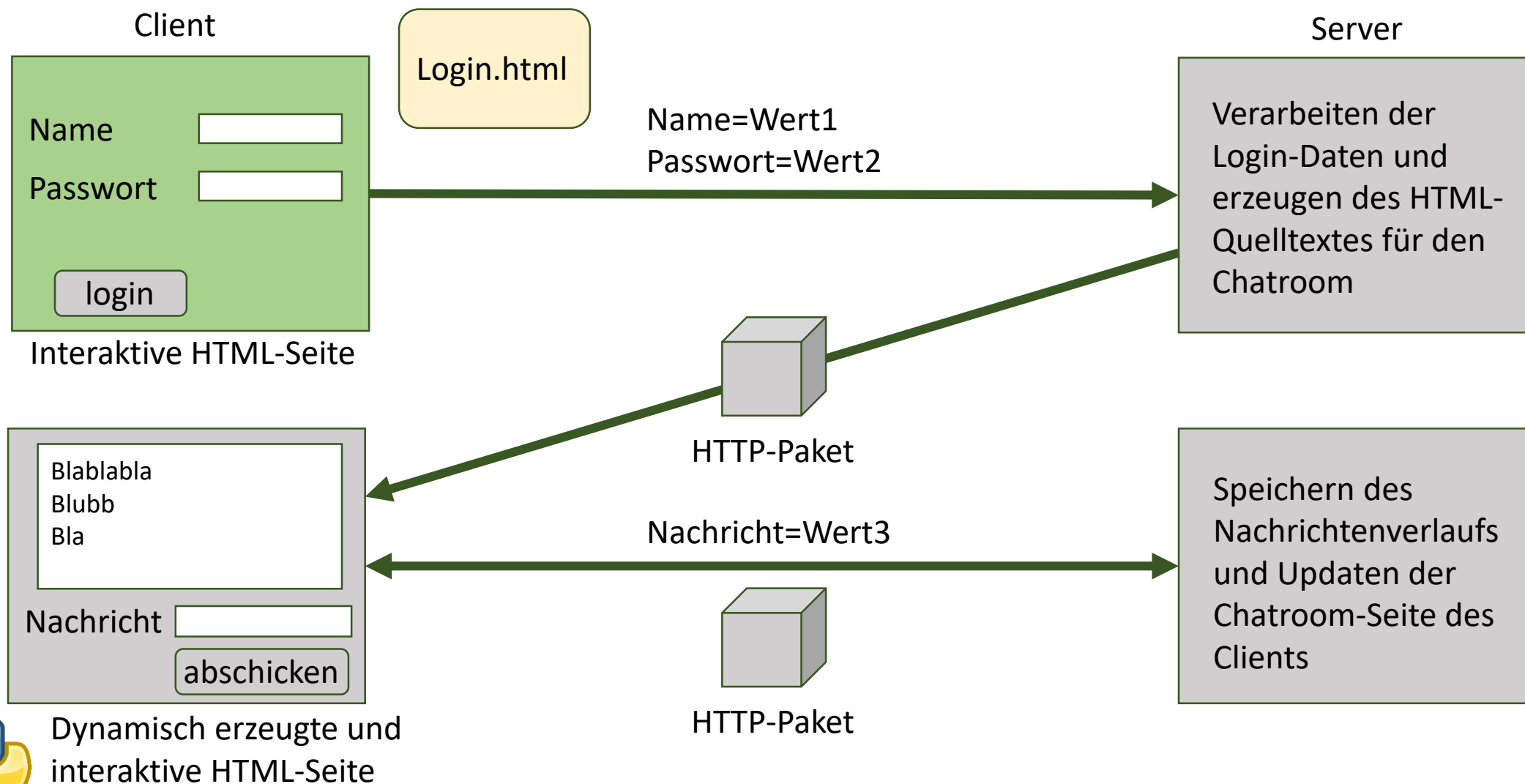
Eigener Python HTTP-Server



Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login



Crashkurs HTML

`<DIV>Q: HOW DO YOU ANNOY A WEB DEVELOPER?`



Crashkurs HTML

- HTML (Hypertext Markup Language) ist DIE Sprache des Internets
- Sie dient zur Strukturierung digitaler Dokumente, mit welcher Text, Bilder, Videos, etc. online via Webseiten dargestellt werden können
- HTML-Dokumente bestehen aus Elementen (ähnlich zu den Widgets aus der GUI-Programmierung), welche wiederum durch so genannte *Tags* repräsentiert werden

