


Não digite: ">>>"

Caso tenha duvida assista a aula ao vivo assim ajuda a replicar o código de sua maneira funcional.

Lendo uma página

Inicie um projeto criando um diretório chamado quotes, dentro do diretório, crie um arquivo que irá conter o código fonte do nosso projeto. O arquivo se chamará `spider_quote.py`.

 Não utilizaremos a estrutura completa padrão de um projeto python em prol da simplicidade e restrição do domínio do problema.

Estrutura no final fica assim:

```
├── spider_quote.py
```

0 directories, 1 file

Como todo início de projeto em python, vamos iniciar criando um ambiente virtual. Para isso, utilizamos o comando:

```
$ python3 -m venv .venv
```

Em seguida, ativamos o ambiente virtual com o comando:

```
$ source .venv/bin/activate
```

Com o ambiente virtual ativado, vamos instalar as três dependências que precisaremos hoje. `requests` para a realização de requisições *web*, `parsel` para analisar conteúdos HTML e `pymongo` para persistirmos nossos dados.

```
$ python3 -m pip install requests
```

```
$ python3 -m pip install parsel
```

```
$ python3 -m pip install pymongo
```

Já navegamos pela página através de um navegador, agora vamos utilizar o python para fazer uma requisição *web* e vamos dar uma olhada em seu conteúdo.

```
>>> import requests
>>>
>>>
>>> response = requests.get("http://quotes.toscrape.com/")
>>> print(response.status_code) # Código de status
>>> print(response.headers["Content-Type"]) # Conteúdo no formato
html
>>> # Conteúdo recebido da requisição
>>> print(response.text)
```

Uma requisição é feita utilizando o método `get`, que tem como único parâmetro obrigatório a URL. Vimos que por padrão a biblioteca adiciona alguns cabeçalhos à requisição.

O código de status `200` indica que a requisição foi bem sucedida. Vamos então criar uma função para recuperar o conteúdo de uma página *web*, escrevendo também os seus testes automatizados.

`spider_quote.py`

```
import requests

def fetch_content(url):
    return requests.get(url).text
```

Porém, só estamos considerando que nossa resposta será bem sucedida. Mas e se ocorrer um erro durante a requisição? Não teremos, em nosso teste, nada definido para retornar.

Podemos verificar se uma requisição foi bem sucedida através do método `raise_for_status` que irá lançar uma exceção caso a requisição falhe. Com isso, podemos capturá-la e tomar alguma medida.

```
>>> import requests
>>>
>>>
>>> response = requests.get("http://httpbin.org/status/404")
>>> print(response.status_code) # Código de status
>>> # Quando o código de status é diferente de 200 lança uma exceção
>>> response.raise_for_status()
```

Vamos modificar o código do arquivo `spider_quote.py` para implementarmos um tratamento de exceção básico.

spider_quote.py

```
import requests

def fetch_content(url):
    try:
        response = requests.get(url)
        response.raise_for_status()
    except requests.HTTPError:
        return ""
    else:
        return response.text
```

spider_quote.py

```
# import requests
```

```
#def fetch_content(url):
#    try:
#        response = requests.get(url)
#        response.raise_for_status()
#    except requests.HTTPError:
#        return ""
#    else:
#        return response.text
```

```
print("conteúdo correto",
      fetch_content("http://quotes.toscrape.com/"))
print("vazio:", fetch_content("http://httpbin.org/status/404"))
```

Adicionamos a verificação se a requisição foi bem sucedida através da função `raise_for_status()`. Além disso, adicionamos a ação para caso de falha, através do `except`, que será o retorno de um conteúdo vazio. Essa nossa implementação é como uma condicional que terá seu fluxo de execução definido pelo lançamento ou não da exceção.

Nossa função para recuperar conteúdos da *web* está pronta. Contudo, como aprendemos, é sempre legal desconsiderarmos uma requisição que está demorando demais. Para isso, vamos adicionar um tempo de limite (`timeout`) para nossa requisição. Assim, garantimos que nosso "raspador" não fique esperando uma resposta.

spider_quote.py

```
# import requests
```

```
def fetch_content(url, timeout=1):
    # try:
    #     response = requests.get(url, timeout=timeout)
    #     response.raise_for_status()
    except (requests.HTTPError, requests.ReadTimeout):
    #     return ""
```

```
# else:
#     return response.text
```

Quando uma requisição lançar um erro por *timeout*, devemos retornar o conteúdo vazio.

spider_quote.py

```
# ...
```

```
print("quando o servidor demora mais de 1 segundo para responder",
      fetch_content("http://httpbin.org/delay/3"))
print("quando o servidor demora menos de 1 segundo para responder",
      fetch_content("http://httpbin.org/get"))
```

