Web scraping and social media scraping – scraping a single, static page

Jacek Lewkowicz, Dorota Celińska-Kopczyńska

University of Warsaw

March 19, 2019

What have we learnt so far?

- The logic of the structure of XML/HTML documents
- How to write easy (and more complex) XPaths
- The principles of responsible scraping

What we will be working on today?

- How to start a project in Scrapy or Beautiful Soup
- How to write a simple scraper
- Especially, how to deal with scraping tables

Note: In this lecture we will show how to scrap statically generated sites. Crawling and simulating the user will be covered later!

Convention

- In snippets, we will highlight in violet the areas where you may put your own content
- In commands, the areas in [] are optional
- UNIX-like systems use "/" as the path separator and DOS uses "\". In this presentation the paths will be written in UNIX-like convention if not stated otherwise

Before we start – useful command line commands

Description	Windows (DOS)	UNIX-like
change directory	`CD, CHDIR`	`cd`
copy files	`COPY`	`cp [-r]`
move files	`MOVE`	`mv`
delete files	`DEL`	`rm`
delete directories	`DELTREE`	`rm -r`
delete empty directories	`RMDIR`	`rmdir`
list files in directory	`DIR`	`ls [-lah]`
create directory	`MD, MKDIR`	`mkdir`

Running Beautiful Soup

```
# import all the necessary libraries
import requests
from bs4 import BeautifulSoup

#specify the url of the website you want to scrape
url = 'http://your-site-here.net'
r = requests.get(url)
html_doc = r.text

#creation of a BeautifulSoup object
soup = BeautifulSoup(html_doc)
pretty_soup = soup.prettify()
print(pretty_soup)

#code the part that extracts the data
#remember to extract responsibly
```

Saving data

```
import csv
from datetime import datetime

# open a csv file with append, so old data will not be erased
with open('filename.csv', 'a') as csv_file:
    writer = csv.writer(csv_file)
    writer.writerow([item1, item2, datetime.now()])
```

Typical approach

- You filter the parts of the HTML code:
  - `findAll(tag, attributes, recursive, text, limit, keywords)`
  - `find(tag, attributes, recursive, text, keywords)`
- Examples:

```
allText = bsObj.findAll(id="text")
print(allText[0].get_text())
```

Xpaths in Beautiful Soup

- Typically you do not use XPaths natively in Beautiful Soup
- You will need to use functions from `lxml` package instead (*not covered in classes*)

```
from lxml import etree

with open('index.html', 'r') as myfile:
    data=myfile.read()

root = etree.fromstring(data)
item = etree.ETXPath("//*[text()='Searched Text']") (root)
```

Scrapy – starting a project

Description	Windows (DOS)	UNIX-like
1. Open the command line tool	menu Start → cmd	open console
2. Navigate to desired location	`cd ..|\disk:\path\to\dir`	`cd ..|/path/to/dir`
3. Start a project	`scrapy startproject projectname`	`scrapy startproject projectname`

Content of the project directory in Scrapy

```
projectname/
    scrapy.cfg            # deploy configuration file

    projectname/          # project's Python module, you'll import your code from here
        __init__.py

        items.py          # project items definition file

        middlewares.py    # project middlewares file

        pipelines.py      # project pipelines file

        settings.py       # project settings file

        spiders/          # a directory where you'll later put your spiders
        __init__.py
```

Structure of the spider in Scrapy

```
import scrapy
import necessary_library

class NameItem(scrapy.Item):
    url               = scrapy.Field()
    field_name        = scrapy.Field()
    another_fiel_name = scrapy.Field()

class Name_of_YourSpider(scrapy.Spider):
    name = 'myFirstSpider'
    # here we will provide the starting urls
    # note the subtle difference between using [] and () while providing lists of urls
    start_urls = [
        ''http://your_site_here.net/1/',
        ''http://your_site_here.net/2/',]

    # here we will (optionally) declare the rules for crawlers and modify the scrapers

    def __init__(self, category=None, *args, **kwargs):
        super(Name_of_YourSpider, self).__init__(*args, **kwargs)
        self.module_you_change = put_your_code_here

    # here we will put our xpaths, we start with initializing the item
    def parse(self,response):
    item = NamenItem()

        item['url']= response.url
        item['field_name'] = response.xpath('//insert/xpath/here').getall()
        item['another_field_name'] = response.xpath('//insert/xpath/here/text()').getall()
        yield item
```

