Packages Teil 2 - Best Of

Proseminar Python

Steffen Beckmann

Arbeitsbereich Wissenschaftliches Rechnen
Fachbereich Informatik
Fakultät für Mathematik, Informatik und Naturwissenschaften
Universität Hamburg

2022-06-28

Gliederung (Agenda)

HTTP

- Protokoll für Kommunikation zwischen Clients und Servern
- Übertragung von Daten
- Nutzung für Webseiten und APIs
- Verschiedene Arten von Requests (GET, POST, ...)

[4]

Überblick

- HTTP "for humans": Im Code einfach verständlich
- Einfaches Versenden von Requests
- Vielfältige Optionen beim Versenden, Attribute bei Responses

```
1 python pip install requests
```

Installation

Beispiel HTTP GET-Request

- Aufruf der Methode sendet Request
- Hier: Nutzen der Service-Website httpbin zum Ausprobieren
- Methode gibt Response-Objekt zurück

```
1 import requests
2 r = requests.get("http://www.httpbin.org/get")
3 print(type(r))
4
5 >> <class 'requests.models.Response'>
```

Eine simple Request

[8]

Parameter

- Reihe an Parametern für die Request
- Query-Strings, Authentifizierung, Cookies, Timeout...

```
1 import requests
2 query = {"stadt": "Hamburg", "land":
      ↪ "Deutschland"}
3 r = requests.get("http://www.httpbin.org/get",
      ↪ params=query)
4 print(r.url)
5 >> http://www.httpbin.org/get?stadt=Hamburg
6                                &land=Germany
```

Query-Strings mit params als Dict übergeben

[7]

Beispiel Authentifizierung

```
1  import requests
2  auth_tuple = ("nutzer", "geheim01")
3  r1 = requests.get("http://www.httpbin.org/basic-auth
4                     /nutzer123/passwort1",
                          ↪ auth=auth_tuple)
5  print(r1.status_code)
6  >> 401
7  auth_tuple = ("nutzer123", "passwort1")
8  r2 = requests.get("http://www.httpbin.org/basic-auth
9                     /nutzer123/passwort1",
                          ↪ auth=auth_tuple)
10 print(r2.status_code)
11 >> 200
```

Tupel zur Authentifizierung mit auth übergeben

Response-Objekt

- Verschiedene Attribute, Informationen über die Rückgabe

```
1 print(r.status_code)
2 >> 200
3 print(r.ok)
4 >> True
```

Status

- Zugriff auf den Inhalt als String (text) oder als Bytes (content)

[7]

Response-Objekt

- Response im JSON-Format direkt als Dictionary durch die dict()-Methode

```
1 ...
2 r_json = r.json()
3 print(r_json)
4 >> {'args': {}, 'headers': {'Accept': '*/*',
    ↪ 'Accept-Encoding': 'gzip, deflate', 'Host':
    ↪ 'www.httpbin.org', 'User-Agent':
    ↪ 'python-requests/2.27.1',
    ↪ 'X-Amzn-Trace-Id':
    ↪ 'Root=1-629a4423-26c1090f5784e44b61d03be5'},
    ↪ 'origin': '34.71.107.197', 'url':
    ↪ 'http://www.httpbin.org/get'}
5 print(r_json['url'])
6 >> http://www.httpbin.org/get
```

Konvertieren einer JSON-Response

HTML

- Auszeichnungssprache zur Beschreibung der Struktur einer Webseite
- Besteht aus ineinander geschachtelten Elementen (Tags) mit Attributen
- Unterstützung für Bilder, Überschriften, Tabellen, ...

```
1 import requests
2 r = requests.get("http://quotes.toscrape.com/")
3 print(r.text)
4 >> <!DOCTYPE html>
5     <html lang="en">
6     <head>
7         <meta charset="UTF-8">
8         <title>Quotes to Scrape</title>
9     ...
```

Zugriff auf HTML-Code durch Request

BeautifulSoup

- Möglichkeit zum objektorientierten Durchschreiten von HTML-Code
- Vielfältige Möglichkeiten zur Suche und Filterung

```
1 python pip install beautifulsoup4
```

Installation

BeautifulSoup

- HTML-Struktur muss bekannt sein
- Tag-Objekte als Repräsentation der einzelnen HTML-Tags
- ToS beachten, nicht spammen!

```
1 import requests
2 from bs4 import BeautifulSoup
3 req = requests.get("http://quotes.toscrape.com/")
4 b = BeautifulSoup(req.text, "html.parser")
5 print(b.title)
6 >> <title>Quotes to Scrape</title>
```

Erstellen einer BeautifulSoup-Objekts

[2]

HTML-Beispiel

than absolutely boring.”

by **Marilyn Monroe** (about)

Tags: be-yourself inspirational

“Try not to become a man of success. Rather become a man of value.”

