Webscraping documentation

Source : Corey Schafer YT Video

<https://www.youtube.com/watch?v=ng2o98k983k>

# Packages we need

* Pip install beatifulsoup4
* Pip install lxml
* Pip install requests

We can use several parsers but we are going to use lxml just like in the tutorial .

# What is a parser ?

A parser is a program which is given a sequence of text and identifies the different parts of it in order to give “meaning” to it.

Most commonly a parser is used to identify the meaning of source code but it is entirely possible to identify the meaning of English text aswell.

In the following project we are going to give our ‘lxml’ parser the contents of the html code of the page we wish to scrape and it will parse it for use.

# Requests module

The requests module is a module that helps you download information from a website .

link = 'https://coreyms.com/'

source = requests.get(link).text

The requests.get() method will download the link we gave it and transform it into text.

A very good practice is to include the raise\_for\_status() – which is a method that will raise an error if the download was not succesfull

Our request part of the code should look something like this:

link = 'https://coreyms.com/'  
source = requests.get(link)  
source.raise\_for\_status()  
source = requests.get(link).text

## Downloading with requests

This has nothing really to do with this projects but when you want to download for example a .txt file from the internet using the requests library we have to write the file in binary mode in order to maintain the texts encoding. Example below:

import requests  
  
link = 'http://www.gutenberg.org/cache/epub/61148/pg61148.txt'  
r = requests.get(link)  
r.raise\_for\_status() # Raises an exception if download was not completed  
  
with open('downloadedtext.txt', 'wb') as f:  
 content = r.text  
 for chunk in r.iter\_content(100000):  
 f.write(chunk)

‘wb’ – Write Binary

The iter\_content() method returns “chunks” of the content on each iteration through the loop. Each chunk is of the bytes data type, and you get to specify how many bytes each chunk will contain. One hundred thousand bytes is generally a good size, so pass 100000 as the argument to iter\_content().

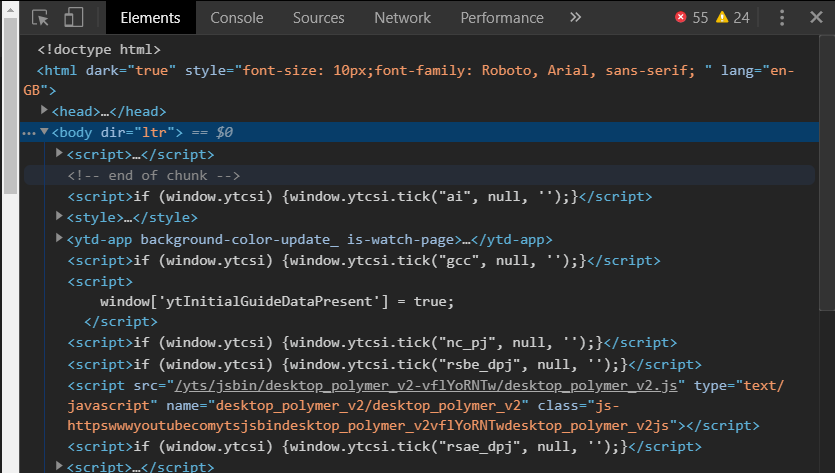
# Knowing the terrain

In order to make our lives easier it is highly recommended to first take a look at the website we want to parse .

First of all – Does it have an API ? if it does we shouldn’t bother scraping it because we can just continue using the request module together with the API and getting the information much easier without much hassle

Next check the website , look at the structure . A good way to do this is to use the built in developer tools of your browser (in chrome press F12)

It should look something like this:



When you hover over an element in this window it will highlight the appropriate part of the website it belongs to.

A good way to parse a website (by Corey Schafer) .

Let’s say you want to parse a website with multiple articles , you should first find the attributes of the first article like : Title, Date Added, Summary etc after you have all of this working and the information is displayed properly go ahead and scrape the website for all URL’s.

