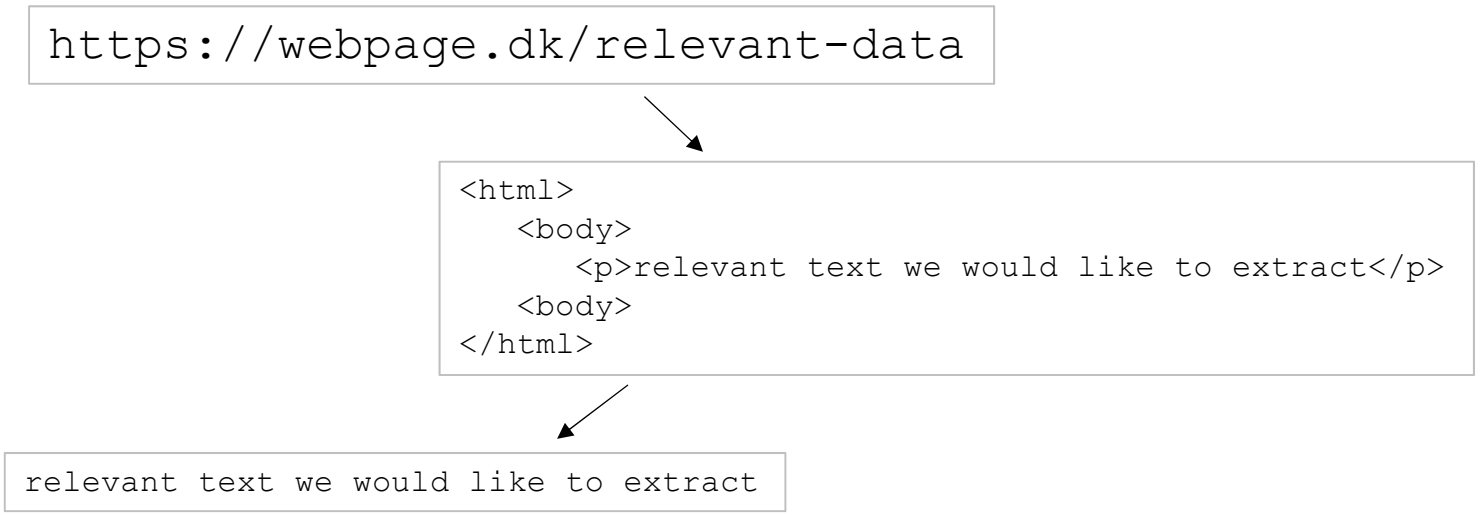
Web-Scraping

Agenda

- Introduction
- Copyright
- Internet and WWW - brief history
- HTML
- CSS
- Python web-scraping
- Exercises

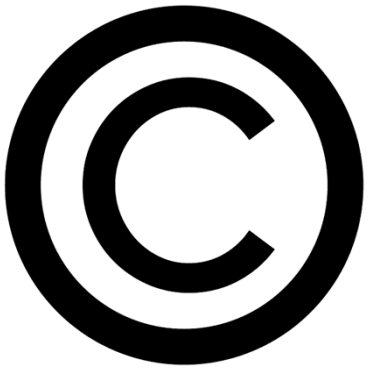
What is Web-Scraping?

- Extract relevant text or images from a website
- Three steps
  1. Fetch the web-page (HTML-document)
  2. Parse the HTML
  3. Query and extract the relevant parts



Copyright

- Even though publicly available many websites have copyright restrictions which must be followed:
  - look at Terms of Service
  - look at /robots.txt
- If we are allowed to scrape data, the following rules must still be followed:
  - **do not** re-distribute collected data without a clear written statement saying you are allowed to
  - **always** make clear where data comes from when used in papers etc. -> use quotation and references

