BIG DATA

LABDATA

Introdução ao Big Data

Tema da Aula: **Web Scraping com Python**

Prof.: **Dino Magri**

Data: **28 de Novembro de 2018**

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

LABDATA

Coordenação:

Prof. Dr. Adolpho Walter Pimazzi Canton

Profa. Dra. Alessandra de Ávila Montini

- Contatos:
  - E-mail: professor.dinomagri@gmail.com
  - Twitter: https://twitter.com/prof_dinomagri
  - LinkedIn:http://www.linkedin.com/in/dinomagri
  - Site: http://www.dinomagri.com

Currículo

- **(2014-Presente)** – Professor no curso de Extensão, Pós e MBA na Fundação Instituto de Administração (FIA) – www.fia.com.br
- **(2013-Presente)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – www.larc.usp.br
- **(2013)** – Professor no MBA em Desenvolvimento de Inovações Tecnológicas para WEB na IMED Passo Fundo – RS – www.imed.edu.br
- **(2012)** – Bacharel em Ciência da Computação pela Universidade do Estado de Santa Catarina (UDESC) – www.cct.udesc.br
- **(2009/2010)** – Pesquisador e Desenvolvedor no Centro de Computação Gráfica – Guimarães – Portugal – www.ccg.pt
- **Lattes:** http://lattes.cnpq.br/5673884504184733

Material das aulas

- Material das aulas estão disponíveis em:
  - https://urls.dinomagri.com/posmba-turma9
    - **Senha**: `turma9`
    - **Data Expiração:** `31-12-2018`

Material das aulas

- Caso esteja utilizando seu próprio computador, realize o download de todos os arquivos e salve na **Área de Trabalho** para facilitar o acesso.
  - Lembre-se de instalar os softwares necessários conforme descrito no documento de Instalação (**`InstalaçãoPython3v1.1.pdf`**).

- Nos computadores da FIA os arquivos já estão disponíveis, bem como a instalação dos softwares necessários.

Conteúdo da Aula

- Objetivo
- Web Scraping
- BeautifulSoup
- Expressão Regular

Conteúdo da Aula

- **Objetivo**
- Web Scraping
- BeautifulSoup
- Expressão Regular

Objetivo

- Objetivo dessa aula é introduzir os **conceitos sobre Web Scraping** e como podemos criar scripts **Python** para recuperar informações sem ter acesso a **API**.

Conteúdo da Aula

- Objetivo
- **Web Scraping**
- BeautifulSoup
- Expressão Regular

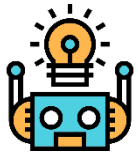
Introdução

- Uma empresa de e-commerce está procurando uma consultoria que crie um sistema automatizado para comparar os preços praticados em seus concorrentes de diversos produtos.

- Porém, essas empresas não disponibilizam uma API para acessar os dados.

- **Como essa empresa poderia realizar essa análise?**

Web Scraping



O que é Web Scraping?

- Web Scraping é a prática de **coletar dados** através de qualquer meio que não seja um programa interagindo com uma API.
- Podemos fazer isso, criando um **programa automatizado** que **consulta** um servidor web, **solicita** os dados para **analisar** e **extrair** as informações necessárias.

O que é Web Scraping?

- Transformar dados não estruturados (HTML) em dados estruturados.

Por que utilizar Web Scraping?

- Quando recuperamos informações de terceiros, é importante utilizarmos uma API. Porém **nem sempre** é oferecida e/ou disponibilizada, pois:
  - API não foi bem definida;
  - O conjunto de **dados** é **muito pequeno**; ou
  - Não tem uma infraestrutura ou habilidade técnica para criar uma API.
- Mas mesmo que exista, ainda assim essas empresas impõem limites de volume e velocidade das solicitações, tipos dos dados fornecidos entre outros.

Por que utilizar Web Scraping?

- Portando podemos utilizar Web Scraping para suprir essas necessidades.

- Salvo algumas exceções, se conseguimos visualizar no navegador web, podemos acessá-lo via Python ☺

Como utilizar Web Scraping?

- Conforme vimos nas aulas anteriores, utilizamos o módulo `requests` para recuperar uma página web.

```
>>> import requests
>>> req = requests.get("http://www.python.org")
>>> req.text # Imprime o texto capturado (HTML)
```

Web Scraping

- Para realizar o Web Scraping, será necessário, além da **recuperação de dados HTML** a partir de um nome de domínio, **analisar os dados** buscando as informações desejadas.
- Além disso, devemos realizar o armazenamento dessas informações.

Web Scraping

Os exemplos apresentados nessa aula foram baseados e/ou retirados do livro: **Web Scraping com Python** – Ryan Mitchell – O'Reilly (Novatec), 2015.

Web Scraping

- O primeiro passo, de recuperar uma determinada página, é relativamente simples, como vimos no exemplo anterior.

- Porém quando realizamos uma chamada **requests**:

```
>>> req = requests.get("http://pythonscraping.com/pages/page1.html")
```

BeautifulSoup

- É possível acontecer duas situações:
  1. **A página** pode **não ser encontrada** no servidor (ou ocorrer algum erro na recuperação)
     - Nesse caso uma mensagem de erro HTTP (**404** Page Not Found, **500** Internal Server Error, entre outros erros.) será retornada e será necessário tratá-la corretamente.
  2. **O servidor não ser encontrado**
     - Se o servidor não for encontrado, uma exceção de erro de conexão será apresentada.

