# Golf Nation Wide Website

## First steps: virtual environment, scrapy, css selectors

1. create virtual environment by:
   1. in VS Code terminal, in designated folder (in our case its *Scrapy*):

>>> python –m venv venv

* 1. then activate with: >>> .\venv\Scripts\activate
  2. press ctrl-shift-p -> python: select interpreter -> select virtual env

1. install scrapy: >>> pip install scrapy
2. start the project >>> scrapy startproject postscrape
3. >>> cd postscrape
4. inside postscrape/postscrape/spiders folder create posts\_spider.py file
5. in the created file:

**import** scrapy

**class** PostsSpider(scrapy.Spider):

    name = 'posts'

    start\_urls = [

        'https//blog.scrapinhub.com/page/1/'

        'https//blog.scrapinhub.com/page/2/'

    ]

**def** parse(self, response):

        # get the page number from the links

        page = response.url.split('/')[-1]

        # create a filename

        filename = 'posts-%s.html' % page

        # create actual file, wb specifies write bianry

        with open(filename, 'wb'):

            f.write(response.body)

1. save. run with: scrapy crawl posts (whatever the name we specified in the file)

## Run commands in terminal

We can run similar commands in the terminal ie:

>>> scrapy shell <https://etc>

## Get title

We can get specific info using css selectors, ie (this will get us all titles on the page):

>>> response.css(‘title’)

This will return the first instance of the title

>>> response.css(‘title’).get()

## Get text without tags

To get just the text without tags:

>>> response.css(‘title::text’).get()

To get the specific h3 tag (in the example 2nd one):

>>> response.css(‘h3::text’)[1].get()

Get all method:

>>> response.css(‘h3::text).getall()

## Get link

Get only links from the body table:

response.css(‘td a::attr(href)’).getall()

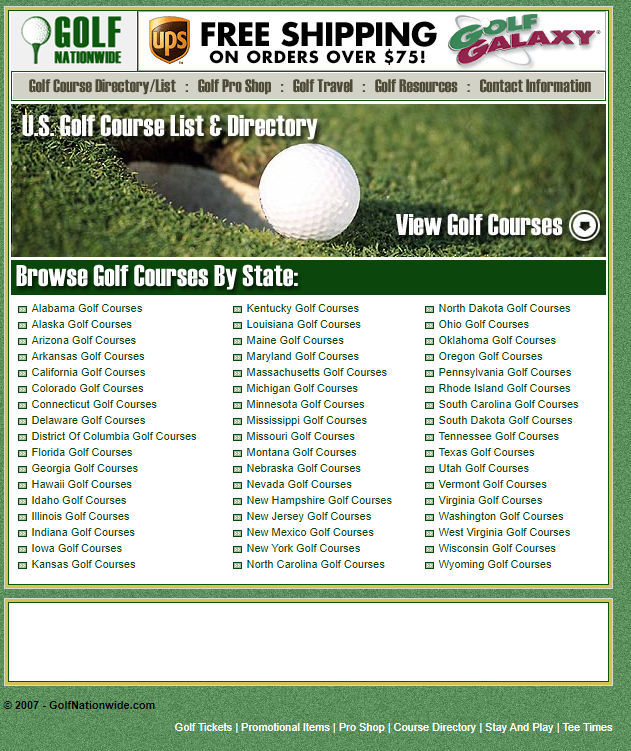
Final file to extract golf course links into json file:

