

Scraping with Python

Isaiah Hull



Sveriges Riksbank

October 13, 2018



References

1. “Web Scraping with Python: Collecting Data from the Modern Web”
 - ▶ Ryan Mitchell (2015)
2. “Python Web Scraping”
 - ▶ Katharine Jarmul & Richard Lawson (2017)

Overview

1. Introduction
2. Scraping
3. Crawling
4. Dynamic Content
5. Logins
6. Storing Data
7. Concurrent Downloading

Introduction

How does a browser work?¹

- ▶ Alice has a website.
- ▶ Bob wants to view Alice's site.
- ▶ Bob's computer creates a header and body.
 - ▶ Header = MAC address + Alice's IP address.
 - ▶ Body = GET request for index.html.
- ▶ Header and body bundled as packet and sent to Alice via intermediary servers.
- ▶ Alice's server locates the file and returns it to Bob.

¹See Mitchell (2015) for additional detail.

Introduction

How does a browser work?

- ▶ A browser isn't needed for anything Bob did.
- ▶ We can do this with Python.

Introduction

How does a browser work?

```
#lecture1_example1.py
```

```
$user python
```

```
>>> from urllib2 import urlopen
```

```
>>> url "http://www.math.unm.edu/  
writingHTML/tut/index.html"
```

```
>>> html = urlopen(url)
```

```
>>> print(html.read())
```

Introduction

How does a browser work?

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 3.2//EN">
<html>
<head>
<title>Writing HTML</title>
<META name="..."> </head>
<body bgcolor="FFFFFF">
...
<p>
We suggest that you proceed through the lessons in order,
but at any time you can return to the index to jump to a
different lesson. Within each lesson you can compare
your work to a sample file for that lesson. ...
<p>
...
</body>
</html>
```

Introduction

Scraping ethics

- ▶ Check robots.txt file before scraping.
 - ▶ Lists suggested restrictions on automated interaction.
 - ▶ Good Internet ethics.
 - ▶ Lowers probability of an IP ban.
 - ▶ Protocol: <http://www.robotstxt.org>
- ▶ Robots.txt examples:
 - ▶ <https://www.ecb.europa.eu/robots.txt>
 - ▶ <http://www.bcb.gov.br/robots.txt>

Introduction

Scraping ethics

```
User-agent:  *
Sitemap:    https://www.ecb.europa.eu/sitemap.xml
Disallow:   /*_content.bg.html$
Disallow:   /*_content.cs.html$
Disallow:   /*_content.da.html$
Disallow:   /*_content.de.html$
Disallow:   /*_content.el.html$
...
Disallow:   /ecb/10ann/shared/movies/
Disallow:   /ecb/educational/pricestab/shared/movie/
Disallow:   /ecb/educational/shared/movies/
...
Crawl-delay: 5
```

Introduction

Scraping ethics

- ▶ **User-agent:** * → Rules apply to all users.
- ▶ **Sitemap:** ... → Sitemap located here.
- ▶ **Disallow:** ... → Don't scrape this URL.
- ▶ **Crawl-delay:** 5 → Use a 5-second delay between requests.

Introduction

Scraping ethics

```
...
User-agent:  Offline Explorer/1.9
Disallow:  /

...
User-agent:  htdig/3.1.4 (****@bcb.gov.br)
Disallow:

...
User-agent:  *
Disallow:  css/*

...
```

Introduction

Scraping ethics

- ▶ **User-agent: Offline Explorer/1.9**
Disallow: /

- ▶ → User-agent: Offline Explorer/1.9 shouldn't scrape any pages.

User-agent: htdig/3.1.4 (**@bcb.gov.br)**
Disallow:

- ▶ → User-agent: htdig/3.1.4 (****@bcb.gov.br) is unrestricted.

- ▶ **User-agent: ***
Disallow: css/*

- ▶ → All other user-agents are restricted.

Introduction

Scraping ethics

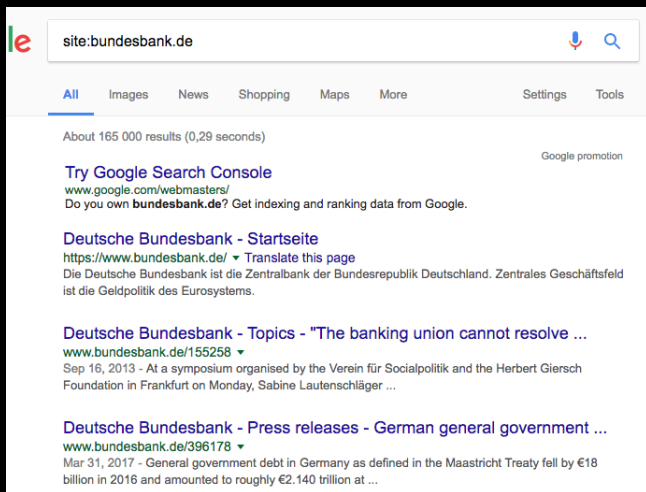
- ▶ Use the sitemap to improve scraping efficiency.
 - ▶ Avoid crawling and scraping duplication.
 - ▶ Avoid visiting old pages if unnecessary.

Introduction

Estimating website size

- ▶ Crawling and scraping strategy may depend on website size.
- ▶ Estimate site size.
- ▶ Use the “site” parameter on Google to obtain estimate.
 - ▶ `site:bundesbank.de`

Estimating website size



The screenshot shows a Google search interface. The search bar contains the text "site:bundesbank.de". Below the search bar, there are tabs for "All", "Images", "News", "Shopping", "Maps", and "More". The "All" tab is selected. Below the tabs, it says "About 165 000 results (0,29 seconds)". To the right of this, there is a "Google promotion" link. The first search result is titled "Try Google Search Console" and includes the URL "www.google.com/webmasters/" and the text "Do you own bundesbank.de? Get indexing and ranking data from Google." The second search result is titled "Deutsche Bundesbank - Startseite" and includes the URL "https://www.bundesbank.de/" and a link to "Translate this page". The text below the title says "Die Deutsche Bundesbank ist die Zentralbank der Bundesrepublik Deutschland. Zentrales Geschäftsfeld ist die Geldpolitik des Eurosystems." The third search result is titled "Deutsche Bundesbank - Topics - 'The banking union cannot resolve ...'" and includes the URL "www.bundesbank.de/155258". The text below the title says "Sep 16, 2013 - At a symposium organised by the Verein für Socialpolitik and the Herbert Giersch Foundation in Frankfurt on Monday, Sabine Lautenschläger ...". The fourth search result is titled "Deutsche Bundesbank - Press releases - German general government ..." and includes the URL "www.bundesbank.de/396178". The text below the title says "Mar 31, 2017 - General government debt in Germany as defined in the Maastricht Treaty fell by €18 billion in 2016 and amounted to roughly €2.140 trillion at ...".

Introduction

Identifying website technology