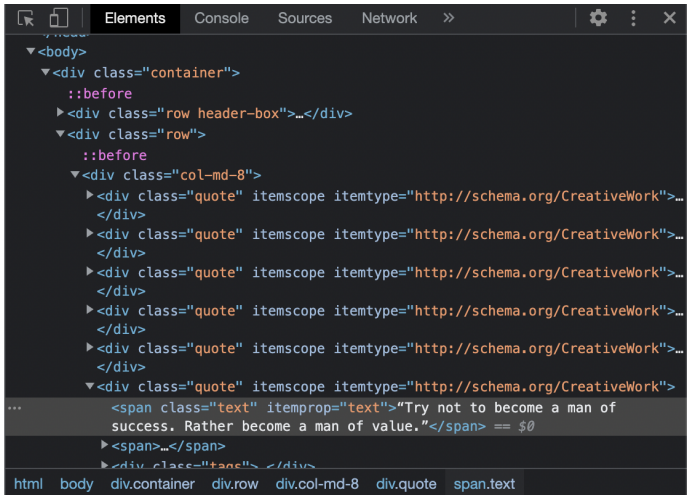
by **Albert Einstein** (about)

Tags: adulthood success value

“It is better to be hated for what you are than to be loved for what you are not.”

by **André Gide** (about)

Tags: life love



Suchen von HTML-Tags

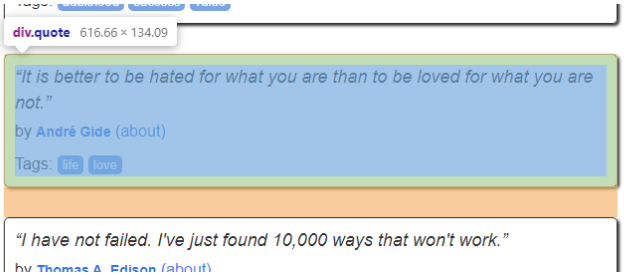
- Suchen nach Tags über Tag-Namen, Attribute, Inhalt
- Rekursive Suche nach unten über .find oder .find_all
- Suche in andere Richtungen möglich -> .find_parents, .find_next_siblings, .find_previous_siblings

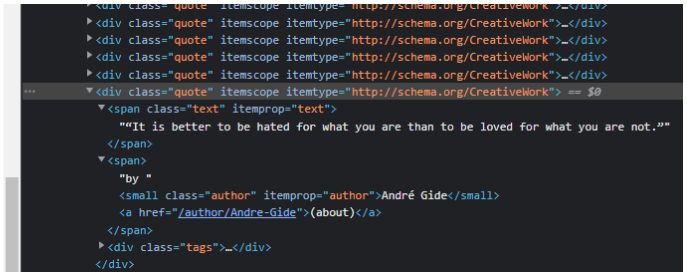
```
1 b = BeautifulSoup(req.text, "html.parser")
2 quotes = b.find_all("div", class_="quote")
3 print(len(quotes))
4 >> 10
```

Suche aller HTML-Tags mit Class-Attribut "quote"

[2]

"Manuelles" Navigieren





- Shortcut für untergeordneten Tag: Tag.(Name)
- Liste der nächsten "Nachfahren" eines Tags durch .contents-Attribut
- Nächster Vorfahre durch .parent, alle Vorfahren durch .parents
- Geschwister-Tags durch .next_siblings, bzw. .previous_siblings
- Text-Inhalt eines Tags durch .string

[2]