Changes in Scrapy 1.6.0

- Modern scrapy projects use `getall()` or `get()` methods
- Those methods have the same effect as `extract()` or `extract_first()` you may find in older tutorials
- All of them are correct, project managers do not plan to deprecate it
- More: `http://docs.scrapy.org/en/latest/topics/selectors.html#extract-and-extract-first`

Interactive mode of scrapy shell

- Scrapy provides an interactive shell where you can try and debug your code quickly without running the spider
- `scrapy shell /absolute/path/to/file`
- `scrapy shell ‘‘http://enter-your-site-here.net’’`
- To end the session type `ctrl-z` in Windows or `ctrl-d` in UNIX-like systems

Note: When using relative file paths be explicit and prepend them with `./` or `../` when applicable

Example of shell session

```
2018-03-06 17:33:07 [scrapy] INFO: Scrapy 1.1.0 started (bot: hiperbug)
2018-03-06 17:33:07 [scrapy] DEBUG: Telnet console listening on 127.0.0.1:6023
2018-03-06 17:33:07 [scrapy] INFO: Spider opened
2018-03-06 17:33:10 [scrapy] DEBUG: Crawled (200) <GET https://www.reddit.com/robots.txt> (referer: None)
2018-03-06 17:33:11 [scrapy] DEBUG: Crawled (200) <GET https://www.reddit.com/r/roguelikes/> (referer: None)
[s] Available Scrapy objects:
[s]   crawler    <scrapy.crawler.Crawler object at 0x7fd53b265cd0>
[s]   item       {}
[s]   request    <GET https://www.reddit.com/r/roguelikes/>
[s]   response   <200 https://www.reddit.com/r/roguelikes/>
[s]   settings   <scrapy.settings.Settings object at 0x7fd53b2658d0>
[s]   spider     <DefaultSpider 'default' at 0x7fd539cea2d0>
[s] Useful shortcuts:
[s]   shelp()           Shell help (print this help)
[s]   fetch(req_or_url) Fetch request (or URL) and update local objects
[s]   view(response)    View response in a browser
>>> response.xpath('//title/text()').getall()
[u'Roguelikes!']
>>> response.xpath('//title').getall()
[u'<title>Roguelikes!</title>']
```

Running the spider and saving the output

- You run the bot from the command line in the directory which contains your spiders
- To save the data you may use `-o` option and provide the name of the file
- Scrapy supports several formats of output files, here we will show how to save `.csv` file
- `scrapy crawl name_of_your_spider [-o resulting_file.csv]`
- `scrapy crawl myFirstSpider [-o resulting_file.csv]`

Procedure of scraping a single page

1. Find the url of your site
2. Investigate if this is a dynamically or statically generated page – there are different approaches to extract data!
3. Decide which framework/library you will use
4. Find the crucial areas and tokens you want to scrap and investigate the source code – if you use XPaths, try to write or copy them
5. Write the scraper
6. Run the scraper, check if the output is correct, debug the code
7. Repeat point 6 until you are satisfied (:
8. Analyze the data or leave it to your ordering party

Tips for writing code

- In Windows better use ‘ ‘ instead of ’
- No matter the operating system, decide whether you use tabs or spaces and be consistent
- Using XPaths or the HTTP tags does not mean that you are not allowed to use other algorithms or functions!
- Tidying data is always a bottleneck – decide what (and if) you can move to analysis stage
- Usually text preprocessing (removing newlines and unnecessary characters) does not significantly slow down the performance of our bot
- Use the loops and predicates wisely and be aware of your hardware limitations

Arbitrary page – Beautiful Soup

```
# Let us scrape an arbitrary Wikipedia page (text)
import requests
from bs4 import BeautifulSoup

url = 'https://en.wikipedia.org/wiki/Edvard_Munch'
r = requests.get(url)
html_doc = r.text
soup = BeautifulSoup(html_doc)
edvard_title = soup.title
print(edvard_title)
edvard_text = soup.get_text()
print(edvard_text)
```

