**Scraping com Python – Coleta De Dados Na Web – Anotações**

1. **Aula 1 – Nosso Primeiro Scraping:**
   1. Vamos fazer o scraping, ou seja, pegar as informações automaticamente de um site fictício da alura que foi desenvolvido para isso:
      1. <https://alura-site-scraping.herokuapp.com/index.php>.
      2. Para fazer o scraping utilizaremos 2 bibliotecas: urllib e bs4. Mais precisamente, para esse primeiro contato, utilizaremos as funções urlopen e BeautifulSoup, de cada uma respectivamente:



* + 1. Para abrir a url precisamos usar a função que importamos passando a url como parâmetro e em seguida, para ler o html da página passada, usar o.read() e tendo como resposta:





* + 1. Para organizar melhor e facilitar a visualização, utilizamos a BeautifulSoup(html, ‘html.parser’), passando a leitura do html e um parâmetro para organizando:





* + 1. Como queremos pegar algo em particular, clicamos com o botão direito em cima da informação que queremos obter e clicamos em “inspecionar elemento” abrindo uma aba que nos mostra exatamente qual a tag html em que essa informação está e suas classes e id’s. Em seguida, com essa informação, utilizamos o método .find(‘tag’, id=’se\_tiver\_e\_quiser’):



* + 1. Como observamos ele nos trouxe a tag inteira. Para pegar somente o texto utilizamos a função .get\_text(). Fizemos isso para o título e parágrafo da página que estamos scraping:



* 1. O que aprendemos:
     1. A configurar um ambiente de desenvolvimento com a distribuição Anaconda;
     2. A verificar as versões de bibliotecas Python;
     3. A executar um web scraping simples, do começo ao fim.

1. **Aula 2 – Obtendo e Limpando o HTML:**
   1. Entendendo a Web:
      1. Obtemos nossos dados pelo web scraping pois fazemos uma requisição a partir do computador/cliente para o servidor e o servidor nos devolve/responde o html da página.
   2. Obtendo o Conteúdo HTML de um Site:
      1. Utilizando o que vimos até agora, não será possível fazer a requisição e obter a informação de qualquer site, pois a maioria possui um tipo de proteção para isso. Com isso temos que utilizar outros métodos da biblioteca urllib, tais como:



* + 1. Nesse caso utilizamos o Request junto com o cabeçalho do site, desse modo ele permite fazermos a abertura e leitura da url. Podemos ainda colocar uma exceção para que ele exiba qual dos erros foi ocasionado:







* + 1. O headers foi feito através da inspeção de elementos:



* 1. Tratamento de Strings:
     1. Quando fazemos esse tipo de requisição e o site nos retorna o html, a nossa variável é sempre do tipo byte:





* + 1. Para converter para str precisamos decodificar ele com o .decode(‘utf-8’):



* + 1. Mas isso não faz mudanças efetivas nos nossos dados, apenas começam a aparecer as acentuações, setas dentre outros caracteres do tipo. Continua uma bagunça:



* + 1. Agora vamos remover esses ‘\r’, ‘\n’, ‘\t’, dentre outros. Para isso primeiro precisamos fazer um split, separando todas as palavras por tabulações e espaços. Para juntar tudo novamente fazemos um “ ”.join() do nosso split, dizendo que cada item da lista deve ser separado por um espaço, quando exibimos o que foi feito, não temos mais tabulações. Em seguida, precisamos remover os espaços existentes entre as tags, pois o BeautifulSoap pode não reconhecer elas se houver esse espaço. Para isso fazemos um replace() das tags de assim ‘> <’ para assim ‘><’, e, ao exibir, todas as tags estão sem espaços e nossos dados sem nenhuma tabulação. Com isso em mente, fizemos uma função que realizasse todo esse processo a partir do nosso input e apenas devolvesse nossos dados corretos:







* 1. O que aprendemos:
     1. Um resumo sobre o funcionamento da web;
     2. Algumas ferramentas para acessar um web site e obter o seu conteúdo HTML;
     3. Uma técnica de tratamento de strings e criação de funções com Python.

