Scraping: requests, BeautifulSoup and others

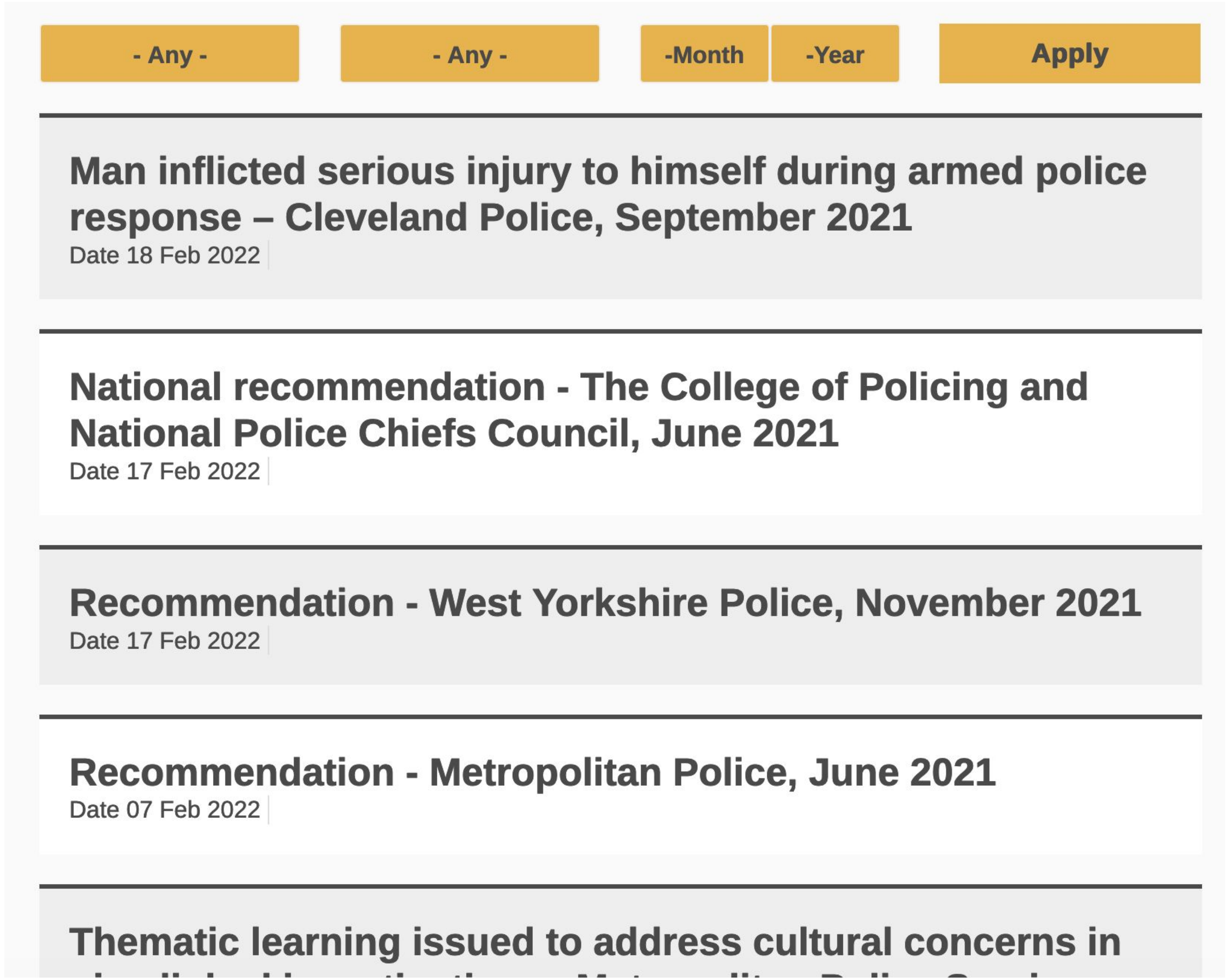
Paul Bradshaw

This week:

- Python libraries for fetching HTML documents…
- …and drilling into them
- …and storing/exporting the results

First, break down the problem(s)

- Fetch a page from a URL
- Drill down into that page
- Extract text/links/etc.
- Store them in a dataframe
- Export them as a CSV



Man inflicted serious injury to himself during armed police response – Cleveland Police, September 2021
Date 18 Feb 2022

National recommendation - The College of Policing and National Police Chiefs Council, June 2021
Date 17 Feb 2022