# Beatifulsoup

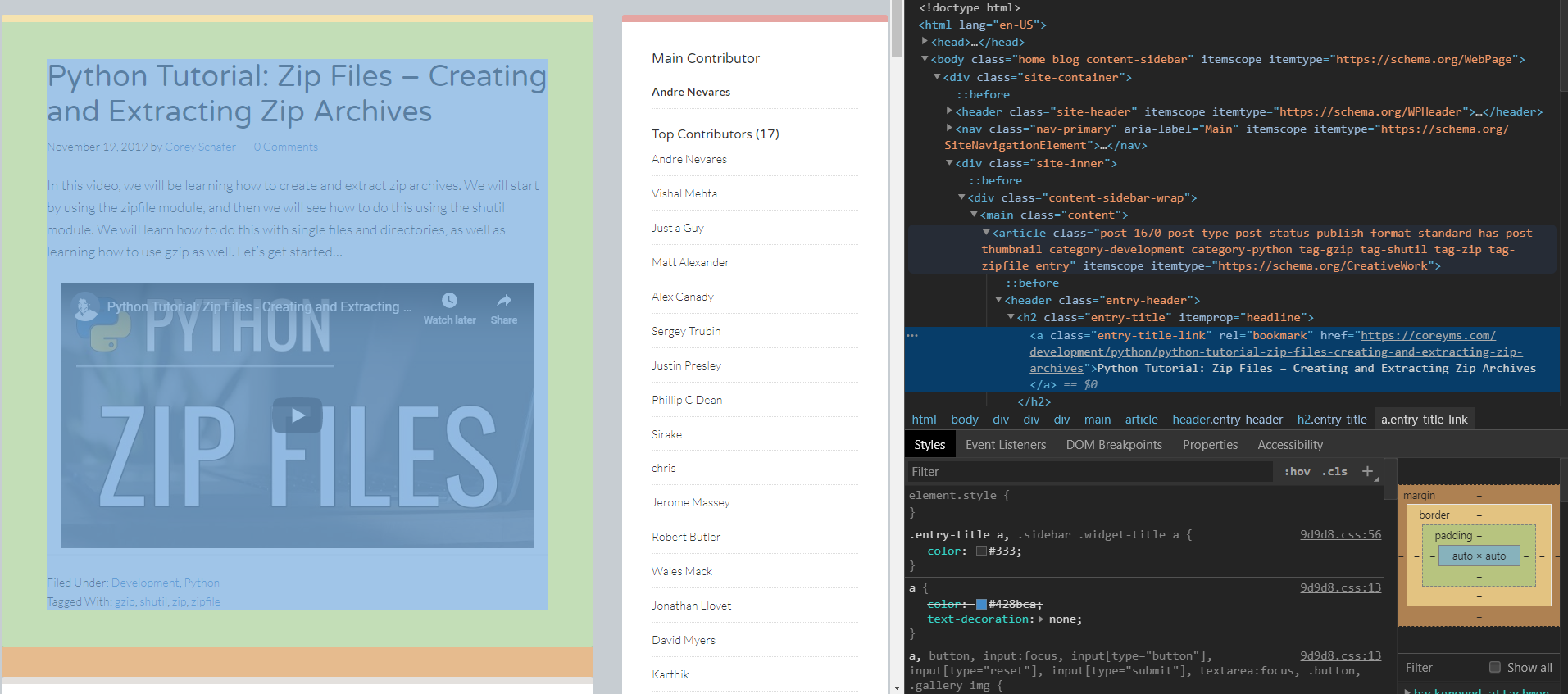
After we done the request part we are going to head on and make an instance of beautiful soup called soup in which we are going to pass in our requests response and the parser we want to use (in our case lxml)

soup = BeautifulSoup(source, 'lxml')

We are going to use Corey Schafers website as an example

## Looking at the Source Code

Next we should look at our page source and see the tag within which one article is containted



## Getting the information about a single article

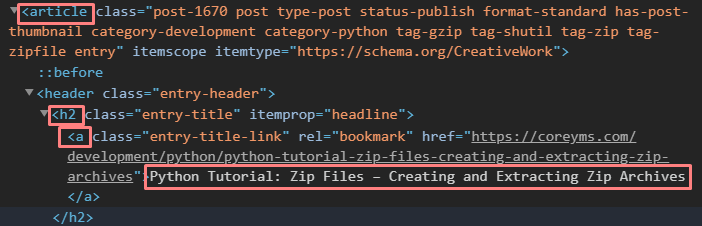
We can see here that a single article is contained within a <article> tag .

So next we can state the following.

Article = soup.find('article')

This means we are going to find the **first** article tag within the page. (We are going first scrape a single article and after we have the wanted outcome we are going to move on to scrape all the rest of the articles)

Next step is finding the heading of the article. We are going to use the dev tool again



As we can see the text of the headline is contained within an <a> (anchor tag) along with several information we don’t need. The way we are going to approach this is surprisingly easy.