```
$user pip install wad
```

```
$user wad -u https://pypi.python.org
```


Introduction

Identifying website technology

```
{  
  "https://pypi.python.org/pypi": [  
    {  
      "type": "cache-tools",  
      "app": "Varnish",  
      "ver": null  
    },  
    {  
      "type": "web-servers",  
      "app": "Nginx",  
      "ver": "1.10.3"  
    }  
  ]  
}
```

Introduction

Identifying website technology

- ▶ Example
 - ▶ Varnish front-end cache tools.
 - ▶ Nginx back-end webserver.
- ▶ Dynamic content
 - ▶ ASP.NET
 - ▶ JQuery
 - ▶ Modernizer

Scraping

BeautifulSoup

- ▶ Python package for HTML and XML parsing.
- ▶ Generates parse tree from extracted code.
- ▶ Allows programmers to access and modify HTML.
- ▶ Fixes HTML errors, layouts, and indenting.

Scraping

BeautifulSoup

- Broken HTML

```
<ul class=country>  
    <li>Area  
    <li>Population  
</ul>
```

Scraping

BeautifulSoup

```
#lecture1_example2.py
```

```
$user pip install bs4
```

```
$user python
```

```
>>> from bs4 import BeautifulSoup
```

```
>>> broken_html = "<ul
```

```
class=country><li>Area<li>Population</ul>"
```

```
>>> soup = BeautifulSoup(broken_html)
```

```
>>> fixed_html = soup.prettify()
```

```
>>> print(fixed_html)
```

Scraping

BeautifulSoup

- Fixed HTML

```
<ul class="country">  
  <li>  
    Area  
  </li>  
  <li>  
    Population  
  </li>  
</ul>
```

BeautifulSoup

Parser	License	Implementation language(s)	Latest date*	HTML parsing ^[1]	HTML5-compliant parsing	Clean HTML**	Update HTML***
Lambda Soup	BSD-2-Clause	OCaml	2016-12-10 ^[2]	Yes	Yes	?	?
html.parser	Python S. F. L.	Python	2016-06-27 ^[3]	Yes	?	No	No
Html Agility Pack	Microsoft Public License	C#	2016-07-14 ^[4]	Yes	?	No	?
Beautiful Soup	Python S. F. L.	Python	2016-08-02 ^[5]	Yes	Partial ^[6]	Yes	Yes
Gumbo	Apache License 2.0	C	2015-05-01	Yes	Yes	?	?
html5ever	Apache License 2.0	Rust	2016-02-23	Yes	Yes	?	?
html5lib	MIT License	Python (and PHP, six years ago)	2016-07-15 ^[7]	Yes	Yes	Yes	No
HTML::Parser	Perl license	Perl	2013-03-28	Yes	No ^[8]	?	?
WebGear	GPL3	Perl	2017-03-10	Yes	Yes	?	?
htmlPurifier	GNU Lesser GPL	PHP	2009-03-25 ^[9]	No	No	Yes	Yes
HTML Tidy	W3C license	ANSI C	2017-03-01 ^[10]	Yes ^[11]	Yes	Yes ^[11]	Yes
HtmlUnit	Apache License 2.0	Java	2016-05-27 ^[12]	Yes	?	No	No
HtmlCleaner	BSD License ^[13]	Java	2015-08-24	No	No	Yes	?
Hubbub	MIT License	C	2016-02-16	Yes	Yes ^[14]	?	?

https://en.wikipedia.org/wiki/Comparison_of_HTML_parsers

Scraping

BeautifulSoup

```
#lecture1_example3.py
```

```
$user python
```

```
>>> from urllib2 import urlopen
```

```
>>> from bs4 import BeautifulSoup
```

```
>>> url = "https://en.wikipedia.org/  
wiki/Richard_Thaler"
```

```
>>> html = urlopen(url)
```

```
>>> soup = BeautifulSoup(html.read())
```

```
>>> print(soup.h1)
```

```
<h1 class="firstHeading" id="firstHeading"  
lang="en">Richard Thaler</h1>
```

```
>>> print(soup.h1.text)
```

```
Richard Thaler
```


Scraping

BeautifulSoup

- ▶ `html.read()` → retrieve HTML content
- ▶ `BeautifulSoup(html.read())` → convert HTML into BeautifulSoup object
 - ▶ `html` → `head` → `body` → `h1` → `div`
 - ▶ `soup.html.body.h1 = soup.body.h1 = soup.html.h1`

Scraping

BeautifulSoup

- ▶ Exception handling
 - ▶ 404 Page Not Found
 - ▶ 500 Internal Server Error
- ▶ `urlopen` throws `HTTPError` in all cases
- ▶ Use `try/except` structure

Scraping

BeautifulSoup

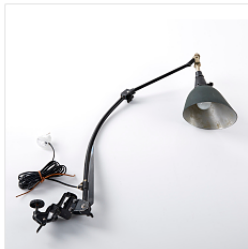
```
#lecture1_example4.py
```

```
$user python
```

```
>>> from bs4 import BeautifulSoup
>>> from urllib2 import urlopen
>>> from urllib2 import HTTPError
>>> import time
>>> url = "http://google.com/404"
>>> try:
    html = urlopen(url)
    soup = BeautifulSoup(html.read())
>>> except HTTPError as e:
    if str(e).find('404') != -1:
        time.sleep(60)
    ...
```

BeautifulSoup

All auctions



492993 CURT FISCHER, bordslampa, "114", Midgard, 1930-tal, ...
Estimate: SEK 5 000 SEK
🕒 < 1 minute left, 2017-10-15 kl 14:42
Slakthusgatan 22, Stockholm
Bid: SEK 3 000 SEK



492971 BÖRGE MOGENSEN, stolar, 6 st, ek, tygklädsel, Karl ...
Estimate: SEK 3 000 SEK
🕒 4 minutes left, 2017-10-15 kl 14:45
Magasin 5, Stockholm



492917 CHANEL, sjal mönstrad med broscher på brun ullbotten med ...
Estimate: SEK 2 000 SEK
🕒 7 minutes left, 2017-10-15 kl 14:48
Magasin 5, Stockholm
Bid: SEK 1 800 SEK

<https://online.auktionsverket.se>

Scraping

BeautifulSoup