Recommendation - West Yorkshire Police, November 2021
Date 17 Feb 2022

Recommendation - Metropolitan Police, June 2021
Date 07 Feb 2022

Thematic learning issued to address cultural concerns in

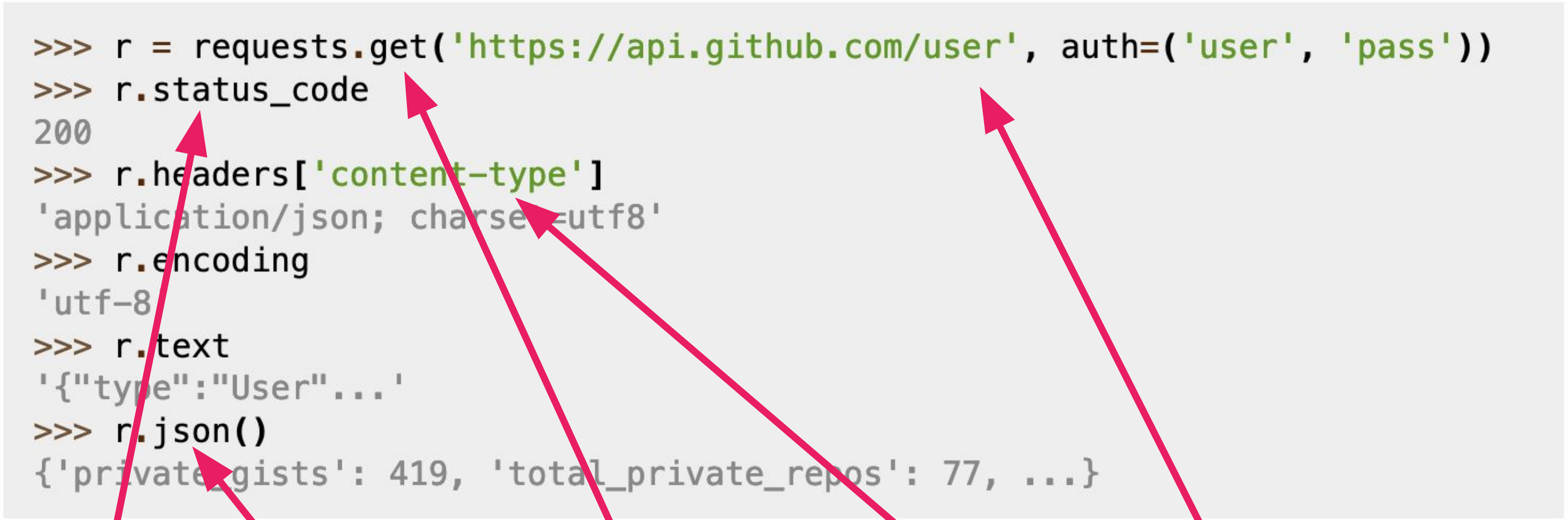
Find libraries to solve problems

- Fetch a page from a URL: **requests**
- Drill down into that page: **BeautifulSoup**
- Extract text/links/etc.: **BeautifulSoup**
- Store them in a dataframe: **pandas**
- Export them as a CSV: **pandas**

The **requests** library

- Fetches a document from a URL
- Not just webpages (HTML document)

docs.python-requests.org/en/latest



```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

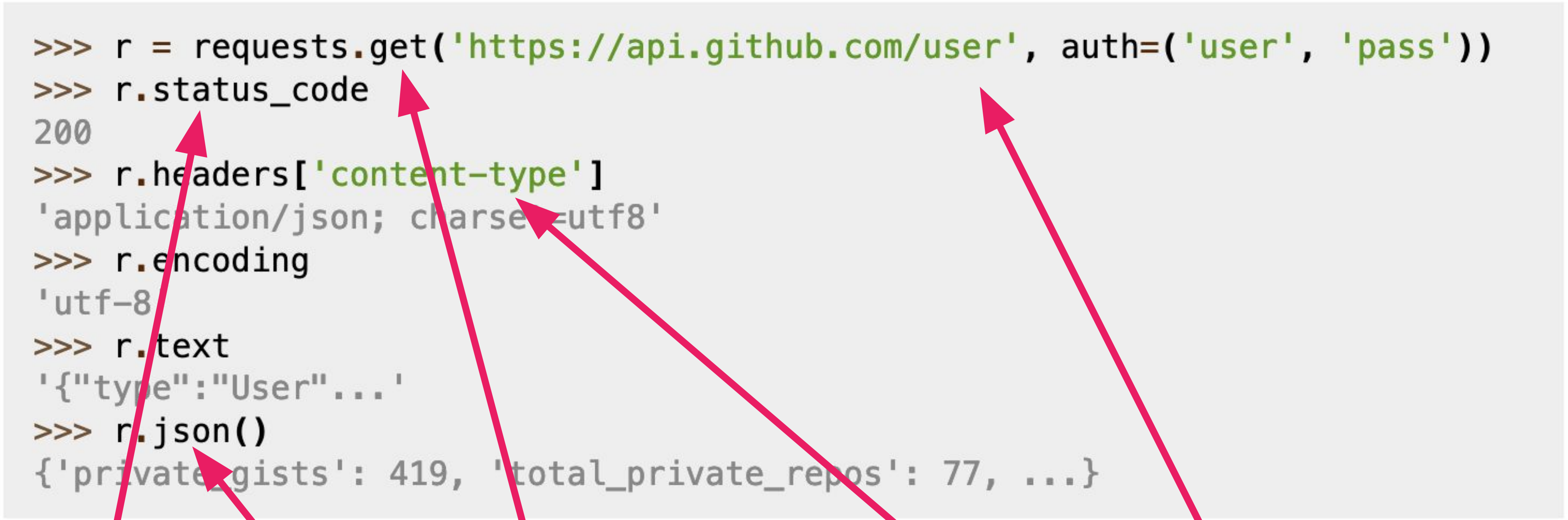
What type of thing is this?

And this?

What type of thing is this?

What clue does this give us about .headers **?**

What type of thing is this?
What's the clue?