Arbitrary page – Beautiful Soup

```
# Now we get hyperlinks from Wikipedia page (text)
import requests
from bs4 import BeautifulSoup

url = 'https://en.wikipedia.org/wiki/Edvard_Munch'
r = requests.get(url)
html_doc = r.text
soup = BeautifulSoup(html_doc)
print(soup.title)
tags = soup.find_all('a')
for link in tags:
    print(link.get('href'))
```

Arbitrary page – Scrapy

- Usually it is not very different from the example code we show earlier
- You just specify the content you want to retrieve (`item` class) and get it with XPaths and some additional coding for control flow (`parse()` function)
- You also provide a starting url in the respective field

Arbitrary page – wget

- *This is an example to boost your curiosity*
- Sometimes it is easier (but also significantly slower) to get the source code via `wget`
- And later parse it with core utils and shell programming

```
#!/bin/zsh

echo "user;date;package" >$2;

for x in $(cat $1); do
    wget -O example.html "https:insert-your-site-here.net/$x";
    grep "commented on"  <example.html | tr -d "t" > comments;
    sed -e "s|$|;$x|" -e 's| commented on |;|g' <comments >>$2;
    rm comments example.html;
done;
```

Scraping tables – easy and amazing part

- Scraping tables will be probably one of the most common situations you will use scraping for
- It can be both quite easy and quite though, amazing and terrifying (depending on the table)
- The easy part is that usually the code of table will be well structured and relatively easy to access
- The amazing part: you can scrap it in a few ways!

Scraping tables – though and terrifying part

- The though part appears if the table's fields differ among your crawled sited
- The terrifying part: you can scrap it in a few ways!
- ...However, for the tables which differ in the placing of the context or you are unable to specify how many entries the table has, you will not go without some basic programming
- ... And you may end with a monster like this: (*it is only part of the code...*)

Scraping tables – though and terrifying part

```
# has or not keywords
    if response.xpath('//*[@id="iinfo"]/tbody/tr[5]/td/a[contains(@class, "keyword")]/text()'):

        item['keyword'] = response.xpath('//*[@id="iinfo"]/tbody/tr[5]/td/a[contains(@class,"keyword")]/text()').re('[-\w+:#]+
        item['licenses'] = response.xpath('//*[@id="iinfo"]/tbody/tr[6]/td/span/text()').re('[-\w:+#]+')

        #has or not groups
        if response.xpath('//*[@id="iinfo"]/tbody/tr[7]/th/text()').re('[-\w+:#]+')[0] == "Groups:":
            item['groups'] =  response.xpath('//*[@id="iinfo"]/tbody/tr[7]/td/span/text()').re('[-\w+:#]+')

            #has or not conflicts and provides
            if response.xpath('//*[@id="iinfo"]/tbody/tr[8]/th/text()').re('[\w+:#]+')[0] == "Conflicts:":
                item['conflicts'] = response.xpath('//*[@id="iinfo"]/tbody/tr[8]/td/span/text()').re('[-\w+:#]+')
                if response.xpath('//*[@id="iinfo"]/tbody/tr[9]/th/text()').re('[\w:+#]+')[0] == "Provides:":
                item['provides'] = response.xpath('//*[@id="iinfo"]/tbody/tr[9]/td/span/text()').re('[-\w:+#]+')
                if response.xpath('//*[@id="iinfo"]/tbody/tr[10]/th/text()').re('[\w:+#]+')[0] == "Replaces:":
                    item['replaces'] = response.xpath('//*[@id="iinfo"]/tbody/tr[10]/td/span/text()').re('[\w:+#]+')
                    item['responsible'] = response.xpath('//*[@id="iinfo"]/tbody/tr[11]/td/text()').getall()
                    item['owner'] = response.xpath('//*[@id="iinfo"]/tbody/tr[13]/td/text()').getall()
                    item['votes'] = response.xpath('//*[@id="iinfo"]/tbody/tr[14]/td/text()').getall()
                    item['popularity'] = response.xpath('//*[@id="iinfo"]/tbody/tr[15]/td/text()').getall()
                    item['first'] = response.xpath('//*[@id="iinfo"]/tbody/tr[16]/td/text()').getall()
                    item['last_updated'] = response.xpath('//*[@id="iinfo"]/tbody/tr[17]/td/text()').getall()
                else:
                    item['replaces'] = "9999"
                    item['responsible'] = response.xpath('//*[@id="iinfo"]/tbody/tr[10]/td/text()').getall()
                    item['owner'] = response.xpath('//*[@id="iinfo"]/tbody/tr[12]/td/text()').getall()
                    item['votes'] = response.xpath('//*[@id="iinfo"]/tbody/tr[13]/td/text()').getall()
                    item['popularity'] = response.xpath('//*[@id="iinfo"]/tbody/tr[14]/td/text()').getall()
                    item['first'] = response.xpath('//*[@id="iinfo"]/tbody/tr[15]/td/text()').getall()
                    item['last_updated'] = response.xpath('//*[@id="iinfo"]/tbody/tr[16]/td/text()').getall()
                else:
                item['provides'] = "9999"
                if response.xpath('//*[@id="iinfo"]/tbody/tr[9]/th/text()').re('[\w:+#]+')[0] == "Replaces:":
                    item['replaces'] = response.xpath('//*[@id="iinfo"]/tbody/tr[9]/td/span/text()').re('[\w:+#]+')
                    item['responsible'] = response.xpath('//*[@id="iinfo"]/tbody/tr[10]/td/text()').getall()
                    item['owner'] = response.xpath('//*[@id="iinfo"]/tbody/tr[12]/td/text()').getall()
                    item['votes'] = response.xpath('//*[@id="iinfo"]/tbody/tr[13]/td/text()').getall()
```