Analizando respostas

Agora, vamos explorar o conteúdo que obtemos da resposta da raspagem que estamos fazendo.

```
>>> import parse
>>> from spider_quote import fetch_content
>>>
>>> page_content = fetch_content("http://quotes.toscrape.com/")
>>> # Criamos um seletor a partir do texto extraído da página.
>>> selector = parse.Selector(page_content)
>>>
>>> # Podemos selecionar todas as citações utilizando.
>>> # O retorno da função `css` são seletores.
>>> quotes = selector.css("div.quote")
>>> print(quotes)
```

Reparem como estamos usando o console interativo aqui para testar a sintaxe de algo que queremos colocar no nosso código! Faça isso com frequência. *Selector* são objetos que lhe permitem selecionar partes (ou inteiro) de um determinado texto HTML/XML. A partir de um seletor, podemos selecionar partes mais específicas, utilizando seletores CSS ou XPATH.

Podemos utilizar os métodos `get` e `getall` para recuperar informações obtidas a partir dos seletores. Quando não especificamos atributos ou o texto, estes métodos nos retornam, de forma textual, a tag ou as tags encontradas.

```
>>> # Podemos recuperar os autores de cada citação
>>> # Note que, aqui, temos o conteúdo retornado em forma de tag
(<small>...</small>)
```

```
>>> selector.css("div.quote small.author").getall()
>>>
>>> # Podemos combinar seletores CSS com uma sintaxe similar a notação
XPATh para extrair somente textos
>>> selector.css("div.quote small.author::text").getall()
```

Cada citação é composta por um texto, nome do seu autor e várias tags relacionadas à citação.

Vamos adicionar uma função no nosso arquivo `spider_quote.py` para extrair uma citação:

`spider_quote.py`

```
import parsel
# import requests

# ...

def extract_quotes(text):
    selector = parsel.Selector(text)
    quotes = []
    return quotes
```

Vimos que as citações podem ser obtidas através do seletor `div.quote` e a partir dela podemos recuperar todas as outras informações. Vamos uma a uma, inspecionando, descobrindo seu seletor, adicionando o código e em seguida vamos fazer o teste da implementação.

Agora que vimos onde está localizado o texto da citação, vamos voltar no código e pegá-lo!

`spider_quote.py`

```
# import parsel
# import requests

# ...
```

```
def extract_quotes(text):
    # selector = parsel.Selector(text)
    # quotes = []
    # Para cada uma das citações
    for quote in selector.css("div.quote"):
        # Recuperamos o texto contigo na tag span de classe text
        text = quote.css("span.text::text").get()
        quotes.append({
            "text": text,
        })
    # return quotes
```

spider_quote.py

```
# ...
```

```
page_content = fetch_content("https://quotes.toscrape.com/")
print(extract_quotes(page_content))
```

O próximo campo é o autor da citação.

Vamos alterar a função `extract_quotes` para adicionar o código de extração do "texto" do nome do autor.

spider_quote.py

```
# import parsel
# import requests
```

```
# ...
```

```
def extract_quotes(text):
    # selector = parsel.Selector(text)
    # quotes = []
    # Para cada uma das citações
    # for quote in selector.css("div.quote"):
        # Recuperamos o texto contigo na tag span de classe text
        # text = quote.css("span.text::text").get()
        author = quote.css("small.author::text").get()
        quotes.append({
            # "text": text,
            "author": author,
        })
    # return quotes
```

Por fim, vamos repetir o mesmo processo para extrair as tags de cada citação.

Agora que vimos onde está localizado o texto da citação, vamos voltar no código e pegá-lo!

spider_quote.py

```
# import parsel
# import requests

# ...

def extract_quotes(text):
    # selector = parsel.Selector(text)
    # quotes = []
    # Para cada uma das citações
    # for quote in selector.css("div.quote"):
    #     # Recuperamos o texto contigo na tag span de classe text
    #     # text = quote.css("span.text::text").get()
    #     # author = quote.css("small.author::text").get()

    # Estamos usando `getall`, pois agora o conteúdo será mais de
um elemento
    tags = quote.css("a.tag::text").getall()
    quotes.append({
        # "text": text,
        # "author": author,
        "tags": tags,
    })
    # return quotes
```

Vamos agora executar um código no terminal para demonstrarmos que as 10 citações estão sendo recuperadas, contendo autor, tag e o texto da citação. Para isso, vamos abrir o REPL do python novamente e escrever o código para vermos.