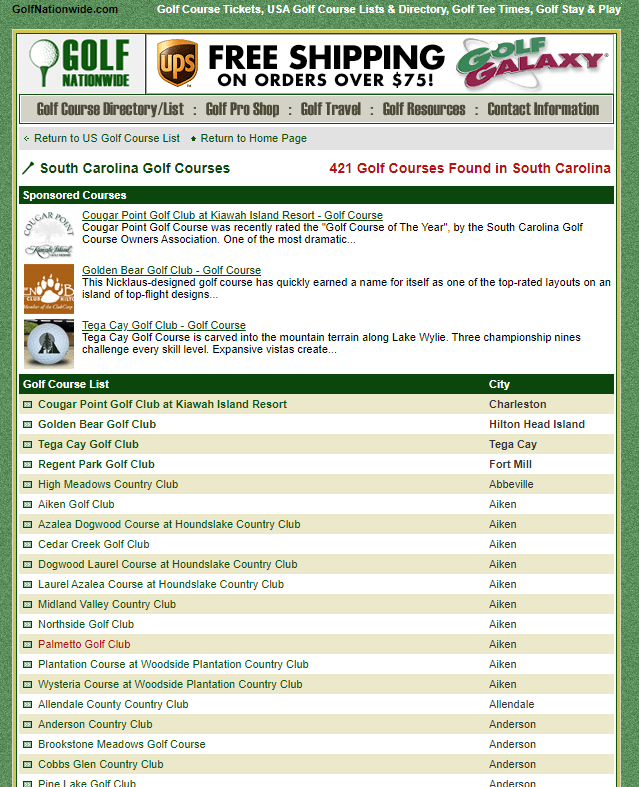
1. **import** scrapy
3. **class** PostsSpider(scrapy.Spider):
4. name = 'items'
6. start\_urls = [
7. 'http://www.golfnationwide.com/Default.aspx'
8. ]
10. **def** parse(self, response):
11. **for** item **in** response.css('td a'):
12. **yield**{
13. 'title': item.css('a::text').get(),
14. 'link': 'http://www.golfnationwide.com/' + item.css('a::attr(href)').get()
15. }

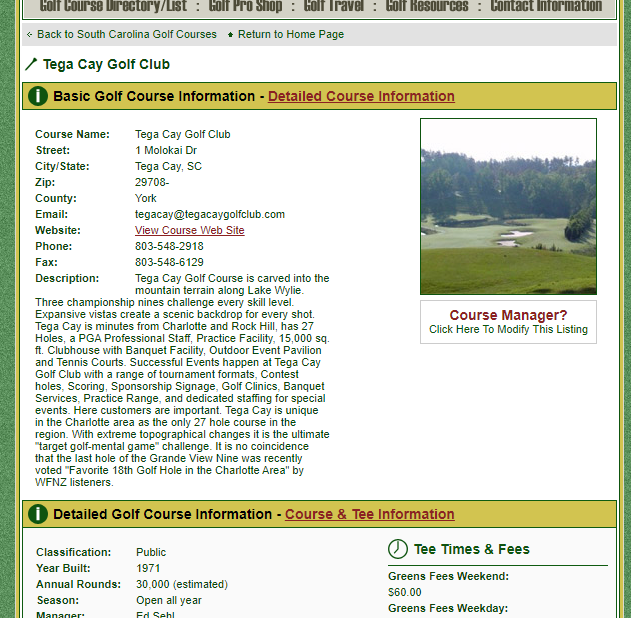
## Export output to csv or json

>>> scrapy crawl items –o items.json

The golf course website was scraped for course details with the following outline:







Final code:

**import** scrapy

**from** urllib.parse **import** urljoin

**class** GolfCourseSpider(scrapy.Spider):

    name = 'golf\_courses'

    start\_urls = [

        'http://www.golfnationwide.com/Default.aspx'

    ]

**def** parse(self, response):

        links = response.css('td a::attr(href)').extract()

**for** link **in** links:

            url = urljoin(response.url, link)

**yield** scrapy.Request(url, callback=self.parse\_second)

**def** parse\_second(self, response):

        links = response.css('td a::attr(href)').extract()

**for** link **in** links:

            url = urljoin(response.url, link)

**yield** scrapy.Request(url, callback=self.parse\_course)

**def** parse\_course(self, response):

        details = [tag.css('::text').get(default='') **for** tag **in** response.css('dl span')]

        website\_link = response.css('dl dd a::attr(href)').get()

