

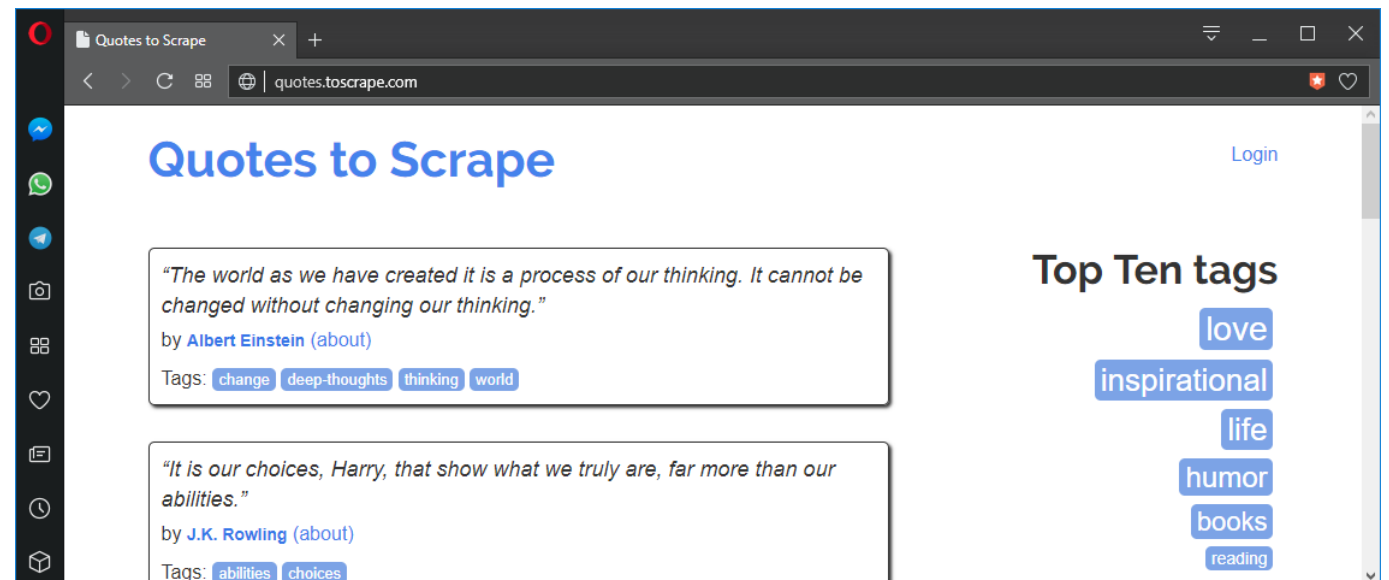
Scrapy

Trabalhando com Scrapy – Parte 1

Scrapy

Em nosso primeiro projeto vamos criar um *spider* para rastrear e extrair dados de um site usado como exemplo no tutorial do projeto do Scrapy que é o site *quotes.toscrape.com*.

Este site lista citações de autores famosos.



Scrapy

Antes de começar a raspagem de dados, teremos que configurar nosso projeto *Scrapy*.

A sintaxe para criar o projeto é:

```
$ scrapy startproject nome_do_projeto
```

Acesse o *prompt* de comando no *Windows* ou o Terminal do *Linux*, entre em uma pasta onde será criado o projeto e execute o seguinte comando:

```
$ scrapy startproject aulascrapy
```

