

# Python

---

Webscraping

# Webscraping

---

- Scraping (auch Web Scraping oder Screen Scraping) bezeichnet den manuellen oder **automatisierten** Vorgang des Extrahierens, Kopierens, Speicherns sowie der Wiederverwendung fremder Inhalte und Daten im Internet.
- Manuell wird bei uns nur das **Konzept**, alles andere soll automatisiert werden.

# Szenario "Digitec"

CASH-BACK

**2199.–** CashBack 300.–

**Sony Alpha 7 IV**

33 Mpx, Vollformat

Bis zu 329.84 sparen ✓  
Gebraucht kaufen ab 1869.16

Bewertungen  129 | Marke Mehr von Sony

Testberichte  
Sehr gut bei 12 Tests

✓ Morgen nach Hunzenschwil geliefert  
Mehr als 10 Stück an Lager

In den Warenkorb

Vergleichen Merken

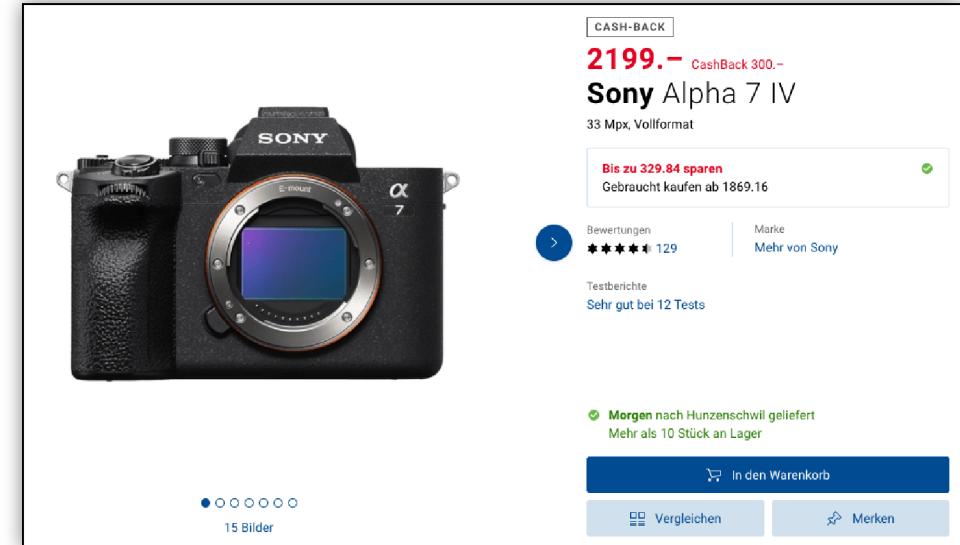


● ○ ○ ○ ○ ○  
15 Bilder

<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

Was könnte hier für mich spannend sein?

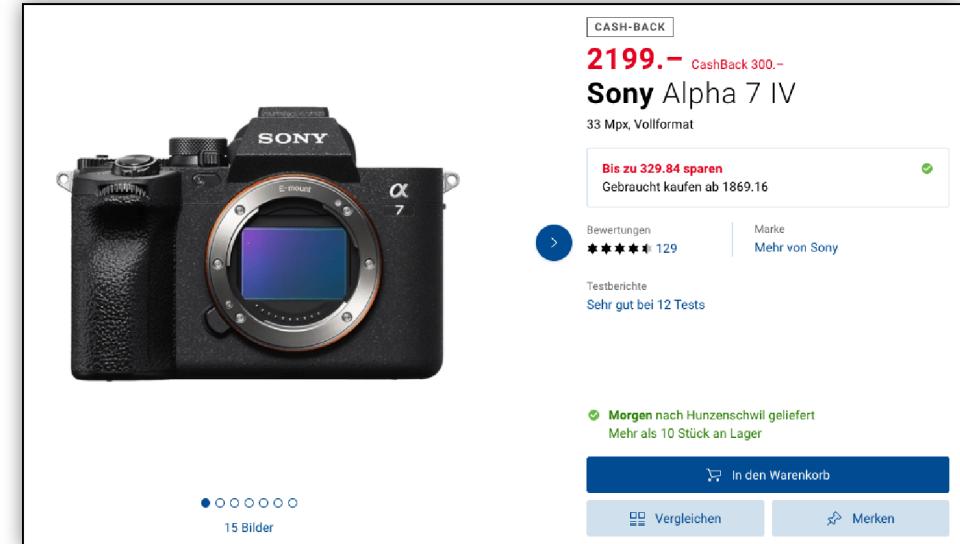


<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

Was könnte hier für mich spannend sein?

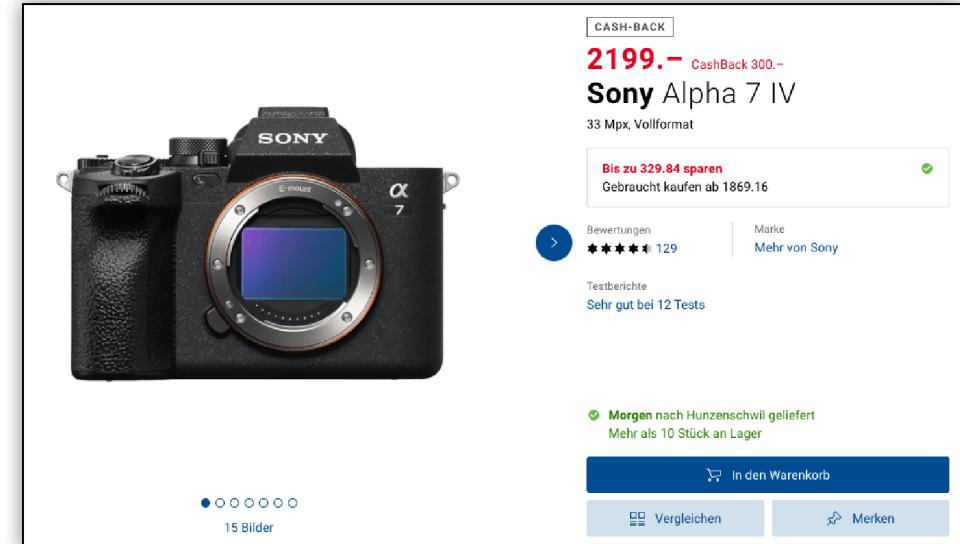
- Preis



# Szenario "Digitec"

Was könnte hier für mich spannend sein?

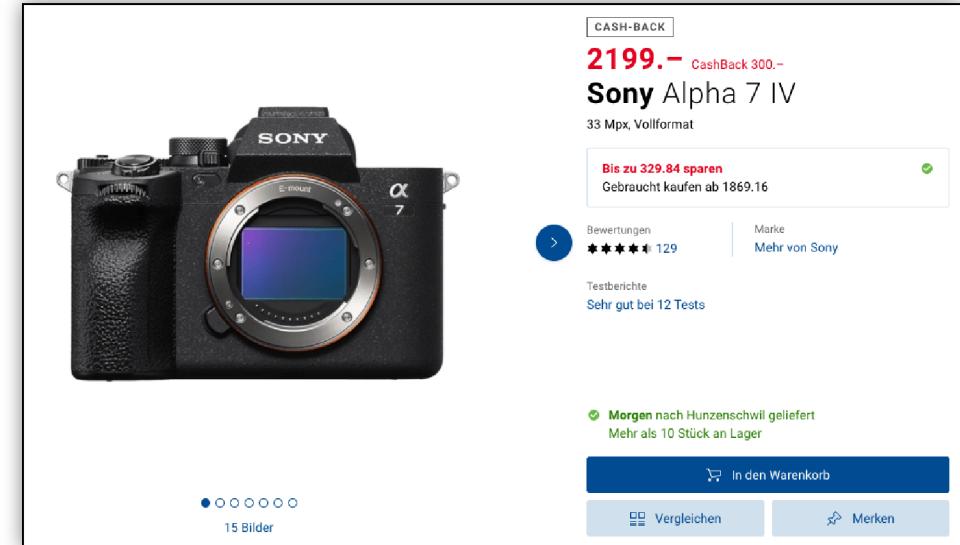
- Preis
- Cashback



# Szenario "Digitec"

Was könnte hier für mich spannend sein?