`<tagname> Inhalt des Tags </tagname>`

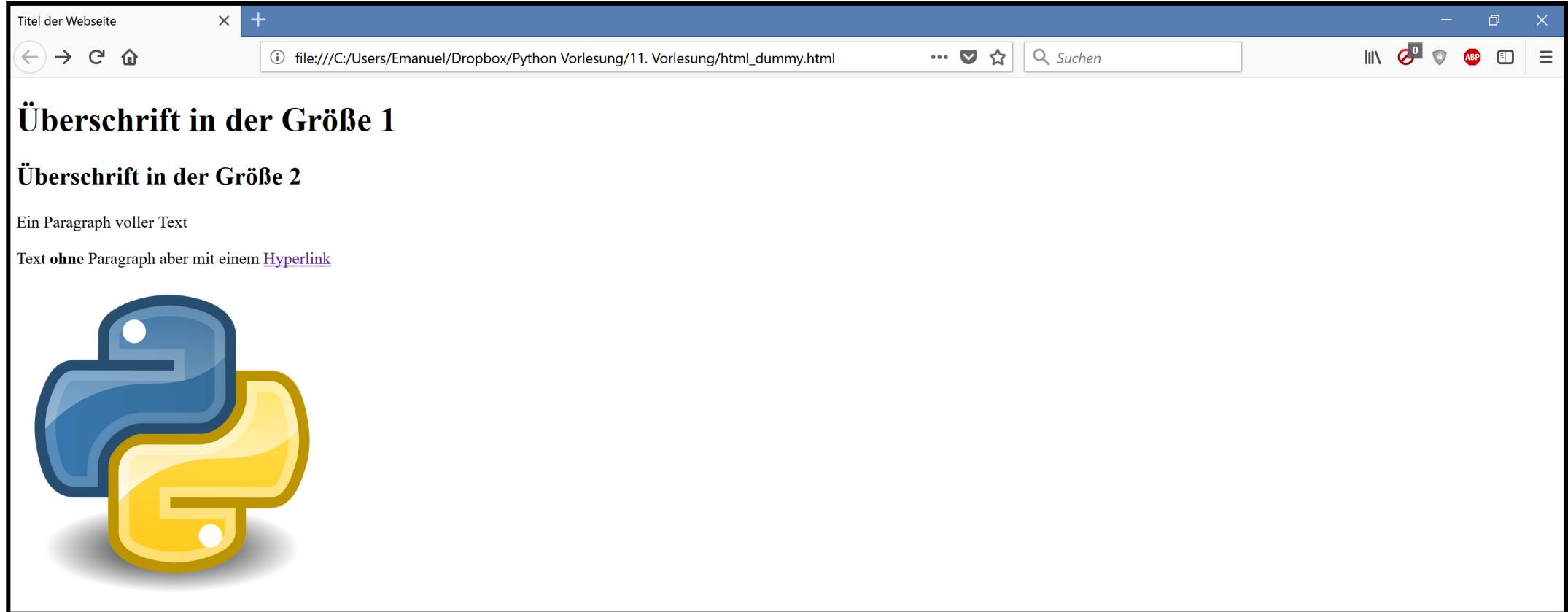


Crashkurs HTML

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
  <head>
    <title> Titel der webseite </title>
  </head>
  <body>
    <h1> Überschrift in der Größe 1 </h1>
    <h2> Überschrift in der Größe 2 </h2>
    <p> Ein Paragraph voller Text </p>
    <p> und noch mehr Text </p>
    Text <b>ohne</b> Paragraph
    aber mit einem
    <a href="www.python.org"> Hyperlink </a>
    <br>
    <br>
    <image src="./python.png" height="300" weight="300" />
  </body>
</html>
```



Crashkurs HTML

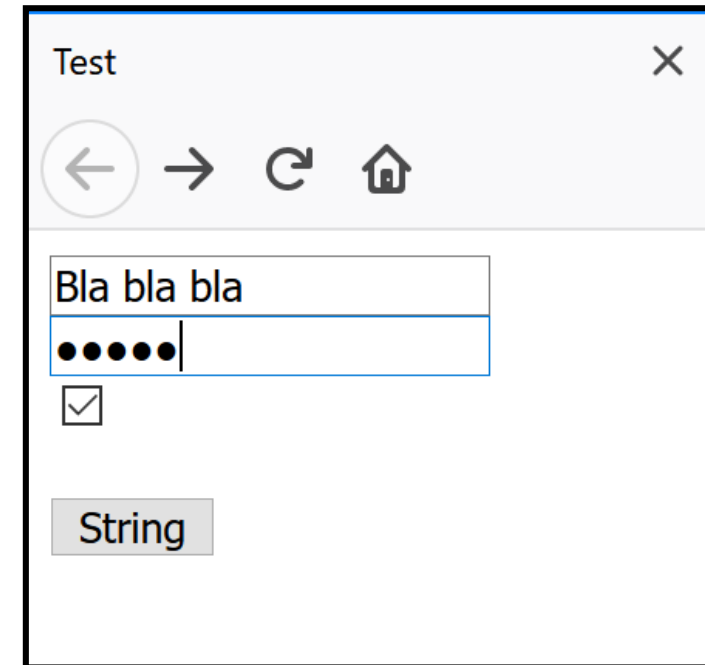


- Unter www.w3schools.com findet man eine gute Übersicht über alle Grundlegenden Tags, mit einigen Codebeispielen

Crashkurs HTML – Kommunikation über interaktive Webseiten

- Über ein HTML-Formular tag kann man relativ einfach Informationen vom Client zu einem CGI-Skript auf einem Server übertragen
- Diese Formulare können verschiedene Dateneingabeelemente enthalten (ähnlich zu den Widgets)

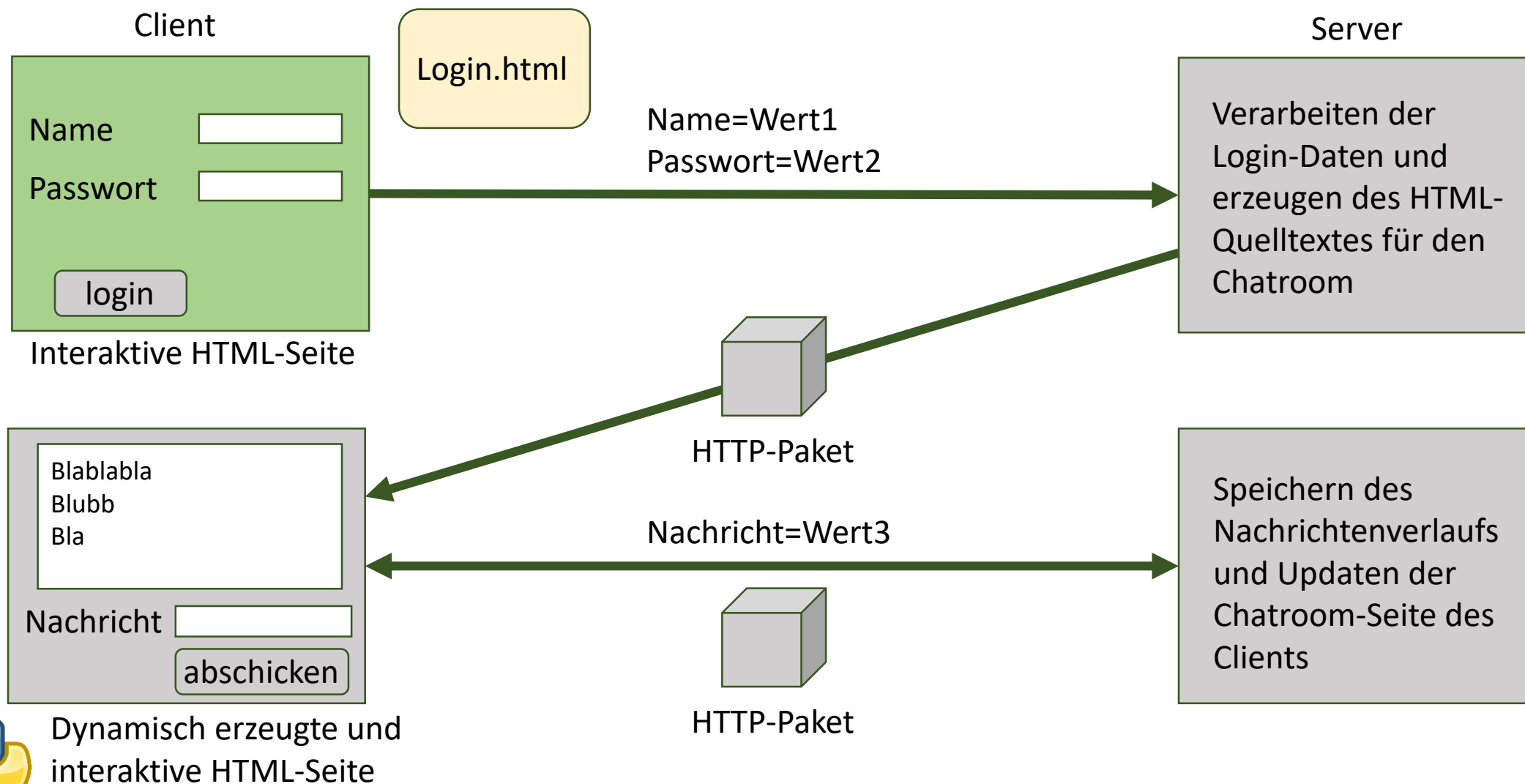
```
<html>
  <body>
    <form method="get" action="CGI-Skript.py">
      <input type="text" name="Var1">
      <input type="Password" name="Var2">
      <input type="checkbox" name="Var3" value="wert1">
      <input type="hidden" name="Var4" value="wert2">
      <input type="submit" value="String">
    </form>
  </body>
</html>
```



The screenshot shows a web browser window titled "Test" with a close button (X) in the top right corner. The browser's address bar contains navigation icons: back, forward, refresh, and home. Below the address bar, the rendered HTML form is displayed. It features a text input field containing "Bla bla bla", a password input field represented by five dots, a checked checkbox, and a submit button labeled "String".



Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
  <head>
    <title> Mein Python Chatroom </title>
  </head>
  <body>
    <h1> willkommen im Python Chatroom </h1>
    <form method="get" action="http://localhost:8080/cgi-bin/login.py">
      <pre>
        Benutzername:   <input type="text" name="user" size="15">
        Passwort:       <input type="password" name="pass" size="15">
                        <input type="submit" value="login">
      </pre>
    </form>
  </body>
</html>
```

Login.html



Beispiel: Chatroom mit Login

Mein Python Chatroom

file:///C:/Users/Emanuel/Dropbox/Python Vorlesung/11. Vorlesung/login.html

Suchen

Benutzername:

Emanuel

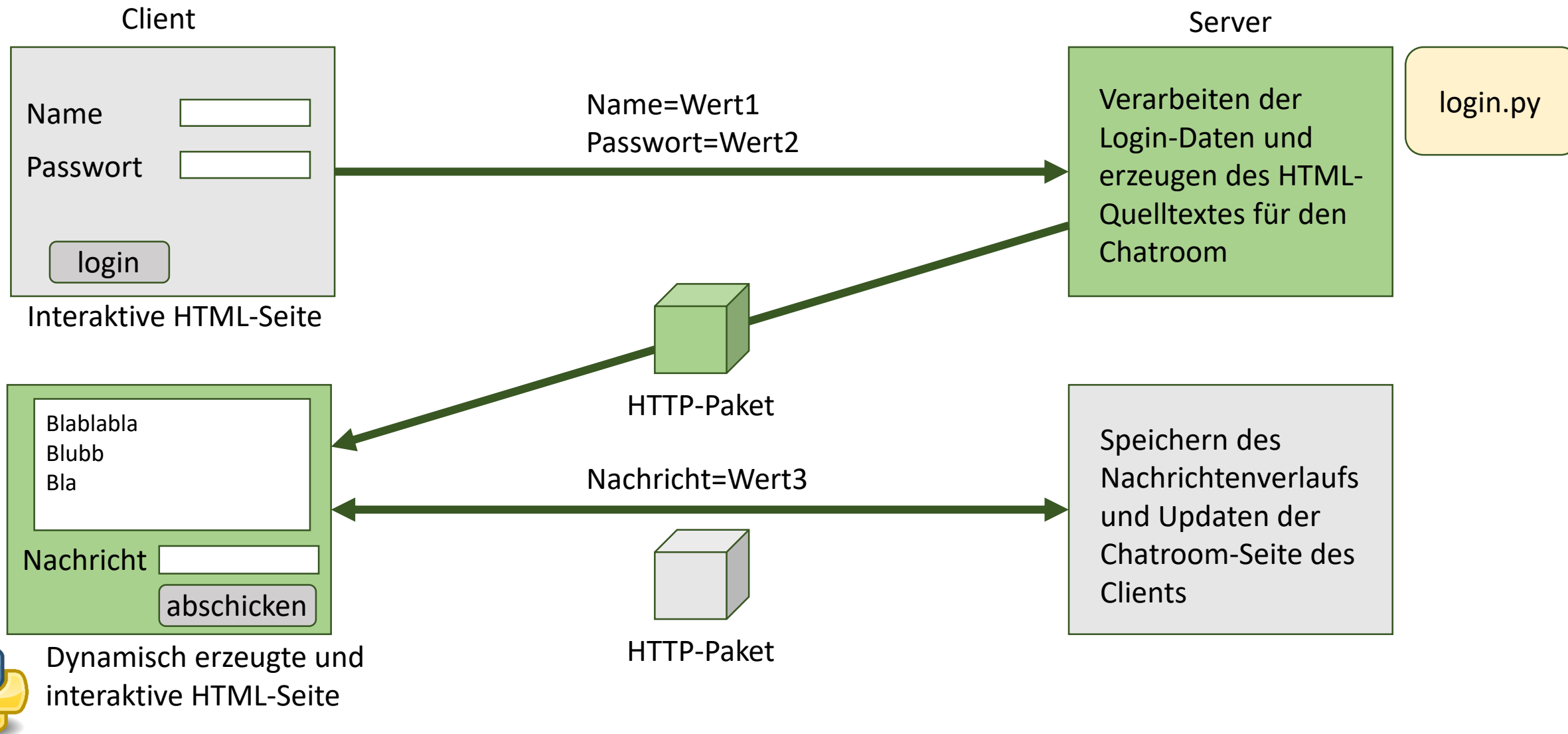
Password:

•••••

login



Beispiel: Chatroom mit Login



Verarbeitung von Eingabedaten im CGI-Skript

- Die von einem HTML-Formular übermittelten Daten können mittels eines *FieldStorage*-Objekts des *cgi* Moduls verwendet werden

```
import cgi  
  
formData = cgi.FieldStorage()
```

- Diese Objekte funktionieren ähnlich wie Dictionaries nach dem *Schlüssel:Wert*-Prinzip

```
formData.getvalue('variablenname', default)
```

- Alle Werte sind als Strings abgespeichert und jede Variable hat entweder genau einen String als Wert oder eine Liste von Strings
- Man erhält *None* wenn man einen Variablennamen abfragt der nicht durch das HTML-Formular übermittelt wurde (oder optional *default*)

```
formData.keys()
```



Beispiel: Chatroom mit Login

```
#!C:\Program Files\Python36\python.exe

import cgi
import cgitb

cgitb.enable()

formData = cgi.FieldStorage()
name = formData.getvalue('user')
pw = formData.getvalue('pass')

knownUsers = {'Emanuel': '12345', 'Monty': 'Python'}

if name not in knownUsers or pw != knownUsers[name]:
    print('Content-Type: text/html; char-set=utf-8\n')
    print('<html> <body>\n')
    print('<h2>Benutzername oder Passwort falsch!</h2>\n')
    print('</body> </html>\n')
```

login.py



Beispiel: Chatroom mit Login

```
else:
    print('<!DOCTYPE html>')
    print('Content-Type: text/html; char-set-utf-8\n')
    print('<html>')
    print('<head>')
    print('<title> Mein Python Chatroom </title>')
    print('</head>')
    print('<body>')
    print(f'<h1>willkommen im Python Chatroom, {name}!</h1>')
    print('<br>')
    print('<form method="post" action="chat.py">')
    print(f'<input type="hidden" name="user" value="{name}">')
    print('<input type="Text" name="beitrag" size="140"')
    print('maxlength="140">')
    print('<input type="submit" value="Absenden">')
    print('</form>')
```

login.py



Beispiel: Chatroom mit Login

login.py

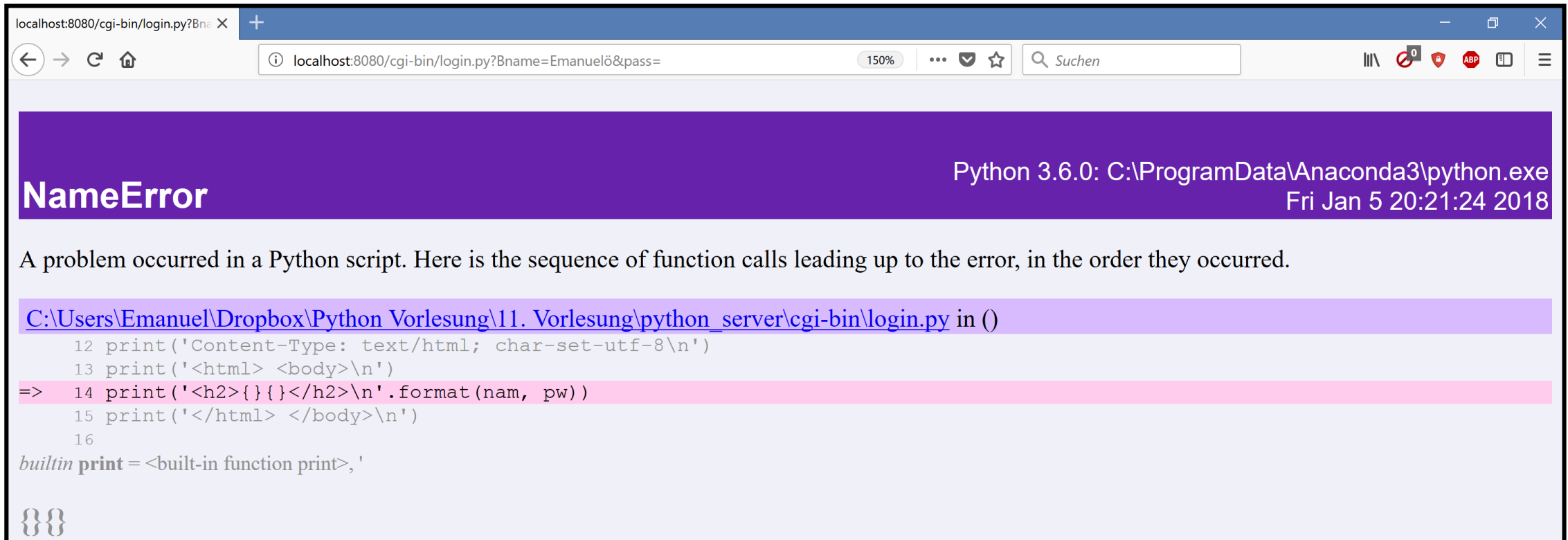
```
print('<br>')
If os.path.exists('C:\\python_server\\chatlog.txt'):
    chatlog = open('C:\\python_server\\chatlog.txt').readlines()
    if len(chatlog) > 0:
        print('<p style="border:3px border-color:#008000 padding: 1em;">')
        for line in chatlog:
            print(line)
        print('</p>')
print('</body>')
print('</html>')
```



Fehler in einem CGI-Skript finden

- Wird ein fehlerhaftes CGI-Skript über einen Browser aufgerufen, erhält man nur eine leere Seite, da das Skript die Ausgabe abbricht

```
import cgi
cgi.enable()
```



The screenshot shows a web browser window with the address bar displaying `localhost:8080/cgi-bin/login.py?Bname=Emanuel&pass=`. The browser's developer console shows a `NameError` message. The error message includes the Python version `Python 3.6.0: C:\ProgramData\Anaconda3\python.exe` and the timestamp `Fri Jan 5 20:21:24 2018`. Below the error message, a text block states: "A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred." This is followed by a stack trace showing the file `C:\Users\Emanuel\Dropbox\Python Vorlesung\11. Vorlesung\python_server\cgi-bin\login.py` and the line numbers 12 through 16. The code for lines 12 through 15 is displayed, with line 14 highlighted in pink. Line 14 is `14 print('<h2>{{}}</h2>\n'.format(nam, pw))`. Line 16 is `16`. The stack trace also shows `builtin print = <built-in function print>, '`. The browser's address bar shows the URL `localhost:8080/cgi-bin/login.py?Bname=Emanuel&pass=`. The browser's status bar shows the Python logo and the text `{{}}`.

localhost:8080/cgi-bin/login.py?Bname=Emanuel&pass=

Python 3.6.0: C:\ProgramData\Anaconda3\python.exe
Fri Jan 5 20:21:24 2018

NameError

A problem occurred in a Python script. Here is the sequence of function calls leading up to the error, in the order they occurred.

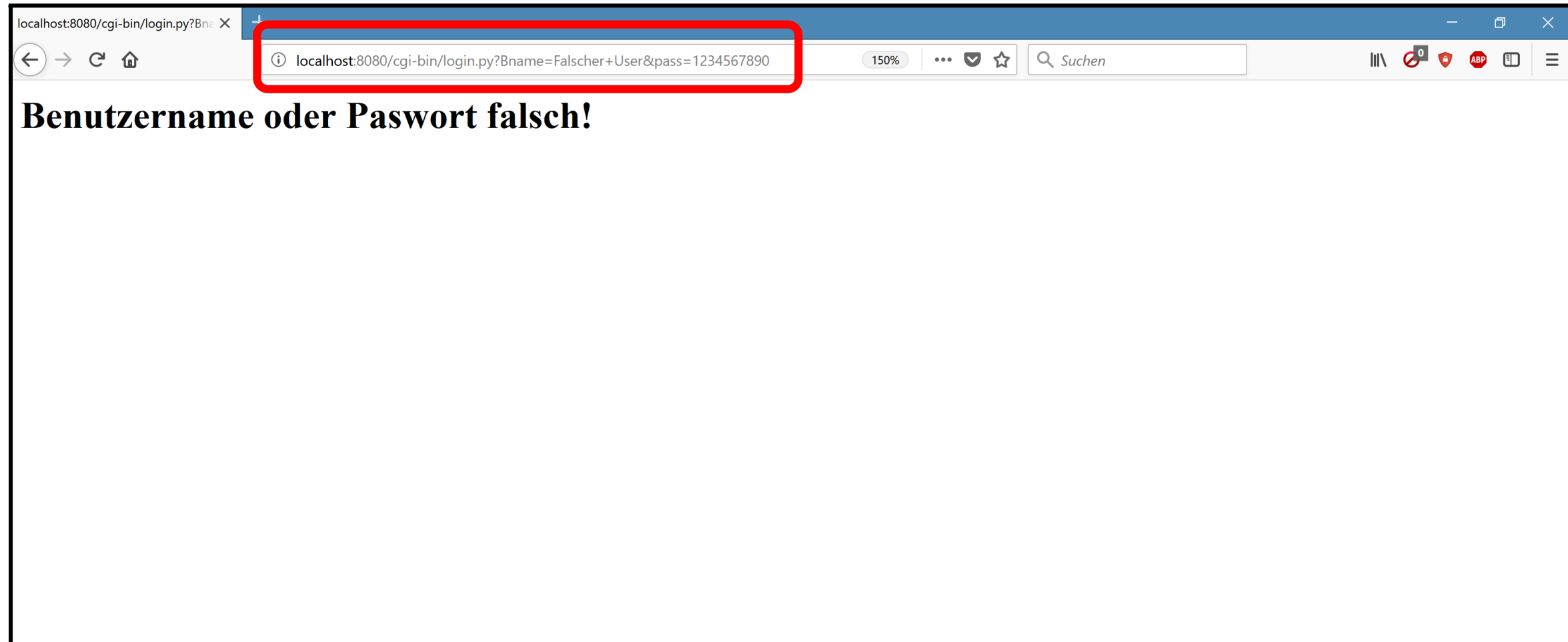
C:\Users\Emanuel\Dropbox\Python Vorlesung\11. Vorlesung\python_server\cgi-bin\login.py in ()

```
12 print('Content-Type: text/html; char-set=utf-8\n')
13 print('<html> <body>\n')
=> 14 print('<h2>{{}}</h2>\n'.format(nam, pw))
15 print('</html> </body>\n')
16
```

builtin print = <built-in function print>, '