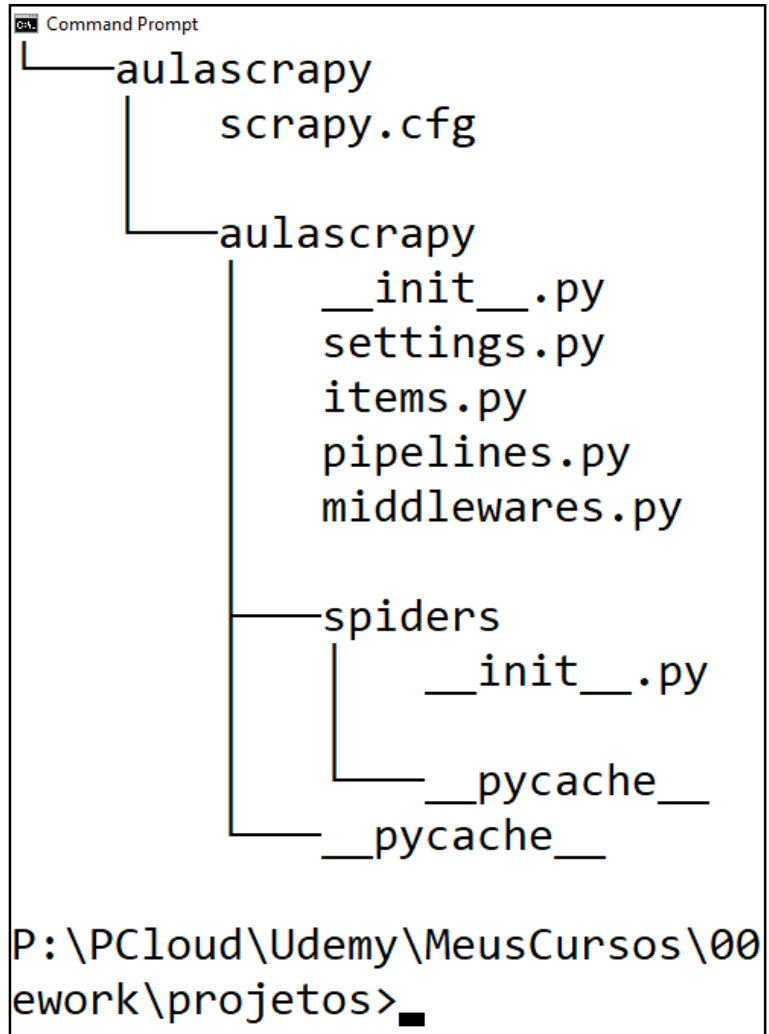
Scrapy

```
Command Prompt
P:\PCloud\Udemy\MeusCursos\002-PythonWebScraping\Secao11-UsandoOScrap
yFramework\projetos>scrapy startproject aulascrapy
New Scrapy project 'aulascrapy', using template directory 'c:\\evaldo
\\ferramentasdesenvolvimento\\python\\python36\\lib\\site-packages\\s
crapy\\templates\\project', created in:
    P:\PCloud\Udemy\MeusCursos\002-PythonWebScraping\Secao11-UsandoOS
crapyFramework\projetos\aulascrapy

You can start your first spider with:
    cd aulascrapy
    scrapy genspider example example.com

P:\PCloud\Udemy\MeusCursos\002-PythonWebScraping\Secao11-UsandoOScrap
yFramework\projetos>
```

Scrapy



scrapy.cfg # Arquivo de configuração de implantação (deploy)

aulascrapy # Módulo Python do projeto, você importará seu código à partir daqui

items.py # Arquivo de definição de itens do projeto

pipelines.py # Arquivo de pipeline do projeto

settings.py # Arquivo de configurações do projeto

spiders # Diretório onde serão criados os spiders

Scrapy

Criando nosso primeiro *Spider (Crawler)*

Spiders são classes que você define e que *Scrapy* usa para raspar informações de um site (ou um grupo de sites). Eles devem ser subclasses de *scrapy.Spider* e definir as solicitações iniciais que devem ser feitas, opcionalmente, como seguir links nas páginas e como analisar o conteúdo da página baixada para extrair dados.

Scrapy

Este é o arquivo de nosso primeiro *spider*.

```
import scrapy

class SpiderCitacoes(scrapy.Spider):
    name = "citacoes"

    def start_requests(self):
        urls = [
            'http://quotes.toscrape.com/page/1/',
            'http://quotes.toscrape.com/page/2/',
        ]
        for url in urls:
            yield scrapy.Request(url=url, callback=self.parse)

    def parse(self, resposta):
        pagina = resposta.url.split("/")[-2]
        nome_arquivo = f'citacoes-{pagina}.html'
        with open(nome_arquivo, 'wb') as f:
            f.write(resposta.body)
        self.log(f'Arquivo salvo {nome_arquivo}')
```

Salve o mesmo na pasta
aulascrapy/spiders
com o nome
citacoes_spider.py

Scrapy

Como podemos observar, nosso *spider* é uma subclasse de *scrapy.Spider* e define alguns atributos e métodos:

name: Identifica o *Spider*. Ele deve ser exclusivo dentro de um projeto, ou seja, você não pode definir o mesmo nome para diferentes *spiders*.

start_requests(): deve retornar um iterável de *Requests* (Você pode retornar uma lista de *Requests* ou escrever uma função geradora/*generator*). Os *requests* (as requisições) subsequentes serão gerados sucessivamente à partir desses *requests* iniciais.

Scrapy

parse(): Um método que será chamado para lidar com a resposta para cada um dos pedidos feitos. O parâmetro de resposta é uma instância de *TextResponse* que armazena o conteúdo da página e possui outros métodos úteis para lidar com ele.

O método *parse()* geralmente analisa a resposta, extraíndo os dados raspados como dicionários e também encontrando novas *URLs* a seguir e criando novas requisições (*Request*) deles.