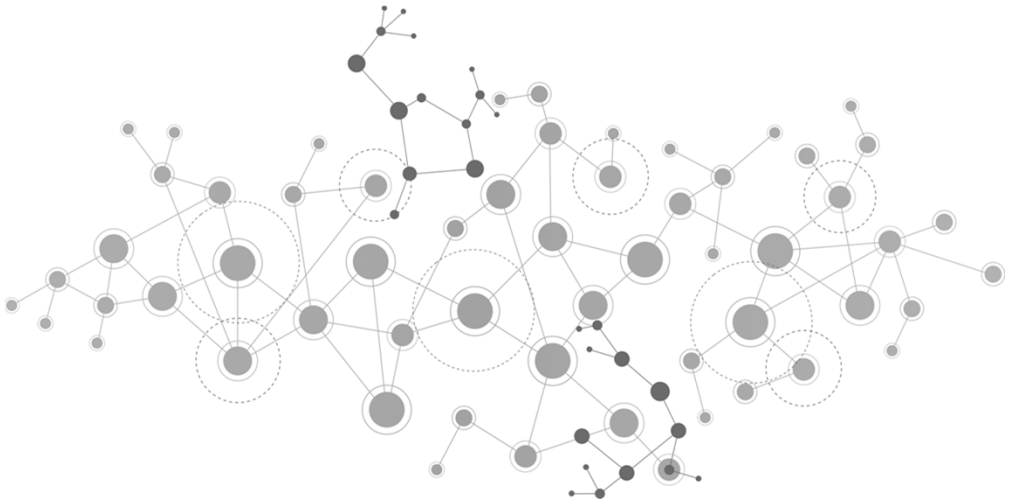


Copyright Examples

- Take a look at: https://www.themoviedb.org/terms-of-use
- https://www.fandom.com/terms-of-use
- https://www.imdb.com/conditions
- What are we allowed to do?

The Internet and WWW

- 1960’es – Arpanet (several other networks followed)
- 1980’es – Different networks switching to the tcp/ip protocol -> global internet
- Protocols:
  - 1970’es – FTP (file transfer)
  - 1980’es – SMTP (email)
  - 1989 – HTTP (WWW)

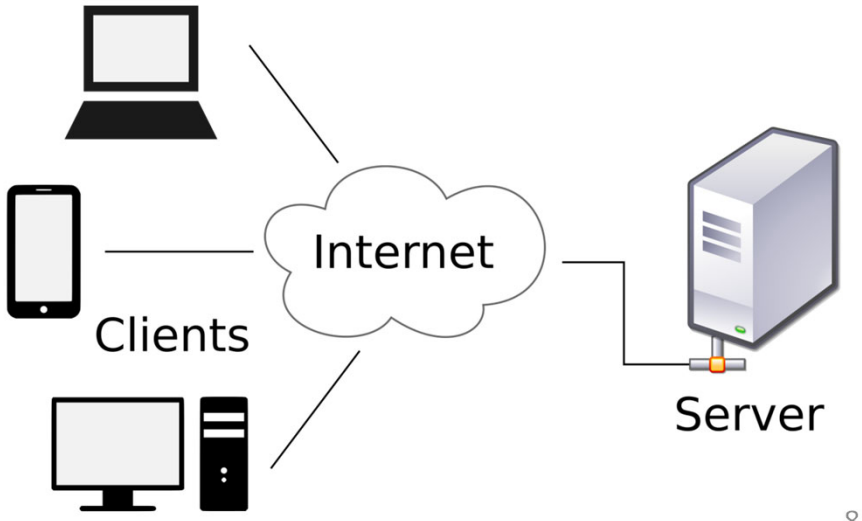


Tim Berners Lee and WWW

- CERN
  - Communication problems
- Solution WWW
  - HTML
    - Hypertext (Vannemar Bush, Ted Nelson etc.)
  - HTTP-protocol
  - Browser
  - Web-Server

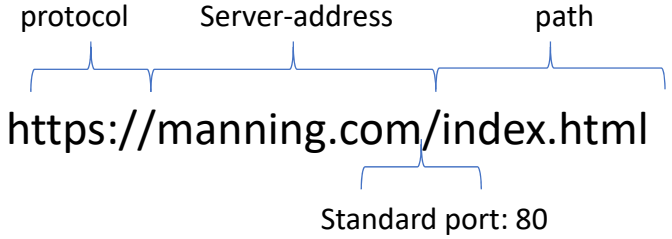
Client - Server

- For a website to become publicly available it must be on a webserver connected to the internet
- To access the web-site we use client software such as a browser, which can interpret the content of the files fetched from the server
- The HTTP-protocol describes how the client and server should communicate