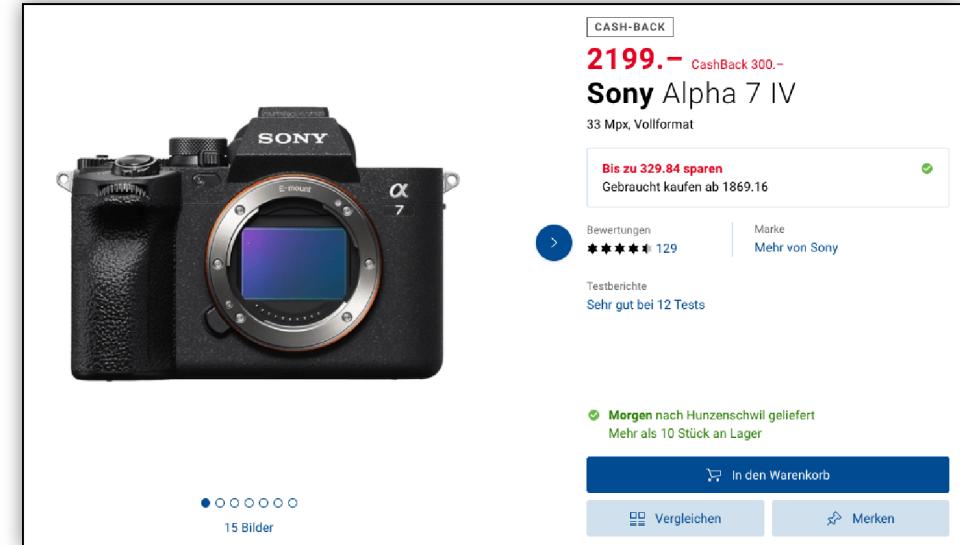
- Preis
- Cashback
- Anzahl Bewertungen



# Szenario "Digitec"

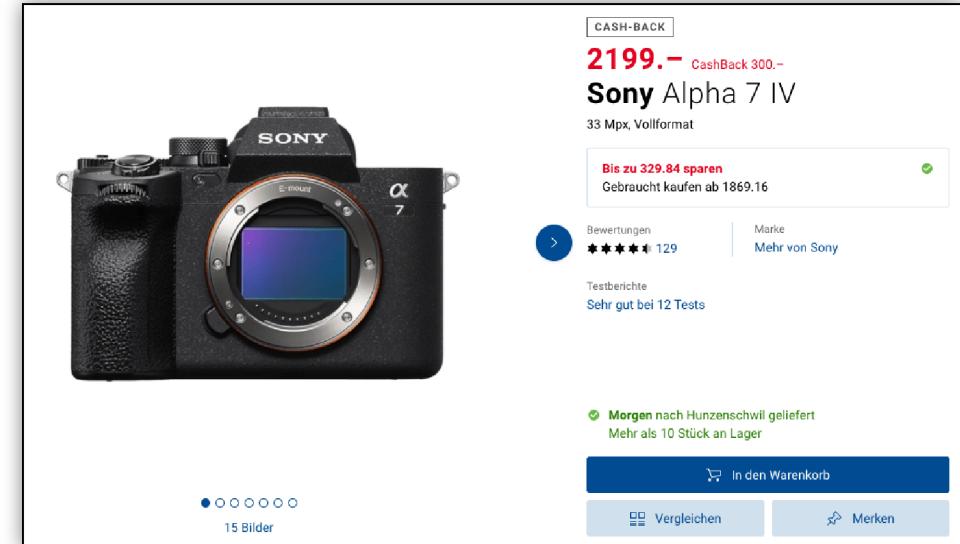
Was könnte hier für mich spannend sein?

- Preis
- Cashback
- Anzahl Bewertungen
- ....



# Szenario "Digitec"

Doch dafür muss ich das Web erstmal (oberflächlich) verstehen.



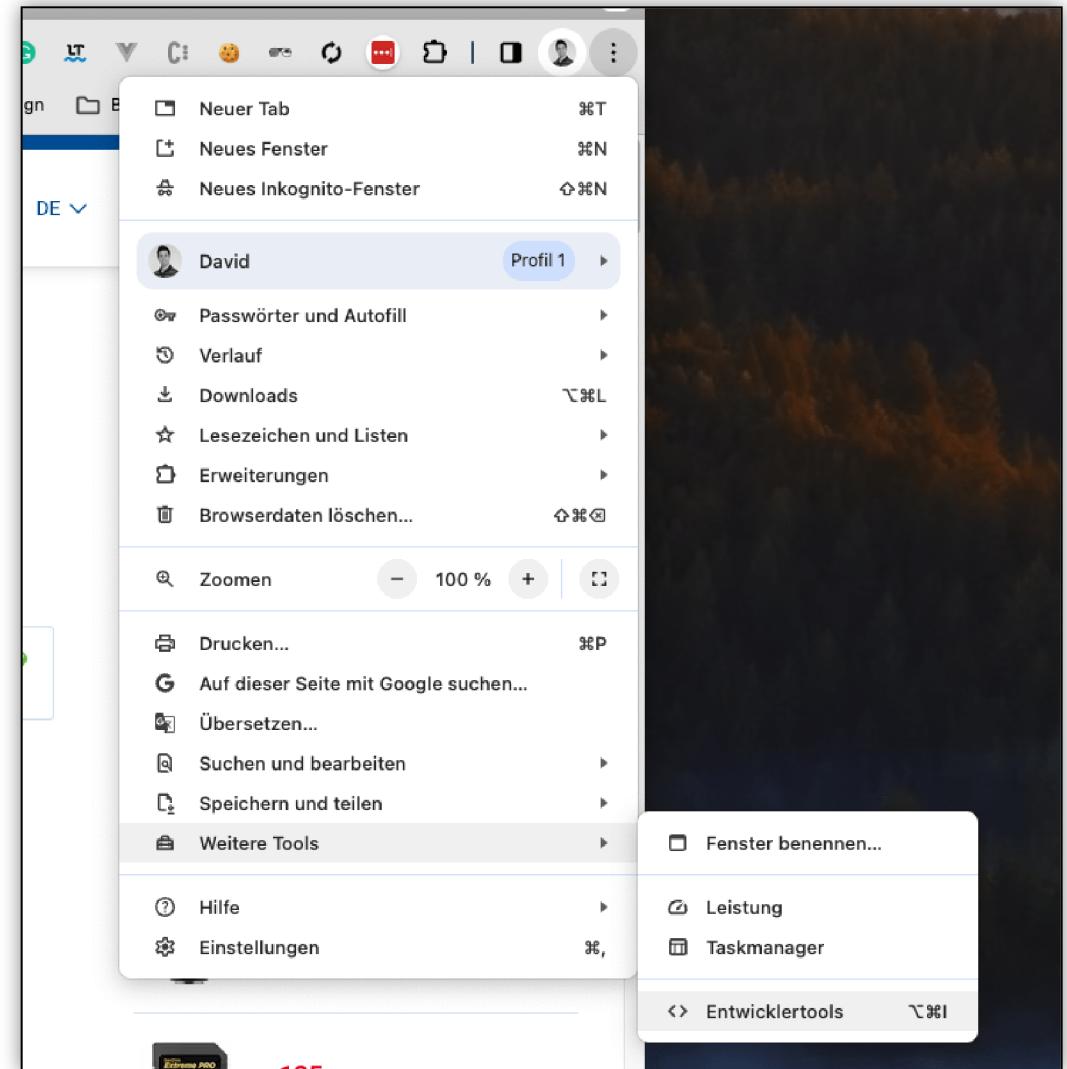
Die URL (Unified Resource Locator) gibt mir eine Adresse, auf die ich via GET zugreife.