```
#lecture1_example5.py
```

```
$user python
```

```
>>> from urllib2 import urlopen
```

```
>>> from bs4 import BeautifulSoup
```

```
>>> url = "https://online.auktionsverket.com/"
```

```
>>> html = urlopen(url)
```

```
>>> soup = BeautifulSoup(html.read())
```

Scraping

BeautifulSoup

```
#lecture1_example5.py (continued)
```

```
>>> soup.title
<title>Stockholms Auktionsverk Online</title>
>>> soup.title.name
'title'
>>> soup.title.string
u'Stockholms Auktionsverk Online'
>>> soup.title.parent.name
'head'
>>> soup.p
<p>Invitation to consign</p>
```

Scraping

BeautifulSoup

```
#lecture1_example5.py (continued)
```

```
>>> soup.a
```

```
<a href="/"><div style="padding-left:0.7em;  
padding-right:0.4em; float:left;">Home</div></a>
```

```
>>> soup.find_all('a')
```

```
[<a href="/"><div style="padding-left:0.7em;  
padding-right:0.4em; float:left;">Home</div></a>, <a  
href="http://auktionsverket.com/news/"><div  
style="padding-left:0.25em; padding-right:0.4em;  
float:left; margin-left:0.4em;">Right now</div></a>,  
...]
```

Scraping

BeautifulSoup

```
#lecture1_example5.py (continued)
```

```
>>> links = soup.find_all('a')
```

```
>>> for link in links[0:5]:
```

```
    print link
```

```
<a href="/"><div style="padding-left:0.7em; padding-right: ...
```

```
<a href="http://auktionsverket.com/news/"> ...
```

```
<a href="http://auktionsverket.com/buy/kop-online...
```

```
<a href="http://auktionsverket.com/salja/">...
```

```
<a href="http://auktionsverket.com/sell/valuations/">...
```


Scraping

BeautifulSoup

- Documentation

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

Scraping

Lxml

- ▶ Built on XML parsing library in C
- ▶ Faster than BeautifulSoup
- ▶ More difficult to install on Windows

Lxml

A



Joseph Aldy

Associate Professor of Public Policy

79 John F. Kennedy St. Taubman-Ofer Bldg 382

| [617-496-7213](tel:617-496-7213)

[Email](#) | [Mailing Address](#)



Graham Allison

Douglas Dillon Professor of Government

79 John F. Kennedy St. Littauer Bldg 346

| [617-496-6099](tel:617-496-6099)

[Email](#) | [Mailing Address](#)

Scraping

Lxml

```
#lecture1_example6.py

$user pip install lxml
$user python
>>> from lxml import html
>>> from urllib2 import urlopen
>>> import cssselect
>>> url = "https://www.hks.harvard.edu/
faculty-directory"
>>> page = urlopen(url).read()
>>> tree = html.fromstring(page)
>>> h2 = tree.cssselect('h2')
>>> print(h2[5].text_content().strip())
Joseph Aldy
```

Scraping

Lxml

```
#lecture1_example6.py (continued)
```

```
>>> for name in h2[5:]:  
        print(name.text_content().strip())
```

Joseph Aldy

Graham Allison

Alan Altshuler

Matthew Andrews

Arthur Applbaum

Cecile Aptel

Christopher Avery

...

Scraping

Lxml

- Documentation

<http://lxml.de/>

Lambda expressions

- ▶ One-line functions
- ▶ May be passed to functions
- ▶ BeautifulSoup accepts as arguments
- ▶ Can be combined with `find_all()`

Scraping

Lambda expressions

```
#lecture1_example7.py
```

```
$user python
```

```
>>> from urllib2 import urlopen
```

```
>>> from bs4 import BeautifulSoup
```

```
>>> import re
```

```
>>> url = "https://online.auktionsverket.com/  
auktion/Bocker/'
```

```
>>> html = urlopen(url)
```

```
>>> soup = BeautifulSoup(html.read())
```


Lambda expressions

Auctions Böcker/Kartor (35)

Subcategories: All auctions, 16th century books, Children's books, Manuscripts, History, Atlases/Maps, Cookery books, Art Reference, Medicine, Military, Natural Sciences, Prints /Illustrated books, Travels, Literature, Swedish Topography, Theology/Philosophy



490931 MAP OF MARTINIQUE. BELLIN - HOMANN HEIRS. Carte de l'Isle ...
Estimate: SEK 1 500 SEK
🕒 47 minutes left, 2017-10-15 kl 20:26
Geijersgatan 14, Gothenburg

Bid: SEK 700 SEK



490926 MAP OF IRELAND. LAURIE & WHITTLE. A New Map of Ireland ...
Estimate: SEK 1 000 SEK
🕒 1 hour 1 minutes left, 2017-10-15 kl 20:40
Geijersgatan 14, Gothenburg

Bid: SEK 800 SEK



490941 SEACHART NORWAY. DONCKER, HENRICK. De Custen Van Noorwegen, ...
Estimate: SEK 3 000 SEK
🕒 1 hour 39 minutes left, 2017-10-15 kl 21:18
Geijersgatan 14, Gothenburg

Bid: SEK 5 600 SEK



490937 MAP OF GREAT BRITAIN. JANSSONIUS, J. Magnae Britanniae et ...
Estimate: SEK 1 500 SEK
🕒 1 hour 55 minutes left, 2017-10-15 kl 21:34
Geijersgatan 14, Gothenburg

Bid: SEK 2 200 SEK

Scraping

Lambda expressions

```
#lecture1_example7.py (continued)
```

```
>>> maps = soup.findAll(lambda tag:  
tag.text.find('IRELAND')>-1)
```

```
>>> ireland_map = maps[1].text
```

```
>>> print(ireland_map)
```

```
490926 MAP OF IRELAND. LAURIE WHITTLE. A New Map of  
Ireland ...Estimate: SEK 1 000 SEK  
1 hour 1 minutes left, 2017-10-15 kl  
20:40Geijersgatan 14, Gothenburg  
Bid: SEK 800 SEK
```

Regular expressions

- ▶ Rules for string matching
- ▶ String satisfies conditions \rightarrow return it
- ▶ Examples:
 - ▶ aa^* \rightarrow the letter a, followed by any number of a's.
 - ▶ bbb \rightarrow three consecutive b's.
 - ▶ $(cb)^*$ \rightarrow any multiple of the pair cb
 - ▶ $(d|)$ \rightarrow d or no d

Regular expressions

► Examples:

- $[A - Za - z]^+$ → contains at least one uppercase letter or one lowercase letter.
- $[A - Za - z0 - 9 \backslash . _ +]^+$ → contains at least one uppercase letter, one lowercase letter, one number 0-9, a period, a plus sign, or an underscore.
- $*$ → matches preceding character(s) 0 or more times
- $+$ → matches preceding character(s) 1 or more times
- $[]$ → matches any character within brackets

Scraping

Regular expressions: examples