```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419, 'total_private_repos': 77, ...}
```

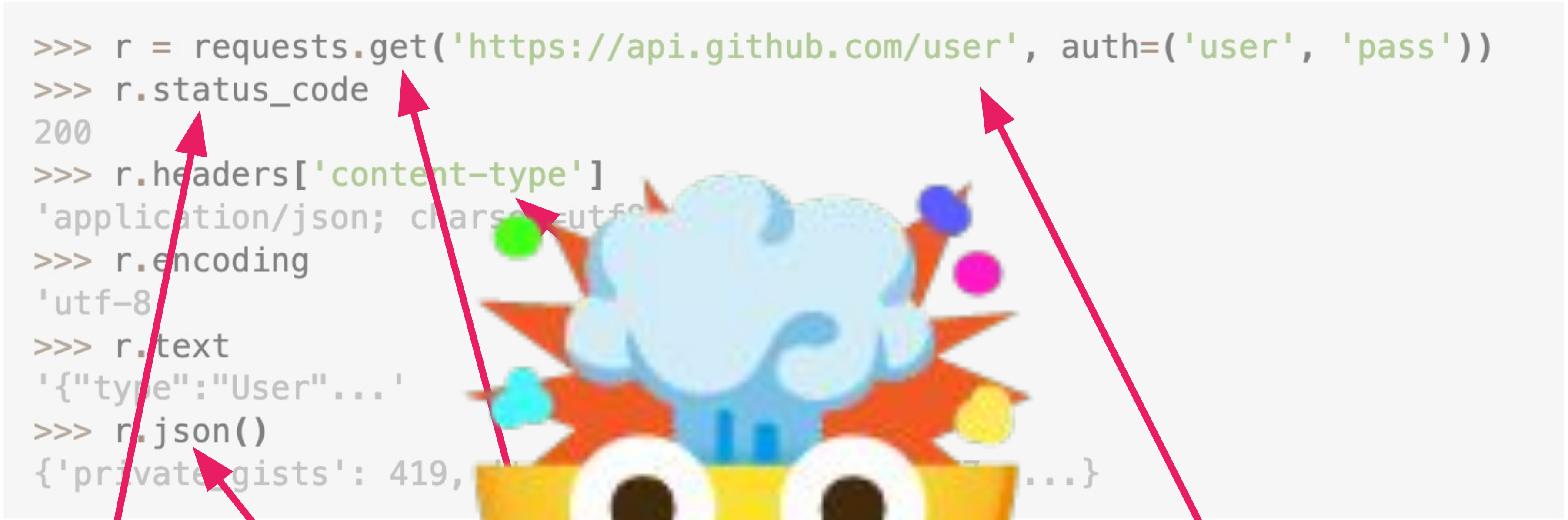
A function from the requests library

An argument/ parameter (for .get())

An attribute (of r)

A method (a function of the object)

It tells us that .headers **is probably a dictionary (it's a key)**



```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8
>>> r.encoding
'utf-8'
>>> r.text
'{"type":"User"...'
>>> r.json()
{'private_gists': 419,                    ...}
```

An attribute
(of r)

A method
(a function of the o

A fun
the re

It tells us that .headers is probably a dictionary (it’s a key)

An argument/
parameter
(for .get())

```
import requests
#fetch a page from a URL
page = 
requests.get("https://www.bbc.co.u
k/news")
```

