Python Workshop:
Gathering data from the web

Ties de Kok

Tilburg University

Agenda

What are we going to discuss today?

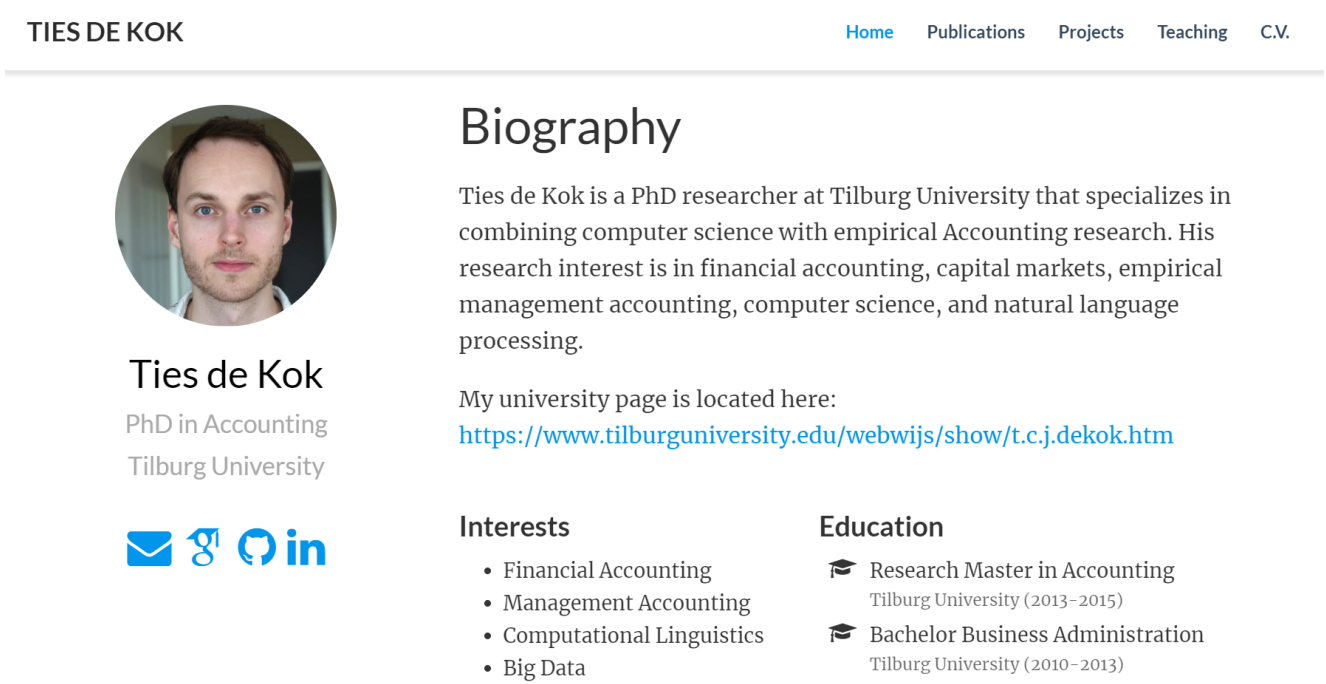
1. How does the web work?
2. Terminology
3. Ethics
4. Tools
5. Specific topics:
    - Interacting with an API
    - Web scrape a page
    - Reverse-engineer HTTP requests
    - Dealing with Javascript elements

Agenda

The web

How does the web work?

What we see:



Agenda

The web

How does the web work?

What computers see:

```
<h1 id="biography">Biography</h1>

<p></p>

<p>Ties de Kok is a PhD researcher at Tilburg University that specializes in combining computer science with empirical Accounting research. His
research interest is in financial accounting, capital markets, empirical management accounting, computer science, and natural language processing.
</p>

<p>My university page is located here: <a href="https://www.tilburguniversity.edu/webwijs/show/t.c.j.dekok.htm"
target="_blank">https://www.tilburguniversity.edu/webwijs/show/t.c.j.dekok.htm</a></p>


    <div class="row">


      <div class="col-sm-5">
        <h3>Interests</h3>
        <ul class="ul-interests">

          <li>Financial Accounting</li>

          <li>Management Accounting</li>

          <li>Computational Linguistics</li>

          <li>Big Data</li>

        </ul>
      </div>



      <div class="col-sm-7">
        <h3>Education</h3>
        <ul class="ul-edu fa-ul">

          <li>
            <i class="fa-li fa fa-graduation-cap"></i>
            <div class="description">
              <p class="course">Research Master in Accounting</p>
              <p class="institution">Tilburg University (2013-2015)</p>
            </div>
          </li>

          <li>
            <i class="fa-li fa fa-graduation-cap"></i>
            <div class="description">
              <p class="course">Bachelor Business Administration</p>
              <p class="institution">Tilburg University (2010-2013)</p>
            </div>
          </li>
```

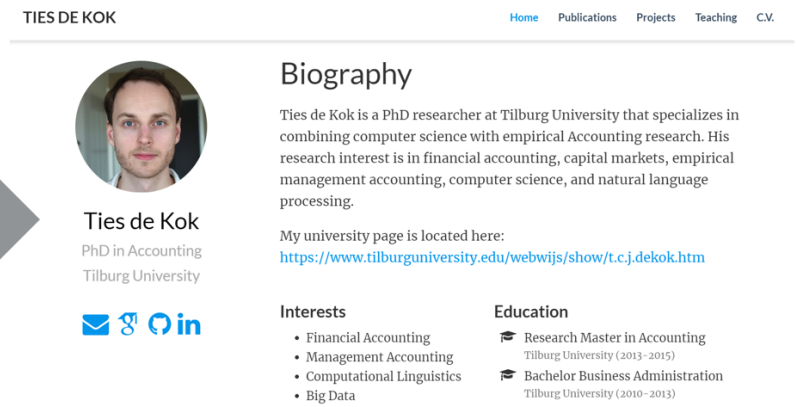
Agenda

The web

How does the web work?