{{}}

Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<head>
<title> Mein Python Chatroom </title>
</head>
<body>
  <h1> willkommen im Python Chatroom </h1>
  <form method="get" action="http://localhost:8080/cgi-bin/login.py">
    <pre>
    Benutzername: <input type="text" name="user" size="15">
    Passwort:     <input type="Password" name="pass" size="15">
                  <input type="submit" value="login">
    </pre>
  </form>
</body>
</html>
```

Login.html



HTTP-Anfragemethoden (request modes)

- Das HTTP-Protokoll besitzt verschiedene Anfragemethoden um verschiedene Formen von Informationen vom Client zum Server zu schicken (Formulare, Daten, ...)

Anfragemethode	Funktion
get	Anfordern einer Ressource vom Server, wobei Parameter (z. B. von HTML-Formularen) der URL hinzugefügt werden
post	Senden großer Datenmengen zum Server, wobei diese als extra HTTP-Pakete verschickt werden und <u>nicht</u> über die URL
head	Abfragen des Headers einer Server Ressource um Gültigkeiten von Dateien im Browser-Cache zu überprüfen
put	Hochladen einer Datei auf einen Server ohne das diese von einem POST-Skript verarbeitet werden
delete	Löschen einer Datei vom Server
options	Anzeigen aller zur Verfügung stehenden Methoden des Servers



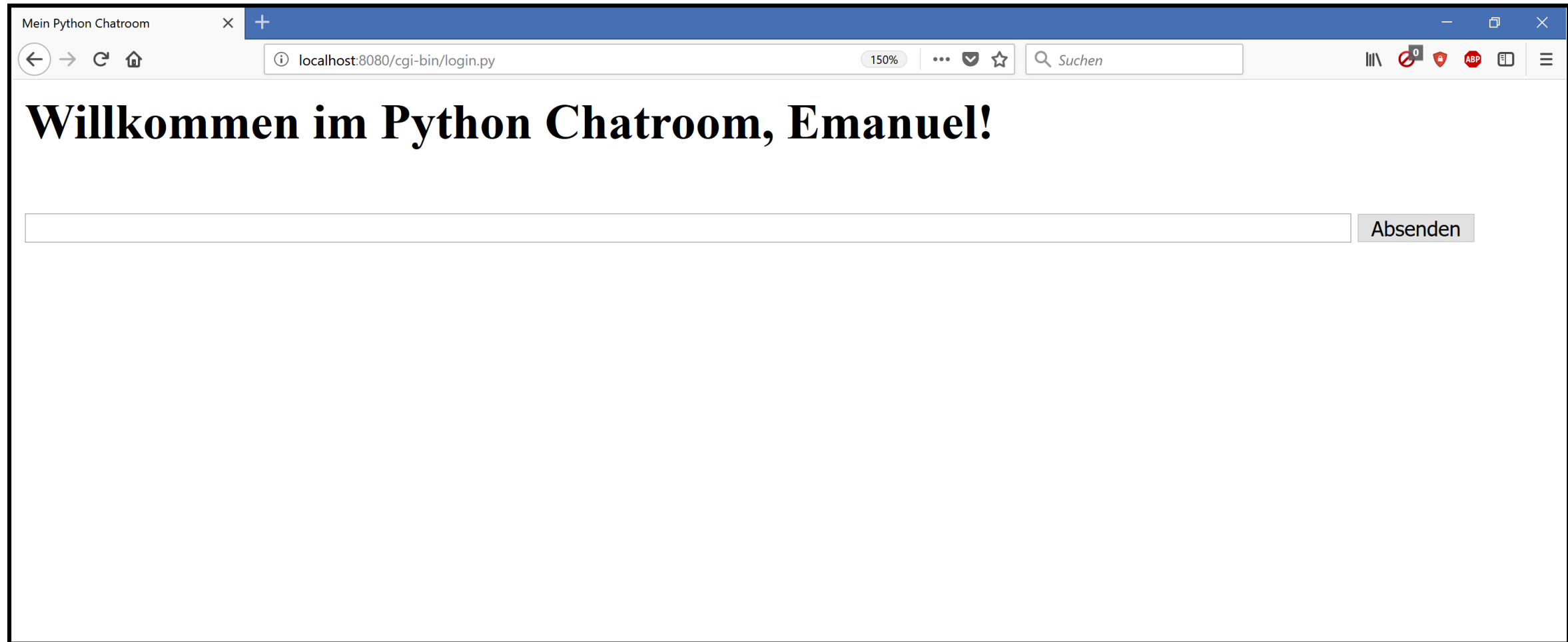
Beispiel: Chatroom mit Login

```
<!DOCTYPE html>
<meta charset="UTF-8">
<html>
<head>
<title> Mein Python Chatroom </title>
</head>
<body>
  <h1> willkommen im Python Chatroom </h1>
  <form method="post" action="http://localhost:8080/cgi-bin/login.py">
    <pre>
    Benutzername: <input type="text" name="user" size="15">
    Passwort:      <input type="Password" name="pass" size="15">
                  <input type="submit" value="login">
    </pre>
  </form>
</body>
</html>
```

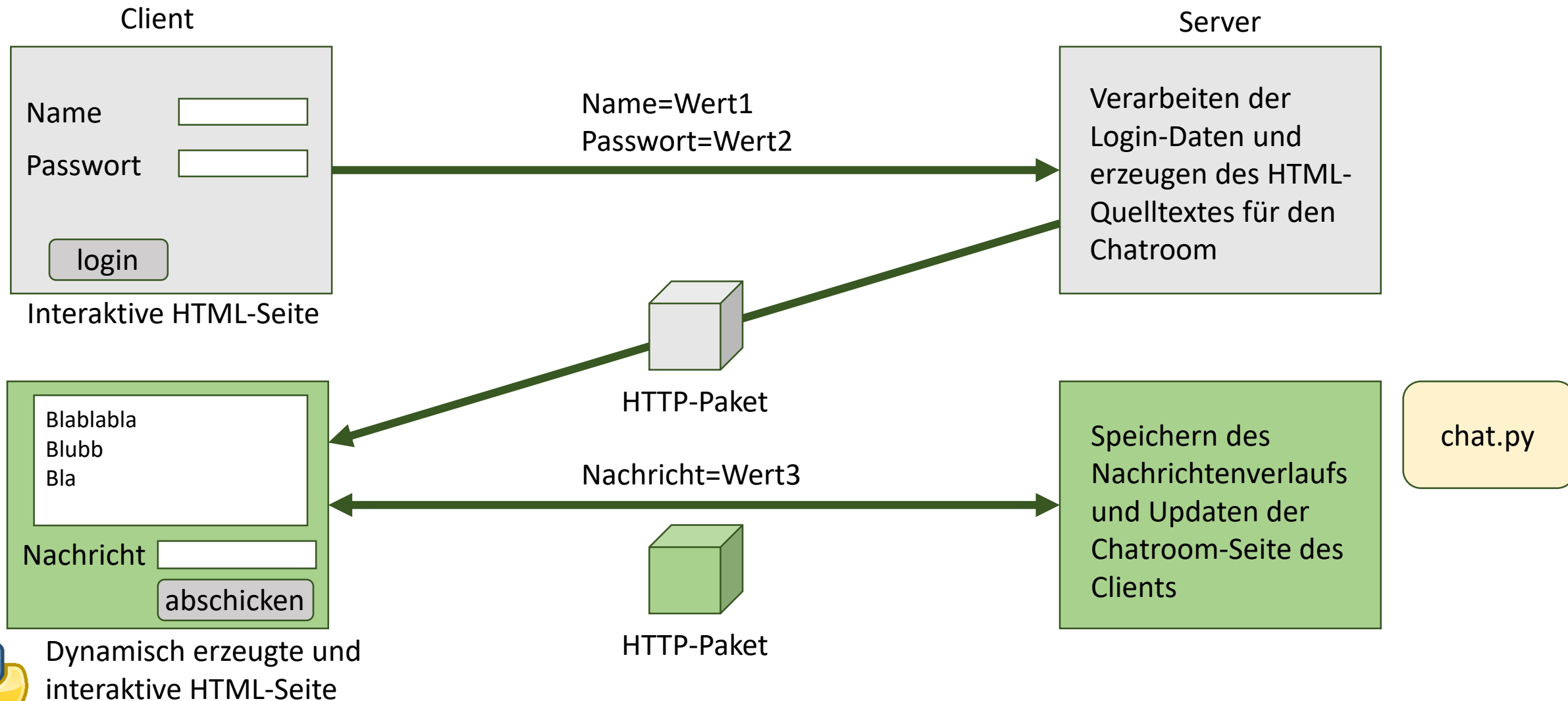
Login.html



Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login