```
#this is now a requests object
page
#use .content to show the contents
of that object
page.content
```

That’s it for requests.

Reuse the code.

All you have to change is the URL.

Scraping libraries

- More than one!
- Beautiful Soup (bs4)
- Scraperwiki + lxml
- Scrapy

The **BeautifulSoup** library

- Converts a webpage into a ‘parsable’ object
- A HTML ‘parser’ (can drill into HTML to fetch contents of specified tags)

crummy.com/software/BeautifulSoup

Imported ‘from’ bs4

```
from bs4 import BeautifulSoup
```

- Means you can use BeautifulSoup function without having to name the bs4 library too, i.e. **bs4.BeautifulSoup( )**

The BeautifulSoup( ) function

```
soup =
BeautifulSoup(page.content,
‘html.parser’)
```

- Two ingredients: the object fetched by `requests`, and the ‘parser’

Using **.select( )** to grab tags

- Applied to Beautiful Soup objects
- Use CSS ‘selectors’ to extract information
- E.g. “*text within <a> tags inside <p> tags*”
- Or “*the href= value inside an <a> tag*”

Some examples

```
h2tags = soup.select(‘h2’)

links = soup.select(‘a’)

boldtext = soup.select(‘p b’)

nums = soup.select(‘li[class=”number”]’)
```

The .select( ) method

- **.select( )** - grab whatever is inside the CSS selector specified
- Results will *always* be a **list** (even if 0 or 1 matches)
- Can extract text or attributes from items

```
#result looks like this

[<h2>Accessibility links</h2>,
 <h2 class="gs-u-vh">News
Navigation</h2>]
```

Selectors

CSS selectors

- HTML tags: a ‘map’ to the content
- E.g. “p” will fetch contents of <p> tags
- “p em” will fetch contents of <em> tags inside <p> tags

(A brief introduction to HTML)

- HTML **tags** can have **attributes** and **values**
- E.g. <img width="400px">
- **Tag**: img
- **Attribute**: width
- **Value**: "400px"

For example: links and images

- The **tag** for a link is <a> and it needs a href= **attribute** to work. The **value** of that is the URL it goes to
- Likewise the <img> tag needs a src= attribute with the address of the image

How can you tell?

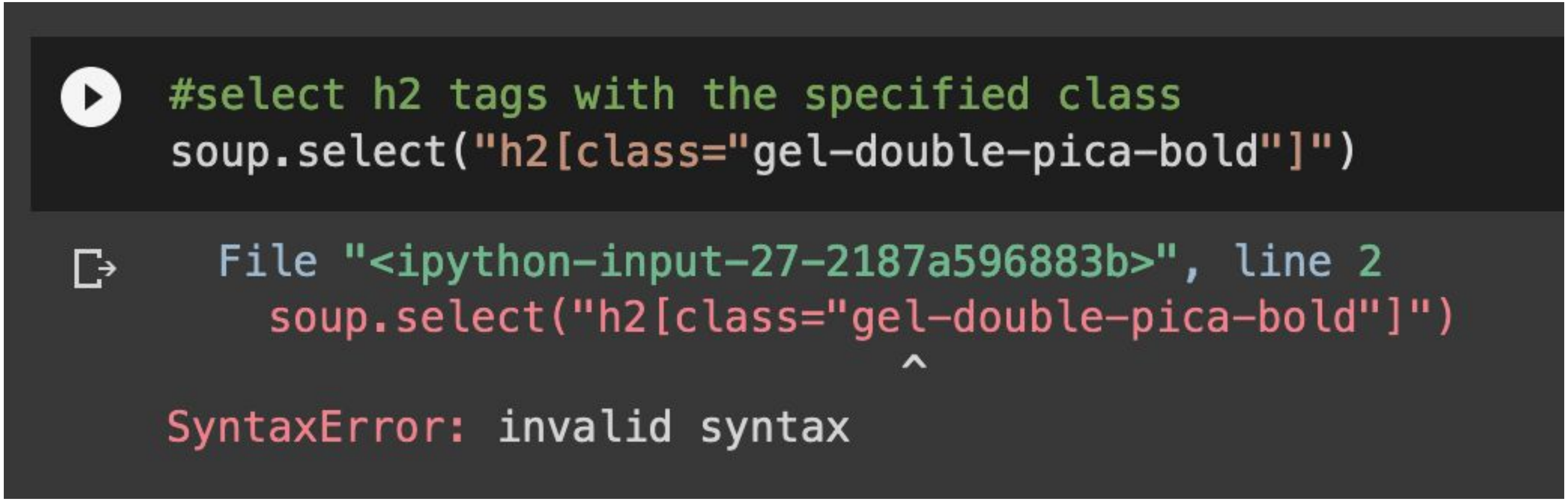
- The **tag** is always the first word after <
- The **attribute** is always followed by an =
- The **value** is always *after* the =
- They are also coloured differently when viewing source code (helpfully)

Selecting by attributes and values

- **soup.select(‘a[target="_blank"]’)** - grab whatever is inside <a target=“blank”>
- **soup.select(‘[title~="flower"]’)** - grab whatever is inside a tag where the value of the title attribute includes the string ‘flower’

⚠️ Quotes inside quotes warning!

- Note that there are two sets of quotation marks in **(‘a[target="_blank"]’)** so they need to use different quotation marks
- The whole string is inside single quotes; so “blank” needs to be inside double quotes