Web browsers are awesome!

```
<h1 id="biography">Biography</h1>

<p></p>

<p>Ties de Kok is a PhD researcher at Tilburg University that specializes in c
research interest is in financial accounting, capital markets, empirical manag
</p>

<p>My university page is located here: <a href="https://www.tilburguniversity.
target="_blank">https://www.tilburguniversity.edu/webwijs/show/t.c.j.dekok.htm

    <div class="row">

      <div class="col-sm-5">
        <h3>Interests</h3>
        <ul class="ul-interests">

          <li>Financial Accounting</li>

          <li>Management Accounting</li>

          <li>Computational Linguistics</li>

          <li>Big Data</li>

        </ul>
      </div>


      <div class="col-sm-7">
        <h3>Education</h3>
        <ul class="ul-edu fa-ul">

          <li>
            <i class="fa-li fa fa-graduation-cap"></i>
            <div class="description">
              <p class="course">Research Master in Accounting</p>
              <p class="institution">Tilburg University (2013-2015)</p>
            </div>
          </li>

          <li>
            <i class="fa-li fa fa-graduation-cap"></i>
            <div class="description">
              <p class="course">Bachelor Business Administration</p>
              <p class="institution">Tilburg University (2010-2013)</p>
            </div>
          </li>
```



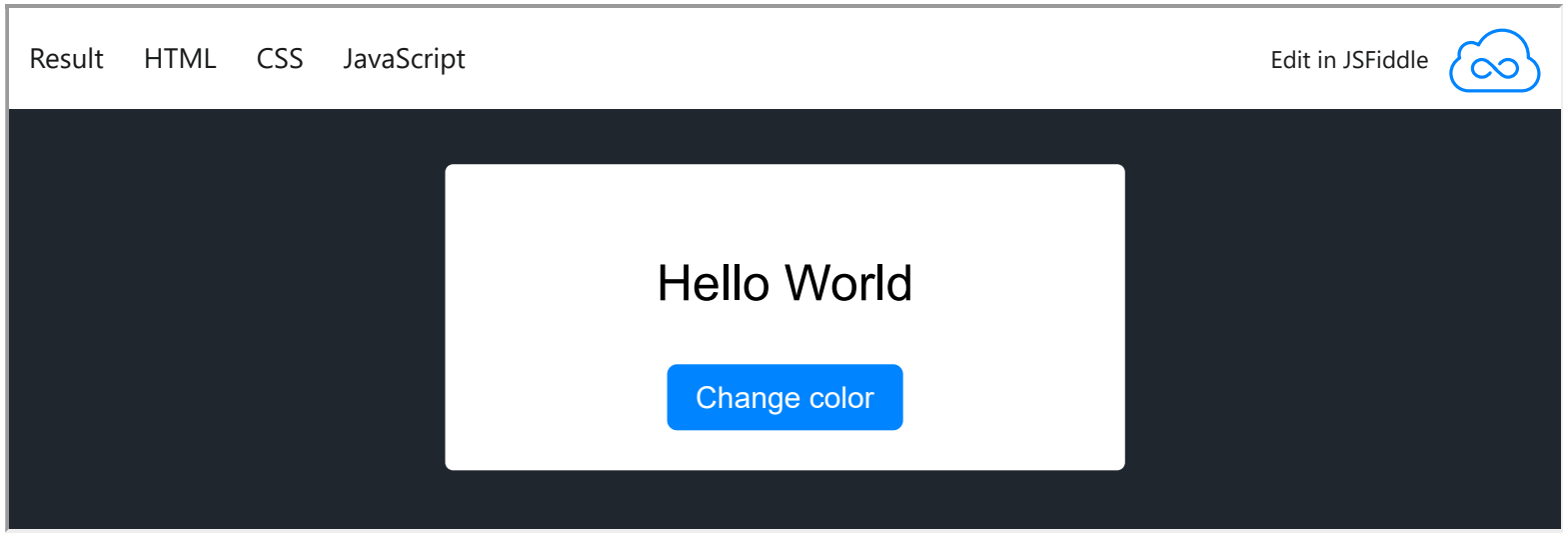
Agenda

The web

Terminology

Web-development terms

- HTML
  - The structure + static content
- CSS
  - Determines the way things look
- Javascript
  - Code to make stuff happen



Agenda

The web

Terminology

Gathering data from the web

- **Web scraping**
  - Extract data from a webpage
- Web crawling
  - Automatically move across webpages

Agenda

The web

Terminology

Gathering data from the web

- **Web scraping**
  - Extract data from a webpage
- Web crawling
  - Automatically move across webpages
- API
  - "Webpages" for computers, not for humans.

Let's say we want to know the current price of Bitcoin:

Humans: Google "Bitcoin price", open website, click on some menu.

Computers: ??

Agenda

The web

Terminology

Computers: https://api.coindesk.com/v1/bpi/currentprice.json

Agenda

The web

Terminology

Ethics

Ethics of gathering data from the web

General rules:

1. Always use the API if there is an API available
2. Never allow your scraper to turn into an unintentional DoS attack
3. Read the ToS, explicitly prohibited? Probably don't do it

There is a lot of conflicting information and advice...

My golden principles:

1. Make sure nothing is harmed by gathering the data
2. Never distribute gathered data without permission

Agenda

The web

Terminology

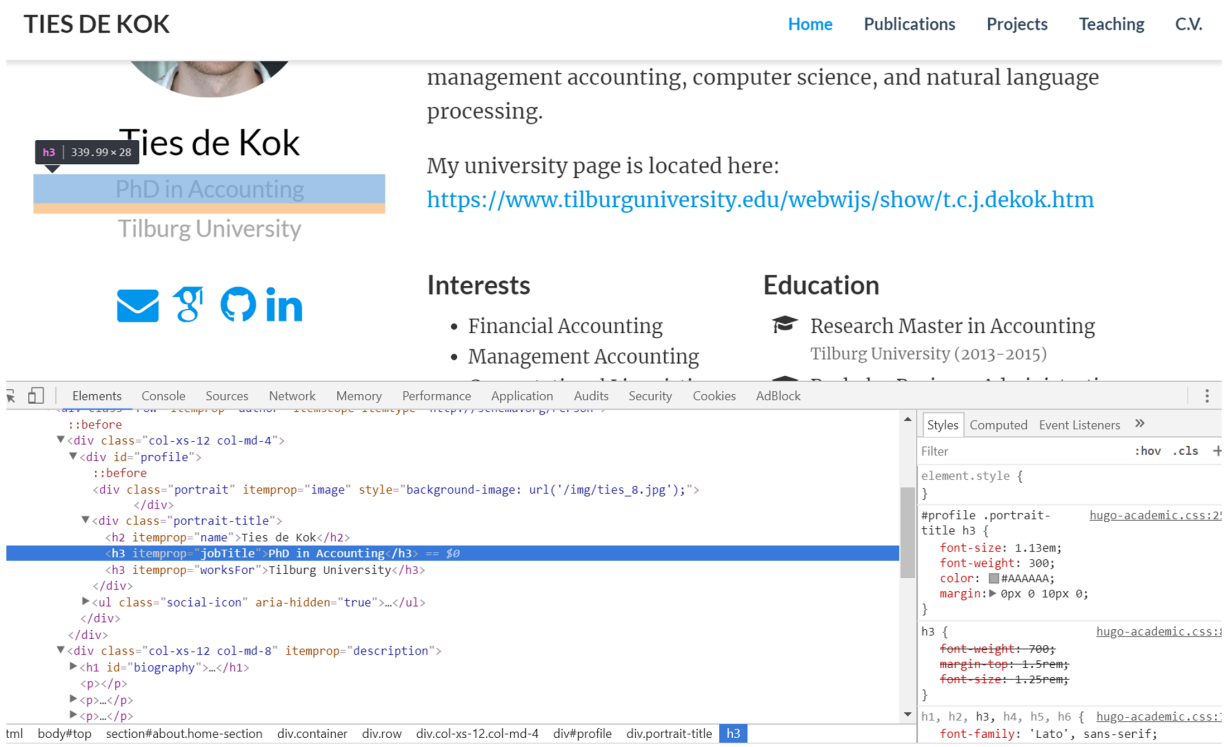
Ethics

Tools

Basic tools

- Chrome + Chrome DevTools

- Open DevTools: `CTRL` + `SHIFT` + `J`



Agenda

The web

Terminology

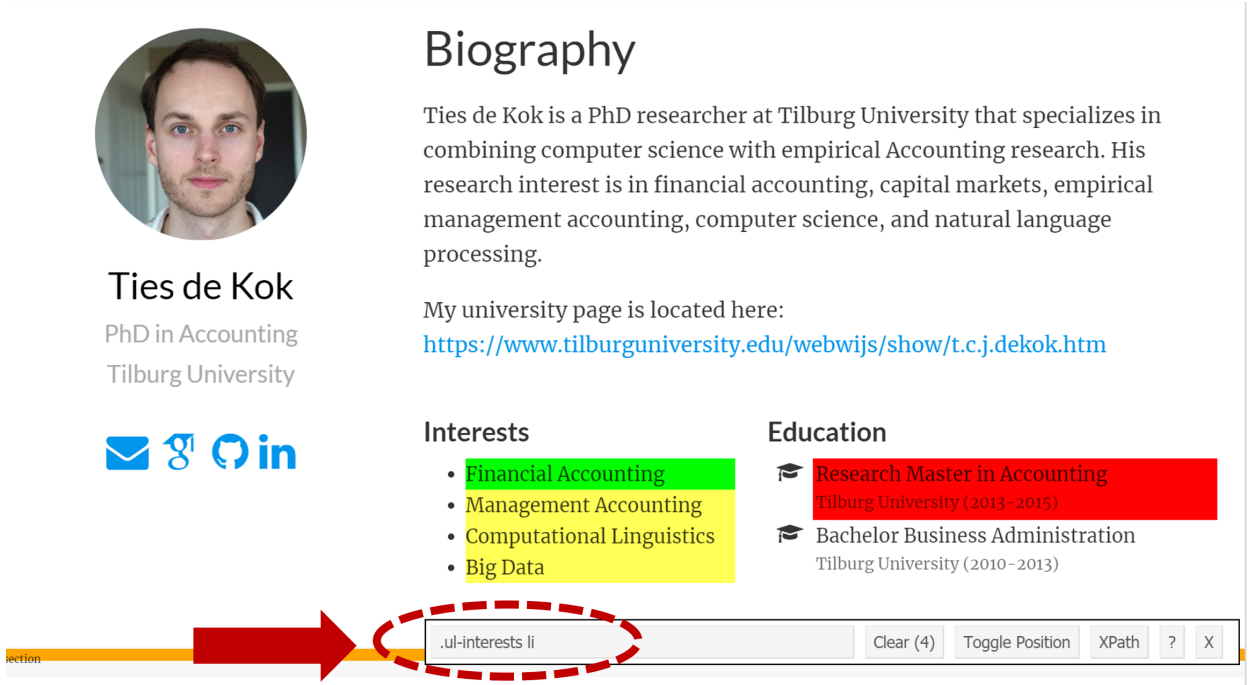
Ethics

Tools

Basic tools

- `SelectorGadget` Chrome extension

▶ Downoad + Install `SelectorGadget`



Agenda

The web

Terminology

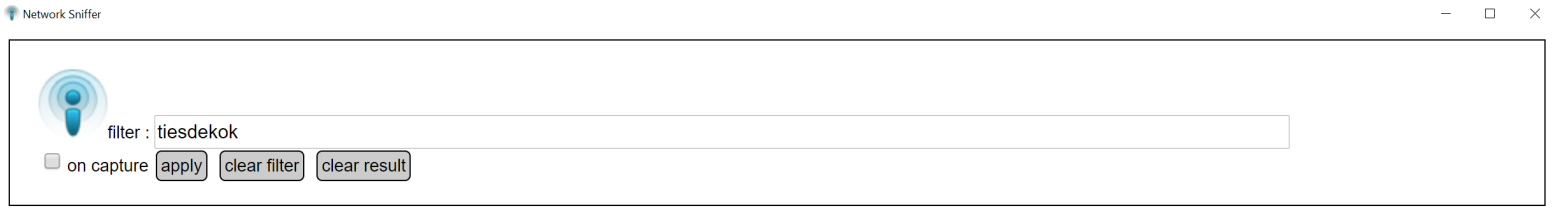
Ethics

Tools

Basic tools

- `Network Sniffer` Chrome extension

▶ Downoad + Install `Network Sniffer`



requestId	statusCode	ip	url	type	timeStamp	tabId	fromCache	method
30797	200	185.199.110.153	https://www.tiesdekok.com/img/icon.png	image	1526261694081.187	338	false	GET
30794	200	74.125.135.156	https://stats.g.doubleclick.net/r/collect?v=1&aip=1&t=dc&_r=3&tid=UA-9085…	image	1526261693842.674	338	false	GET
30791	200	185.199.110.153	https://www.tiesdekok.com/fonts/academicons.ttf?v=1.5.0	font	1526261693608.339	338	false	GET
30790	200	185.199.110.153	https://www.tiesdekok.com/fonts/fontawesome-webfont.woff2?v=4.7.0	font	1526261693580.492	338	false	GET
30787	200	185.199.110.153	https://www.tiesdekok.com/img/ties_8.jpg	image	1526261693957.175	338	false	GET
30785	200	185.199.111.153	https://www.tiesdekok.com/js/hugo-academic.js	script	1526261693166.877	338	true	GET
30783	200	185.199.111.153	https://www.tiesdekok.com/js/isotope.pkgd.min.js	script	1526261693167.4111	338	true	GET
30782	200	185.199.111.153	https://www.tiesdekok.com/js/bootstrap.min.js	script	1526261693162.49	338	true	GET
30781	200	185.199.111.153	https://www.tiesdekok.com/js/jquery-1.12.3.min.js	script	1526261693339.983	338	true	GET
30778	200	185.199.111.153	https://www.tiesdekok.com/css/hugo-academic.css	stylesheet	1526261693146.528	338	true	GET
30776	200	185.199.111.153	https://www.tiesdekok.com/css/academicons.min.css	stylesheet	1526261693146.997	338	true	GET
30775	200	185.199.111.153	https://www.tiesdekok.com/css/font-awesome.min.css	stylesheet	1526261693145.237	338	true	GET
30774	200	185.199.111.153	https://www.tiesdekok.com/css/bootstrap.min.css	stylesheet	1526261693149.069	338	true	GET
30773	200	185.199.111.153	https://www.tiesdekok.com/css/highlight.min.css	stylesheet	1526261693142.9329	338	true	GET
30771	200	185.199.111.153	https://www.tiesdekok.com/	main_frame	1526261693044.897	338	true	GET