<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

Hinter dieser URL verbirgt sich auch ein Quelltext. Dieser lässt sich am besten mit den "Developer Tools" untersuchen.

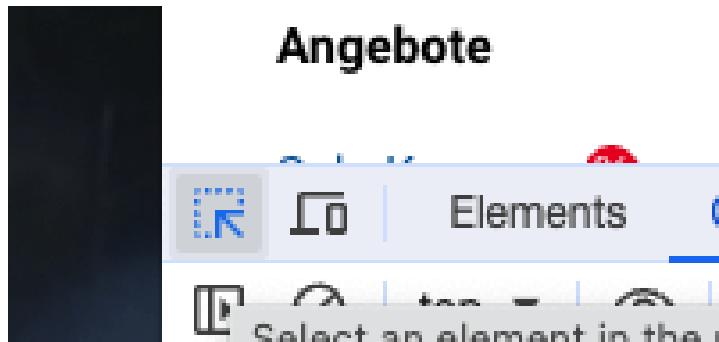
Wer mit HTML vertraut ist, kann sich auch den Quelltext anschauen.



<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

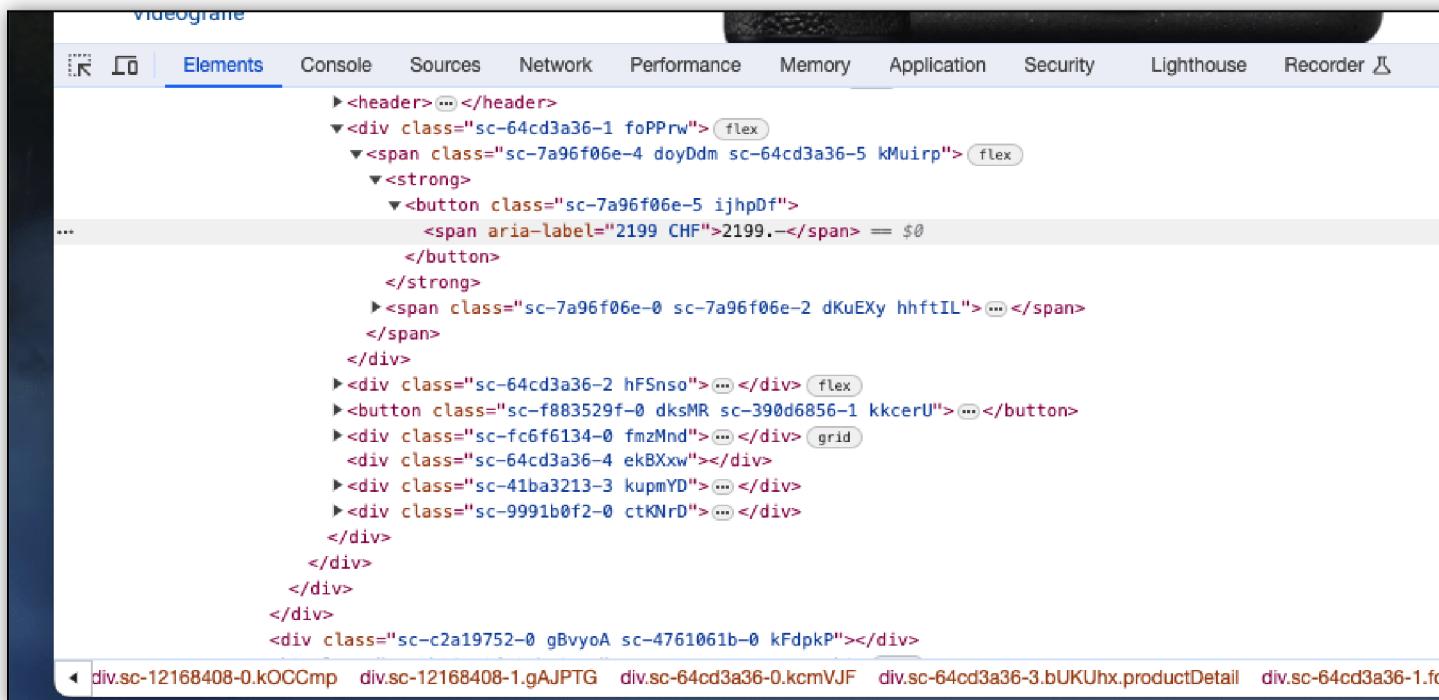
Ist der Pfeil angewählt, können wir Elemente (HTML-Elemente) in der Seite markieren und betrachten:



The screenshot shows a product listing for a Sony Alpha 7 IV camera. The price '2199.-' is highlighted with a blue selection bar, and a context menu is open above it. The menu provides detailed information about the selected element, including its type ('span'), dimensions (107.28 x 38), color (#E60021), font ('32px Roboto, Arial, sans-serif'), and accessibility details (Name: 2199 CHF, Role: generic, Keyboard-focusable). Below the menu, the product name 'Sony Alpha 7 IV' is displayed in large bold letters, followed by '33 Mpx, Vollformat'. A promotional box offers a discount of 'Bis zu 329.84 sparen' (Up to 329.84 saved) when bought used for 1869.16. At the bottom, there are sections for 'Bewertungen' (Reviews) with a rating of 129 and 'Marke' (Brand) with a link to 'Mehr von Sony' (More from Sony). Test reports indicate a 'Sehr gut bei 12 Tests' (Very good in 12 tests) rating.

# Szenario "Digitec"

Ein weiterer Klick führt uns auf den entsprechenden Source-Code.



<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

Wir sehen HTML-Elemente (z. B. **button**) und viele classes (z. B. **sc-7a96f06c-5**)

The screenshot shows the Chrome DevTools Elements tab with the following details:

- Page Title:** videogramme
- Toolbar:** Elements (selected), Console, Sources, Network, Performance, Memory, Application, Security, Lighthouse, Recorder, P
- HTML Structure:**

```
> <header> ... </header>
  <div class="sc-64cd3a36-1 foPPrw"> (flex)
    <span class="sc-7a96f06e-4 doyDdm sc-64cd3a36-5 kMuirp"> (flex)
      <strong>
        <button class="sc-7a96f06e-5 ijhpDf">
          <span aria-label="2199 CHF">2199.-</span> = $0
        </button>
      </strong>
      <span class="sc-7a96f06e-0 sc-7a96f06e-2 dKuEXy hhftIL"> ... </span>
    </span>
  </div>
  <div class="sc-64cd3a36-2 hFSnso"> ... </div> (flex)
  <button class="sc-f883529f-0 dksMR sc-390d6856-1 kkcerU"> ... </button>
  <div class="sc-fc6f6134-0 fmzMnd"> ... </div> (grid)
    <div class="sc-64cd3a36-4 ekBXxw"> ... </div>
  <div class="sc-41ba3213-3 kupmYD"> ... </div>
  <div class="sc-9991b0f2-0 ctKNrD"> ... </div>
    </div>
  </div>
</div>
<div class="sc-c2a19752-0 gBvyoA sc-4761061b-0 kFdpkP"> ... </div>
```
- Bottom Bar:** div.sc-12168408-0.kOCCmp div.sc-12168408-1.gAJPTG div.sc-64cd3a36-0.kcmVJF div.sc-64cd3a36-3.bUKUhx.productDetail div.sc-64cd3a36-1.foP

# Szenario "Digitec"

Wir sehen HTML-Elemente (z. B. **button**) und viele classes (z. B. **sc-7a96f06c-5**)

Ähnlich wie in Büchern (Seitenzahlen, Kapiteln) können wir diese Informationen nutzen, um nur bestimmte Teile einer Webseite zu extrahieren.

Selten sind wir an der ganzen HTML-Seite interessiert...

