8 - Prática: Web Scraping com Python p/ Ciência de Dados (II)

- HTML

  Através da analise das páginas da Web, será possível identificar os campo que desejamos extrair, utilizando o formato HTML definido em página. Nele, conseguimos identificar as DIVs das quais dividem os parágrafos, informações e afins.

- PYTHON

  - Básico

    Para o python, é necessário as bibliotecas beautifulsoup4 e a lxml.

    Havendo as bibliotecas instaladas, é possível iniciar importando a biblioteca BeautifulSoup:

    ```
    from bs4 import BeautifulSoup
    ```

    A partir disso, será aberto o arquivo referente a esse Webscraping:

    ```
    with open('Nome_arquivo.html', 'r') as html_file:
    ```

    O with open serve para abertura de arquivos;

    Já o ‘r’ indicado dentro do mesmo serve para indicar qual método será utilizado, leitura somente, escrita somente ou leitura e escrita.

    Ao final, é indicado o alias para a variável que irá assumir esse arquivo.

    Após indicar variavel, devemos indicar as demais, que irão receber os conteúdos dos arquivos:

    ```
    content = html_file.read()
    ```

    Isto fará com que todo conteúdo do html seja printado em tela.

Porém, se desejamos somente uma informação especifica da estrutura do nosso HTML, devemos utilizar outro parâmetro:

```
soup = BeautifulSoup(content, 'lxml')
```

Isto #fará com que consigamos definir as estruturas e os conteúdos do nosso html.

```
# deste modo irá puxar somente o primeiro valor
courses_html_tags = soup.find('h5')

# deste modo trará todas
courses_html_tags2 = soup.find_all('h5')
```

Feito isto, basta utilizar tanto o que geramos a partir da variável courses_html_tags uma estrutura de repetição, que poderá printar ou inserir essas informações em algum local:

```
for course in courses_html_tags2:
    print(course.text)
```

Printando o valor Text, teremos o exato conteúdo do texto da tag h5 definida.

▼ Valores específicos:

Como existe um parâmetro em python chamado class, quando vamos definir um web scraping que pegará esses dados, definimos ‘class_’ ,desse modo evitando erros

Para pegar valores específicos, foram utilizados seguinte parâmetros:

```
course_cards = soup.find_all('div', class_='card' )
```

Isto, faz com que consigamos pegar somente o conteúdo das div, e não todo conteúdo da página.

```
for course in course_cards:
    course_name = course.h5.text
```

```
    course_price = course.a.text.split()[-1]
```

Desde modo, conseguimos definir, tanto o nome do conteúdo encontrado em h5 quanto o preço dele, encontrado em a.

▼ Para sites

Para iniciar é necessário que tenha instalado a biblioteca requests, esta que é a fundamental para as solicitações Web:

```
from bs4 import BeautifulSoup
import request
```

Para solicitação no site, é possível utilizar o seguinte parâmetro:

```
html_text = requests.get('https://www.meusdividendos.co
```

O request é utilizado para que a consulta do corpo de um site seja encaminhado para uma variável do python.

Para isto, também utilizei a variável Headers, da qual detém do perfil base de um navegador de sites. Realizei isto, pois, nas consultas que tentei via alguns sites, os mesmos impossibilitavam acesso. Com esse parâmetro no request.get pude consultar sem problemas.

```
headers = {
    'User-Agent': 'Mozilla/5.0 (Windows NT 10.0; Win64;
}
```

É importante dentro da consulta, estar ligado a tag pai das demais, esta que ira permitir buscar as demais

Após o request, utilizamos da biblioteca BeautifulSoup para pegar esse html, e estruturar ele de modo que possamos acessa-lo posteriormente:

```
soup = BeautifulSoup(html_text, 'lxml')
```

Após isto, podemos pegar na estrutura desse HTML utilizando a opção de inspecionar página, e ao encontrar a flag que é a “pai” dos registros

que precisamos:

```
fundos = soup.find('tbody')
```

Ao pegar ela, realizei definição de 2 listas:

```
nomes_fii = []
valores_fii = []
```

Estas que utilizarei posteriormente para anexar os valores que estou extraindo deste site.

Na sequência, é trazido alguns processos mais específicos, utilizando o módulo “find”.

O find, basicamente busca uma informação contida na estrutura desse HTML, como exemplo podemos ter o caso acima do qual atribui para variável fundos o conteúdo de tbody.

Logo, é como se o texto de formação do HTML estivesse na variável tbody.

Havendo esse processo, é possível buscarmos mais um campo dentro dele:

```
nome = fundos.find('div', class_='label label-default')
```

Esse passo, realiza a consulta encima da DIV e busca o conteúdo com a class_ = ‘label label-default’.

Este processo faz somente com que o primeiro registro entre para essa variável, para realizarmos processo de inserção como lista(Ou print) geramos uma estrutura de repetição, da qual enquanto houver valores disponíveis para consulta, ele permanecerá consultando:

```
for fundo in fundos:

  nome = fundo.find('div', class_='label label-default'

  valor = fundo.find('span' , class_='moeda')

  if valor:
```

```
    valor = fundo.find('span', class_='moeda').text
  valores_fii.append(valor)
  nomes_fii.append(nome)
```

Existe outra variável junto desta, porém o conteúdo principal do código é realizar a busca.
Ao realizar a busca do nome, ele passa esses valor para outra variável que é a lista mencionada anteriormente, acrescendo esse valor a lista através do módulo append.

A partir disso, crio 2 listas, das quais, posso estar gerando um dataframe posteriormente.

Alguns problemas que encontrei neste processo, foram que nem todos os sites detém de estruturas de html que possibilitam a consulta das informações, alguns ambientes contém de formações em java script que barram as consultas, ou tornam ocultos os dados.

No video é indicado a continuação destes testes. Realizando um processo de construção de função, esta que é construida com uma função if **__name__** == **__main__**:

E contém um while que faz função de timer para próxima vez que irá rodar esta função.

Ficando do seguinte modo:

```
if __name__ == '__main__':
    while True:
            consulta()
            time_wait = 10
            print(f'Waiting {time_wait} minutes...')
            time.sleep(time_wait * 60)
```

Deste modo, definimos um temporizado especifico.

No video também foi definido uma variavel, para que a mesma seja um filtro para conteúdo da vaga.

Ela é gerada a partir de um input no inicio do código:

```
unfamiliar_skill = input('>')
```

Porém, acredito que não seja tão valida para meu uso.

O mesmo também indica a possibilidade de gravar através uma pasta estes dados. Estes que são gerados através do comando:

```
with open(f'posts/{index}.txt', 'w') as f:
# indica a abertura do arquivo e o modo escrita nele
        f.write(f'texto: {variavel}\n')
        f.write(f'texto: {variavel}\n')
        f.write(f'texto: {variavel}\n')
print(f'File Saved: {index}')
```

Achei super interessante a ideia do uso do web Scraping, acredito que só não fui tão feliz, por conta dos ambientes que gostaria de pegar informações terem estes bloqueios indicados anteriormente.