Agenda

The web

Terminology

Ethics

Tools

Python packages

To retrieve a webpage:

1) `Requests`

To process a webpage:

2) `LXML` and the higher-level wrapper `Requests-HTML`

To deal with JavaScript heavy-webpages:

3) `Selenium with Python`

(Note: Selenium is not covered in this workshop)

Agenda

The web

Terminology

Ethics

Tools

Topics

Specific topics:

- Interacting with an API
- Web scrape a page
- Reverse-engineer HTTP requests
- Dealing with Javascript elements

API Requests

Basics of using an API

An API, in a simplified sense, has two characteristics:

1. A request is made using a URL + parameters
2. A response is returned in a machine-readable format.

The machine-readable formats are usually either:

- JSON
- XML
- (sometimes plain text)

API Requests

Making API requests using `requests`

API Endpoint

API Parameters

JSON →

In [1]:
```
import requests
```

In [2]:
```
API_endpoint = 'https://api.coindesk.com/v1/bpi/historical/close.json'
payload = {'start' : '2013-09-01', 'end' : '2013-09-05'}
```

In [3]:
```
res = requests.get(API_endpoint, params=payload)
```

In [4]:
```
print(res.url)
```

```
https://api.coindesk.com/v1/bpi/historical/close.json?start=2013-09-01&end=2013-09-05
```

In [7]:
```
bitcoin_2013 = res.json()
bitcoin_2013['bpi']
```

Out[7]:
```
{'2013-09-01': 128.2597,
 '2013-09-02': 127.3648,
 '2013-09-03': 127.5915,
 '2013-09-04': 120.5738,
 '2013-09-05': 120.5333}
```

Basics of web scraping

Scraping a page consists of 4 steps:

1. Construct / determine the URL
2. Retrieve the webpage data (usually HTML)
3. Parse the HTML
4. Extract information using HTML structure

Basics of web scraping

Scraping a page consists of 4 steps:

1. Construct / determine the URL
2. Retrieve the webpage data (usually HTML)
3. Parse the HTML
4. Extract information using HTML structure

Step 1 and Step 2:

- Mostly the same compared to interacting with an API.

Step 3 and Step 4:

- Different, a HTML page is meant for humans!

API
Requests

Web
Scraping

Retrieve webpage data (step 1 & 2):

In [8]:

```
import requests
url = 'https://www.tiesdekok.com/'
res = requests.get(url)
```

API Requests

Web Scraping

Retrieve webpage data (step 1 & 2):

In [8]:

```
import requests
url = 'https://www.tiesdekok.com/'
res = requests.get(url)
```

Retrieve data (step 3 & 4):