```
test_string = """
At 7:00 on 10/10/2015, the central bank announced a
$40,000,000,000 bond purchase. It also lowered its
target rate to 0.25%.
"""
```

Scraping

Regular expressions: examples

1. Extract time string.
2. Find the dollar amount of bonds.
3. Extract the date string.
4. Extract the interest rate.
5. Remove \$ and % symbols.

Scraping

Regular expressions: examples

```
# Extract the time string
```

```
$user python
```

```
>>> import re
```

```
>>> test_string = """
```

```
At 7:00 on 10/10/2015, the central bank announced a  
$40,000,000,000 bond purchase. It also lowered its  
target rate to 0.25%.
```

```
"""
```

Scraping

Regular expressions: examples

```
# Extract the time string
```

```
>>> pattern = re.compile('\d:\d\d')
```

```
>>> re.search(pattern, test_string).group()
```

```
7:00
```


Regular expressions: examples

```
# Extract the time string
```

```
>>> pattern = re.compile('\d{1,2}:\d{2}')
```

```
>>> re.search(pattern, test_string).group()
```

```
7:00
```

Scraping

Regular expressions: examples

```
# Extract the time string
```

```
>>> pattern = re.compile('[0-9]{1,2}:[0-9]{2}')
```

```
>>> re.search(pattern, test_string).group()
```

```
7:00
```

Scraping

Regular expressions: examples

```
# Find the dollar amount of bonds
```

```
>>> pattern = re.compile('\$\d+')
>>> re.search(pattern, test_string).group()
```

```
$40
```

Scraping

Regular expressions: examples

```
# Find the dollar amount of bonds
```

```
>>> pattern = re.compile('\$(\d|,)+')
```

```
>>> re.search(pattern, test_string).group()
```

```
$40,000,000,000
```

Regular expressions: examples

```
# Extract the date string
```

```
>>> pattern = re.compile('\d{1,2}/\d{1,2}/\d{4}')  
>>> re.search(pattern, test_string).group()  
'10/10/2015'
```

Scraping

Regular expressions: examples

```
# Extract the interest rate
```

```
>>> pattern = re.compile('\d+\.\d+%')
```

```
>>> re.search(pattern, test_string).group()
```

```
0.25%
```

Regular expressions: examples

```
# Remove all % and $ symbols
```

```
>>> pattern = re.compile('(\%|\$)')  
>>> no_symbols = re.sub(pattern, '',  
test_string).group()  
>>> print no_symbols
```

Scraping

Regular expressions: examples

At 7:00 on 10/10/2015, the central bank announced a 40,000,000,000 bond purchase. It also lowered its target rate to 0.25.

Regular expressions

- ▶ Documentation
 - ▶ <https://docs.python.org/2/library/re.html>
- ▶ Regex Tester
 - ▶ <https://pythex.org/>

Scraping

Regular expressions

```
#lecture1_example7.py (continued)
```

```
>>> map_ascii = re.sub(r`[^\\x00-\\x7F]+`,` `, map)
```

```
>>> re.findall("[0-9]+ hour [0-9]+ minutes",
```

```
map_ascii)
```

```
[u'1 hour 1 minutes']
```

```
>>> re.findall("SEK [0-9]+",map_ascii)[1]
```

```
u`SEK 800`
```

XPath selectors

- ▶ Parse trees often fail to properly clean up code
- ▶ Difficult to reliably extract same element
- ▶ Alternative to CSS selector
- ▶ Based on XML document hierarchy

XPath selectors

- ▶ Examples:

- ▶ All link \rightarrow ``//a``
- ▶ All divs with class “green” \rightarrow ``//div[@class="green"]``
- ▶ Select ul with id “country” \rightarrow ``//ul[@id="country"]``
- ▶ Select text from all paragraphs \rightarrow ``//p/text()``

Scraping

CSS selectors

- ▶ Examples:
 - ▶ All link \rightarrow ``a``
 - ▶ All divs with class “green” \rightarrow ``div.green``
 - ▶ Select ul with id “country” \rightarrow ``ul#country``
 - ▶ Select text from all paragraphs \rightarrow ``p`*`

Crawling

Overview

- ▶ Scrapers target a specific site or sites to collect detailed information
- ▶ Crawlers collect more generic information from many sites
 - ▶ Entire Internet
 - ▶ Targeted top-level domain

Crawling

Single-Domain Crawling

```
#lecture1_example8.py
```

```
$user python
```

```
>>> from urllib2 import urlopen
>>> from bs4 import BeautifulSoup
>>> import re
>>> url = "https://en.wikipedia.org/wiki/Commodity"
>>> html = urlopen(url)
>>> soup = BeautifulSoup(html.read())
>>> for link in soup.findAll("a"):
    if "href" in link.attrs:
        print(link.attrs["href"])
```

Crawling

Single-Domain Crawling

```
#mw-head
#p-search
...
/wiki/Legal_personality
/wiki/Cooperative
/wiki/Corporation
...
/wiki/Economic_development
/wiki/Economic_statistics
/wiki/File:Coffee_Beans_Phographed_in_Macro.jpg
/wiki/File:Loose_leaf_darjeeling_tea_twinings.jpg
/wiki/Yerba_mate
/wiki/Coffee_bean
```


Crawling

Single-Domain Crawling

```
#lecture1_example8.py (continued)

>>> for link in soup.find("div",
{"id":"bodyContent"}).findAll("a",
href=re.compile("^(/wiki/)((?!:).)*$")):
    if "href" in link.attrs:
        print(link.attrs["href"])
```

Crawling

Single-Domain Crawling

/wiki/Commodity_(album)
/wiki/Business_administration
/wiki/Management
/wiki/Business
/wiki/Accounting
/wiki/Management_accounting
/wiki/Financial_accounting
/wiki/Financial_audit
/wiki/Legal_personality
/wiki/Cooperative
/wiki/Corporation
/wiki/Limited_liability_company

Crawling

A Random Crawl