A opção -i irá carregar tudo que foi definido em um módulo, como funções, variáveis etc e em seguida irá abrir o terminal interativo. Ela permite que você use todos os recursos de um módulo no REPL. Com isso, conseguimos verificar o que está acontecendo naquele módulo de perto.

Observe que -i faz a função do import para no terminal.

```
>>> page_content = fetch_content("http://quotes.toscrape.com")
>>> quotes = extract_quotes(page_content)
>>> print(quotes)
```

Já estamos extraindo as citações de uma página, porém temos 100 citações neste site e seu conteúdo está paginado. Ou seja, estamos extraindo as citações apenas da primeira página do site.

Vamos fazer a navegação página a página para extrair as citações de cada uma delas.

Precisamos fazer a navegação de página em página, mas como podemos descobrir a URL para a próxima página, se existir?

Na primeira página do `http://quotes.toscrape.com`, vamos inspecionar o botão next. Percebam que seu seletor é uma *tag* `li` com classe `next`. Dentro, temos um elemento âncora (*a*) em que seu atributo `href` contém o caminho para a próxima página.

Repare que a referência para a próxima página é algo como `/page/2` por isso no código vamos concatenar com a URL base.

Vamos escrever então uma função que recupere uma página de conteúdo, extraia suas citações, busque uma nova página e continue até não restar mais páginas. Para isso vamos criar uma nova função chamada `get_all_quotes` dentro do arquivo `spider_quote.py`.

`spider_quote.py`

```
# ...
```

```
def get_all_quotes():
    base_url = "http://quotes.toscrape.com"
    next_page = "/"
    quotes = []
    while next_page:
        content = fetch_content(base_url + next_page)
        # Extraí e adiciona as novas citações à lista
        quotes.extend(extract_quotes(content))

        next_page = (
            parse1.Selector(content).css("li.next >
a::attr(href)").get()
        )
    return quotes
```

`spider_quote.py`

```
# ...
```

```
print(get_all_quotes())
```


Extração de conteúdos a partir de conteúdos

Ainda precisamos extrair as informações dos autores das citações. Estas informações podem ser obtidas a partir do conteúdo das páginas de citação. Será necessário buscar, no conteúdo, o caminho para ir à página de detalhes sobre um autor.

Vamos entrar na primeira página <http://quotes.toscrape.com/>. Em seguida, vamos navegar até um autor e vamos ver as informações disponíveis que vamos extrair.

A página de detalhes de um autor possui seu nome, sua data de nascimento, local de nascimento e uma descrição.

Vamos escrever o código que realiza a extração de um autor a partir de uma página de detalhe:

spider_quote.py

#...

```
def extract_author(text):
    selector = parsel.Selector(text)
    return {
        "name": selector.css("h3.author-title::text").get(),
        "birth-date": selector.css("span.author-born-
date::text").get(),
        "birth-location": selector.css(
            "span.author-born-location::text"
        ).get(),
        "description": selector.css("div.author-
description::text").get(),
    }
```

spider_quote.py

...

```
page_content =
fetch_content("https://quotes.toscrape.com/author/Marilyn-Monroe/")
author = extract_author(page_content)
print(author)
```

Para recuperarmos as URLs de todos os autores presentes em uma página, recuperamos seu conteúdo, fazemos uma busca utilizando seletores CSS e em seguida podemos percorrer esta lista, fazendo uma nova requisição à página de detalhes e extraíndo sua informação.

Vamos então criar uma nova função para extrair todos os autores do site!

spider_quote.py

```
# ...
```

```
def get_all_authors():
    authors = []
    base_url = "http://quotes.toscrape.com"
    content = fetch_content(base_url)
    selector = parsel.Selector(content)
    authors_url = selector.css("div.quote > span >
a::attr(href)").getall()

    for url in authors_url:
        detail_content = fetch_content(base_url + url, timeout=2)
        authors.append(extract_author(detail_content))

    return authors
```

Vamos acrescentar, também, a paginação à nossa função:

spider_quote.py

```
# ...
```

```
# def get_all_authors():
#     authors = []
#     base_url = "http://quotes.toscrape.com"
#     next_page = "/"
#     while next_page:
#         content = fetch_content(base_url + next_page)
#         selector = parsel.Selector(content)
#         authors_url = selector.css("div.quote > span >
a::attr(href)").getall()

#         for url in authors_url:
#             detail_content = fetch_content(base_url + url,
timeout=2)
#             authors.append(extract_author(detail_content))

#         next_page = (
#             selector.css("li.next > a::attr(href)").get()
#         )

#     return authors
```

spider_quote.py

```
# ...
```

```
print(get_all_authors())
```