Url (request)

- When we write an url in the address bar or click a link in the browser, the browser performs a HTTP-request to the server
- The server returns the content of the requested file along with some meta-data (http-response)
- The browser interprets the HTML in the response and renders it on the screen
- If the HTML contains references to images, stylesheets, javascript-files etc. these are fetched as well
- Url-structure:



URL	Status	Domain	Size	Remote IP	Timeline
GET index.html	200 OK	manning.com	95,3 KB	184.173.95.35:80	788ms
GET mainhome.css	200 OK	manning.com	12,0 KB	184.173.95.35:80	264ms
GET brand?form=cse-search-b	302 Found	google.com	260 B	216.58.209.100:80	36ms
GET navblog.js	200 OK	manning.com	4,2 KB	184.173.95.35:80	266ms
GET masthead.jpg	200 OK	manning.com	12,5 KB	184.173.95.35:80	392ms
GET top_button_ordering.jpg	200 OK	manning.com	1,2 KB	184.173.95.35:80	388ms
GET top_button_shoppingcart.j	200 OK	manning.com	1,6 KB	184.173.95.35:80	389ms
GET brand?form=cse-search-b	200 OK	cse.google.com	1,1 KB	216.58.209.110:80	240ms
GET google_custom_search_wa	200 OK	cse.google.com	2,0 KB	216.58.209.110:80	19ms
GET mailbox.gif	200 OK	manning.com	74 B	184.173.95.35:80	132ms
GET twitter-16x16.png	200 OK	manning.com	3,0 KB	184.173.95.35:80	131ms
GET facebookF.gif	200 OK	manning.com	1,0 KB	184.173.95.35:80	206ms
GET gupta2_3d.gif	200 OK	manning.com	5,8 KB	184.173.95.35:80	203ms
GET bdavis_3d.gif	200 OK	manning.com	6,7 KB	184.173.95.35:80	203ms
GET nicolette_3d.gif	200 OK	manning.com	7,7 KB	184.173.95.35:80	249ms

HTML

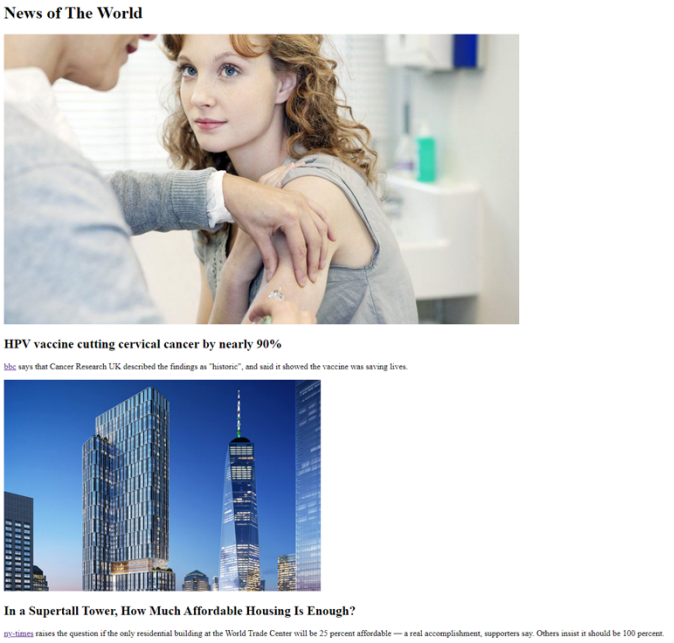
- Describes the structure of a page/document
- The first version of HTML was only intended for online documents and not web-pages with complicated layouts
- The structure is a hierarchy of nested elements
  - An element is made of a start- and end-tag
  - Most elements have semantics
- Start-tags can have attributes

```
<!DOCTYPE html>
<html>
<head>
   <meta charset="utf-8">
   <title>A title here</title>
</head>
<body>
   <h1>Heading</h1>
   <p>
      some regular text <a href="http://dr.dk">with a link</a>
      and an image
      <img src="images/image.jpg" alt="test"/>
   </p>
</body>
</html>
```