```python
#select h2 tags with the specified class
soup.select("h2[class="gel-double-pica-bold"]")
```

```
  File "<ipython-input-27-2187a596883b>", line 2
    soup.select("h2[class="gel-double-pica-bold"]")
                             ^

SyntaxError: invalid syntax
```

Selecting by div or class

- **soup.select("p.hello")** - grab whatever is inside <p class=”hello”>
- **soup.select("p#hello")** - grab whatever is inside <p id=”hello”>

(But can also just use class/id)

- **soup.select(‘p[class="hello"]’)** - grab whatever is inside <p class=”hello”>
- **soup.select(‘p[id="hello"]’)** - grab whatever is inside <p id=”hello”>

`div`

= select all <div> tags

`div.job`

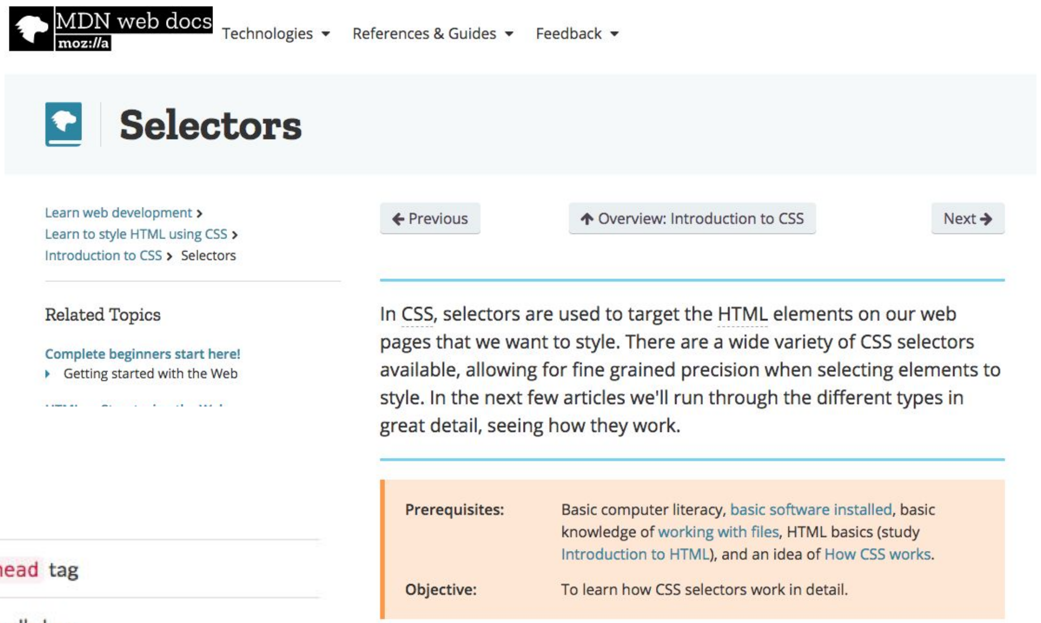
= select all <div class=”job”>

`div#job`

= select all <div id=”job”>

`div a`

= select all <a> tags within <div>



Selectors

Learn web development ›
Learn to style HTML using CSS ›
Introduction to CSS › Selectors
← Previous
↑ Overview: Introduction to CSS
Next →

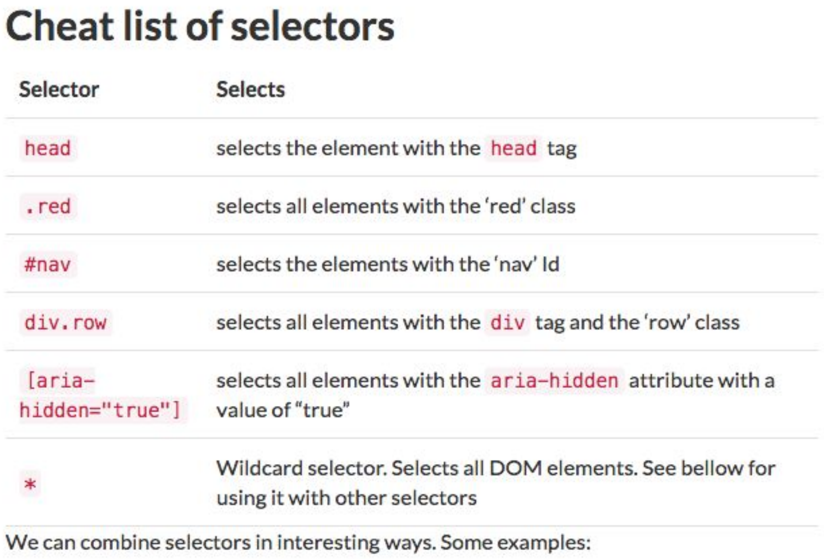
Related Topics

Complete beginners start here!

- Getting started with the Web

In CSS, selectors are used to target the HTML elements on our web pages that we want to style. There are a wide variety of CSS selectors available, allowing for fine grained precision when selecting elements to style. In the next few articles we'll run through the different types in great detail, seeing how they work.

Prerequisites:	Basic computer literacy, basic software installed, basic knowledge of working with files, HTML basics (study Introduction to HTML), and an idea of How CSS works.
Objective:	To learn how CSS selectors work in detail.



Cheat list of selectors

Selector	Selects
`head`	selects the element with the `head` tag
`.red`	selects all elements with the 'red' class
`#nav`	selects the elements with the 'nav' Id
`div.row`	selects all elements with the `div` tag and the 'row' class
`[aria-hidden="true"]`	selects all elements with the `aria-hidden` attribute with a value of "true"
`*`	Wildcard selector. Selects all DOM elements. See bellow for using it with other selectors

We can combine selectors in interesting ways. Some examples:

https://developer.mozilla.org/en-US/docs/Learn/CSS/Introduction_to_CSS/Selectors
https://guide.freecodecamp.org/css/tutorials/css-selectors-cheat-sheet/

Loop or use index to access item

Grab the item index 0 in that list of matches