**yield**{

            'Course Name': details[0],

            'Street': details[1],

            'City/State': details[2] +'/'+ details[3],

            'Zip': details[4] +'-'+ details[5],

            'County': details[6],

            'Email': details[7],

            'Website': website\_link,

            'Phone': details[8],

            'Fax': details[9],

            'Description': details[10],

            'Classification': details[11],

            'Year Built': details[12],

            'Annual Rounds': details[13],

            'Season': details[14],

            'Manager': details[15],

            'Club Pro': details[16],

            'Superintendant': details[17],

            'Guest Policy': details[18],

            'Designer': details[19],

            'Pro Shop Hours': details[20],

            'Dress Code': details[21],

            'Greens Fees Weekend': details[22],

            'Greens Fees Weekday': details[23],

            'Tee Time Reservation': details[24],

            'Online Reservation': details[25],

            'Earliest Tee Time': details[26],

            'Holes': details[27],

            'Greens': details[28],

            'Fairways': details[29],

            'Water Hazards': details[30],

            'Bunkers': details[31],

            'Metal Spikes': details[32],

            'Greens Aerated': details[33],

            'Overseeding': details[34],

            'Five Somes': details[35],

        }

# Golf Advisor Website

## Using Xpath instead of css selector:

Goal is to extract text “Lounge, Showers, Lockers ” from the following html:

**<div** class="CourseAbout-otherFacilities more"**>**

**<h3** class="CourseAbout-otherFacilities-title"**>**Available Facilities**</h3>** " Lounge, Showers, Lockers "

**</div>**

instead of using css selector:

response.css('.CourseAbout-foodAndBeverage.more::text').get()

I was suggested the following:

There are **three** text elements in your target div (matched by your CSS expression):

<div class="CourseAbout-otherFacilities more">FIRST<h3

<h3 class="CourseAbout-otherFacilities-title">SECOND</h3>

</h3>THIRD</div>

By using .get() you're telling Scrapy to return **first** match.

I recommend to use XPath expression here instead and match your element by text:

//h3[.="Available Facilities"]/following-sibling::text()[1]'

## Empty string while using strip() method

You will get error if you will try to strip empty string. To avoid it, you should set get() method to get(‘’), so if the string is missing we are passing on empty string. e.g.:

'Description': response.css('.CourseAbout-description p::text').get('').strip(),

## Using CrawlSpider and LinkExtractor

CrawlSpider class allows to include rules that include link selection: what links should be ignored and what links should be visited for scraping

importing libraries:

from scrapy.spiders import CrawlSpider, Rule

from scrapy.linkextractors import LinkExtractor

include rules. allow parameter specifies what links should be visited, deny – ignored

rules = [

        Rule(LinkExtractor(allow=('courses/'), deny=('page=')), callback='parse\_filter\_course', follow=True),

    ]

## Sitemaps

Turns out almost every site has a sitemap for our spiders to crawl. Sitemap is a dictionary of all urls the website has. Initializing sitemapspider:

class GolfCourseSpider(scrapy.spiders.SitemapSpider):

    name = 'golfadvisorcom'

    custom\_settings = {"DOWNLOAD\_DELAY": 0.25,

    }

    sitemap\_urls = [

        'https://www.golfadvisor.com/sitemap1.xml',

        'https://www.golfadvisor.com/sitemap2.xml'

    ]

### Finding sitemaps URLs

If you are as green as I am, you are probably asking where do I like for the sitemap urls? Usually you can find them by simply adding /sitemap.xml or /sitemap or sitemap\_index.xml to the website domain, like so:

<https://www.golfadvisor.com/sitemap.xml>

## Sitemap filters

After you initialized sitemap class and specified the sitemap urls, you want to crawl specific pages, in my case the pages with golf course info. Adding filters:

def sitemap\_filter(self, entries):

        for entry in entries:

            if  ('/courses/bvr/' or '/courses/badges/' or '/courses/feed/') in entry.get('loc'):

                continue

            if '/courses/' in entry.get('loc'):

                yield entry