In [10]:

```
res.text
```

Out[10]:

```
'<!DOCTYPE html>\n<html lang="en-us">\n<head>\n\n  <meta charset="utf-8">\n  <meta name="viewport
ntent="IE=edge">\n  <meta name="theme" content="hugo-academic">\n  <meta name="generator" content
n" content="PhD in Accounting">\n\n  <link rel="stylesheet" href="/css/highlight.min.css">\n  <li
ont-awesome.min.css">\n  <link rel="stylesheet" href="/css/academicons.min.css">\n  <link rel="st
\n  <link rel="stylesheet" href="/css/hugo-academic.css">\n  \n\n  <link rel="alternate" href="ht
k rel="feed" href="http://www.tiesdekok.com/index.xml" type="application/rss+xml" title="Ties de
uch-icon" type="image/png" href="/img/apple-touch-icon.png">\n\n  <link rel="canonical" href="htt
data-spy="scroll" data-target="#navbar-main" data-offset="71">\n\n<nav class="navbar navbar-defau
="navbar-header">\n      <button type="button" class="navbar-toggle collapsed" data-toggle="colla
class="sr-only">Toggle navigation</span>\n        <span class="icon-bar"></span>\n        <span c
a class="navbar-brand" href="/">Ties de Kok</a>\n    </div>\n\n    \n    <div class="collapse nav
     \n\n        <li class="nav-item">\n          <a href="/#about" data-target="#about" target="
       \n        \n\n        \n\n        <li class="nav-item">\n          <a href="/#publications
ons</span>\n          </a>\n        </li>\n\n        \n        \n\n        \n\n        <li class=
         \n            <span>Projects</span>\n          </a>\n        </li>\n\n        \n
="#teaching" target="_blank">\n            \n            <span>Teaching</span>\n          </a>\n
  <a href="/Vitae.pdf" data-target="http://www.tiesdekok.com/Vitae.pdf" target="_blank">\n
\n      </ul>\n\n    </div>\n  </div>\n</nav>\n\n\n\n<span id="homepage" style="display: none"></
er">\n      \n\n\n\n<div class="row" itemprop="author" itemscope itemtype="http://schema.org/Pers
ass="portrait" itemprop="image"\n          style="background-image: url(\'/img/ties_8.jpg\');">\
Ties de Kok</h2>\n        <h3 itemprop="jobTitle">PhD in Accounting</h3>\n        <h3 itemprop="w
en="true">\n        \n        \n        <li>\n          <a href="mailto:t.c.j.dekok@uvt.nl" targe
</li>\n        \n        \n        <li>\n          <a href="https://scholar.google.co.uk/citation
n"></i>\n          </a>\n        </li>\n        \n        \n        <li>\n          <a href="//gi
```

???

API
Requests

Web
Scraping

Parse HTML (step 3):

HTML has a clear structure, but we need to parse (i.e. interpret) it first!

The `lxml.html` library (or `requests-html` wrapper) make this very easy.

Using `LXML.HTML`

In [17]:

```
import lxml.html
import requests

res = requests.get('https://www.tiesdekok.com')
tree = lxml.html.fromstring(res.text)
```

Using `Requests-HTML`

In [15]:

```
from requests_html import HTMLSession
session = HTMLSession()
res = session.get('https://www.tiesdekok.com')
```

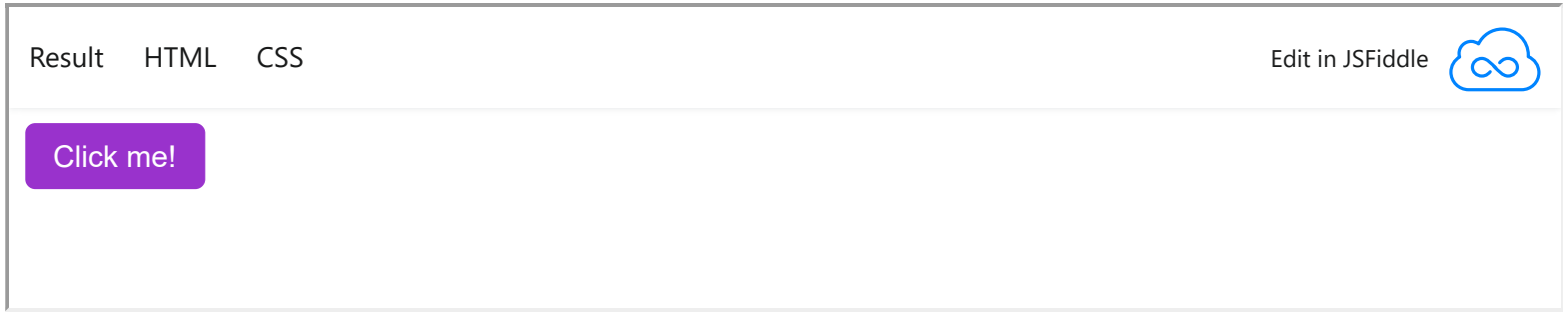
Note: as the names implies `requests-html` combines `requests` with `lxml.html`

We can now use the HTML structure to extract information!

API
Requests

Web
Scraping

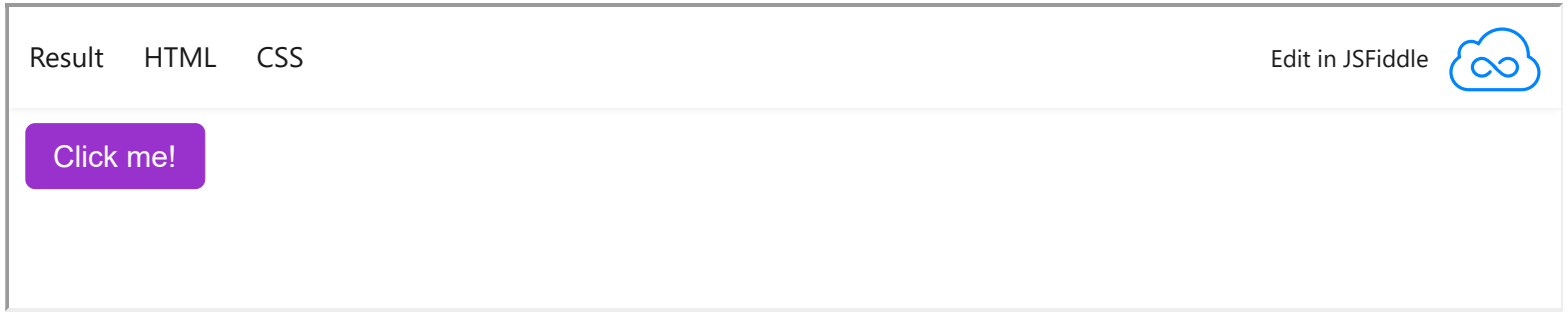
CSS Selectors (step 4, extract information)



API
Requests

Web
Scraping

CSS Selectors (step 4, extract information)



CSS Selector make it very easy to select elements to extract!

