Python For Data Science *Cheat Sheet*

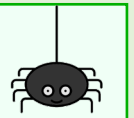
Scraping the web with Scrapy

1 Scraping principle

Fundamental scraping steps

Scraping is a succession of two steps :

1. Parsing : Extracting information from a web page
2. Crawling : going from pages to pages using URLs



We call spiders methods which combine both of these steps
The name comes from the algorithm purpose: crawling the web

HTML Inspection

HTML : **H**yper **T**ext **M**ark-up **L**anguage
It is the most basic web programming language used by every website

HTML is a succession of basic tags which contains the page content. Tag syntax is : *<tag> content </tag>*

```
<html>
  <head>
<title>Google</title>
<style>...</style>
  </head>
  <body>
    <div>...</div>
<center>...</center>
    <div>...</div>
  </body>
</html>
```

The most usual tags are the following ones:
*<html> <body> <title> <p> <a> *

Each tag can have different attributes

1. id : unique identifier for each html component
2. class : to define groups of tags
3. href : contains URL if tag is clickable
4. src : contains the location for a picture

Syntax : <tag-name attribute_name= 'attribute_info'> contents </tag-name>
Example : <div id='spoon_a', class= 'spoons'> Spoon A

Scraping with Python

Python tools


Scrapy


Requests


Beautiful Soup


Selenium

Code snippet

```
import requests
from scrapy.selector import Selector

# Define the URL
url = 'https://etomal.com/blog/'

# Create selector object
html = requests.get(url).content
sel = Selector(text=html)

# Getting the
css_locator = 'h1 ::text'
rep = sel.css(css_locator).extract_first()
print(rep)
```

2 Parsing HTML Files

XPath - Memento

`'/html/body/div'`	Each div into body into html
`'/html/body/div[2]'`	Each 2nd div into body's located into html
`'//div'`	Each div bloc
`'//div[1]'`	Each 1st div bloc in the html tree
`'//div[@class="my-class"]'`	Each div with the class « my-class »
`'//div[@id="my-id"]'`	Each div with id « my-id »
`'//div[1]/@class'`	Class attribute of each 1st div in the tree
`'//div[1]/text()'`	Text into the 1st div tag
`'//div[1]//text()'`	Text into the 1st div tag and its children tags
`'//div[contains(@class, "my-class")]'`	Each div with 'my-class' into it's class attribute

CSS Selectors - Memento

`'html > body > div'`	Each div into body into html
`'html > body > div:nth-of-type(2)'`	Each 2nd div into body's located into html
`'div'`	Each div bloc
`'div:nth-of-type(1)'`	Each 1st div bloc in the html tree
`'div.my-class'`	Each div that exactly match class my-class
`'div#my-id'`	Each div with id « my-id »
`'div:nth-of-type(1).class'`	Class attribute of each first div in the tree
`'div:nth-of-type(1)::text'`	Text into the first div tag
`'div:nth-of-type(1) ::text'`	Text into the 1st div tag and its children tags
`'div ::attr(href)'`	Content of the href attribute of each div tag

Extracting information

Using extractors on a selector results in a smaller (or empty) selector.
To get the information, an extraction is needed

1. `my_selector.extract()` List with all the concerned elements as strings
2. `my_selector.extract_first()` String with the first concerned element

⚠ Once a content has been extracted it isn't possible anymore to select anything into it

Method chaining

It is possible to combine multiple CSS or XPath selectors : the 4 above lines product the same

```
sel.xpath('html/body/div//a[2]/@href')
sel.xpath('html/body/div').xpath('//a[2]').xpath('/@href')
sel.css('html > body > div').xpath('//a[2]/@href')
sel.css('html > body > div a:nth-of-type(2) ::attr(href)')
```

Using navigator for exploration

Depending on the OS, these shortcuts open the "inspect" tab in the browser

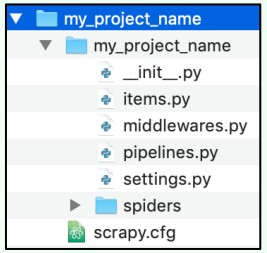

CTRL + SHIFT + I


OPTION + CMD + I

3 Crawling – Scrapy specific tools

Create a project

```
scrapy startproject my_project_name
```



To be seen as a package by python
Define structure to yield scraped items
Modify scrapy internal process
Process and store yielded items
Define the way scrapy will run the project

Crawling must be ran from here (with this file)

Spiders

The **name** is used for crawling. Each spider name of a project must be different.

The **start_requests** method is automatically called by Scrapy. The crawling process will start from this function.

```
# Create the Spider class
class CheatSheetSpider(scrapy.Spider):
  name = 'CSSpider'

  # start_requests method
  def start_requests( self ):
    my_url = 'https://example.com'
    yield scrapy.Request(url=my_url, callback=self.parse)

  # parse method
  def parse(self, response):
    # Create an extracted list of course author names
    example = response.css('div > p.sheet::text').extract()

    # Here we will just return the text from example
    yield example
```

The **parse** function is commonly used as a call-back which extract data from webpages and yield items or other pages to scrap.

Running a spider – Command Line Tools

1. Exploration

`scrapy shell www.google.com` : explore a webpage as a Scrapy response object

2. Crawling (execute a spider)

`scrapy run_spider spiderName` : in a stand-alone way (next to the spider file)

`scrapy crawl spiderName` : within a Scrapy project (next to the *scrapy.cfg* file)

3. Storing (saving scraped items)

`scrapy crawl spiderName -o my_file.json` Yield item in a *.jl file allows to store
`scrapy crawl spiderName -o my_file.jl` multiple scraping in the same file

Setting.py – Get data from protected websites

```
# Crawl responsibly by identifying yourself (and your website) on the user-agent
USER_AGENT = 'Mozilla/5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36' + /
    '(KHTML, like Gecko) Chrome/70.0.3538.110 Safari/537.36'

# Obey robots.txt rules
ROBOTSTXT_OBEY = False
```

Designed by Thibaud Lamothe | etomal.com | scraping-corner