1. **Aula 3 – Introdução ao BeautifulSoap:**
   1. Vimos sobre estrutura básica de HTML.
   2. Criando um Objeto BeautifulSoup:
      1. O html.parser que passamos como segundo argumento do BeautifulSoup, sendo o primeiro a leitura do html da página que fizemos a requisição, serve para padronizar o comportamento dele. Se não passar esse argumento ele fará e exibirá a leitura de qualquer forma, mas mostrará uma mensagem dizendo que, por não ter passado/especificado o parser, pode ter resultados diferentes já que a própria lib escolherá o um parser que achar melhor. Para exibir o html de forma melhor visível, podemos usar o prettify():



* 1. Acessando Tags:
     1. Uma maneira de se acessar uma tag, é colocando ela logo após o nome da variável que você armazenou seu objeto BeautifulSoup, considerando o primeiro registro dele:



* + 1. Ou então, como ele acessa o primeiro registro da nossa html, podemos colocar somente o title:



* + 1. Se tivessem outros titles e você quisesse o segundo, teria que passar o caminho completo. Se tiver um ninho de divs, por exemplo, para acessar a última temos que fazer o seguinte:



* 1. Acessando o Conteúdo das Tags:
     1. Para pegar somente o conteúdo da tag, sem ela englobando o texto, utilizamos tanto o get\_text() quanto o getText():



* + 1. Podemos utilizar isso com o objeto inteiro, mas não vale a pena porque teríamos que fazer diversos tratamentos de str até chegar no resultado, coisa que o próprio BS nos ajuda a fazer:



* 1. Acessando os Atributos de Uma Tag:
     1. Para acessar os atributos de uma tag utilizamos o attrs. Esse atributo nos devolverá um dict com todas as chaves sendo o atributo e os valores sendo o que está sendo atribuído a esse atributo:



* + 1. Se quisermos somente as chaves ou somente os valores, fazemos igual manipulação de dicts normal. Podemos ainda usar sistema de indexação de dicionários para obter os valores de uma chave específica:



* 1. O que aprendemos:
     1. Como um arquivo HTML é formado, sua estrutura básica e principais tags;
     2. A biblioteca BeautifulSoup;
     3. Como criar um objeto BeautifulSoup;
     4. As formas básicas de navegação e pesquisa em objetos BeautifulSoup;
     5. A acessar tags, seus atributos e conteúdos.

1. **Aula 4 – Pesquisando Com BeautifulSoup:**
   1. Os Métodos find() e findAll():
      1. O find() é exatamente equivalente a apenas colocar sou.img, por exemplo:



* + 1. Já no caso do findAll() ele trás uma lista com todos os itens encontrados no DOM do HTML da página equivalente ao que estamos procurando:



* + - 1. Argumento limit: Limita a quantidade de resultados do findAll() no número desejado:



* + - * 1. Continua devolvendo uma lista. Podemos fazer seleção de dados como em listas utilizando os [] pós ().
      1. Um atalho para esse comando, já que é o que mais utilizamos, é basicamente colocar soup(‘item\_desejado’):



* + - * 1. Exatamente o mesmo resultado que antes.
      1. Podemos passar uma lista para ele trazer tudo o que desejamos de uma vez:



* + - * 1. Se ele não encontrar o item, simplesmente não faz nada.
      1. Argumento attributes: Podemos passar um dicionário com chave e valor referente ao atributo da HTML que queremos pegar:



* + - 1. Por conteúdo na tag:



* + - 1. Utilizando ainda atributos diretos, como a descrição alternativa dos img:



* + - 1. Como são iteráveis podemos ainda pegar somente o src das imagens:



* + - 1. Também existe o atributo class, mas tome cuidado pois o próprio python possui ele, portanto, com scraping, precisamos colocar um \_ depois:



* + - 1. Text = True: Trás todos os textos encontrado em todas as classes para uma lista:



* 1. Outros métodos de Pesquisa:
     1. Tudo o que precisa saber está resumido no notebook da aula 4, sem mais nem menos. Não vou reescrever tudo porque tem zero necessidades.
  2. O que aprendemos:
     1. Técnicas avançadas de pesquisa com objetos BeautifulSoup;
     2. Os métodos find() e findAll();
     3. Algumas configurações de argumentos e dicas de pesquisa para os métodos find() e findAll();
     4. Outros métodos de pesquisa disponibilizados pela biblioteca BeautifulSoup:
        1. findParent e findParents;
        2. findNextSiblings e findNextSiblings;
        3. findPreviousSibling e findPreviousSiblings;
        4. findNext e findAllNext;
        5. findPrevious e findAllPrevious.