In [27]:

```
from requests_html import HTMLSession
session = HTMLSession()
res = session.get('https://www.tiesdekok.com')

for element in res.html.find('.ul-interests > li'):
    print(element.text)
```

```
Financial Accounting
Management Accounting
Computational Linguistics
Big Data
```

This is the CSS selector

Syntax of a css selector

Most frequent options:

1. Use a dot to select based on **class**: `.classname`
2. Use a hash to select based on **id**: `#idname`
3. Selected directly on the **element**: `p`, `span`, `h1`

You can chain multiple conditions together using: `>`, `+`, and `~`.

Example: get `<p>` elements with the `title` class and `<div>` parent:

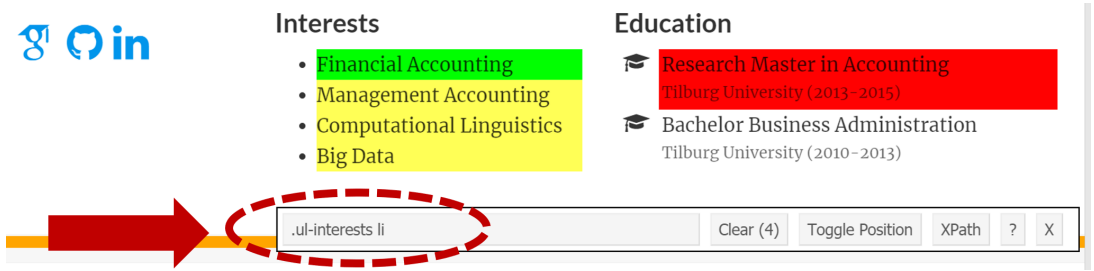
- CSS Selector: `div > p.title`

For a full overview I recommend checking this page:
https://www.w3schools.com/cssref/css_selectors.asp

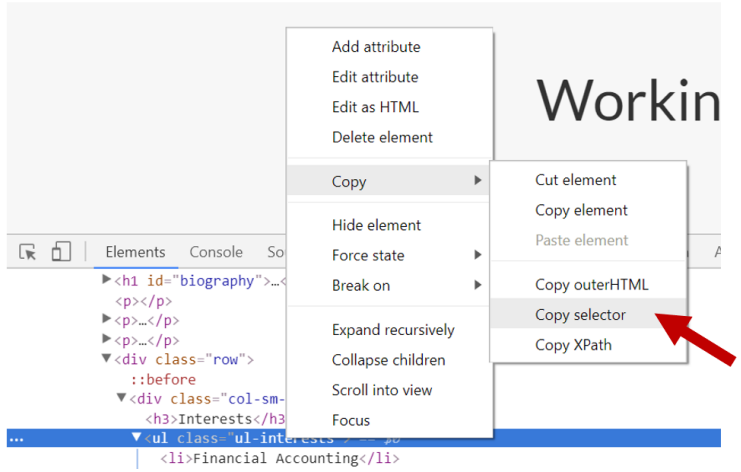
API Requests

Web Scraping

`SelectorGadget` Chrome extension:



Chrome DevTools:



API
Requests

Web
Scraping

Recap: web-scraping a page

Step 1: determine the URL of the page you need

> URL = https://www.tiesdekok.com

Step 2 and Step 3: download and parse the HTML of the webpage

```
from requests_html import HTMLSession
session = HTMLSession()
res = session.get('https://www.tiesdekok.com')
```

Note: the `Requests-HTML` does the HTML parsing automatically for you.

Step 4: use CSS Selectors to extract information

```
for element in res.html.find('.ul-interests > li'):
    print(element.text)
```

```
Financial Accounting
Management Accounting
Computational Linguistics
Big Data
```

API
Requests

Web
Scraping

HTTP
Requests

HTTP Requests

Modern webpages often "load" data to the page using HTTP Requests.

Tip: reverse-engineer the APIs that are used and mimic them!

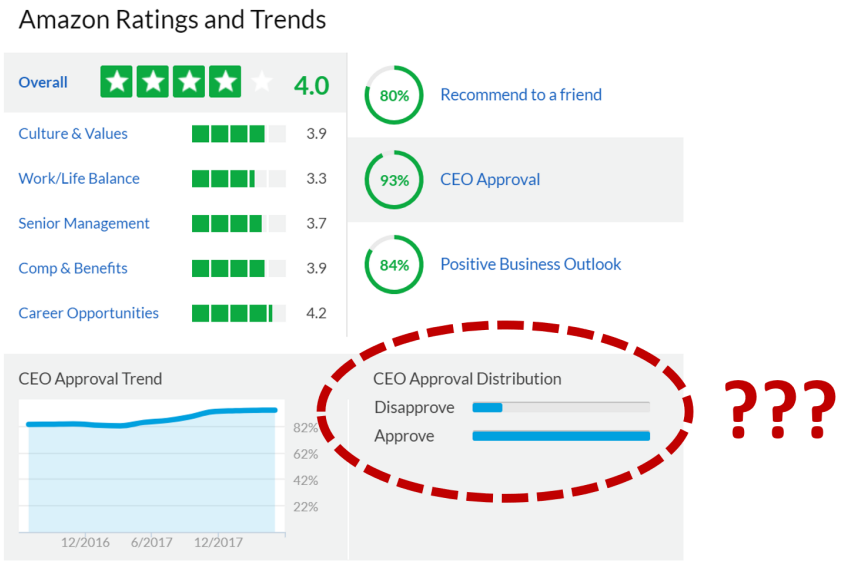
HTTP Requests

Modern webpages often "load" data to the page using HTTP Requests.

Tip: reverse-engineer the APIs that are used and mimic them!