```
    h2s[0]
```

Loop through and print each item

```
    for i in h2s:
        print(i)
```

Extract text or links

- **i.get_text( )** - grab the text contents of the targeted tags
- **i[‘href’]** - grab the URL that’s linked to (the href= value)

Reuse the code.

All you have to change is the selectors.

Creating empty lists to fill with data

- Create an empty list before loop
- Each time loop runs, **append** new item using the **.append( )** method on the list

E.g. **mylist.append(“Paul”)** adds “Paul” to list

```
#create an empty list
headlines = []
for i in h2s:
    #extract the text for that item
    h2text = i.get_text()
    #add it to the list
    headlines.append(h2text)
```

```
#grab the h2 tags with the specified class

h2s = soup.select('h2[class="gel-double-pica-bold"]')

#create an empty list

headlines = []

#loop through the h2 matches

for i in h2s:

  #extract the text for that h2

  h2text = i.get_text()

  #add it to the list

  headlines.append(h2text)
```

Saving the results: **pandas**

- Create a dataframe using **pd.DataFrame( )**
- Ingredients: a dictionary
- Key(s) are your column heading(s)
- Value(s) are your list(s)

```
#create two lists
list1 = ['Paul','Maeve','Alice']
list2 = [20,30,40]
#create a dataframe
simpledf =
pd.DataFrame({"names":list1,
                     "scores":list2})
```

```
h2s = soup.select('h2[class="gel-double-pica-bold"]')
headlines = []

for i in h2s:
  h2text = i.get_text()
  headlines.append(h2text)

