Scrapy

Trabalhando com Scrapy – Parte 4

Scrapy

Seguindo links

Até agora nós raspamos material de páginas específicas do *site* http://quotes.toscrape.com.

Agora vamos ver como navegar para outras páginas seguindo os *links* que apontam para a próxima página.

Scrapy

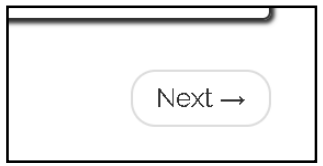
Seguindo links

Acesse quotes.toscrape.com, clique com o botão direito e selecione a opção para exibir o código-fonte, veja a parte destacada na imagem:

Este é o *link* para a próxima página.

Acionado quando clicamos no botão "*Next*".

```
207         </div>
208     </div>
209
210     <nav>
211         <ul class="pager">
212
213
214             <li class="next">
215                 <a href="/page/2/">Next <span aria-hidden="true">&rarr;</span></a>
216             </li>
217
218         </ul>
219     </nav>
220     </div>
221     <div class="col-md-4 tags-box">
222
223             <h2>Top Ten tags</h2>
224
```



Scrapy

Seguindo links

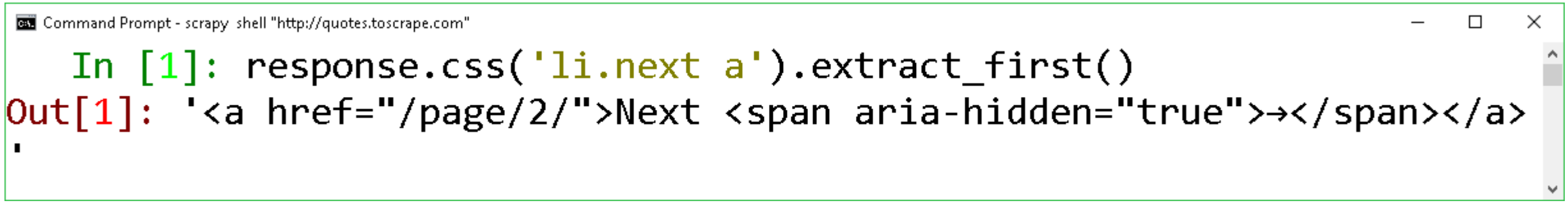
Vamos usar o *scrapy shell*.

$ scrapy shell “http://quotes.toscrape.com”

Dentro do *shell* digite o comando:

response.css(‘li.next a’).extract_first()

Com este comando retornamos a *tag* “a” inteira. Veja o resultado:

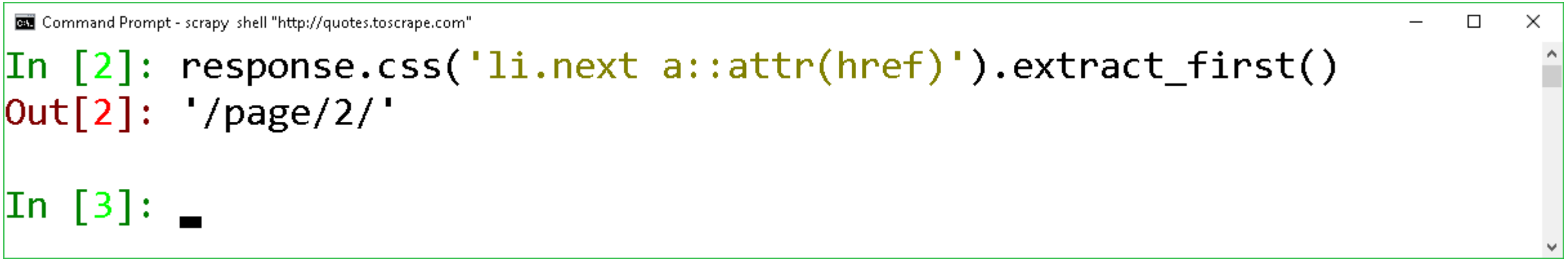


Scrapy

Seguindo links

Precisamos do atributo *href*. O *Scrapy* oferece suporte a uma extensão *CSS* que permite selecionar o conteúdo do atributo, veja:

$ response.css('li.next a::attr(href)').extract_first()



Scrapy

Seguindo links

Vamos agora implementar um *spider* para seguir recursivamente para as próximas páginas, extraindo os dados das mesmas (o sistema executará até a última página, a página 10).

Crie o projeto:

$ scrapy startproject aulascrapy

Crie o arquivo “citacoes.py” na pasta spiders.

Scrapy

Seguindo links

```
import scrapy
class QuotesSpider(scrapy.Spider):
    name = "citacoes"
    start_urls = ['http://quotes.toscrape.com/page/1/',]

    def parse(self, response):
        for quote in response.css('div.quote'):
            yield {
                'texto': quote.css('span.text::text').extract_first(),
                'autor': quote.css('small.author::text').extract_first(),
                'tags': quote.css('div.tags a.tag::text').extract(),
            }
        pagina = response.url.split("/")[-2]
        nome_arquivo = f'citacoes-{pagina}.html'
        with open(nome_arquivo, 'wb') as f:
            f.write(response.body)
        next_page = response.css('li.next a::attr(href)').extract_first()

        if next_page is not None:
            next_page = response.urljoin(next_page)
            yield scrapy.Request(next_page, callback=self.parse)
```

Scrapy

Seguindo links

Depois de extrair os dados, o método *parser* procura o *link* para a próxima página, cria uma *URL* completa usando o método *urljoin()*, uma vez que os *links* podem ser relativos e produz uma nova solicitação para a próxima página.

Podemos utilizar esta mesma ideia para navegar em *blogs*, fóruns ou outros *sites* com paginação.

Scrapy

Seguindo links

Podemos usar o *response.follow* como um atalho para as requisições.

Ao contrário de *scrapy.Request*, *response.follow* suporta *URLs* relativas diretamente, não sendo necessário utilizar o método *urljoin*.

Podemos passar também um *seletor* para o *response.follow* em vez de uma string.

Veja:

```
for href in response.css('li.next a::attr(href)'):
    yield response.follow(href, callback=self.parse)
```

Scrapy

Seguindo links

Para elementos <a> existe um atalho. O *response.follow* usa seu atributo *href* automaticamente, então, o código pode ser melhorado ainda mais.

```
for a in response.css('li.next a'):
    yield response.follow(a, callback=self.parse)
```

FIM