```
#!C:\Program Files\Python36\python.exe
```

chat.py

```
import cgi  
import cgitb
```

```
cgitb.enable()
```

```
formData = cgi.FieldStorage()  
text = formData.getvalue('text')  
name = formData.getvalue('user')
```

```
If text != None:  
    with open('C:\\python_server\\chatlog.txt', 'a') as chatlog:  
        chatlog.write(f'{name} ::: {text}\n<br>\n')
```

```
chatlog = open('C:\\chatlog.txt').readlines()
```



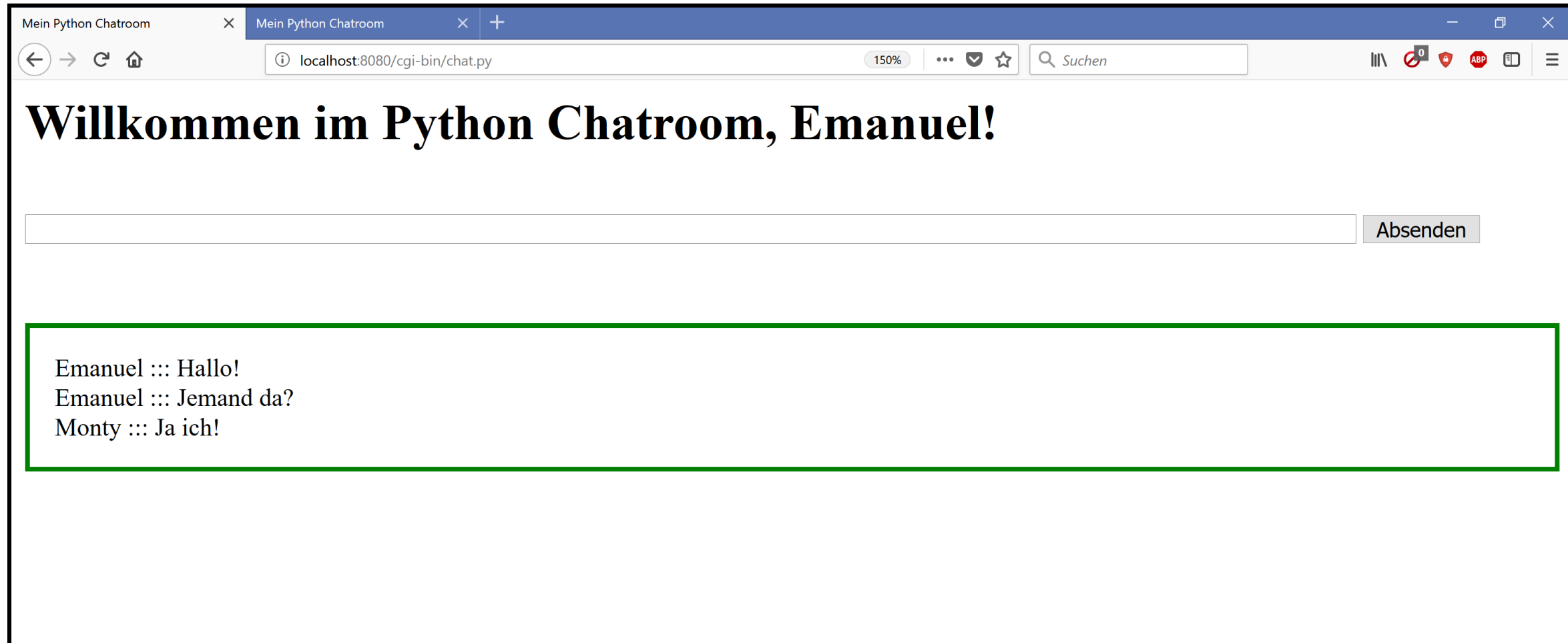
Beispiel: Chatroom mit Login

```
print('<!DOCTYPE html>')
print('Content-Type: text/html; char-set-utf-8\n')
print('<html> <head> <title> Mein Python Chatroom </title>')
print('</head>')
print('<body>')
print(f'<h1>willkommen im Python Chatroom, {name}!</h1>')
print('<br>')
print('<form method="post" action="chat.py">')
print(f'<input type="hidden" name="user" value="{name}">')
print('<input type="Text" name="beitrag" size="140" maxlength="140">')
print('<input type="submit" value="Absenden">')
print('</form>')
print('<br>')
print('<p style="border:3px; border-color:#008000; padding: 1em;">')
for line in chatlog:
    print(line)
print('</p> </body> </html>')
```

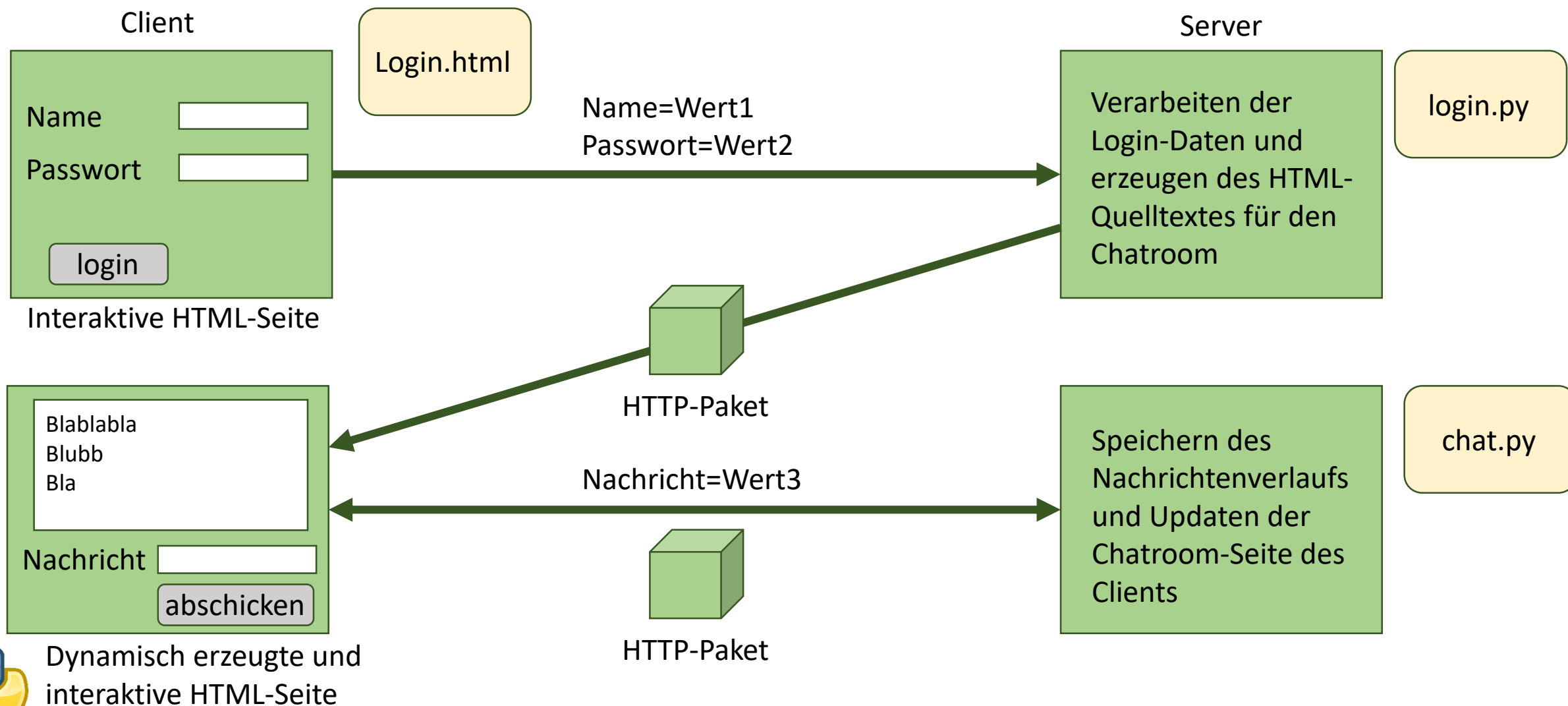
chat.py



Beispiel: Chatroom mit Login



Beispiel: Chatroom mit Login



CGI-Skript mit Cookies

- Cookies sind Informationen die ein Server (meist) in Form einer Textdatei als Schlüssel-Wert-Paare beim Client speichert
- Sie werden entweder im Header einer HTML-Seite mitgeschickt oder per JavaScript auf dem Client Rechner erzeugt
- Meistens enthalten Cookies Informationen zu früheren Besuchen einer Seite und optional ein Verfallsdatum
- Wir verwenden hier nur die *SimpleCookie*-Klasse, welche Cookie-Objekte wie Dictionaries behandelt und nur Strings abspeichert