# Szenario "Digitec"

Wir sehen HTML-Elemente (z. B. **button**) und viele classes (z. B. **sc-7a96f06c-5**)

Sowohl HTML-Elemente als auch class="..." kann mehrmals auf der Seite vorkommen. Besser wären id="...", den diese dürfen nur einmal (unique) pro Seite vorkommen.

*Warum? Digitec macht sehr viel Analyse der Nutzung, daher gibt es hier viele generierte Klassen. Diese können wir aber fast wie id's nützen :-)*

# Szenario "Digitec"



```
1 from bs4 import BeautifulSoup
2 from urllib.request import urlopen
3 import ssl
4 ssl._create_default_https_context = ssl._create_unverified_context
5
6 url = "https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964"
7 page = urlopen(url)
8 html = page.read().decode("utf-8")
9 soup = BeautifulSoup(html, "html.parser")
10
11 # first level
12 # <button class="sc-7a96f06e-5 ijhpDf"><span aria-label="2199 CHF">2199.-</span></button>
13 price = soup.find("button", class_="sc-7a96f06e-5 ijhpDf")
14 print(price)
```

Ergibt:

<button class="sc-7a96f06e-5 ijhpDf"><span aria-label="2199 CHF">2199.-</span></button>

<https://www.digitec.ch/de/s1/product/sony-alpha-7-iv-33-mpx-vollformat-kamera-17071964>

# Szenario "Digitec"

Als HTML betrachtet:



```
1 <button class="sc-7a96f06e-5 ijhpDf">
2   <span aria-label="2199 CHF">2199.-</span>
3 </button>
```

Wir sehen die Hierarchie, button => span

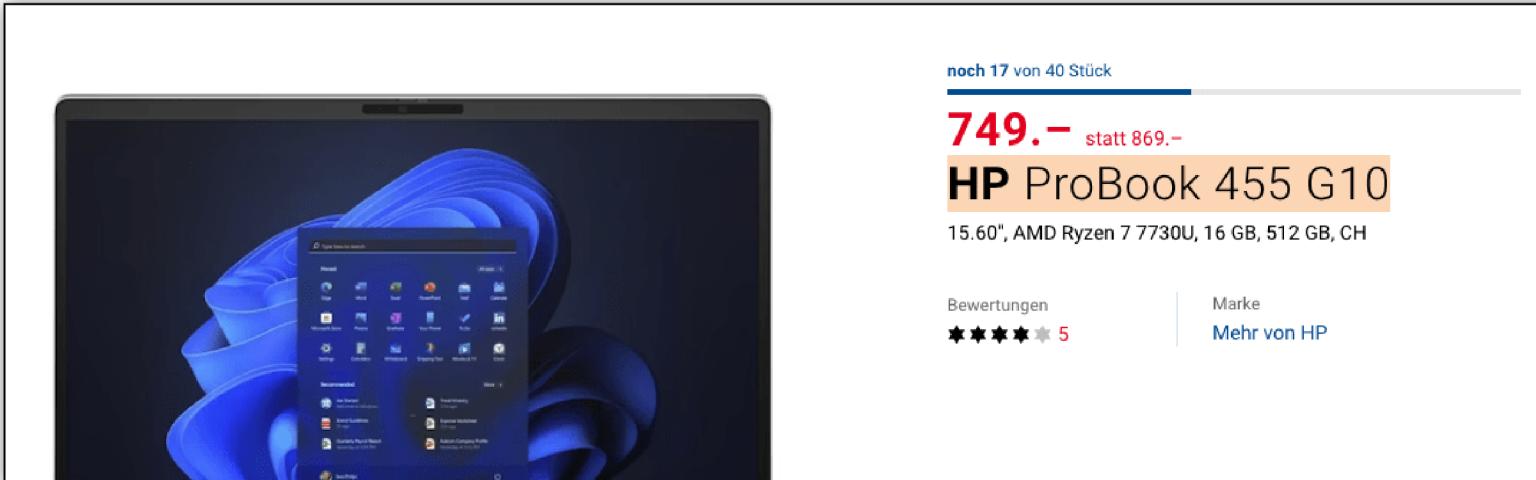
# Szenario "Digitec"

Also müssen wir noch ein wenig nach "unten" graben:

```
● ● ●  
1 # first level  
2 # <button class="sc-7a96f06e-5 ijhpDf"><span aria-label="2199 CHF">2199.—</span></button>  
3 price = soup.find("button", class_="sc-7a96f06e-5 ijhpDf")  
4 print(price)  
5  
6 # second, third level  
7 # <span aria-label="2199 CHF">2199.—</span>  
8 # 2199.—  
9 for d in price.descendants:  
10     print(d)
```

# Aufgabe Digitec

- 1) Versucht **00\_scrap\_digitec.py** unter **exercises** auszuführen.
- 2) Versucht die Datei in den "userfolder" zu kopieren, und dort mit einem anderen, Digitec-Produkt eurer Wahl zu versehen und **nur den Preis** zu erhalten.
- 3) Versucht anstelle des Preises, den Titel eines Produktes zu erhalten.



The image shows a screenshot of a web browser displaying a product listing for an HP ProBook 455 G10 laptop on the Digitec website. The laptop is shown from a top-down perspective, featuring a dark blue finish and a large, vibrant blue flower graphic on its screen. To the right of the image, product details are listed:

- Stock status: noch 17 von 40 Stück
- Price: 749.- statt 869.-
- Product name: HP ProBook 455 G10
- Specs: 15.60", AMD Ryzen 7 7730U, 16 GB, 512 GB, CH
- Rating: Bewertungen ★★★★☆ 5
- Brand: Marke Mehr von HP

# Ablauf

---

Mit dem Digitec Use Case können wir nun einen generellen Ablauf definieren:

- 1) Ziel definieren (Ich möchte vollautomatisiert xy abrufen)
- 2) Das Ziel untersuchen (URL / HTML) + Stabilität
- 3) Einen "Proof of Concept" => Muss funktionieren, aber nicht perfekt sein
- 4) Den PoC weiterentwickeln, schärfen, verbessern
- 5) Den Skript vollautomatisiert ausführen lassen (z. B. mit Cron auf Linux)

# Wichtig!

Zu viel / zu oft kann dazu führen das eine Webseite "denkt" es handelt sich um einen **DoS (Denial of Service) Attacke**.

Auch wichtig, aber seltener: Urheberrechte und Datenschutz muss stets beachtet werden, auch beim Web Scraping!

# Scraper Rules

Jede grössere Webseite besitzt ein robots.txt. Dieses definiert die Regeln, an die sich Crawler (im wesentlichen WebScraper der Suchmaschinen) halten.



```
1 User-agent: *
2 Crawl-delay: 10
3 Allow: /pages/
4 Disallow: /scripts/
5
6 # more stuff
```

- User-agent: Sagt uns welche "Browser" bzw. Agenten erlaubt sind (\* heisst alle)
- Crawl-delay: Sekunden die gewartet werden muss, zwischen Crawl-Vorgängen
- Allow: Was dürfen die Bots "crawlen"
- Disallow: Was ist für die Bots nicht erlaubt

# Aufgabe Scraper Rules

Finden Sie die robots.txt Datei von [www.helsana.ch](http://www.helsana.ch) (Tipp: /robots.txt).

Was finden Sie heraus?

Was ist eine "Sitemap"?

# Verschiedene Scraper

---

Folgende "Scraper" werden oft verwendet:

- request
- BeautifulSoup
- Selenium\*

Die Empfehlung ist meistens BeautifulSoup bzw. bs4, da es viel einfacher zu handhaben ist als request. Selenium ist eher fürs "testen" gedacht und hat daher einen anderen Fokus und ist auch eher im JavaScript Kontext.

# Der Erste Scraper

Wir wollen nun einen ersten Scraper mit request bauen.