Beispiel: Alle Autoren in eine Liste

```
1 ...
2 quotes = b.find_all("div", class_="quote")
3 authors = []
4 for quote in quotes:
5   author = quote.find(class_="author")
6   if author is not None:
7     authors.append(author.string)
8 print(authors)
9 >> ['Jane Austen', 'Eleanor Roosevelt', 'Marilyn
    ↪ Monroe', 'Albert Einstein', 'Haruki Murakami',
    ↪ 'Alexandre Dumas fils', 'Stephenie Meyer',
    ↪ 'Ernest Hemingway', 'Helen Keller', 'George
    ↪ Bernard Shaw']
```

argparse

- Argument: Parameter beim Aufruf über Kommandozeile
- Das Package argparse verwaltet diese Argumente
- Kümmert sich um Fehlerbehandlung bei falscher Nutzung
- Bildet automatisch Hilfsseiten
- In Standard-Python-Bibliothek enthalten - keine Installation nötig!

[1]

Argument-Arten

- positionell
  - implizit, keine Angabe des Namens nötig
- optional
  - Name muss spezifiziert werden
  - Konvention: ”–” für ganzer Name, ”-” für Abkürzung
- Flags
  - optionales Boolean-Argument
  - Benutzung heißt wahr, Nichtbenutzung falsch

[3]

Argumente hinzufügen

- ArgumentParser-Objekt
- Mit .add_argument Argumente hinzufügen
- Verschiedene Möglichkeiten für Spezifizierung durch Parameter
- type-Parameter für Datentyp, muss dann eingehalten werden

[1] [3]

Beispiel

```
1  import math
2  import argparse
3  parser = argparse.ArgumentParser("Kreis berechnen")
4  parser.add_argument("radius", type=int, help="Der
       ↪ Radius des Kreises")
5  args = parser.parse_args()
6
7  def calc_circle_area(r):
8    return math.pi*r**2
9
10 if __name__ == "__main__":
11   print(calc_circle_area(args.radius))
```

Positionelles Argument

Beispiel

```
python main.py 5
>> 78.53981633974483
```

Korrekter Aufruf

```
python main.py
>> usage: Kreis berechnen [-h] radius
   Kreis berechnen: error: the following arguments
   are required: radius
```

Inkorrekter Aufruf

optionales Argument

- argparse erkennt durch Striche-Konvention Art des Arguments
- Aufruf durch Name, gefolgt vom Inhalt des Arguments
- Standard-Wert mit default-Parameter
- Optionale Parameter können auch Voraussetzung sein (required-Parameter)

```
1 parser.add_argument("--radius", "-r", type=int,
      ↪ default=2, help="Der Radius des Kreises")
```

optionales Argument

[1] [3]

optionales Argument

```
python main.py --radius 5
>> 78.53981633974483
```

Aufruf optionales Argument

```
python main.py
>> 12.566370614359172
```

Zurückfallen auf Default-Wert

Flag

- optionale Boolean-Argumente meistens als Flag
- Verwendung normalerweise: Default-Wert false, Angabe impliziert true
- Dafür action-Parameter "store_true", Gegensatz "store_false"

```
1 parser.add_argument("-v", "--verbose",
      ↪ action="store_true", help="Mehr
      ↪ Informationen in Ausgabe")
```

Flag

[3]

Flag

```
1 if args.verbose:
2     print("Flaeche des Kreises:
          ↪ {}".format(calc_circle_area(args.radius)))
3 elif not args.verbose:
4     print(calc_circle_area(args.radius))
```

Zugriff auf Flag im Code

```
1 python main.py 5 -v
2 >> Flaeche des Kreises: 78.53981633974483
```

Aufruf mit Flag

Hilfe

- argparse bietet immer eine Hilfsausgabe
- Auflistung mögliche Argumente mit Reihenfolge
- Aufrufbar durch Flag "-h"

```
1 python main.py -h
2 >> usage: Kreis berechnen [-h] [-v] radius
3
4 positional arguments:
5   radius          Der Radius des Kreises
6
7 optional arguments:
8   -h, --help      show this help message and exit
9   -v, --verbose   Mehr Informationen in Ausgabe
```

Hilfsausgabe

PyTest

- Simple und einfach zu realisierende Test-Möglichkeit
  - Testmethoden
  - Testklassen
- Leicht skalierbar
- Baut auf Konventionen bei der Namensgebung auf

```
1 python pip install PyTest
```