```
#Setzen eines Cookies
>>> from http.cookies import SimpleCookie
>>> c = SimpleCookie()
>>> c['ID'] = 'Py5th19on36VL'
>>> print(c)
Set-Cookie: ID=Py5th19on36VL
```



CGI-Skript mit Cookies

- Cookies sind Informationen die ein Server (meist) in Form einer Textdatei als Schlüssel-Wert-Paare beim Client speichert
- Sie werden entweder im Header einer HTML-Seite mitgeschickt oder per JavaScript auf dem Client Rechner erzeugt
- Meistens enthalten Cookies Informationen zu früheren Besuchen einer Seite und optional ein Verfallsdatum
- Wir verwenden hier nur die *SimpleCookie*-Klasse, welche Cookie-Objekte wie Dictionaries behandelt und nur Strings abspeichert

```
#Lesen und Abfragen eines Cookies
>>> from http.cookies import SimpleCookie
>>> c = SimpleCookie()
>>> c.load(os.environ['HTTP_COOKIE'])
>>> c['ID'].value
'H5A19BE36'
```



CGI-Skript mit Cookies

```
#!/C:\Program Files\Python35\python.exe

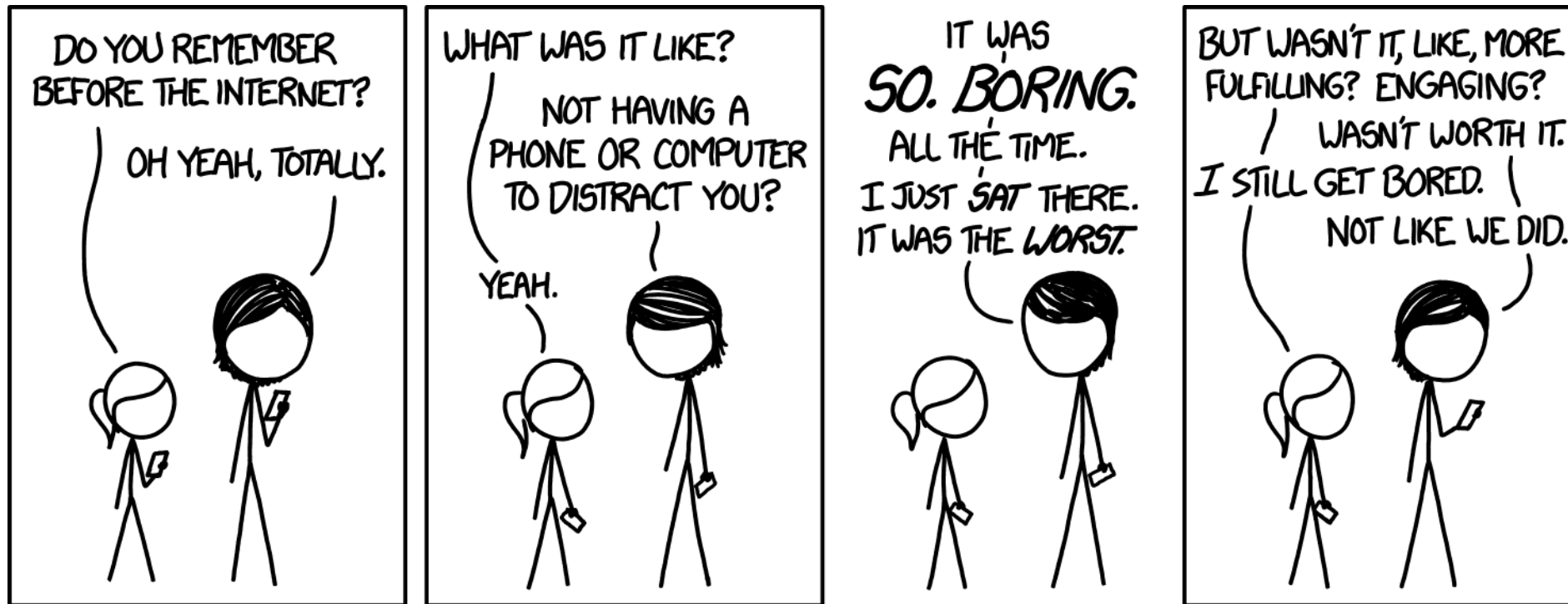
import os
from http.cookies import SimpleCookie

c = SimpleCookie()
try:
    c.load(os.environ['HTTP_COOKIE'])
    c['counter'] = int(c['counter'].value)+1
except:
    c['counter'] = 1

print('<!DOCTYPE html>')
print('Content-Type: text/html; char-set-utf-8\n')
print(f'<meta http-equiv="set-cookie" content="counter={c["counter"].value};">')
print('<html> <head>')
print('<title> Zähler </title>')
print('</head> <body>')
print('<h1>Du besuchst diese Seite zum {c["counter"].value}. Mal!</h1>')
print('</body> </html>')
```



Internet-Programmierung



Internet-Programmierung

- Zur Kommunikation von Rechnern über das Internet existieren verschiedene *Protokolle*
- Für jede Art von Datenkommunikation gibt es ein eigenes Protokoll, welche sich an Referenzmodellen orientieren (RFC – Request for Comments; www.rfc-editor.org)
- Hierbei wird jedes Protokoll hierarchisch einer bestimmten Kommunikationsschicht zugeordnet, z. B. das TCP/IP-Modell

Schicht des TCP/IP-Modells	Protokolle (Beispiele)
Verarbeitung	HTTP, FTP, SMTP, POP3, TELNET
Transport	TCP, UDP
Internet	IP (IPv4, IPv6)
Netzzugang	Ethernet, Token Bus, IPoAC



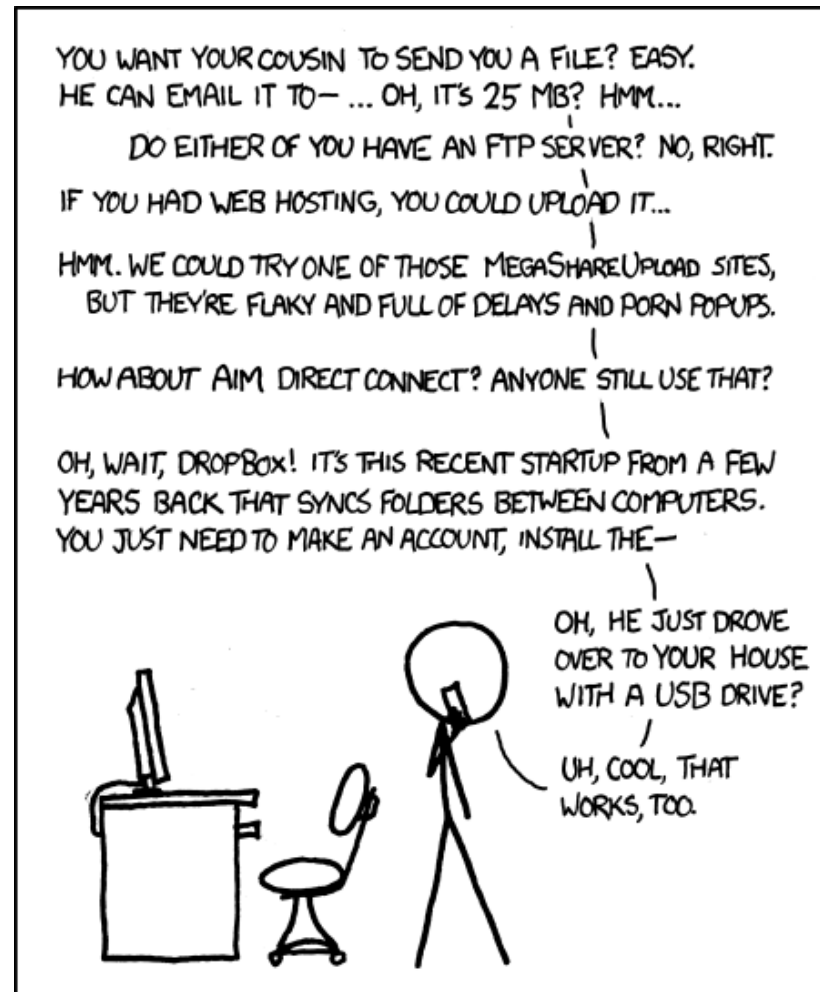
Internet-Programmierung

- Für uns sind nur Protokolle aus der Verarbeitungsschicht relevant
- Zu jedem dieser Protokolle gibt es Python-Module, welche Klassen und Funktionen bereitstellen um mit diesen zu arbeiten

Protokoll	Port	RFC	Python-Modul	Erklärung
FTP	21	959	ftplib	File Transfer Protocol, Übertragung von Daten
HTTP	80	2616	httplib BaseHTTPServer request	Hyper Text Transport Protocol, Kommunikation mit Webservern
IMAP4	143	2060	imaplib	Internet Message Access Protocol, Abrufen von E-Mails
POP3	110	1725	poplib	Post Office Protocol, Abrufen von E-Mails
SMTP	25	821	smtplib	Simple Mail Transfer Protocol, Senden von E-Mails
TELNET	23	854	telnetlib	Nutzung eines Rechners aus der Ferne



Übertragen von Dateien mit FTP

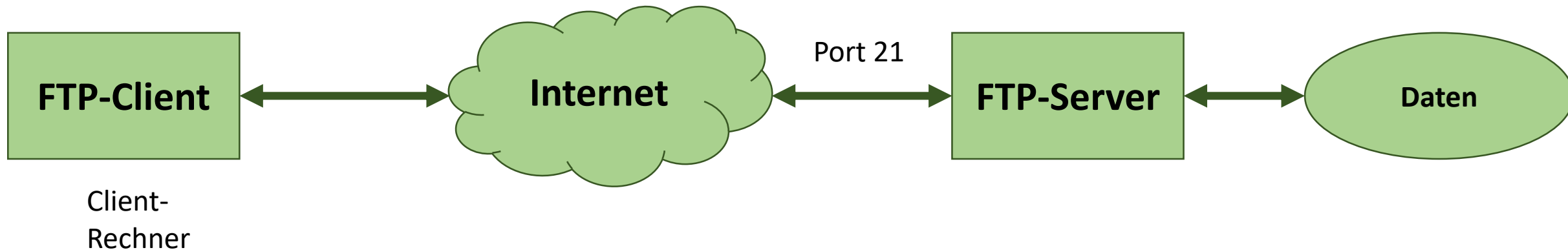


I LIKE HOW WE'VE HAD THE INTERNET FOR DECADES,
YET "SENDING FILES" IS SOMETHING EARLY
ADOPTERS ARE STILL FIGURING OUT HOW TO DO.



Übertragen von Dateien mit FTP

- Das *File Transfer Protocol* regelt das Übertragen von beliebigen Dateien zwischen einem Client und einem Server über das Internet



Übertragen von Dateien mit FTP

- Die Klasse *FTP* des *ftplib* Moduls stellt ein FTP-Client Objekt dar, mit welchem man Verbindung zu einem FTP-Server aufnehmen kann