```
#lecture1_example9.py

$user python
>>> from urllib2 import urlopen
>>> from bs4 import BeautifulSoup
>>> import datetime
>>> import random
>>> import time
>>> import re
>>> html = urlopen("https://en.wikipedia.org")
>>> soup = BeautifulSoup(html.read())
>>> links = soup.find("div",
{"id": "bodyContent"}).findAll("a",
href=re.compile("^(/wiki/)((?!:).)*$"))
```

A Random Crawl

#lecture1_example9.py (continued)

```
>>> while len(links) > 0:
    link = links[random.randint(0,
    len(links)-1)].attrs["href"]
    print(link)
    html = urlopen("https://en.wikipedia.org/"
+link)
    soup = BeautifulSoup(html.read())
    links = soup.find("div",
    {"id":"bodyContent"}).findAll("a", href =
    re.compile("^(/wiki/)((?!:).)*$"))
    time.sleep(5)
```

Crawling

A Random Crawl

[/wiki/Tropical_Storm_Arthur_\(2008\)](#)
[/wiki/Grangemoor_Park](#)
[/wiki/List_of_Pok%C3%A9mon](#)
[/wiki/President_of_the_United_States](#)
[/wiki/Sea_surface_temperature](#)

Additional Tools

- ▶ Scrapy → web crawling
 - ▶ <https://docs.scrapy.org/en/latest/>
- ▶ Portia → visual scraping
 - ▶ <https://github.com/scrapinghub/portia>
- ▶ Scrapely → automated scraping
 - ▶ <https://pypi.python.org/pypi/scrapely>

Dynamic Content

JavaScript

- ▶ JavaScript most common form of dynamic content
 - ▶ jQuery
 - ▶ Google Analytics
 - ▶ Google Maps
- ▶ Runs in browser, rather than on web server
- ▶ Browser reads and executes script
- ▶ Does not require user to reload page
- ▶ Enclosed in `<script> ... </script>` tags

Dynamic Content

JavaScript

- ▶ JavaScript console in Chrome
 - ▶ Windows/Linux: Ctrl + Shift + J
 - ▶ Mac: Cmd + Opt + J
- ▶ Example

```
> function square(x){  
    var y = x**2;  
    return y  
}  
⏪ undefined  
> y = square(10)  
⏪ 100
```


Dynamic Content

JavaScript

The screenshot shows the Rome2rio website interface. At the top, the logo "Rome2rio" is on the left, and navigation links "Hotels", "Cars", and "About" are on the right. Below the logo, there are two input fields: "FROM Beijing, China" and "TO London, England", separated by a double-headed arrow icon. Below the "FROM" field, there are three flight options listed:

Destination	Price Range (kr)	Duration
Fly to London Heathrow	2035 kr - 4945 kr	13 h 28 min
Fly to London City	2745 kr - 6580 kr	15 h 46 min
Fly to London Gatwick, train	2115 kr - 5835 kr	16 h 15 min

Each flight option includes an icon representing the mode of transport (airplane for flights, train for the train option) and a right-pointing arrow. To the right of the flight options is a map showing the route from Beijing, China to London, England, with labels for Greenland, Iceland, Sweden, Finland, and Norway.

<https://www.rome2rio.com/s/Beijing/London>

Dynamic Content

JavaScript

```
>>> wad -u https://www.rome2rio.com/s/Beijing/London
{
...
    {
        "type": "javascript-frameworks",
        "app": "jQuery",
        "ver": "1.9.1"
    },
...
}
```

Dynamic Content

JavaScript

- ▶ jQuery → dynamic HTML generation
 - ▶ Not executed → missing HTML
- ▶ More dynamic content
 - ▶ Ajax → asynchronous javascript and XML
 - ▶ DHTML → dynamic HTML

Dynamic Content

Selenium

- ▶ Originally developed for website testing
- ▶ Used for browser automation
- ▶ Can execute dynamic elements in browser
- ▶ May be run in headless mode
 - ▶ PhantomJS
 - ▶ Firefox
 - ▶ Chrome

Dynamic Content

Selenium

```
#lecture1_example10.py
```

```
$user python
```

```
>>> from selenium import webdriver
>>> from selenium.webdriver.common.keys import Keys
>>> driver = webdriver.PhantomJS()
>>> driver.set_window_size(1200, 600)
>>> driver.get("https://www.google.com/")
>>> driver.find_element_by_id("a").
send_keys("python")
```

Dynamic Content

Selenium