Scraping tables – Beautiful Soup way

- The easy way to get the whole table (basic programming skills):

```
soup = BeautifulSoup(HTML)

# the first argument to find tells it what tag to search for
# the second you can pass a dict of attr->value pairs to filter
# results that match the first tag
table = soup.find( "table", {"title":"TheTitle"} )

rows=list()
for row in table.findAll("tr"):
   rows.append(row)

# now rows contains each tr in the table (as a BeautifulSoup object)
# and you can search them to pull out the times
```

Source: https://stackoverflow.com/questions/2935658/beautifulsoup-get-the-contents-of-a-specific-table

- The more complex (but pretty) way to get the whole table (functional programming involved):

```
html = urllib2.urlopen(url).read()
bs = BeautifulSoup(html)
table = bs.find(lambda tag: tag.name=='table' and tag.has_key('id') and tag['id']=="Table1")
rows = table.findAll(lambda tag: tag.name=='tr')
```

Source: https://stackoverflow.com/questions/2935658/beautifulsoup-get-the-contents-of-a-specific-table

Scraping tables – Beautiful Soup way #2

```
# another example
import requests
from bs4 import BeautifulSoup

web_url = requests.get('https://en.wikipedia.org/wiki/List_of_European_countries_by_area').text
soup = BeautifulSoup(web_url,'lxml')
print(soup.prettify())

table = soup.find('table',{'class':'wikitable sortable'})
table

# extract all the links within <a>
links = table.findAll('a')

# do the list of countries
for link in links: Countries.append(link.get('title'))
print(Countries)

# how about data frame?
import pandas as pd
df = pd.DataFrame()
df['Country'] = Countries
df
```

Scraping tables – the Scrapy way

- If the information is always stored in one determined place: copy XPath
- If you are going to crawl more pages: check if the tables contain the same fields
- If not: copy a few XPaths, investigate the changes
- If XPath contains non-deterministic parts, try rewrite it (*you know how*)
- Consider using loops and conditions if you cannot determine the number of the fields/rows
- Use `{}.format(var)` to your advantage! See the example:

```
def parse(self,response):
      i = 1
      while (response.xpath('//*[@id="some-id"]/table/tbody/tr[{}]/td[1]/text()'.format(i)).getall()):
            item['name']=response.xpath('//*[@id="some-id"]/table/tbody/tr[{}]/td[3]/a/text()'.format(i)).getall()
```

Sad conclusion

- I suppose that Beautiful Soup way may be more appealing to you so far...
- But at some point (and in more complex projects) you will probably end with using Scrapy
- Or anything else what was not covered during classes
- Mostly because of the performance issues and amount of coding to the efectiveness of bot ratio
- You are also not able to predict what you will be using in even five years from now
- So – learn the ideas and workarounds, not the exact code