```
>>> import ftplib  
>>> ftpConnection = ftplib.FTP(host, user, password)
```

- Mit den Methoden *cwd()* sowie *retrbinary()* und *retrlines()* lässt sich auf einem FTP-Server navigieren und Daten austauschen

retrlines(*command*, *callback*)



Befehlsaufruf auf
dem FTP-Server



Funktion mit welcher
die erhaltenen Daten
verarbeitet werden sollen



Übertragen von Dateien mit FTP

```
>>> import ftplib
>>> ftp = ftplib.FTP('ftp.ncbi.nlm.nih.gov', 'anonymous', '@')

>>> ftp.retrlines('LIST')
dr-xr-xr-x   4 ftp      anonymous      4096 Jan 11 03:49 1000genomes
-r--r--r--   1 ftp      anonymous 104858648576 Oct 14  2014 100GB
-r--r--r--   1 ftp      anonymous 10486808576 Oct 14  2014 10GB
-r--r--r--   1 ftp      anonymous 1049624576 Oct 14  2014 1GB
-r--r--r--   1 ftp      anonymous 52429848576 Oct 14  2014 50GB
-r--r--r--   1 ftp      anonymous 5243928576 Oct 14  2014 5GB
-r--r--r--   1 ftp      anonymous      2037 Aug 12  2015 README.ftp

>>> liste = []
>>> ftp.retrlines('LIST', liste.append)
>>> liste[0]
'-r--r--r--   1 ftp      anonymous      41397 Feb 20  2009 1.xml'
```



Übertragen von Dateien mit FTP

```
>>> import ftplib
>>> ftp = ftplib.FTP('ftp.ncbi.nlm.nih.gov', 'anonymous', '@')

>>> ftp.pwd()
 '/'

>>> ftp.cwd('/blast/documents')
>>> ftp.pwd()
 '/blast/documents'

>>> ftp.dir()
dr-xr-xr-x   2  ftp      anonymous      4096 Nov 12 2015 NEWXML
-r--r--r--   1  ftp      anonymous      1014 Mar 28 2016 README
-r--r--r--   1  ftp      anonymous    457674 Mar 27 2012 acss2012.pdf
-r--r--r--   1  ftp      anonymous     95699 Oct 13 2004 blast-sc2004.pdf
dr-xr-xr-x   2  ftp      anonymous      4096 Jul 27 2016 developer
```



Übertragen von Dateien mit FTP

```
>>> import ftpplib
>>> ftp = ftpplib.FTP('ftp.ncbi.nlm.nih.gov', 'anonymous', '@')

>>> ftp.dir()
dr-xr-xr-x    2 ftp      anonymous      4096 Nov 12 2015 NEWXML
-r--r--r--    1 ftp      anonymous      1014 Mar 28 2016 README
-r--r--r--    1 ftp      anonymous    457674 Mar 27 2012 acss2012.pdf
-r--r--r--    1 ftp      anonymous     95699 Oct 13 2004 blast-sc2004.pdf
dr-xr-xr-x    2 ftp      anonymous      4096 Jul 27 2016 developer

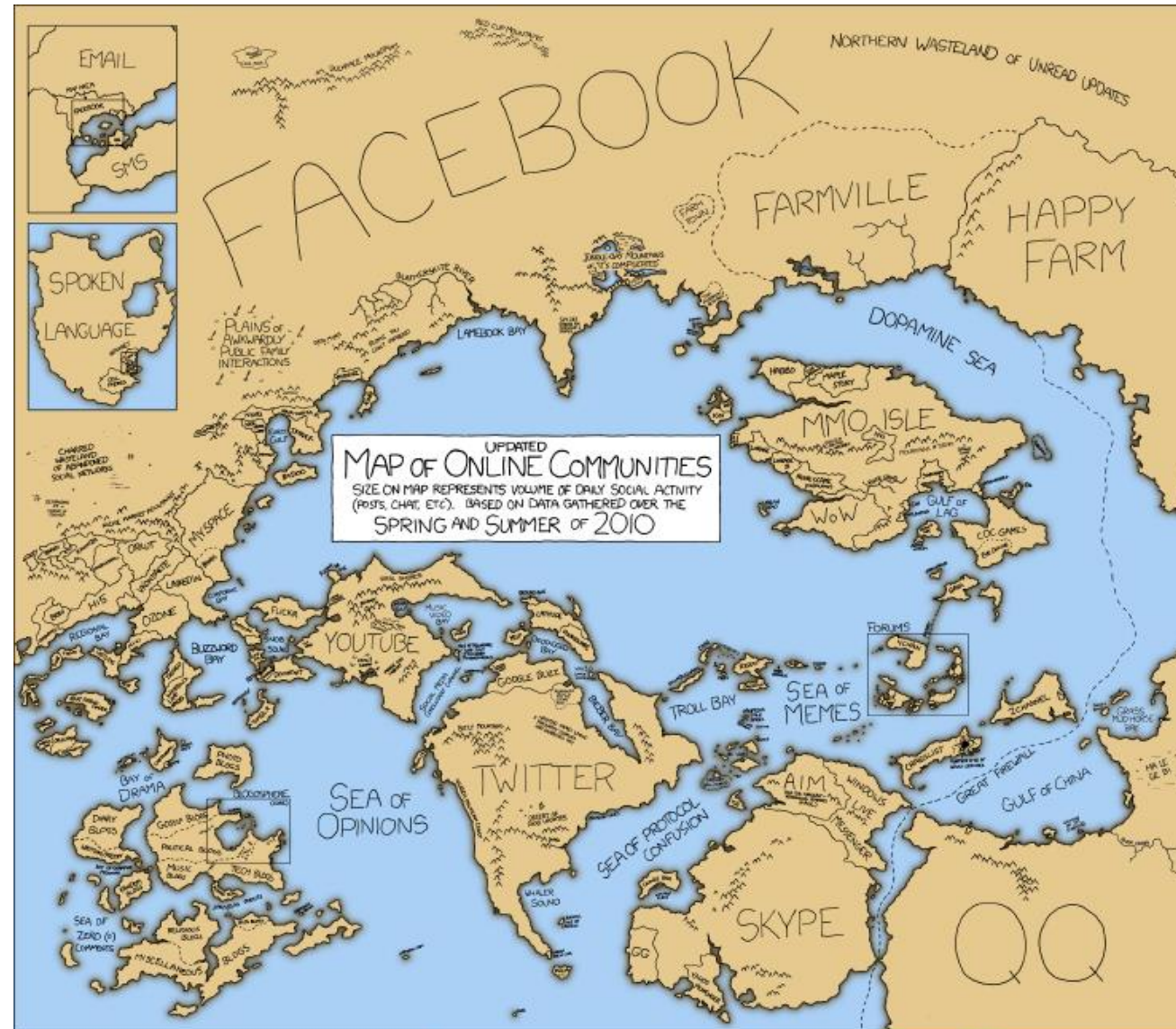
>>> file1 = open('/home/emanuel/blast_readme', 'w')
>>> file2 = open('/home/emanuel/acss2012.pdf', 'wb')

>>> ftp.retrlines('RETR README', file1.write)
>>> ftp.retrbinary('RETR /blast/documents/acss2012.pdf', file2.write)

>>> file1.close()
>>> file2.close()
>>> ftp.close()
```



Zugriff auf Webseiten mit HTTP



Zugriff auf Webseiten mit HTTP

- Das *Hypertext Transfer Protocol* regelt die Übertragung von Webseiten aus dem Internet
- Das Modul *http.client* enthält u. A. die Klasse *HTTPConnection*, deren Objekte eine Verbindung von einem HTTP-Client zu einem HTTP-Server darstellt

```
>>> from http.client import HTTPConnection
>>> verbindung = HTTPConnection('www.python.org')
>>> verbindung.request('GET', '/index.html')
>>> antwort = verbindung.response()
>>> verbindung.close()
>>> inhalt = antwort.read()