```
#lecture1_example10.py (continued)
```

```
>>> driver.find_element_by_id("a").  
send_keys(Keys.RETURN)
```

```
>>> print driver.current_url  
https://www.google.com/?q=python
```

```
>>> driver.quit()
```

Dynamic Content

Selenium selectors

Single elements:

- ▶ `find_element_by_id`
- ▶ `find_element_by_name`
- ▶ `find_element_by_xpath`
- ▶ `find_element_by_link_text`

Dynamic Content

Selenium selectors

Single elements:

- ▶ `find_element_by_partial_link_text`
- ▶ `find_element_by_tag_name`
- ▶ `find_element_by_class_name`
- ▶ `find_element_by_css_selector`

Dynamic Content

Selenium selectors

Multiple elements:

- ▶ `find_elements_by_name`
- ▶ `find_elements_by_xpath`
- ▶ `find_elements_by_link_text`

Dynamic Content

Selenium selectors

Multiple elements:

- ▶ `find_elements_by_partial_link_text`
- ▶ `find_elements_by_tag_name`
- ▶ `find_elements_by_class_name`
- ▶ `find_elements_by_css_selector`

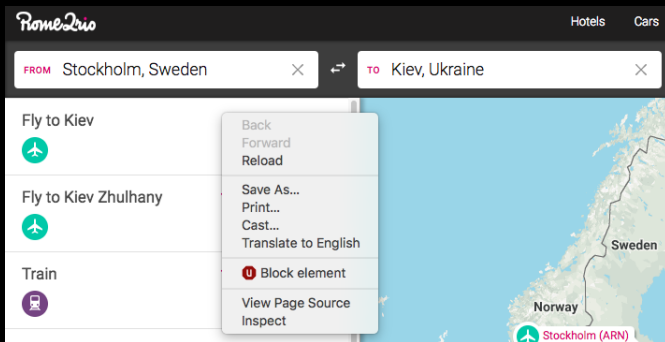
Finding selectors in Chrome

The screenshot displays the Rome2Rio website interface. At the top, the logo 'Rome2Rio' is on the left, and 'Hotels' and 'Cars' links are on the right. Below the logo, there are two input fields: 'FROM Stockholm, Sweden' and 'TO Kiev, Ukraine', each with a close button (X). A double-headed arrow icon is between the fields. Below the input fields, there is a list of travel options, each with an icon, a title, a price range in Swedish Krona (kr), and a duration. To the right of the list is a map of Northern Europe showing Sweden and Norway, with a marker for 'Stockholm (ARN)'.

Mode	Option	Price Range (kr)	Duration
Fly	Fly to Kiev	740 kr - 1960 kr	5 h 20 min
	Fly to Kiev Zhulhany	1021 kr - 3042 kr	7 h 25 min
Train	Train	1805 kr - 3025 kr	37 h 8 min
Car ferry, bus	Car ferry, bus	1136 kr - 1621 kr	39 h 26 min

<https://www.rome2rio.com/s/Stockholm/Kiev>

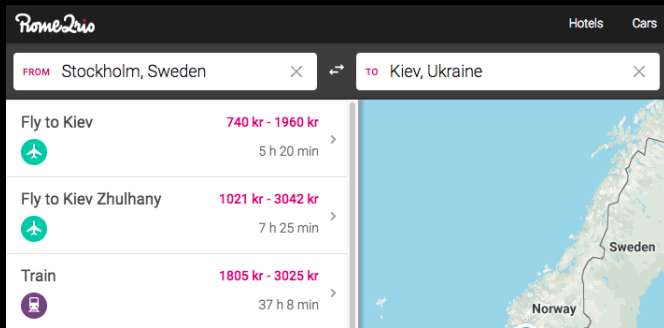
Finding selectors in Chrome



<https://www.rome2rio.com/s/Stockholm/Kiev>

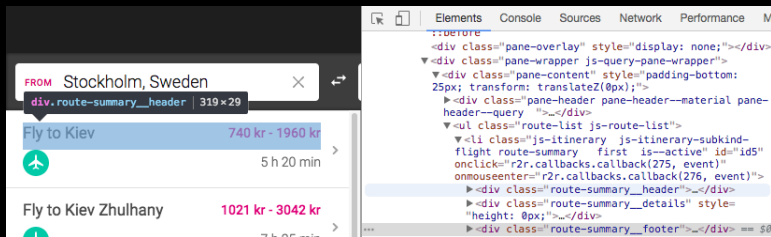
Scraping

Finding selectors in Chrome



<https://www.rome2rio.com/s/Stockholm/Kiev>

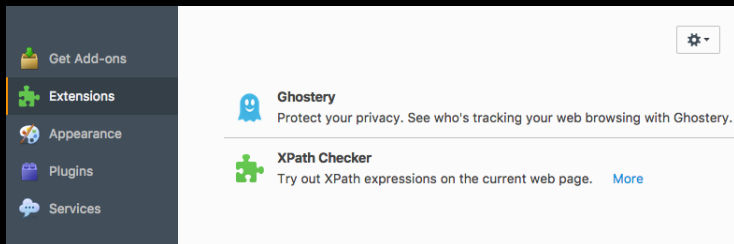
Finding selectors in Chrome



<https://www.rome2rio.com/s/Stockholm/Kiev>

Scraping

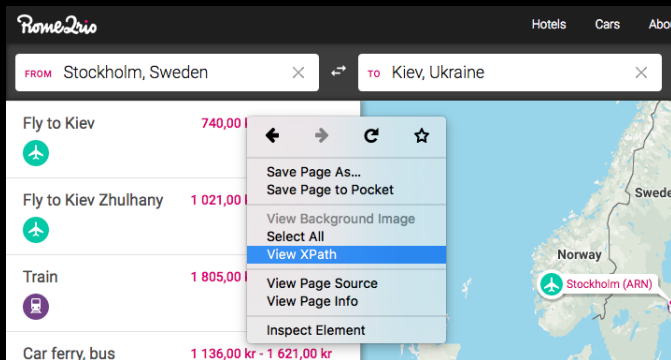
Finding selectors in Firefox



<https://www.rome2rio.com/s/Stockholm/Kiev>

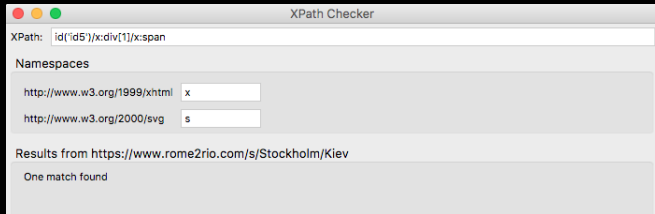
Scraping

Finding selectors in Firefox



<https://www.rome2rio.com/s/Stockholm/Kiev>

Finding selectors in Firefox



<https://www.rome2rio.com/s/Stockholm/Kiev>

Dynamic Content

Xpath selectors

```
#lecture1_example11.py
```

```
$user python
```

```
>>> from selenium import webdriver
>>> driver = webdriver.PhantomJS()
>>> driver.set_window_size(1200, 600)
>>> driver.get("https://www.rome2rio.com/
s/Stockholm/Kiev")
>>> xpath_element =
driver.find_element_by_xpath("//*[@id='id5']/div[1]")
>>> print xpath_element.text
Fly to Kiev 740 kr - 1960 kr
```

Dynamic Content

CSS Selectors

```
#lecture1_example11.py (continued)
```

```
>>> css_element = driver.\nfind_element_by_css_selector("div.route-\nsummary__header")
```

```
>>> print css_element.text\nFly to Kiev 740 kr - 1960 kr
```

```
>>> css_element = driver.\nfind_elements_by_css_selector("div.route-\nsummary__header")
```