Live Code –HTML

- Create a document with a few articles.
- Each article has:
  - A heading (<h1>)
  - body-text (<p>)
    - With some links (<a href="LINK">)
  - An image (<img src=“IMG-FILE">)

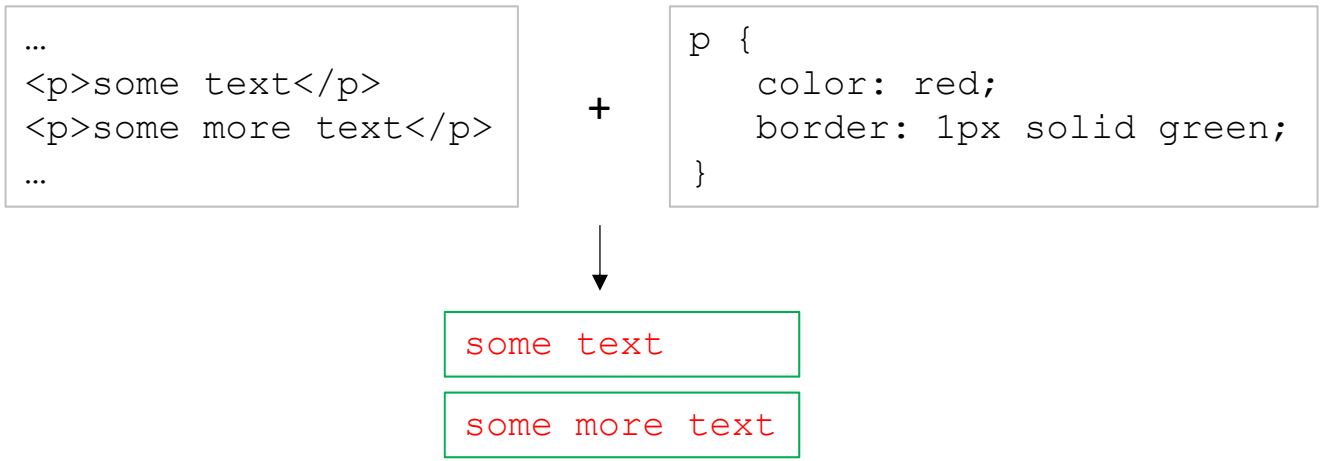
Live Code Solution



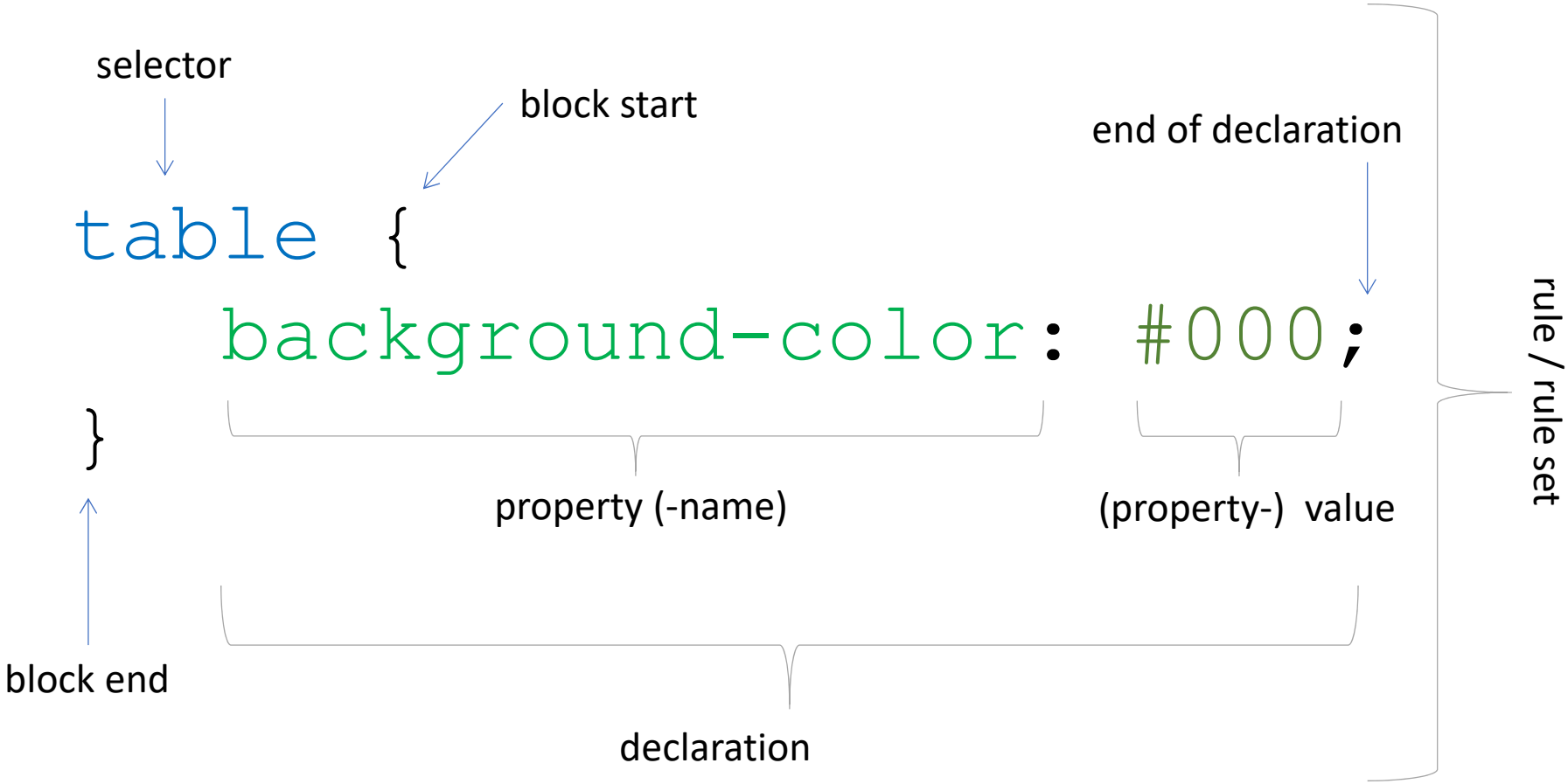

```
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <title>News of the World</title>
</head>
<body>
    <h1>News of The World</h1>
    <article>
        <h2>HPV vaccine cutting cervical cancer by nearly 90%</h2>
        <p><a href="https://www.bbc.com/news/health-59148620">bbc</a> says that Cancer
Research UK described the findings as "historic", and said it showed the vaccine was saving lives. </p>
        <img src="https://ichef.bbci.co.uk/news/976/cpsprodpb/116B9/production/_121335317_gettyimages-705014435.jpg">
    </article>

    <article>
        <h2>In a Supertall Tower, How Much Affordable Housing Is Enough?</h2>
        <p><a href="https://www.nytimes.com/2021/10/29/realestate/5-world-trade-center-affordable.html">ny-times</a>
writes that the only residential building at the World Trade Center will be 25 percent affordable —
a real accomplishment, supporters say. Others insist it should be 100 percent.</p>
        <img src="https://static01.nyt.com/images/2021/10/31/realestate/29affordable1/29affordable1-threeByTwoSmallAt2
    </article>

</body>
</html>
```

CSS (Cascading Style Sheets)

- The popularity of WWW led to further demands for styling
  - WWW was soon not only used for online documents but for websites with various layouts
  - The initial styling applied directly to the start-tag could not be reused
- CSS separates styling from structure and makes the styling reusable
  - CSS can be inline, internal and external
  - CSS-selectors targets HTML-element and applies styling rules



CSS Syntax (CSS Rule Set)