>>> verbindung = HTTPConnection('www.bing.com')
>>> verbindung.request('GET', '/search?q=python')
>>> open('/home/result.html', 'wb').write(verbindung.getresponse().read())
```



Das Modul requests

- **DIE** Schnittstelle für HTTP unter Python, ermöglicht eine sehr einfache Integration von Webdiensten
- Einfaches Anfordern von Webseiten und manipulieren von URLs
- Ermöglicht auch komplexere POST-Anfragen relativ einfach und automatisiert abzuschicken
- *'Recreational use of other HTTP libraries may result in dangerous side-effects, including: security vulnerabilities, verbose code, reinventing the wheel, constantly reading documentation, depression, headaches, or even death.'* – requests Team



Das Modul requests

- Requests unterstützt alle HTTP-Anfragemethoden

```
>>> import requests

>>> r = requests.get('http://httpbin.org/get')
>>> r = requests.post('http://httpbin.org/post')
>>> r = requests.put('http://httpbin.org/put')
>>> r = requests.delete('http://httpbin.org/delete')
>>> r = requests.head('http://httpbin.org/get')
>>> r = requests.options('http://httpbin.org/get')

>>> r = requests.get('https://www.xkcd.com/386/')
>>> r
<Response [200]>

>>> r.status_code
200
```



Das Modul requests

- Eine vom Server angeforderte Ressource besteht aus einem Header-Attribut, welches Informationen über den Aufruf und die Ressource enthält

```
>>> import requests

>>> r = requests.get('https://www.xkcd.com/386/')
>>> r.headers
{'etag': '"2812302078"', 'x-timer': 'S1466683778.318269,VS0,VE174', 'date': 'Thu, 23 Jun 2016 12:09:38 GMT', 'last-modified': 'Wed, 22 Jun 2016 04:00:02 GMT', 'accept-ranges': 'bytes', 'connection': 'keep-alive', 'content-length': '2659', 'age': '0', 'expires': 'Thu, 23 Jun 2016 11:16:37 GMT', 'content-type': 'text/html; charset=utf-8', 'vary': 'Accept-Encoding', 'x-cache': 'MISS', 'x-cache-hits': '0', 'cache-control': 'public, max-age=300', 'server': 'lighttpd/1.4.28', 'x-served-by': 'cache-fra1236-FRA', 'via': '1.1 varnish', 'content-encoding': 'gzip'}

>>> r.encoding
'utf-8'
```



Das Modul requests

- Eine vom Server angeforderte Ressource besteht aus einem Content-Attribut, welches den Inhalt der Ressource enthält

```
>>> import requests

>>> r = requests.get('https://www.xkcd.com/386/')
>>> r.content
b'<!DOCTYPE html>\n<html>\n<head>\n<link rel="stylesheet" type="text/css"
href="/s/b0dcca.css" title="Default"/>\n<title>xkcd: Duty Calls</title>\n<meta
http-equiv="X-UA-Compatible" content="IE=edge"/>\n<link rel="shortcut icon" hr
ef="/s/919f27.ico" type="image/x-icon"/>\n<link rel="icon" href="/s/919f27.ico"
type="image/x-icon"/>\n<link rel="alternate" type="application/atom+xml"
title="Atom 1.0" href="/atom.xml"/>\n<link rel="alternate,,
type="application/rss+xml" title="RSS 2.0,,
href="/rss.xml"/>\n<script>\n(function(i,s,o,g,r,a,m){i[\''GoogleAnalyticsObject
\'']=r;i[r]=i[r]||function(){\n(i[r].q=i[r].q||[]).push(arguments) '

...
```



Das Modul requests

```
>>> import requests
>>> from PIL import Image
>>> from io import StringIO

>>> r = requests.get('https://www.xkcd.com/386/')
>>> i = Image.open(StringIO(r.content.decode('utf-8'))))

>>> r = requests.get('https://www.xkcd.com/386/info.0.json')
>>> r.json()
{'alt': 'what do you want me to do?  LEAVE?  Then they\'ll keep being wrong!',
 'day': '20',
 'img': 'http://imgs.xkcd.com/comics/duty_calls.png',
 'link': '',
 ...
}
```



Das Modul requests

```
>>> import requests

>>> login = ('mu42cuq', 'AlsObIchJema1sMeinPasswortVeratenwürde')
>>> r = requests.get('https://webmail.uni-jena.de/login.php', auth=login)

>>> formular = {'key1': 'value1', 'key2': 'value2'}
>>> r = requests.get('http://httpbin.org/post', data=formular)
>>> r.url
u'http://httpbin.org/get?key2=value2&key1=value1'

>>> r = requests.post('http://httpbin.org/post', data=formular)
```

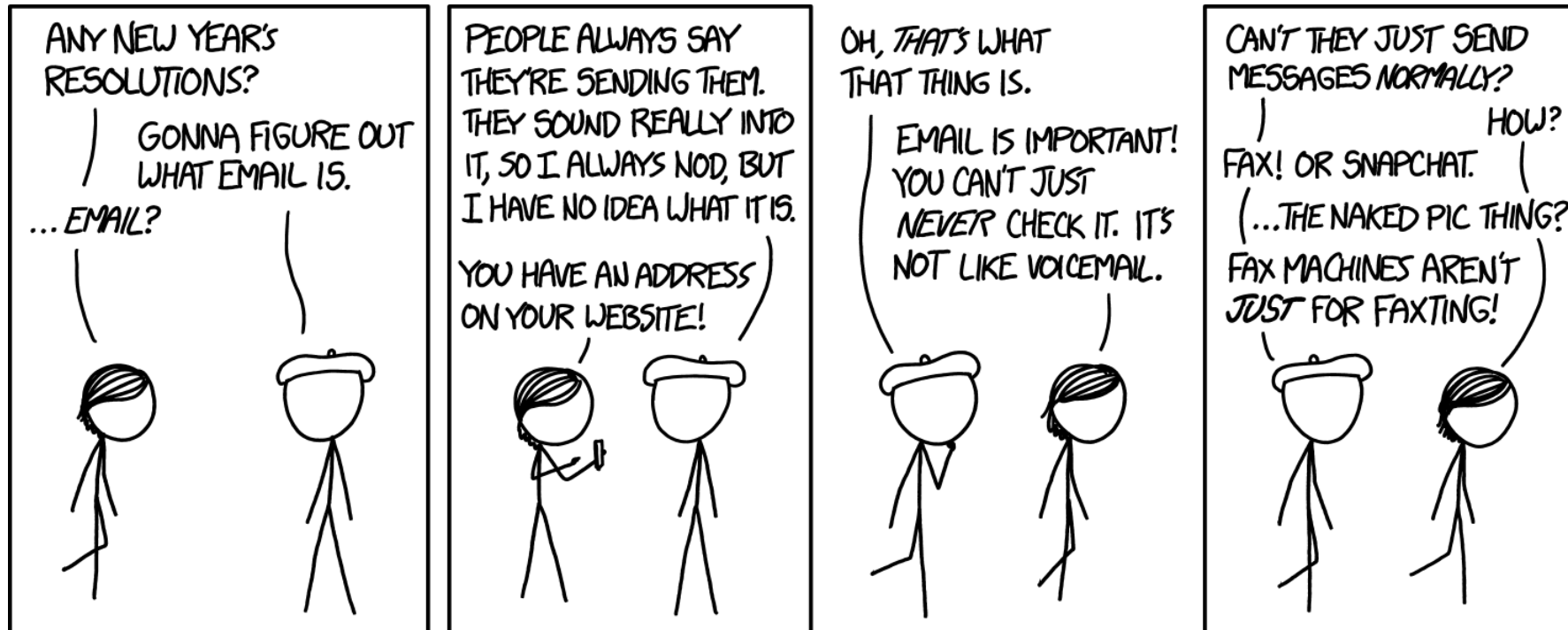


Das Modul requests

- Was requests sonst noch kann:
 - Auslesen und manipulieren von Cookies
 - Multipart Datei Upload
 - SSL Verifizierung
 - Proxy Unterstützung
 - Download streaming
 - ...



E-Mails senden mit SMTP



E-Mails senden mit SMTP

- Das *Simple Mail Transfer Protocol* regelt das Senden von E-Mails an einen Mail-Server
- Die in dem Modul *smtplib* enthaltene Klasse *SMTP* erzeugt Objekte mit denen man sehr einfach einen Mail-Client programmieren kann

```
>>> from smtplib import SMTP  
>>> SMTP(host)
```

- Die wichtigsten Methoden sind:

Methode	Erklärung
<code>login(user, password)</code>	Einloggen auf einem SMTP-Server mittels Authentifizierung.
<code>sendmail(fromAddr, toAddr, text)</code>	Absenden einer Mail.
<code>set_debuglevel(1)</code>	Anzeigen des vollständigen Dialogs mit dem SMTP-Server.
<code>quit()</code>	Trennen der Verbindung.



E-Mails senden mit SMTP

```
>>> from smtplib import SMTP
>>> s = SMTP('smtp.uni-jena.de')
>>> s.login('mu42cuq', 'MeinTotalGeheimesPasswort')
>>> s.sendmail('mu42cuq@uni-jena.de', 'deine@adresse.de', 'Hallo!')

>>> s.sendmail('mu42cuq@uni-jena.de', 'deine@adresse.de',
               'From:x@y.z\nSubject: Betreff\nHaupttext')

>>> s.quit()
```