#create a dataframe with that list
bbcheadlines = pd.DataFrame({"headlines": headlines})
```

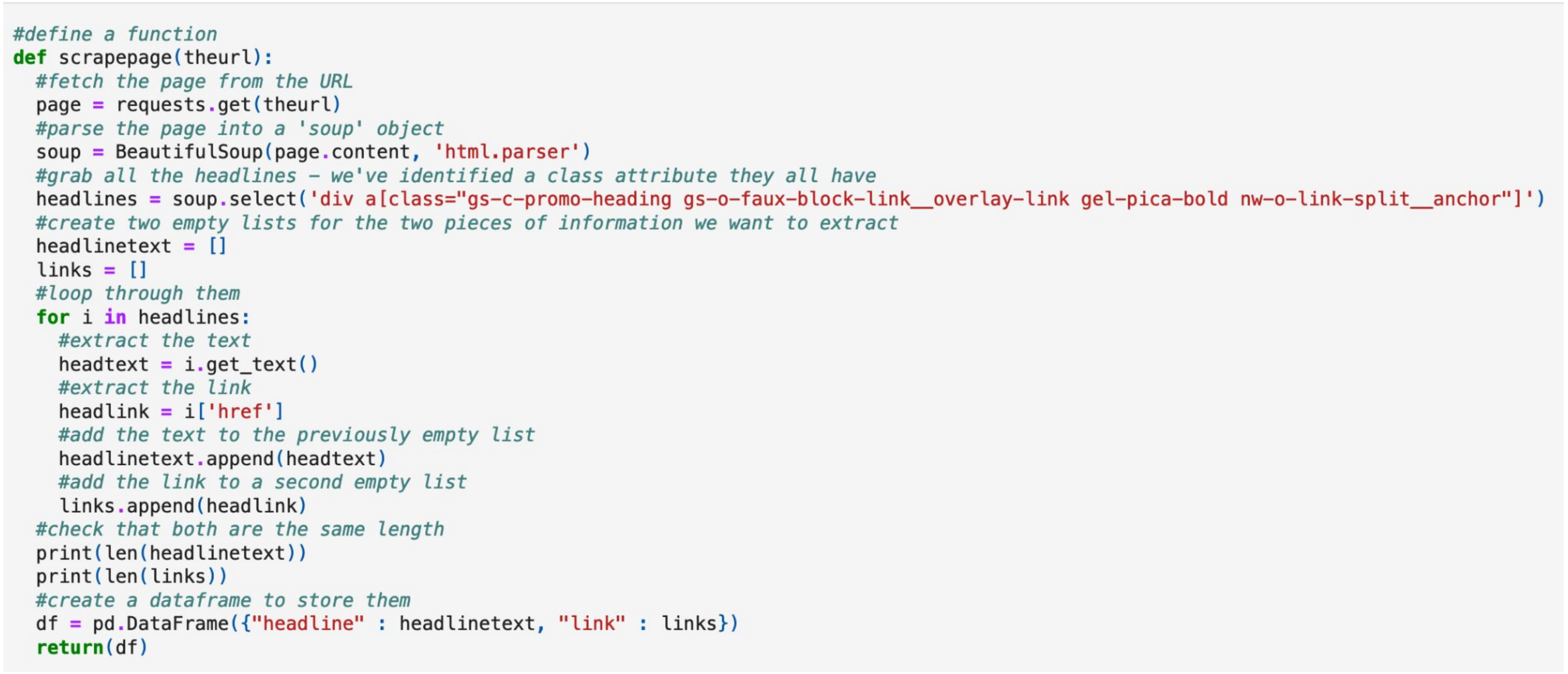
Reuse the code.

All you have to change are the column names

(and you don’t even have to do that).

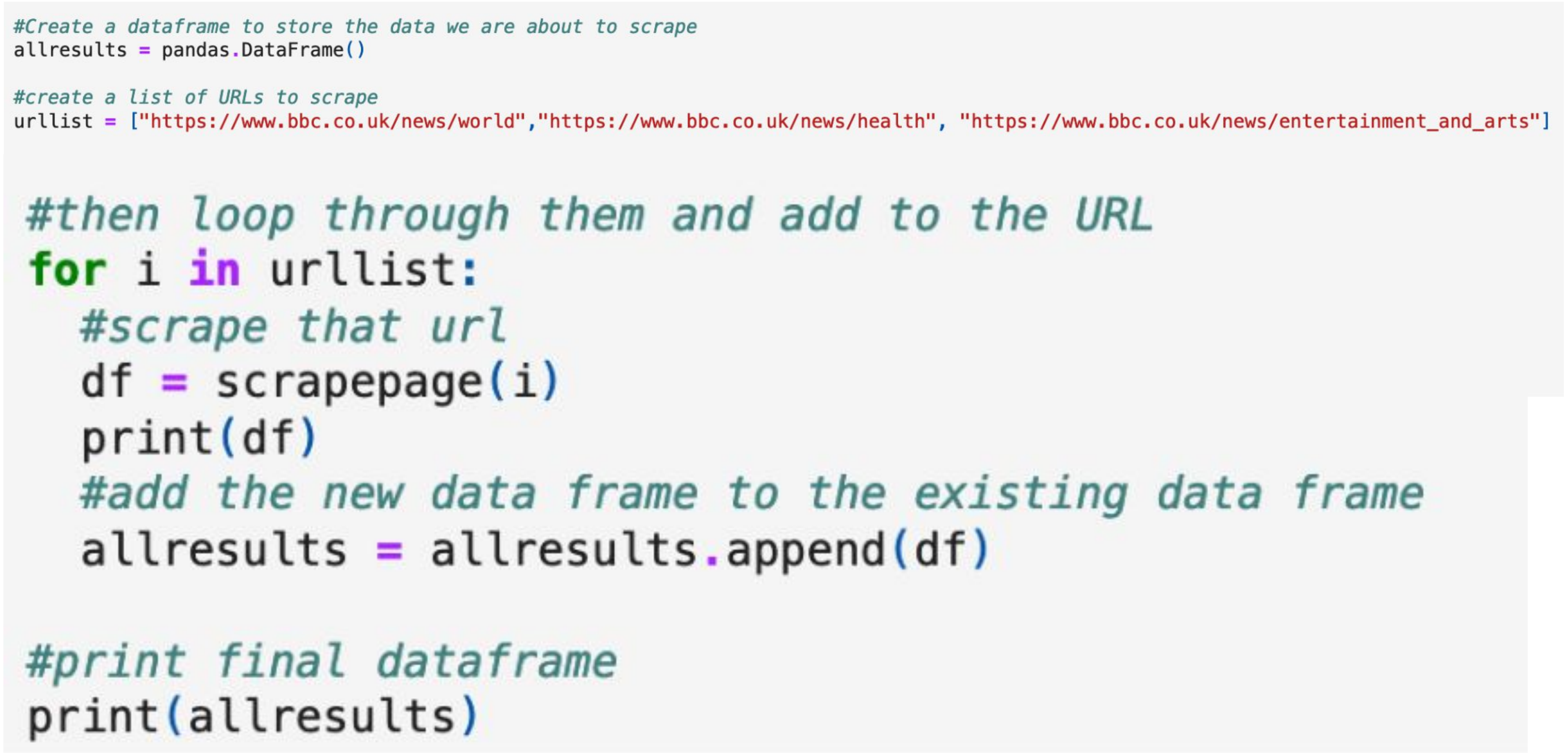
Scraping multiple pages

- Once you’ve scraped one page, you can store the ‘steps’ in your own **function**
- **Loop** through a list of URLs and apply that function to each
- **Append** the results to a dataframe as you go