```
>>> for element in css_elements:\n    print element.text
```

Dynamic Content

Selenium selectors

Fly to Kiev 740 kr - 1960 kr

Fly to Kiev Zhulhany 1021 kr - 3042 kr

Train 1805 kr - 3025 kr

Car ferry, bus 1136 kr - 1621 kr

Bus 900 kr - 1350 kr

Car ferry, night train 1400 kr - 2430 kr

Dynamic Content

Selenium

- ▶ Documentation
 - ▶ <https://selenium-python.readthedocs.io/>

Logins

Overview

- ▶ HTTP POST requests
- ▶ Check robots.txt
- ▶ Content is often gated behind logins
 - ▶ Forums
 - ▶ Social media
 - ▶ User-generated content

POST request

vBulletin Message

Register at Small Business Forums

You have entered an invalid username or password. Please enter the correct details and try again.
Don't forget that the password is case sensitive. Forgotten your password? [Click here!](#)

You have used 1 out of 5 login attempts. After all 5 have been used, you will be unable to login for 15 minutes.

The administrator may have required you to register before you can view this page.

Log In

User Name:

Password:

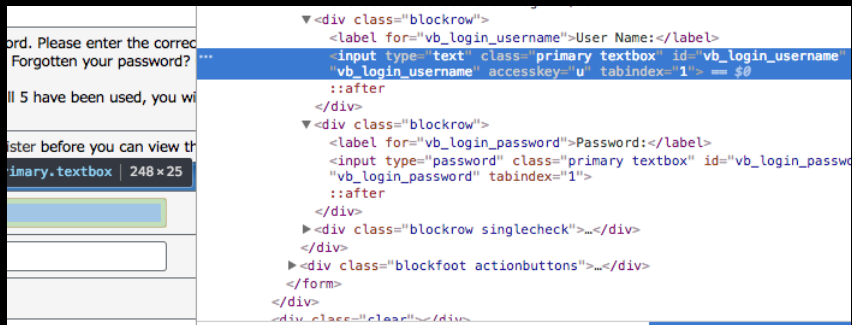
☐ Remember Me?

Log in

Reset Fields

Logins

POST request

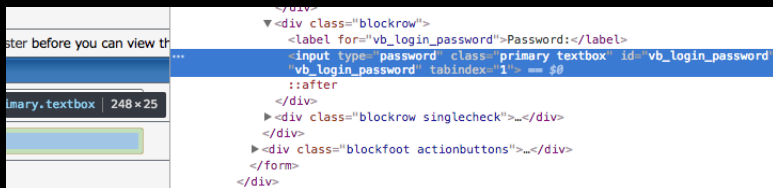


The image shows a web browser window on the left and its HTML source code on the right. The browser window displays a login form with the following text: "ord. Please enter the correc", "Forgotten your password?", "ll 5 have been used, you wi", "ister before you can view th", and "primary.textbox | 248 x 25". The source code on the right is a snippet of HTML, with the following lines highlighted in blue:

```
<div class="blockrow">  
  <label for="vb_login_username">User Name:</label>  
  <input type="text" class="primary textbox" id="vb_login_username"  
    "vb_login_username" accesskey="u" tabindex="1"> == $0  
  ::after  
</div>  
<div class="blockrow">  
  <label for="vb_login_password">Password:</label>  
  <input type="password" class="primary textbox" id="vb_login_passwo  
    "vb_login_password" tabindex="1">  
  ::after  
</div>  
  <div class="blockrow singlecheck">...</div>  
</div>  
<div class="blockfoot actionbuttons">...</div>  
</form>  
</div>  
<div class="clear"></div>
```


Logins

POST request



Logins

POST request

```
#lecture1_example12.py
$user python
>>> import requests
>>> params = {'vb_login_username':
'user@gmail.com', 'vb_login_password': 'mypassword'}
>>> r =
requests.post("https://www.example.com/...",
data=params)
```




Storing Data

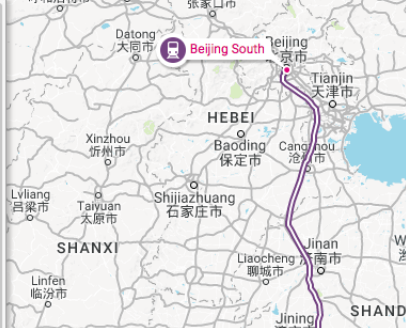
CSV Files

Rome2rio

Hotels Cars About ▾ Help ▾ LOGIN

FROM Beijing, China × ↔ TO Shanghai, China × +

Train China Railways G-Class	1105 kr - 1307 kr	>
	5 h 9 min	
Train China Railways D-Class	600 kr - 950 kr	>
	11 h 37 min	
Drive	850 kr - 1300 kr	>
	12 h 16 min	
Fly Beijing to Shanghai	619 kr - 2041 kr	



The map displays the geographical route from Beijing to Shanghai. A purple line indicates the path, starting at Beijing South and passing through Tianjin and Jinan before reaching Shanghai. The map includes labels for various cities and provinces, such as Beijing, Tianjin, Jinan, Hebei, Shanxi, and Shandong.

Storing Data

CSV files

```
#lecture1_example13.py

$user pip install numpy
$user pip install pandas
$user python
>>> from selenium import webdriver
>>> import numpy as np
>>> import pandas as pd
>>> driver = webdriver.PhantomJS()
>>> driver.set_window_size(1200, 600)
>>> driver.get("https://www.rome2rio.com/s/
Beijing/Shanghai")
```

Storing Data

CSV files

```
#lecture1_example13.py (continued)
```

```
>>> prices = driver.find_elements_by_  
css_selector('span.routesummary__price.js-itinerary-  
price.tip-west')
```

```
>>> modes = driver.find_elements_by_  
css_selector('h2.routesummary__title')
```

```
>>> durations = driver.find_elements_by_  
css_selector('span.routesummary__duration.tip-west')
```

Storing Data

CSV files