Scrapy

yield:

Retorna um objeto *generator*.
O sistema retorna ao chamador
após encontrar um *yield*,
dando sequência em seguida
até o próximo *yield*.

```
# yield retorna um objeto generator
def teste1():
    yield

a = teste1()
print(a)
# <generator object teste1 at 0x000001542ECF68E0>

# Podemos criar um objeto do tipo teste2 e iterar sobre o mesmo
def teste2():
    yield 1
    yield 2
    yield 3
    yield 4

a = teste2()

for x in a:
    print(x)
    # 1
    # 2
    # 3
    # 4

def teste3():
    yield 10, 20, 30 # Retorna uma tupla

a = teste3()

print(next(a))
# (10, 20, 30)
```

Scrapy

Executando nosso spider

A sintaxe para executar o spider é:

```
$ scrapy crawl nome_do_spider
```

Para executar o spider, acesse a pasta de nível superior do projeto (onde está o arquivo *scrapy.cfg*) e execute:

```
$ scrapy crawl citacoes
```

Este comando executa o *spider* denominado “citações” que definimos em:

```
name = “citacoes”
```

Scrapy






Serão enviados alguns pedidos para o domínio *quotes.toscrape.com*.

Ao executar você terá como resultado algo como isto:

```
scrapy crawl citacoes
2017-12-02 17:30:07 [scrapy.utils.log] INFO: Scrapy 1.4.0 started (bot: aulascrapy)
..... (parte omitida)
2017-12-02 17:30:07 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.0.0.1:6023
2017-12-02 17:30:08 [scrapy.core.engine] DEBUG: Crawled (404) <GET http://quotes.toscrape.com/robots.txt> (referer: None)
2017-12-02 17:30:08 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrape.com/page/1/> (referer: None)
2017-12-02 17:30:08 [citacoes] DEBUG: Arquivo salvo citacoes-1.html
2017-12-02 17:30:08 [scrapy.core.engine] DEBUG: Crawled (200) <GET http://quotes.toscrape.com/page/2/> (referer: None)
2017-12-02 17:30:08 [citacoes] DEBUG: Arquivo salvo citacoes-2.html
2017-12-02 17:30:08 [scrapy.core.engine] INFO: Closing spider (finished)
..... (parte omitida)
'start_time': datetime.datetime(2017, 12, 2, 19, 30, 7, 521489)}
2017-12-02 17:30:08 [scrapy.core.engine] INFO: Spider closed (finished)
```

Scrapy

Você irá observar que foram criados dois arquivos na pasta onde executamos o comando que são “citacoes-1.html” e “citacoes-2.html”.

 .idea	02-Dec-17 2:40 PM	File folder	
 aulascrapy	02-Dec-17 2:13 PM	File folder	
 citacoes-1.html	02-Dec-17 5:30 PM	Opera Web Docu...	11 KB
 citacoes-2.html	02-Dec-17 5:30 PM	Opera Web Docu...	14 KB
 scrapy.cfg	02-Dec-17 2:13 PM	CFG File	1 KB

Scrapy

O que aconteceu afinal de contas?

O *Scrapy* agendou objetos *scrapy.Request* retornados pelo método *start_requests* do *spider*.

Ao receber uma resposta para cada um, instanciou os objetos de resposta e chamou o método de chamada associado ao *Request* (o método *parse*, definido como parâmetro *call-back* do método *Request*) passando sua resposta como argumento.

Scrapy

Se você colocar prints como segue:

```
def start_requests(self):  
    print("Executando start_requests")
```

```
def parse(self, resposta):  
    print("Executando parse")
```

```
for url in urls:  
    print("Percorrendo o loop")  
    yield scrapy.Request(url=url, callback=self.parse)
```

Terá o seguinte resultado:

Executando start_requests

Percorrendo o loop

Percorrendo o loop

Executando parse

Executando parse

Scrapy

Uma alternativa ao *start_requests*:

Não é necessário implementar o método *start_requests*, podemos simplesmente definir um atributo de classe *start_urls* com uma lista de endereços (*URLs*). Esta lista será usada pela implementação padrão de *start_requests()* para criar os pedidos iniciais para o *spider*.

Scrapy

E quanto à chamada ao método *parse()*?

O método *parse()* será chamado para lidar com cada um dos pedidos para os endereços (URLs), mesmo que não informamos explicitamente ao *Scrapy*.

Isso acontece porque *parse()* é o método de retorno padrão para o *Scrapy*, que é chamado para requisições sem um retorno de chamada (*callback*) definido explicitamente.

Scrapy

Criando nosso segundo *spider*:

Execute com:

\$ scrapy crawl citacoes2

Veja que os dois arquivos
foram gerados
corretamente.

```
import scrapy

class SpiderCitacoes(scrapy.Spider):
    name = "citacoes2"

    start_urls = [
        'http://quotes.toscrape.com/page/1/',
        'http://quotes.toscrape.com/page/2/',
    ]

    def parse(self, resposta):
        pagina = resposta.url.split("/")[-2]
        nome_arquivo = f'citacoes-{pagina}.html'
        with open(nome_arquivo, 'wb') as f:
            f.write(resposta.body)

        self.log(f'Arquivo salvo {nome_arquivo}')
```

FIM