Type Selectors

- The different HTML elements can be referenced by using their name as selector in a CSS rule

```
table {
    margin-bottom: 20px;
    border 1px solid red;
}

p {
    color: red;
}

h1 {
    border: 1px solid #eee;
}
```

```
<table>
<tr><td>cell 1</td><td>cell 2</td></tr>
</table>

<p>
Some text
</p>

<h1>Heading 1</h1>
```

cell 1 cell 2

Some text

Heading 1

Class Selectors

- Can be applied across different HTML element types
- Makes it possible to only style specific elements
- Is applied using the class attribute
- Can be used in context with a type selector, so it only applies to elements with both the type and class

Class selectors starts with ”.”

```
.attention {
    border: 4px solid #f00;
    background-color: #ff0;
}

p.attention {
    color: green;
}
```

```
<table class="attention">
<tr><td>cell 1</td><td>cell 2</td></tr>
</table>

<p class="attention">
Some text
</p>

<h1 class="attention">Heading 1</h1>
```

cell 1 cell 2

Some text

Heading 1

Id Selectors

- Can only be applied to one element in each HTML document
- Is applied using the id attribute
- Can coexist with the class attribute
- Id is more specific than class and therefore takes precedence if the same property is declared in both class- and id-rule

Id- selectors starts with ”#”

```
#introduction {
    font-style: italic;
    font-size: 18px;
    background-color: #fff;
    color: #333;
}
```

```
<p class="attention" id="introduction">
Some text
</p>
```

Some text

Overrides the rules from “p” and “.attention”

Context Specific Selectors

- descendant selector -> elements which matches the selector and is descendant of the outer selector
- child selector -> elements which matches the selector and is direct child of the outer selector

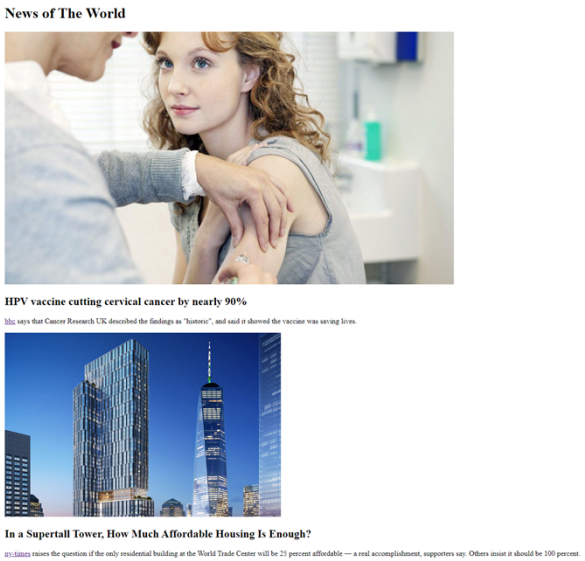
```
p .mini {
   font-size: 10px;
   background-color: red;
}

p > em {
   color: white;
   background-color: black;
}
```

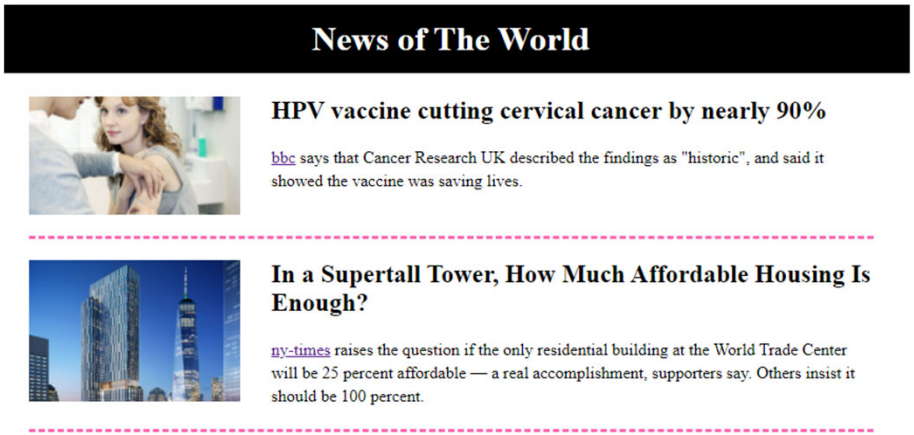
```
<p>
   With <em>innovative</em>
   <span> new <em>performance</em></span>
   features,
   <span><em class="mini">advanced</em></span>
   layering and synchronization
</p>
```