```
#lecture1_example13.py (continued)
```

```
>>> prices = [price.text for price in prices]
>>> modes = [mode.text for mode in modes]
>>> durations = [duration.text for duration in
durations]
>>> data = pd.DataFrame(np.vstack([modes,
prices,durations]).T, columns=['mode',
`price`,`duration`])
>>> data.to_csv('../Beijing-Shanghai.csv',
index=None)
>>> print data
```

Storing Data

CSV files

	Mode	Price	Duration
0	Train China Railways G-Class	1105 kr - 1307 kr	5 h 9 min
1	Train China Railways D-Class	600 kr - 950 kr	11 h 37 min
2	Drive	850 kr - 1300 kr	12 h 16 min
3	Fly Beijing to Shanghai Hongqiao	619 kr - 2041 kr	4 h 1 min
4	Fly Beijing Nanyuan Apt to Shanghai Hongqiao	840 kr - 2560 kr	3 h 46 min
5	Fly Beijing Nanyuan Apt to Shanghai Pudong	940 kr - 2490 kr	3 h 50 min
6	Fly Beijing to Shanghai Pudong	819 kr - 3171 kr	4 h 14 min
7	Train to Tianjin, fly to Shanghai Hongqiao	827 kr - 2531 kr	5 h 4 min

Storing Data

Additional Storage Methods

- ▶ JSON

- ▶ <https://docs.python.org/2/library/json.html>

- ▶ Pandas

- ▶ <http://pandas.pydata.org/pandas-docs/version/0.18.0/>

- ▶ MySQL

- ▶ <https://pymysql.readthedocs.io/en/latest/>

Concurrent Downloading

Multithreading and Multiprocessing

- ▶ Scraping and crawling tasks may require data collection from thousands of URLs
- ▶ Concurrent downloading can substantially reduce the time to complete tasks
- ▶ Python has several options
 - ▶ `from multiprocessing import Pool` → multiprocessing
 - ▶ `import multiprocessing.dummy`
- ▶ No need for delays if URLs are distributed across multiple sites

Concurrent Downloading

Multithreading and Multiprocessing



The screenshot shows the official website of the Bureau of Labor Statistics (BLS). The header is red with the BLS logo and the text "UNITED STATES DEPARTMENT OF LABOR" and "BUREAU OF LABOR STATISTICS". Below the header is a navigation bar with links: Home, Subjects, Data Tools, Publications, and Economic Releases. The main content area features the "Monthly Labor Review" logo, which includes the letters "mlr" in a large, stylized font and a blue line graph with an upward arrow. To the right of the logo are links for "HOME" and "ARCHIVE". Below the logo is a section titled "ARCHIVE BY DATE" with a list of publications for the year 2017.

UNITED STATES DEPARTMENT OF LABOR
BUREAU OF LABOR STATISTICS

Home Subjects Data Tools Publications Economic Releases

Monthly Labor Review mlr BLS

HOME ARCHIVE

ARCHIVE BY DATE

2017

- Benchmarking the Current Employment Statistics survey: the past, present, and future 10/30/2017
- Do Consumer Price Index headlines matter? They just might 10/30/2017
- Book review interest? 10/30/2017
- Projections overview and highlights, 2016–26 10/24/2017
- The cost effectiveness of job-creating economic policies 10/02/2017


Concurrent Downloading


Multithreading and Multiprocessing

ARCHIVE BY DATE		
2017	2016	2015
2014	2013	2012
2011	2010	2009
2008	2007	2006
2005	2004	2003
2002	2001	2000
1999	1998	1997
1996	1995	1994
1993	1992	1991
1990	1989	1988
1987	1986	1985

Concurrent Downloading

Multithreading and Multiprocessing

 Secure | <https://www.bls.gov/opub/mlr/2017/>

 Secure | <https://www.bls.gov/opub/mlr/2016/>

Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py
```

```
$user python
```

```
>>> from urllib2 import urlopen  
>>> from bs4 import BeautifulSoup  
>>> from multiprocessing import Pool  
>>> import multiprocessing.dummy
```

Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py (continued)
```

```
>>> threads = 5
```

```
>>> processes = 5
```

Concurrent Downloading

Multithreading and Multiprocessing

#lecture1_example14.py (continued)

```
>>> date_range = range(1980, 1985)
```

```
>>> thread_pl = multiprocessing.dummy.Pool(thread)
```

```
>>> process_pl = multiprocessing.Pool(process)
```

Concurrent Downloading

Multithreading and Multiprocessing

#lecture1_example14.py (continued)

```
>>> def url_opener(year):  
    html = urlopen("https://www.bls.gov  
    /opub/mlr/"+str(year)).read()  
    soup = BeautifulSoup(html)  
    links = soup.findAll("a")  
    links = [link for link in links if  
    link['href'].find('bls.gov/opub/mlr')>-1]  
    return links
```


Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py (continued)
```

```
>>> thread_links = thread_pl.map(url_opener,  
date_range)
```

```
>>> process_links = process_pl.map(url_opener,  
date_range)
```

```
>>> thread_links = sum(thread_links, [])
```

```
>>> process_links = sum(process_links, [])
```

Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py (continued)
```

```
>>> print len(thread_links)
```

```
360
```

```
>>> print len(process_links)
```

```
360
```

Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py (continued)
```

```
>>> thread_links[:5]
```

Concurrent Downloading

Multithreading and Multiprocessing

```
[<a href="https://www.bls.gov/opub/mlr/1981/12/art4full.pdf">The  
unemployment insurance system: its financial structure</a>,  
<a  
href="https://www.bls.gov/opub/mlr/1981/12/art1full.pdf">Unemployment,  
labor force trends, and layoff practices in 10 countries</a>,  
<a href="https://www.bls.gov/opub/mlr/1981/12/art3full.pdf">Bargaining  
calendar will be heavy in 1982</a>,  
<a  
href="https://www.bls.gov/opub/mlr/1981/12/art2full.pdf">International  
comparisons of trends in productivity and labor costs</a>,  
<a href="https://www.bls.gov/opub/mlr/1981/12/art5full.pdf">Employment  
created by construction expenditures</a>]
```

Concurrent Downloading

Multithreading and Multiprocessing

```
#lecture1_example14.py (continued)
```

```
>>> process_links[:5]
```

Concurrent Downloading

Multithreading and Multiprocessing

```
[<a href="https://www.bls.gov/opub/mlr/1981/12/art4full.pdf">The  
unemployment insurance system: its financial structure</a>,  
<a  
href="https://www.bls.gov/opub/mlr/1981/12/art1full.pdf">Unemployment,  
labor force trends, and layoff practices in 10 countries</a>,  
<a href="https://www.bls.gov/opub/mlr/1981/12/art3full.pdf">Bargaining  
calendar will be heavy in 1982</a>,  
<a  
href="https://www.bls.gov/opub/mlr/1981/12/art2full.pdf">International  
comparisons of trends in productivity and labor costs</a>,  
<a href="https://www.bls.gov/opub/mlr/1981/12/art5full.pdf">Employment  
created by construction expenditures</a>]
```

Summary

- ▶ Introduction
 - ▶ How a browser works
 - ▶ Scraping ethics
 - ▶ Estimating a website's size
 - ▶ Identifying technologies used

Summary

- ▶ Scraping
 - ▶ BeautifulSoup
 - ▶ Lxml
 - ▶ Lambda expressions
 - ▶ Regular expressions
 - ▶ XPath and CSS selectors

Summary

- ▶ Crawling
 - ▶ Single-Domain Crawling
 - ▶ Random Crawling
 - ▶ Scrapy
 - ▶ Portia
 - ▶ Scrapely

Summary

- ▶ Dynamic Content
 - ▶ JavaScript
 - ▶ Ajax
 - ▶ Selenium
 - ▶ Selectors

Summary

- ▶ Logins
 - ▶ Sending POST requests

Summary

- ▶ Storing Data
 - ▶ CSV
 - ▶ JSON
 - ▶ Pandas
 - ▶ MySQL

Summary

- ▶ Concurrent Downloading
 - ▶ Multithreading
 - ▶ Multiprocessing