```python
#define a function
def scrapepage(theurl):
  #fetch the page from the URL
  page = requests.get(theurl)
  #parse the page into a 'soup' object
  soup = BeautifulSoup(page.content, 'html.parser')
  #grab all the headlines - we've identified a class attribute they all have
  headlines = soup.select('div a[class="gs-c-promo-heading gs-o-faux-block-link__overlay-link gel-pica-bold nw-o-link-split__anchor"]')
  #create two empty lists for the two pieces of information we want to extract
  headlinetext = []
  links = []
  #loop through them
  for i in headlines:
    #extract the text
    headtext = i.get_text()
    #extract the link
    headlink = i['href']
    #add the text to the previously empty list
    headlinetext.append(headtext)
    #add the link to a second empty list
    links.append(headlink)
  #check that both are the same length
  print(len(headlinetext))
  print(len(links))
  #create a dataframe to store them
  df = pd.DataFrame({"headline" : headlinetext, "link" : links})
  return(df)
```

https://github.com/paulbradshaw/pythonin12parts/blob/main/part4/08createFunctionsScraperBS.ipynb



```
#Create a dataframe to store the data we are about to scrape
allresults = pandas.DataFrame()

#create a list of URLs to scrape
urllist = ["https://www.bbc.co.uk/news/world","https://www.bbc.co.uk/news/health", "https://www.bbc.co.uk/news/entertainment_and_arts"]

#then loop through them and add to the URL
for i in urllist:
  #scrape that url
  df = scrapepage(i)
  print(df)
  #add the new data frame to the existing data frame
  allresults = allresults.append(df)

#print final dataframe
print(allresults)
```

https://github.com/paulbradshaw/pythonin12parts/blob/main/part4/08createFunctionsScraperBS.ipynb

What happens when it doesn’t work?

- The scraper might be being **blocked**
- The content might be loaded **dynamically**
- Look at `page.content` to see if the text is different to what’s on screen (e.g. captcha, information not there)

Tackling those problems

- Scraper being blocked? Try adding user agents
- Content being loaded dynamically? Try using the browser inspector to access it
- If those don’t work, phone a friend…

Key points

- Use **template code** to fetch & scrape a URL
- Change the **URL**, change the **selectors** for your target page(s) — that's it
- Loop through results and **append** to empty list
- Create a **dataframe** to export/analyse