With *innovative* new *performance* features, advanced layering and synchronization

Live code - CSS

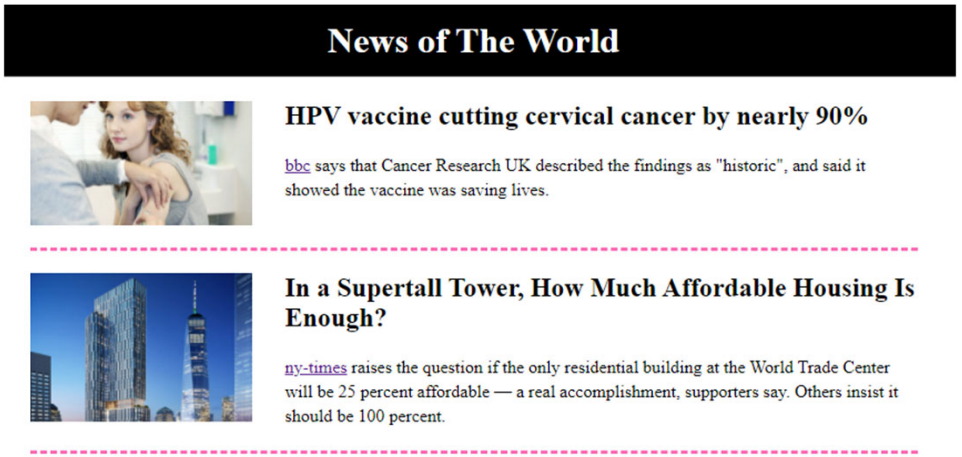
- Apply styling to our web-page





Live Code Solution

```
h1 {
    color: #fff;
    background-color: #000;
    text-align: center;
    line-height: 2em;
}

article {
    margin: auto;
    width: 800px;
    border-bottom: 3px dashed hotpink;
    padding-bottom: 20px;
}

article:after {
    content: "";
    clear: both;
    display: table;
}

article p {
    line-height: 1.4em;
    margin-bottom: 0px;
}

article img {
    float: left;
    width: 200px;
    margin-right: 30px;
}
```



Web-Scraping Python

- We use 2 modules
- requests
  - Request a HTML-document using an url
- beautilfulsoup4
  - Parse the html-page into a set of python-objects we can query using CSS-selector syntax

```
> pip install requests beautifulsoup4
```

requests usage

- requests can be used to make HTTP-requests and fetch the response
  - Documentation: https://docs.python-requests.org/en/latest/
- If the encoding seems wrong use:
  - `response.encoding = response.apparent_encoding`

```
import requests

response = requests.get('https://dr.dk')
# response.encoding = response.apparent_encoding
print(response.text)
```