Article = soup.find('article')

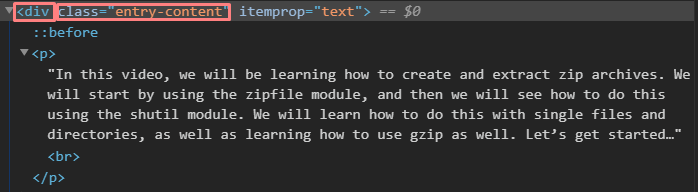
headline = article.h2.a.text

This is basically saying article is within <h2> 🡪 <a> and .text get the text contained within the <a> tag.

For finding the time and other information we are going to use a different approach . We are going to use the find() function where we pass in the tag we are looking for and the class of the specific tag to narrow down the search

summary = article.find('div', class\_='entry-content').p.text.strip()

date\_published = article.find('time', class\_='entry-time').text

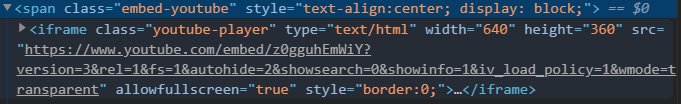


As you can see we just pass in the arguments and put .p at the end to narrow down the search then .strip() to strip all unwanted whitespace.

Same goes for date\_published

## Finding the Link

Finding the youtube link for each video is going to be tricky since it’s an embedded video.



We could just get the link but it will just open a windows with the embedded video and not directly youtube, but since we know how to use youtube we are going to split the text of the link and make our own formatted youtube link.

link = article.find('iframe', class\_='youtube-player')['src']

By typing [‘src’] at the end, it will gives us the src part of the tag.

After this we are going to split the text on each forward slash ‘/’

vid\_id = link.split('/')[4]

vid\_id = vid\_id.split('?')[0]

We are getting the 4th index of our split which is the actual unique id of the video



Then we split the video again so we get rid of the ‘?’ and then we use the first index of the list which is the actual unique id.

Formatting the link is easy we just add the video id after /watch?v= in the youtubelink

youtube\_link = f'https://www.youtube.com/watch?v={vid\_id}'

And there we have our youtube link . Now we have all the information we need and we can move on to scraping all the articles

## Scraping all articles

We are going to use the find\_all() function to find all the articles in our webpage instead of find() which gives us only the first one.

for article in soup.find\_all('article'):  
 headline = article.h2.a.text  
 date\_published = article.find('time', class\_='entry-time').text  
 summary = article.find('div', class\_='entry-content').p.text.strip()  
 try:  
 link = article.find('iframe', class\_='youtube-player')['src']  
 vid\_id = link.split('/')[4]  
 vid\_id = vid\_id.split('?')[0]  
 youtube\_link = f'https://www.youtube.com/watch?v={vid\_id}'  
 except TypeError:  
 youtube\_link = None

We are using a try/except block for the YT link since not all of the articles have a link , if a link is missing we are just going to set the link to none.

## Scraping Multiple Pages (not the best way)

Since we have more pages on the website we are going to format the link in order to open each page and continue scraping away.

for pagenumber in range(0,11):  
 link = f'https://coreyms.com/page/{pagenumber}'  
 source = requests.get(link).text  
  
 soup = BeautifulSoup(source, 'lxml')

We can manually set the range to the number of pages we want to scrape , in the example above we are going to scrape the first 10 pages.

Quick tip: Just go to the second page of the website and take a look at the url formatting to quickly find out how to scrape more pages

## Writing the files to a .CSV file

We are going to use the csv module for this .

file = "corey\_schafer\_scrape\_contextmanager.csv"  
  
with open(file, 'w') as f:  
 csv\_writer = csv.writer(f)  
 csv\_writer.writerow(['headline', 'date\_published', 'summary', 'youtube\_link'])  
  
 for pagenumber in range(0,11):  
 link = f'https://coreyms.com/page/{pagenumber}'  
 source = requests.get(link).text  
  
 soup = BeautifulSoup(source, 'lxml')  
  
 for article in soup.find\_all('article'):  
 headline = article.h2.a.text  
 date\_published = article.find('time', class\_='entry-time').text  
 summary = article.find('div', class\_='entry-content').p.text.strip()  
 try:  
 link = article.find('iframe', class\_='youtube-player')['src']  
 vid\_id = link.split('/')[4]  
 vid\_id = vid\_id.split('?')[0]  
 youtube\_link = f'https://www.youtube.com/watch?v={vid\_id}'  
 except TypeError:  
 youtube\_link = None  
  
 csv\_writer.writerow([headline, date\_published, summary, youtube\_link])

Before entering our for loop we are going to write our headings so it will not write a heading for each iteration of the loop.

In the end we just tell our csv\_writer.writerow() function to write each our extracted data.