Bitte eine neue Datei "first\_scraper.py" im Userfolder erstellen und folgendes einfügen:



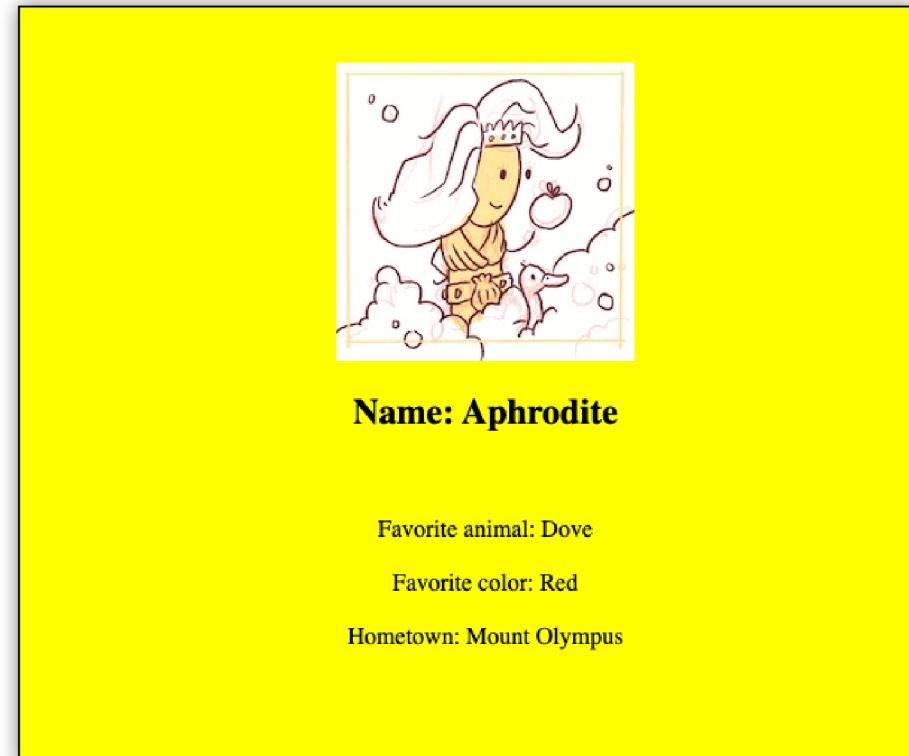
```
1 # 02_first_scraper.py
2
3 from urllib.request import urlopen
```

urllib.request ist Teil der Standard Python Library (<https://docs.python.org/3/library/index.html>) und daher immer installiert.

# Der Erste Scraper

Unser Ziel ist folgende Webseite:

<http://olympus.realpython.org/profiles/aphrodite>



# Der Erste Scraper

Unser Ziel ist folgende Webseite:

<http://olympus.realpython.org/profiles/aphrodite>



```
1 <html>
2   <head>
3     <title>Profile: Aphrodite</title>
4   </head>
5   <body bgcolor="yellow">
6     <center>
7       <br><br>
8       
9       <h2>Name: Aphrodite</h2>
10      <br><br>
11      Favorite animal: Dove
12      <br><br>
13      Favorite color: Red
14      <br><br>
15      Hometown: Mount Olympus
16    </center>
17  </body>
18 </html>
```

Auch der Quelltext ist überschaubar

# Der Erste Scraper

Wir entwickeln unser Skript weiter mit der URL:

```
● ● ●  
1 # 02_first_scraper.py  
2  
3 from urllib.request import urlopen  
4  
5  
6 url = "http://olympus.realpython.org/profiles/aphrodite"
```

# Der Erste Scraper

Und laden nun die Seite "herunter" (das ist was request wirklich macht)

```
● ● ●  
1 # 02_first_scraper.py  
2  
3 from urllib.request import urlopen  
4  
5  
6 url = "http://olympus.realpython.org/profiles/aphrodite"  
7 page = urlopen(url)
```

```
>>> page  
<http.client.HTTPResponse object at 0x105fef820>
```

Damit können wir noch sehr wenig anfangen

# Der Erste Scraper

Wir lesen die Webseite und stellen sicher, dass das utf-8 encoding stimmt:



```
1 # 02_first_scraper.py
2
3 from urllib.request import urlopen
4
5
6 url = "http://olympus.realpython.org/profiles/aphrodite"
7 page = urlopen(url)
8
9 html_bytes = page.read()
10 html = html_bytes.decode("utf-8")
11
12 print(html)
```



```
1 <html>
2 <head>
3 <title>Profile: Aphrodite</title>
4 </head>
5 <body bgcolor="yellow">
6 <center>
7 <br><br>
8 
9 <h2>Name: Aphrodite</h2>
10 <br><br>
11 Favorite animal: Dove
12 <br><br>
13 Favorite color: Red
14 <br><br>
15 Hometown: Mount Olympus
16 </center>
17 </body>
18 </html>
```

# Der Erste Scraper

---

Was können wir nun tun, um an Informationen (z. B. den Titel) zu kommen?



**Name: Aphrodite**

Favorite animal: Dove

Favorite color: Red

Hometown: Mount Olympus

# Der Erste Scraper

Eine Möglichkeit sind "String" Methoden von Python.

Wir suchen nach dem Titel, berechnen dessen Länge und das Ende (immer als Zeichenangabe):

```
>>> start_index = html.find("<title>") + len("<title>")
>>> end_index = html.find("</title>")
>>> title = html[start_index:end_index]
>>> title
```

# Der Erste Scraper

Eine Möglichkeit sind "String" Methoden von Python.

Wir suchen nach dem Titel, berechnen dessen Länge und das Ende (immer als Zeichenangabe):

```
● ● ●  
1 # 03_first_scraper_string.py  
2  
3 from urllib.request import urlopen  
4  
5  
6 url = "http://olympus.realpython.org/profiles/aphrodite"  
7 page = urlopen(url)  
8  
9 html_bytes = page.read()  
10 html = html_bytes.decode("utf-8")  
11  
12 # String Method  
13 start_index = html.find("<title>") + len("<title>")  
14 end_index = html.find("</title>")  
15 title = html[start_index:end_index]  
16 title
```

# Der Erste Scraper

Versuchen wir dasselbe mal noch mit poseidon:



```
1 # 04_first_scraper_string_poseidon.py
2
3 from urllib.request import urlopen
4
5
6 url = "http://olympus.realpython.org/profiles/poseidon"
7 page = urlopen(url)
8
9 html_bytes = page.read()
10 html = html_bytes.decode("utf-8")
11
12 # String Method
13 start_index = html.find("<title>") + len("<title>")
14 end_index = html.find("</title>")
15 title = html[start_index:end_index]
16 title
```



**Name: Poseidon**

Favorite animal: Dolphin

Favorite color: Blue

Hometown: Sea

# Der Erste Scraper

Versuchen wir dasselbe mal noch mit poseidon:

```
<head>
```

```
<title>Profile: Poseidon
```

Warum ist das so?



**Name: Poseidon**

Favorite animal: Dolphin

Favorite color: Blue

Hometown: Sea

# Der Erste Scraper

Versuchen wir dasselbe mal noch mit poseidon:

```
<head>  
<title>Profile: Poseidon
```

Warum ist das so?

Der öffnende `<title>`-Tag hat ein zusätzliches Leerzeichen vor der schliessenden spitzen Klammer (`>`), wodurch er als `<title>` dargestellt wird.



**Name: Poseidon**

Favorite animal: Dolphin

Favorite color: Blue

Hometown: Sea

# Der Erste Scraper

---

Versuchen wir dasselbe mal noch mit poseidon:

`html.find("<Titel>")` gibt -1 zurück, weil die genaue Teilzeichenkette "<Titel>" nicht existiert. Wenn -1 zu `len("<title>")` addiert wird, das 7 ist, wird der Variablen `start_index` der Wert 6 zugewiesen.

Das Zeichen bei Index 6 der Zeichenfolge `html` ist ein Zeilenumbruch (`\n`) direkt vor der öffnenden spitzen Klammer (`<`) des `<head>`-Tags. Das bedeutet, dass `html[start_index:end_index]` den gesamten HTML-Code zurückgibt, der mit diesem Zeilenumbruch beginnt und kurz vor dem `</title>`-Tag endet.