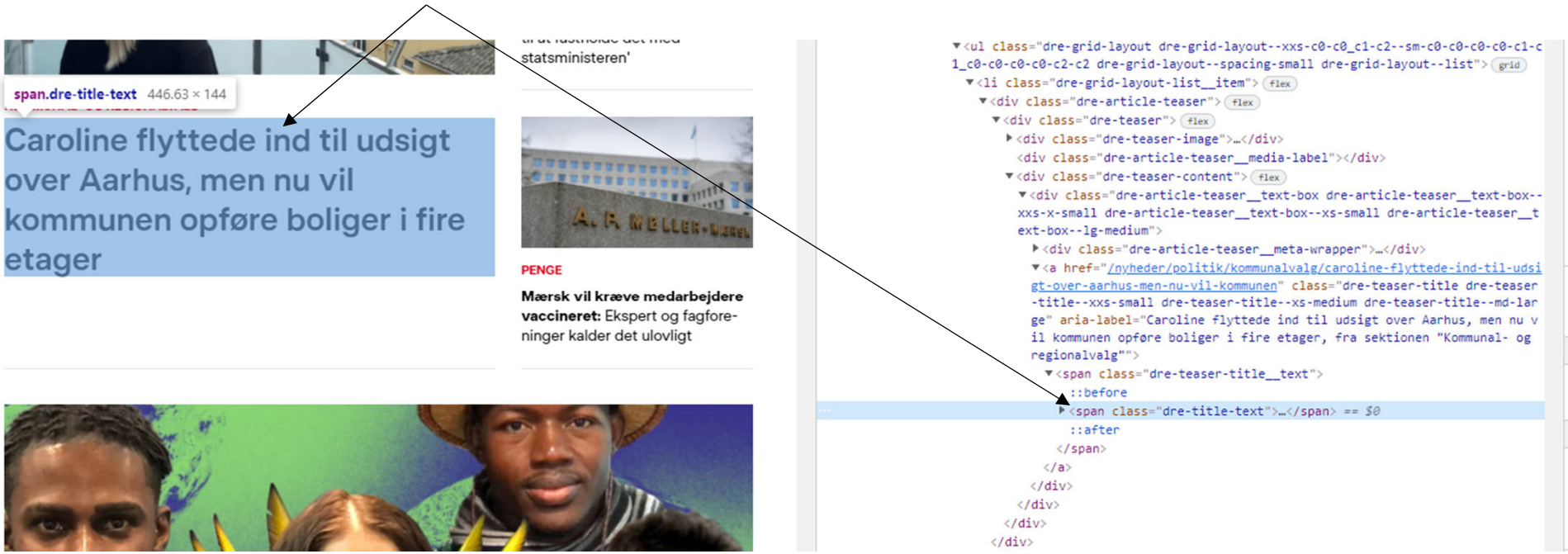
beautifulsoups4 usage

- bs4 parses an HTML-string into a structure of python objects which can queried using CSS-selectors (like the DOM for JS)
  - Documentation: https://www.crummy.com/software/BeautifulSoup/bs4/doc/
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/#css-selectors
- The `select()` method is used to get a list og tag-objects
  - The whole document and each tag can be queried using `select()`
- The `text` property (or method `get_text()`) gives us the text-content of the tag and nested tags
- We can access attributes of a Tag-object using the `get()` method

```
import bs4

html = 'some html…'
soup = bs4.BeautifulSoup(html, 'html.parser')
ps = soup.select('p')
for p in ps:
  print(p.text)
  print(p.get('id'))
```

Find the CSS-path

- In the browser
  - right-click the element we are interested in and click “inspect”
  - We can now see the HTML-structure for the element clicked
  - Try if the type/class/id selector is enough to isolate the wanted text
    - if not, add more context



Live Code – Web Scraping

- Fetch all article headings from dr.dk
- Fetch the title and year-span of the first 100 collections matching “Hasselager” from arkiv.dk
  - We need to paginate
  - We need to break when we have found the first 100 or if there are no more results

dr.dk solution

```
import requests
import bs4


def scrape():
    response = requests.get("https://dr.dk")
    soup = bs4.BeautifulSoup(response.text, 'html.parser')
    for h2 in soup.select('.dre-teaser-title'):
        print(h2.text)


if __name__ == "__main__":
    scrape()
```

Arkiv.dk solution

```
import requests
import bs4


def scrape():
    page = 1
    count = 0
    while True:
        response = requests.get("https://arkiv.dk/soeg?searchstring=Hasselager&page=" + str(page))
        soup = bs4.BeautifulSoup(response.text, 'html.parser')
        rows = soup.select('#resultater tr’)
        for row in rows:
            divs = row.select('div > div')
            year_span = divs[0].text.strip().replace(' ', '')
            title = divs[1].text.strip()
            print(year_span)
            print(title)
            print(100 * '-')
            count += 1
            if count == 100:
                break

        if len(rows) == 0 or count == 100: # no more collections
            break
        else:
            page += 1


if __name__ == "__main__":
    scrape()
```

Exercises

- Extract all headlines from https://tv2.dk
- Extract all names and titles of researchers from https://www.au.dk/om/presse/ekspertlister/corona
- Extract all 116 titles from http://comicscontainer.dk/comicbooks?dynamicSearch=ru&page=1
  - Consider how you will determine when you reached the last page (last page + 1), so you don't get an infinite loop
  - The markup is not very well-structured so you probably need to make several sub-queries with bs4