Example:

Let's say we want to get data on the approval rating for Jeff Bezos:

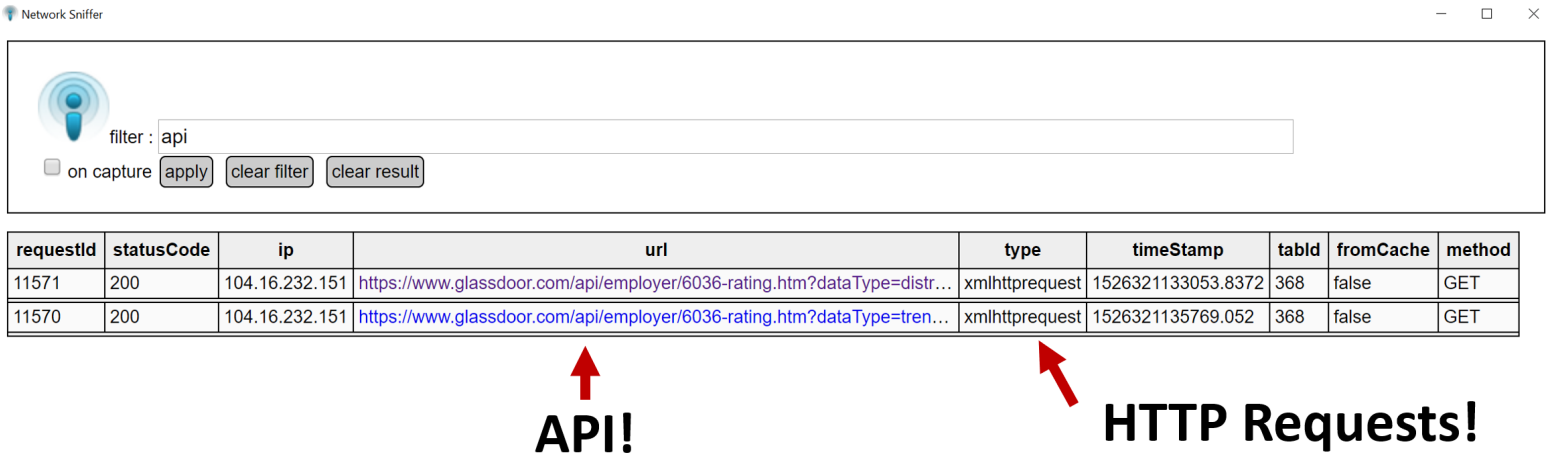


API Requests

Web Scraping

HTTP Requests

What do we see in the `Network Sniffer` Chrome extension?



requestId	statusCode	ip	url	type	timeStamp	tabId	fromCache	method
11571	200	104.16.232.151	https://www.glassdoor.com/api/employer/6036-rating.htm?dataType=distr…	xmlhttprequest	1526321133053.8372	368	false	GET
11570	200	104.16.232.151	https://www.glassdoor.com/api/employer/6036-rating.htm?dataType=tren…	xmlhttprequest	1526321135769.052	368	false	GET

API!

HTTP Requests!

API Requests

Web Scraping

HTTP Requests

What do we see in the `Network Sniffer` Chrome extension?

requestId	statusCode	ip	url	type	timeStamp	tabId	fromCache	method
11571	200	104.16.232.151	https://www.glassdoor.com/api/employer/6036-rating.htm?dataType=distr…	xmlhttprequest	1526321133053.8372	368	false	GET
11570	200	104.16.232.151	https://www.glassdoor.com/api/employer/6036-rating.htm?dataType=tren…	xmlhttprequest	1526321135769.052	368	false	GET

API!

HTTP Requests!

API Requests

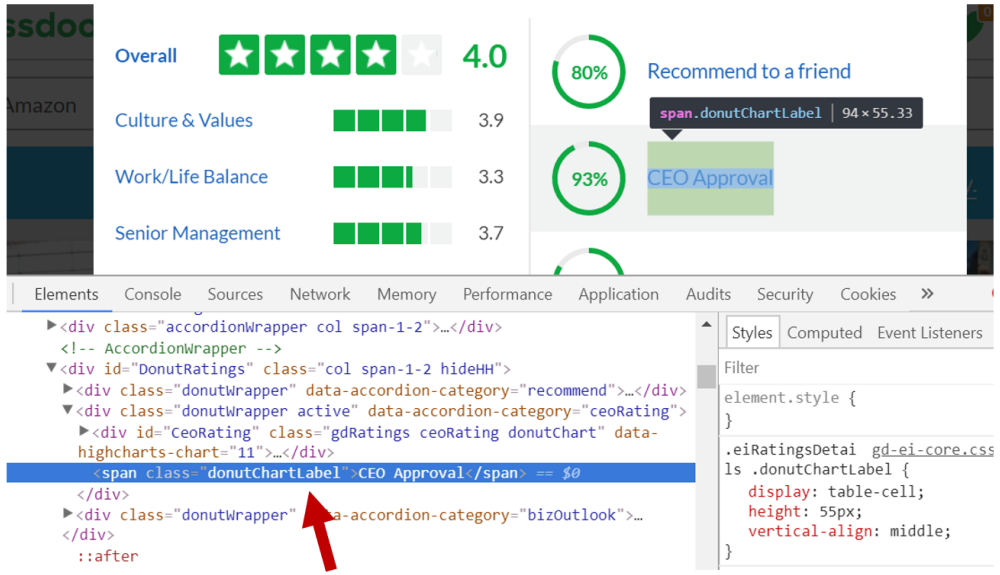
Web Scraping

HTTP Requests

JavaScript Pages

JavaScript heavy webpages

Some webpages rely heavily on JavaScript to load in data-elements:



We can see it

But Python cannot!?

```
from requests_html import HTMLSession
session = HTMLSession()
res = session.get('https://www.glassdoor.com/Re
res.html.find('.donutChartLabel')
```

```
Out[5]: []
```

API Requests

Web Scraping

HTTP Requests

JavaScript Pages

JavaScript heavy webpages

Can we still scrape them?