# Der Erste Scraper

---

Zusammengefasst: Unsere Variante funktioniert für Aphrodite, aber womöglich nur genau für diesen Fall...

# Regular Expressions

Wann immer wir nach Texten und Inhalten suchen, sollten auch Regular Expressions (re) in Betracht gezogen werden. Diese sind deutlich flexibler als die vorherigen String-Operationen.

Auch die Regular Expressions, Python Paket **re** sind Teil der Standard-Library und müssen nicht installiert werden.

Für Regular Expressions liegt ein Cheat-Sheet im Kursordner.

# Regular Expressions

Was bedeutet der Stern (\*)?



```
1 >>> re.findall("ab*c", "abcd")
2 ['abc']
3
4 >>> re.findall("ab*c", "acc")
5 ['ac']
6
7 >>> re.findall("ab*c", "abcac")
8 ['abc', 'ac']
9
10 >>> re.findall("ab*c", "abdc")
11 []
```

# Regular Expressions

Was bedeutet der re.IGNORECASE?



```
1 >>> re.findall("ab*c", "ABC")
2 []
3
4 >>> re.findall("ab*c", "ABC", re.IGNORECASE)
5 [ 'ABC' ]
```

# Regular Expressions

Was bedeutet der Punkt (.)?



```
1 >>> re.findall("a.c", "abc")
2 ['abc']
3
4 >>> re.findall("a.c", "abbc")
5 []
6
7 >>> re.findall("a.c", "ac")
8 []
9
10 >>> re.findall("a.c", "acc")
11 ['acc']
```

# Regular Expressions

Nun Punkt und Stern kombiniert, ergibt das Sinn?



```
1 >>> re.findall("a.*c", "abc")
2 ['abc']
3
4 >>> re.findall("a.*c", "abbc")
5 ['abbc']
6
7 >>> re.findall("a.*c", "ac")
8 ['ac']
9
10 >>> re.findall("a.*c", "acc")
11 ['acc']
```

# Regular Expressions

Das ist alles noch sehr theoretisch, lasst uns <https://regex101.com/> ausprobieren:



```
1 <html>
2 <head>
3 <title>Profile: Poseidon</title>
4 </head>
5 <body bgcolor="yellow">
6 <center>
7 <br><br>
8 
9 <h2>Name: Poseidon</h2>
10 <br><br>
11 Favorite animal: Dolphin
12 <br><br>
13 Favorite color: Blue
14 <br><br>
15 Hometown: Sea
16 </center>
17 </body>
18 </html>
```

<title.\*?>.\*?<\title.\*?>

<title.\*?>.\*?<\title.\*?>

# Regular Expressions

<title.\*?> stimmt mit dem einleitenden <TITLE> Tag in html überein.

.\*? passt ohne Einschränkung auf den gesamten Text nach dem einleitenden <TITLE> und hört bei der ersten Übereinstimmung für </title.\*?> auf.

</title.\*?> unterscheidet sich vom ersten Muster nur durch die Verwendung des Zeichens /, sodass es auf den abschliessenden </title />-Tag in html passt.



The screenshot shows a terminal window with three colored window control buttons (red, yellow, green) at the top. The window contains an HTML document with the following code:

```
1 <html>
2 <head>
3 <title>Profile: Poseidon</title>
4 </head>
5 <body bgcolor="yellow">
6 <center>
7 <br><br>
8 
9 <h2>Name: Poseidon</h2>
10 <br><br>
11 Favorite animal: Dolphin
12 <br><br>
13 Favorite color: Blue
14 <br><br>
15 Hometown: Sea
16 </center>
17 </body>
18 </html>
```

# Regular Expressions

Das ist alles noch sehr theoretisch, lasst uns <https://regex101.com/> ausprobieren:

```
● ● ●  
1 <html>  
2 <head>  
3 <title>Profile: Aphrodite</title>  
4 </head>  
5 <body bgcolor="yellow">  
6 <center>  
7 <br><br>  
8   
9 <h2>Name: Aphrodite</h2>  
10 <br><br>  
11 Favorite animal: Dove  
12 <br><br>  
13 Favorite color: Red  
14 <br><br>  
15 Hometown: Mount Olympus  
16 </center>  
17 </body>  
18 </html>
```

<title.\*?>.\*?<\title.\*?>

MATCH INFORMATION

Match 1 14-47 <title>Profile:·Aphrodite</title>

# Der Erste Scraper

Versuchen wir dasselbe mal noch mit poseidon:

```
● ● ●  
1 #05_first_scraper_re_poseidon.py  
2  
3 import re  
4 from urllib.request import urlopen  
5  
6 url = "http://olympus.realpython.org/profiles/dionysus"  
7 page = urlopen(url)  
8 html = page.read().decode("utf-8")  
9  
10 pattern = "<title.*?>.*?</title.*?>"  
11 match_results = re.search(pattern, html, re.IGNORECASE)  
12 title = match_results.group()  
13 title = re.sub("<.*?>", "", title) # Remove HTML tags  
14  
15 print(title)
```



**Name: Poseidon**

Favorite animal: Dolphin

Favorite color: Blue

Hometown: Sea

# Quizfrage

Wofür ist folgende Regular Expression?

/^([a-zA-Z0-9.\_%-]+@[a-zA-Z0-9.-]+\.[a-zA-Z]{2,6})\*/\$

# Beautiful Soup

Beautiful Soup ist ein Python-Paket zum Parsen von HTML- und XML-Dokumenten (auch mit fehlerhaftem Markup, d. h. nicht geschlossenen Tags, daher der **Name Stichwort Suppe "Tag Soup"**).

Es erstellt einen Parse-Baum für geparsete Seiten, der verwendet werden kann, um Daten aus HTML zu extrahieren, was für Web Scraping nützlich ist.

# Beautiful Soup

Wir starten mit dem gleichen Testobjekt:



```
1 # 06_bs_scraper.py
2
3 from bs4 import BeautifulSoup
4 from urllib.request import urlopen
5
6 url = "http://olympus.realpython.org/profiles/dionysus"
7 page = urlopen(url)
8 html = page.read().decode("utf-8")
9 soup = BeautifulSoup(html, "html.parser")
10
11 print(soup.get_text())
```

- Öffnet die URL `http://olympus.realpython.org/profiles/dionysus` unter Verwendung von `urlopen()` aus dem Modul `urllib.request`.
- Liest den HTML-Code der Seite als String und ordnet ihn der Variablen `html` zu.
- Erzeugt ein `BeautifulSoup`-Objekt und weist es der Variablen `soup` zu.

# Beautiful Soup

Lassen wir mal das get\_text() weg, was passiert?



```
1 # 06_bs_scraper.py
2
3 from bs4 import BeautifulSoup
4 from urllib.request import urlopen
5
6 url = "http://olympus.realpython.org/profiles/dionysus"
7 page = urlopen(url)
8 html = page.read().decode("utf-8")
9 soup = BeautifulSoup(html, "html.parser")
10
11 print(soup.get_text())
```

# Beautiful Soup

Beautiful Soup bietet von Haus aus viele Funktionen, die das Suchen bzw. "Parsen" für uns erleichtern:

<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>

- Beautiful Soup Documentation
  - Getting help
- Quick Start
- Installing Beautiful Soup
  - Installing a parser
- Making the soup
- Kinds of objects
  - Tag
    - Tag.name
    - Tag.attrs
  - NavigableString
  - BeautifulSoup
  - Special strings
    - Comment
    - For HTML documents
      - Stylesheet
      - Script
      - Template
    - For XML documents
      - Declaration
      - Doctype
      - CData
      - ProcessingInstruction
- Navigating the tree
  - Going down
    - Navigating using tag names
    - .contents and .children
    - .descendants
    - .string
    - .strings and .stripped\_strings
  - Going up
    - .parent
    - .parents
  - Going sideways
    - .next\_sibling and .previous\_sibling
    - .next\_siblings and .previous\_siblings
  - Going back and forth
    - .next\_element and .previous\_element
    - .next\_elements and .previous\_elements

# Aufgabe img

Wir wollen uns nun ein wenig mit beautiful soup beschäftigen. Dabei soll 01\_bs\_scraper\_starter.py aus exercises in den userfolder kopiert werden.