Installation

Test-Methoden

- PyTest führt alle Dateien + Methoden mit Name "test_*" oder "*_test" aus
- Test-Methode gibt Testszenario an, Vergleich mit assert

```
1 def sum_range(a):
2   sum = 0
3   for i in range(a):
4     sum += i
5   return sum
```

main.py

```
1 import pytest
2 from main import sum_range
3 def test_sum_range_4():
4   assert sum_range(4) == 6
```

test_sum_range.py

Ausführung

- Starten über Kommandozeile
- Auch Ausführung spezifischer Datei möglich

```
>> pytest
======= test session starts =======
platform linux -- Python 3.8.12, pytest-7.1.2,
    ↪ pluggy-1.0.0
rootdir: /home/runner/test
collected 1 item

test_run.py .          [100%]

======= 1 passed in 0.07s =======
```

Erfolgreiche Ausführung

Ausführung

```
1 def test_floating_point():
2     assert sum_range(2.3)
```

Test-Methode die fehlschlägt

```
1 ======== FAILURES ========
2 __ test_floating_point __
3     def test_floating_point():
4 >       assert sum_range(2.3)
5 (...)
6 main.py:3: TypeError
7 ======= short test summary info =======
8 FAILED test_run.py::test_floating_point - TypeError:
    ↪ 'float' object cannot be interpreted ...
9 ======= 1 failed, 1 passed in 0.19s =======
```

Fehlgeschlagene Ausführung

Exceptions abfangen

- pytest.raises(Exception-Typ)
- Für Tests in denen eine Exception erwartet wird
- match-Parameter filtert nach bestimmten Fehlermeldungen

```
1 def test_floating_point_raises():
2   with pytest.raises(TypeError):
3     sum_range(4.3)
```

Testmethode die eine Exception erwartet

[5]

Test-Klassen

- Gruppierung von Test-Methoden durch Test-Klassen
- Namenskonvention: "Test*"
- Wird automatisch ausgeführt
- Jede Methode eigene Instanz

[5]

Fixtures

- Testdaten mehrmals verwenden
- Als Methode mit pytest.fixture markieren
- Methode als Parameter, wird ausgeführt und zurückgegeben

```
1 @pytest.fixture
2 def number_fixture():
3   return [34, 345, -234, 6]
```

Fixture

[6]

Zusammenfassung

- Requests
  - Einfache Möglichkeit, HTTP-Anfragen über Code zu versenden
  - Übersichtliche Handhabung der Antwort
- BeautifulSoup
  - Automatisiertes Durchschreiten von HTML-Seiten
  - Sehr mächtig für verschiedene Zwecke
- argparse
  - Realisierung von Argumenten bei Aufruf über Kommandozeile
  - Komfortabel für Realisierung von Tools
- PyTest
  - Sinnvolle und saubere Möglichkeit für Tests in Python
  - Für kleine und große Projekte

Literatur I

[1] argparse. argparse Dokumentation und Tutorials. URL: https://docs.python.org/3/library/argparse.html.

[2] BeautifulSoup. BeautifulSoup Documentation. URL: https://www.crummy.com/software/BeautifulSoup/bs4/doc/.

[3] Databraineo. Python-Tutorial für ein Kommandozeilen-Tool mit argparse. URL: https://databraineo.com/ki-training-resources/python/python-tutorial-fuer-ein-kommandozeilen-tool/.

[4] Heise.de. HTTP - Was ist das? URL: https://www.heise.de/tipps-tricks/HTTP-was-ist-das-4607102.html.

Literatur II

[5] PyTest. PyTest Docs - Getting Started, further reading. URL: https://docs.pytest.org/en/7.1.x/getting-started.html.

[6] RealPython. Effective Python Testing With Pytest - Fixtures. URL: https://realpython.com/pytest-python-testing/#fixtures-managing-state-and-dependencies.

[7] Requests. ReadTheDocs - Requests API. URL: https://requests.readthedocs.io/en/latest/api/.

[8] Requests. ReadTheDocs - Requests Quickstart. URL: https://requests.readthedocs.io/en/latest/user/quickstart/.