Sure, but with a different approach:

Option 1: try to reverse-engineer the HTTP Requests

- As demonstrated on previous slide

Option 2: use browser automation tools

- Two primary tools:

1. Use a headless browser (`requests-html` can do this)
2. Use `Selenium` with Chrome bindings

API Requests

Web Scraping

HTTP Requests

JavaScript Pages

Option 2: use browser automation tools

- Use a headless browser (`requests-html` can do this)

Webpage opened by a headless (i.e. hidden) browser

```
In [7]:
from requests_html import HTMLSession
session = HTMLSession()
res = session.get('https://www.glassdoor.com/Reviews/Ama
res.html.render()
res.html.find('.donutChartLabel')

Out[7]: [<Element 'span' class=('donutChartLabel',)>,
         <Element 'span' class=('donutChartLabel',)>,
         <Element 'span' class=('donutChartLabel',)>,
         <Element 'span' class=('donutChartLabel',)>,
         <Element 'span' class=('donutChartLabel',)>,
         <Element 'span' class=('donutChartLabel',)>]
```

Now Python can see it!

Note: first time you run `html.render()` it will download some dependencies.

API Requests

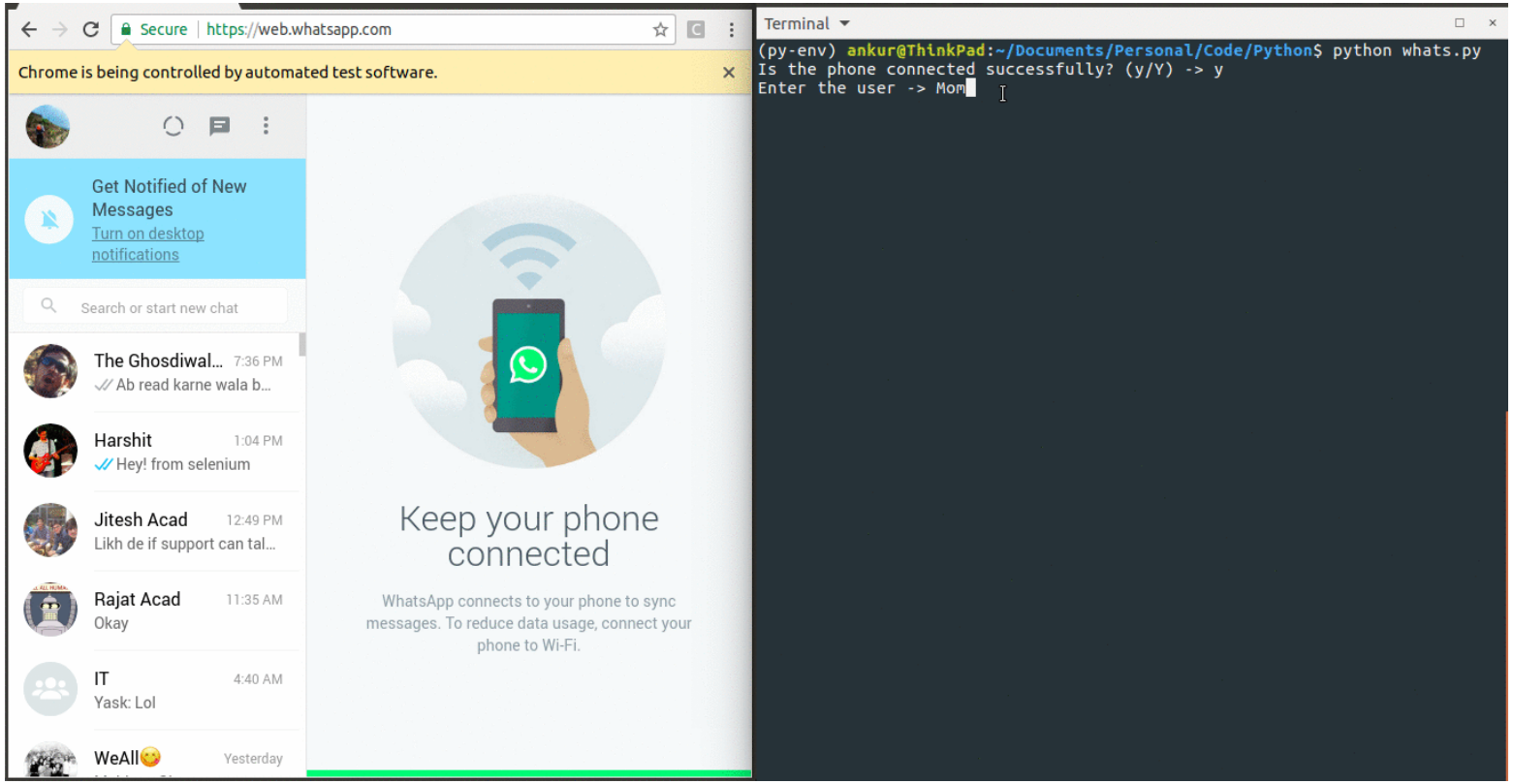
Web Scraping

HTTP Requests

JavaScript Pages

Option 2: use browser automation tools

- Use `Selenium` with Chrome bindings



GIF courtesy of the PyWhatsapp GitHub page

Closing remarks

Questions?

Closing remarks

Setup + Get Started

Setup:

1. Make sure you have Chrome installed
2. Install `SelectorGadget` extension
3. Install `networksniffer` extension

Get Started:

Goal: retrieve my name, job title, and institution from my personal page: http://www.tiesdekok.com/

1. Open up a notebook and import `requests_html`
2. Retrieve website data + parse HTML using `requests_html`
3. Use Chrome DevTools + `SelectorGadget` to find the right CSS Selectors
4. Retrieve name, job title, and institution

For help: see notebook on webscraping

Closing remarks

Setup + Get Started

Mini-Task Instructions

Mini Task

Goal: Get hands-on experience with extracting real data from the web.

Instructions

1. Open (start) a Jupyter Notebook in the `Materials` folder
2. Solve tasks in: `day_2 > mini_task > webscraping_mini_task.ipynb`

For help:

- Python tutorial
- Python Basics Notebook
- Data handling with Pandas
- Web scraping with Python