Gesucht werden alle Bilder (img) aus der dionysus Webseite?

<a href="#">Beautiful Soup Documentation</a>
▪ <a href="#">Getting help</a>
<a href="#">Quick Start</a>
<a href="#">Installing Beautiful Soup</a>
▪ <a href="#">Installing a parser</a>
<a href="#">Making the soup</a>
<a href="#">Kinds of objects</a>
▪ <a href="#">Tag</a>
▪ <a href="#">Tag.name</a>
▪ <a href="#">Tag.attrs</a>
▪ <a href="#">NavigableString</a>
▪ <a href="#">BeautifulSoup</a>
▪ <a href="#">Special strings</a>
▪ <a href="#">Comment</a>
▪ <a href="#">For HTML documents</a>
▪ <a href="#">Stylesheet</a>
▪ <a href="#">Script</a>
▪ <a href="#">Template</a>
▪ <a href="#">For XML documents</a>
▪ <a href="#">Declaration</a>
▪ <a href="#">Doctype</a>
▪ <a href="#">CData</a>
▪ <a href="#">ProcessingInstruction</a>
<a href="#">Navigating the tree</a>
▪ <a href="#">Going down</a>
▪ <a href="#">Navigating using tag names</a>
▪ <a href="#">.contents and .children</a>
▪ <a href="#">.descendants</a>
▪ <a href="#">.string</a>
▪ <a href="#">.strings and .stripped_strings</a>
▪ <a href="#">Going up</a>
▪ <a href="#">.parent</a>
▪ <a href="#">.parents</a>
▪ <a href="#">Going sideways</a>
▪ <a href="#">.next_sibling and .previous_sibling</a>
▪ <a href="#">.next_siblings and .previous_siblings</a>
▪ <a href="#">Going back and forth</a>
▪ <a href="#">.next_element and .previous_element</a>
▪ <a href="#">.next_elements and .previous_elements</a>

# Lösung img

Wir wollen uns nun ein wenig mit beautiful soup beschäftigen. Dabei soll 05\_bs\_scraper.py in den userfolder kopiert werden.

Gesucht werden alle Bilder (img) aus der dionysus Webseite?

```
soup.find_all("img")
```

```
[, ]
```

Beautiful Soup Documentation

- Getting help
- Quick Start
- Installing Beautiful Soup
  - Installing a parser
  - Making the soup
  - Kinds of objects
- Tag
  - Tag.name
  - Tag.attrs
  - NavigableString
  - BeautifulSoup
  - Special strings
    - Comment
    - For HTML documents
      - Stylesheet
      - Script
      - Template
    - For XML documents
      - Declaration
      - Doctype
      - CData
      - ProcessingInstruction
  - Navigating the tree
    - Going down
      - Navigating using tag names
      - .contents and .children
      - .descendants
      - .string
      - .strings and .stripped\_strings
    - Going up
      - .parent
      - .parents
    - Going sideways
      - .next\_sibling and .previous\_sibling
      - .next\_siblings and .previous\_siblings
    - Going back and forth
      - .next\_element and .previous\_element
      - .next\_elements and .previous\_elements

# Lösung img

Wir wollen uns nun ein wenig mit beautiful soup beschäftigen. Dabei soll 05\_bs\_scraper.py in den userfolder kopiert werden.

Gesucht werden alle Bilder (img) aus der dionysus Webseite?

Zusatztipp:

```
image1, image2 = soup.find_all("img")
```

```
image1["src"]  
'/static/dionysus.jpg'
```

Beautiful Soup Documentation  
▪ Getting help  
Quick Start  
Installing Beautiful Soup  
▪ Installing a parser  
Making the soup  
Kinds of objects  
▪ Tag  
▪ Tag.name  
▪ Tag.attrs  
▪ NavigableString  
▪ BeautifulSoup  
▪ Special strings  
▪ Comment  
▪ For HTML documents  
▪ Stylesheet  
▪ Script  
▪ Template  
▪ For XML documents  
▪ Declaration  
▪ Doctype  
▪ CData  
▪ ProcessingInstruction  
Navigating the tree  
▪ Going down  
▪ Navigating using tag names  
▪ .contents and .children  
▪ .descendants  
▪ .string  
▪ .strings and .stripped\_strings  
▪ Going up  
▪ .parent  
▪ .parents  
▪ Going sideways  
▪ .next\_sibling and .previous\_sibling  
▪ .next\_siblings and .previous\_siblings  
▪ Going back and forth  
▪ .next\_element and .previous\_element  
▪ .next\_elements and .previous\_elements

# Beautiful Soup

Gewisse Elemente erlauben einen vereinfachten Zugriff:

```
soup.title  
<title>Profile: Dionysus</title>
```

```
soup.title.string  
'Profile: Dionysus'
```

# Beautiful Soup

Aber wir interessieren uns vor allem für eine sehr genaue Suche:

```
soup.find_all("img", src="/static/dionysus.jpg")
[]
```

Es wird geprüft, ob ein Tag mit diesem  
Inhalt existiert.

# Aufgabe Multisite

Wir wollen nun mit 06\_bs\_multi.py arbeiten.

Bitte kopieren Sie diesen Code in ihren userfolder und erweitern Sie ihn so, dass folgendes Ergebnis zu sehen ist:

```
http://olympus.realpython.org/profiles/aphrodite  
http://olympus.realpython.org/profiles/poseidon  
http://olympus.realpython.org/profiles/dionysus
```

# Lösung Multisite



```
1 from urllib.request import urlopen
2 from bs4 import BeautifulSoup
3
4 base_url = "http://olympus.realpython.org"
5 html_page = urlopen(base_url + "/profiles")
6 html_text = html_page.read().decode("utf-8")
7
8 soup = BeautifulSoup(html_text, "html.parser")
9
10 for link in soup.find_all("a"):
11     link_url = base_url + link["href"]
12     print(link_url)
```

# Mechanical Soup

Ein anderes Thema, aber auch sehr oft gefragt, ist die **Interaktion** mit Formularen.

Zu diesem Zweck werden wir uns ein weiteres Paket von PyPI anschauen, [MechanicalSoup](#).



# Mechanical Soup

Ein anderes Thema, aber auch sehr oft gefragt, ist die **Interaktion** mit Formularen.

Zu diesem Zweck werden wir uns ein weiteres Paket von PyPI anschauen, [MechanicalSoup](#).



# Mechanical Soup

Als Erstes brauchen wir ein Browser-Objekt:



```
1 import mechanicalsoup  
2 browser = mechanicalsoup.Browser()
```

Browser-Objekte stellen den Headless-Webbrowser dar. Man kann diese verwenden, um eine Seite aus dem Internet aufzurufen, indem man eine URL an die Methode `.get()` übergibt:



```
1 url = "http://olympus.realpython.org/login"  
2 page = browser.get(url)
```

# Mechanical Soup

---

Und so sieht das Formular aus: <http://olympus.realpython.org/login>

The image shows a login interface with a yellow background. At the top, the text "Please log in to access Mount Olympus:" is displayed in bold black font. Below this, there are two input fields: one for "Username" and one for "Password", both represented by white rectangular boxes with black outlines. At the bottom center is a single button labeled "Submit".

Gerne hierfür 03\_mech\_soup.py ausprobieren.

# Mechanical Soup

Wir sehen nun folgendes:

<Response [200]>

Was hat das zu bedeuten?

# Mechanical Soup

---

Wir sehen nun folgendes:

<Response [200]>

Was hat das zu bedeuten?

Die Zahl 200 steht für den von der Anfrage zurückgegebenen Statuscode.  
Ein Statuscode von 200 bedeutet, dass die Anfrage erfolgreich war.

# Mechanical Soup

Das Login, netterweise auf realpython.com zur Verfügung gestellt, sieht folgendermassen aus:

<b>Username</b>	<b>Password</b>
zeus	ThunderDude

Gerne kurz auf <http://olympus.realpython.org/login> ausprobieren.

# Mechanical Soup

Und hier ist noch das HTML dazu:

```
● ● ●  
1 <html>  
2   <head>  
3     <title>Log In</title>  
4   </head>  
5   <body bgcolor="yellow">  
6     <center>  
7       <br><br>  
8       <h2>Please log in to access Mount Olympus:</h2>  
9       <br><br>  
10      <form name="login" action="/login" method="post">  
11        Username: <input type="text" name="user"><br>  
12        Password: <input type="password" name="pwd"><br><br>  
13        <input type="submit" value="Submit">  
14      </form>  
15    </center>  
16  </body>  
17 </html>
```

# Mechanical Soup

Und nun wollen wir ein wenig Code dazu schreiben (im userfolder):

```
● ● ●  
1 # 08_mech_soup_login.py  
2 import mechanicalsoup  
3  
4 # 1  
5 browser = mechanicalsoup.Browser()  
6 url = "http://olympus.realpython.org/login"  
7 login_page = browser.get(url)  
8 login_html = login_page.soup  
9  
10 # 2  
11 form = login_html.select("form")[0]  
12 form.select("input")[0]["value"] = "zeus"  
13 form.select("input")[1]["value"] = "ThunderDude"  
14  
15 # 3  
16 profiles_page = browser.submit(form, login_page.url)  
17  
18 # 4  
19 print(profiles_page.url)
```

Die Rückgabe scheint unspektakulär:  
<http://olympus.realpython.org/profiles>

Jedoch ist das die URL die man erhält,  
wenn man sich **erfolgreich**  
**eingeloggt** hat.

# Hinweis

---

Hacker können automatisierte Programme wie das zuvor beschriebene verwenden, um Anmeldungen zu erzwingen, indem sie **schnell viele verschiedene Benutzernamen und Kennwörter ausprobieren**, bis sie eine funktionierende Kombination gefunden haben.

Abgesehen davon, dass dies höchst illegal ist, sperren heutzutage fast alle Websites solche Angriffe und melden Ihre IP-Adresse, wenn sie sehen, dass Sie zu viele fehlgeschlagene Anfragen stellen, also bitte nicht versuchen...

Man nennt diese Attacke "Brute Force"

# Mechanical Soup

Zu guter Letzt wollen wir noch die Links auslesen:

```
 1 # 09_mech_soup_login_ext.py
 2 import mechanicalsoup
 3
 4 # 1
 5 browser = mechanicalsoup.Browser()
 6 url = "http://olympus.realpython.org/login"
 7 login_page = browser.get(url)
 8 login_html = login_page.soup
 9
10 # 2
11 form = login_html.select("form")[0]
12 form.select("input")[0]["value"] = "zeus"
13 form.select("input")[1]["value"] = "ThunderDude"
14
15 # 3
16 profiles_page = browser.submit(form, login_page.url)
17
18 # 4
19 print(profiles_page.url)
20
21 # 5
22 links = profiles_page.soup.select("a")
23 for link in links:
24     address = link["href"]
25     text = link.text
26     print(f"{text}: {address}")
```

# Capstone Project

Zu guter Letzt wollen wir nun noch ein Projekt miteinander realisieren.

Eine Idee um was es gehen wird?



# Capstone Project

Wir wollen herausfinden, wie viele Lifte geöffnet sind und welche Temperatur herrscht :-)

<https://www.flumserberg.ch/de/informationen/Geoeffnete-Anlagen/Winter>



# Capstone Project

The screenshot shows a weather website interface with a sidebar on the left containing icons for live info cards: 17/17, Betriebszeiten, -1 °C, Webcams, and Karte. The main content area displays a large red banner with the text "WINTERBERG Mittwoch, 10". Below the banner, there are two tables showing snow depth information:

Maschgenkamp	
Schneehöhe:	100 cm
Neuschnee:	0 cm

At the bottom, a developer tools overlay shows the DOM structure for the "live-infos\_\_wrapper js-midnight" element, which contains a list of items with icons and text.

```
> <div class="skip-links"></div>
> <header></header>
> <nav id="table-of-content" class="js-toc toc"></nav>
<main id="main-content" role="main" class="content-block main-content">
...   <div class="live-infos__wrapper js-midnight" style="position: fixed; top: 0px; left: 0px; right: 0px; overflow: hidden; height: 352px;"> == $0
    <div class="midnightHeader default" style="position: absolute; overflow: hidden; inset: 0px; transform: translateY(0%) translateZ(0px);">
      <div class="midnightInner" style="position: absolute; overflow: auto; inset: 0px; transform: translateY(0%) translateZ(0px);">
        <ul class="list-unstyled live-infos__list">
          <li class="live-infos__item">
            <a href="/de/informationen/Geoeffnete-Anlagen/Winter" class="target=_self">
              <span class="icon icon-facility live-infos__item-icon" aria-label="Anlagen" title="Anlagen">
                :before
              </span>
              <span class="live-infos__item-text"></span>
            </a>
          </li>
        <li class="live-infos__item"></li>
        <li class="live-infos__item"></li>
        <li class="live-infos__item"></li>
        <li class="live-infos__item"></li>
      </ul>
```

# Capstone Project

Eine kleine Starthilfe besteht unter capstone flums.py, diese kann man in den userfolder kopieren oder auch ganz von vorne beginnen

```
● ● ●

1 # flums.py
2 from bs4 import BeautifulSoup
3 from urllib.request import urlopen
4 import ssl
5 ssl._create_default_https_context = ssl._create_unverified_context
6
7
8 base_url = "https://www.flumserberg.ch/de/informationen/Geoeffnete-Anlagen"
9 html_page = urlopen(base_url + "/Winter")
10 html_text = html_page.read().decode("utf-8")
11
12 soup = BeautifulSoup(html_text, "html.parser")
13 live_items = soup.find_all("li", class_="live-infos__item")
14
15 print(live_items)
```

# Capstone Project - Solution



```
1 from bs4 import BeautifulSoup
2 from urllib.request import urlopen
3 import ssl
4
5 ssl._create_default_https_context = ssl._create_unverified_context
6
7 base_url = "https://www.flumserberg.ch/de/informationen/Geoeffnete-Anlagen"
8 html_page = urlopen(base_url + "/Winter")
9 html_text = html_page.read().decode("utf-8")
10
11 soup = BeautifulSoup(html_text, "html.parser")
12 live_items = soup.find_all("li", class_="live-infos__item")
13
14 slopes = live_items[0]
15 temperature = live_items[2]
16
17
18 # print(slopes)
19 # print(temperature)
20
21 def clean_and_print_data(soup_element):
22     print(soup_element.find("span", class_="live-infos__item-text").get_text().strip())
23
24
25 clean_and_print_data(slopes)
26 clean_and_print_data(temperature)
```

# Zusatzaufgabe

Wer noch mehr Challenge sucht, darf gerne auch noch versuchen die Schneehöhe abzuholen:

WINTERBERICHT VOM  
**Mittwoch, 10. Januar 2024**

<b>Maschgenkamm</b> Schneehöhe: 100 cm Neuschnee: 0 cm	<b>Tannenboden</b> 80 cm 0 cm	 Totale Pistenlänge <b>62/65 km</b>
 Offene Anlagen <b>17/17</b>		

# Vielen Dank

Das war alles für den Moment

---

Ich liebe es, über Vue, Python und mehr zu diskutieren

Sie finden mich unter @dpinezich