BeautifulSoup

- Para o **primeiro cenário**, precisamos verificar qual é o tipo de erro, se a página não existir o valor atribuído em `req.status_code` será o `404`. Desta forma podemos realizar uma verificação simples.

```
req = requests.get("http://pythonscraping.com/pages/page1.html")
if req.status_code != 200:
    print(req.status_code)
```

- Existem diversos códigos HTTP, sendo que cada um representa uma ação. Uma lista completa pode ser visualizada em: https://goo.gl/vjM5ye

BeautifulSoup

- Já no **segundo cenário**, precisamos de fato tratar a exceção `ConnectionError` através do **try... except**

```
try:
    req = requests.get("http://pythonscraping.com/pages/page1.html")
except ConnectionError as e:
    print(e)
    # Retorna nulo, finaliza, ou executa novamente?!
```

Web Scraping

- A primeira página que iremos recuperar é uma página HTML com apenas alguns elementos.

```
>>> import requests
>>> req = requests.get("http://pythonscraping.com/pages/page1.html")
>>> print(req.text)
<html><head>
<title>A Useful Page</title>
</head>
<body>
<h1>An Interesting Title</h1>
... continua ...
```

Web Scraping

- A primeira página que iremos recuperar é uma página HTML com apenas alguns elementos, porém de fácil entendimento.

</> Abra o arquivo **"aula6-parte1-web-scraping.ipynb"**

Conteúdo da Aula

- Objetivo
- Web Scraping
- **BeautifulSoup**
- Expressão Regular

BeautifulSoup

- Para facilitar a analise e recuperação dos dados que estão inseridos dentro das marcações HTML, iremos utilizar a biblioteca **BeautifulSoup** do Python.

- Fornece métodos que facilitam a navegação, pesquisa, e modificação na árvore de análise.

BeautifulSoup

- Para instalar essa biblioteca abra CMD ou Terminal e digite:
  - `pip3 install bs4`

- Agora já podemos importar em nosso notebook
  - `from bs4 import BeautifulSoup`

BeautifulSoup

- O objeto que iremos utilizar da biblioteca bs4 é o próprio BeautifulSoup

```
>>> bs = BeautifulSoup(html, "html.parser")
>>> print(type(bs))
<class 'bs4.BeautifulSoup'>
```

- Existem diversos analisadores (*parsers*) que podem ser utilizados além do **html.parser** que já vem incluído na biblioteca padrão do Python.
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/#installing-a-parser

BeautifulSoup

- Para visualizar a página capturada, basta imprimir a variável bs.

```
>>> print(bs)
```

BeautifulSoup

```
<html>
        <head>
                <title>A Useful Page</title>
        </head>
        <body>
                <h1>An Interesting Title</h1>
                <div>
                Lorem ipsum dolor sit magna aliqua ... est laborum.
                </div>
        </body>
</html>
```

</> Abra o arquivo **"aula6-parte2-bs4.ipynb"**

BeautifulSoup

- A biblioteca facilita o acesso as principais *tags* HTML por meio de atributos pré-definidos.

```
>>> bs = BeautifulSoup(html, "html.parser")
>>> print(bs.body.h1)
<h1>An Interesting Title</h1>
>>> type(bs.h1)
bs4.element.Tag
```

BeautifulSoup

- **Simples assim!** Agora podemos acessar as tags HTML e recuperar o valor existente em cada uma dessas tags.
- Antes de continuar, é importante salientar alguns problemas que podem acontecer durante o processo de análise do HTML.
- Existem dois principais erros de analisadores: `HTMLParser` e `HTMLParseError`
  - `malformed start tag` (inicio da tag mal formatada)
  - `bad end tag` (tag errada no fim)

BeautifulSoup

- Ambos os erros podem acontecer, uma vez que o BeautifulSoup utiliza analisadores externos.
- Desta forma as **exceções** serão lançadas por esses analisadores externos.
- Caso encontre algum erro relacionado, sugere-se utilizar outros analisadores, como o `lxml` ou `html5lib`, para tentar solucionar o problema.

BeautifulSoup

- Além disso, o BeautifulSoup irá retornar **`None`**, caso tente acessar uma tag HTML que **não exista**.
- Realize o seguinte teste:

```
>>> print(bs.body.h4)
```

`None`

BeautifulSoup

- Quando estamos criando **scrapers**, é importante pensar no padrão geral do seu código, tratando possíveis exceções.

- Também é muito importante a reutilização de código, uma vez que iremos recuperar diversas vezes uma determinada informação, como por exemplo, o título da página.

- Nesse caso, funções como `download_html` e `recuperar_titulo` facilitam a execução rápida e confiável do Scraping na Web.

Exercício de fixação

- Vamos criar uma função chamada `recuperar_titulo(url)`, que deverá retornar o título da URL (página) passada por parâmetro. Lembre-se de tratar os erros necessários. Reutilize códigos!

</> Abra o arquivo **"aula6-parte2-bs4.ipynb"**

BeautifulSoup

- Como vimos, o acesso as tags HTML é relativamente simples.
- No primeiro exemplo, recuperamos a tag H1 que representava o título da página.
- Mas vimos que podem existir diversas tags de mesmo nome. **Como podemos fazer para encontrar essas tags?**

BeautifulSoup

- A biblioteca fornece duas funções, `find()` e `findAll()` que podem **filtrar** facilmente as **páginas** para encontrar as tags que queremos de acordo com seus diversos atributos.
  - `find` – retorna apenas o primeiro filho da tag pesquisada.
  - `findAll` – extrai uma lista de tags que correspondem a um dado critério.

</> Abra o arquivo **"aula6-parte3-tags.ipynb"**

BeautifulSoup

- Além do parâmetro `name` é possível utilizar os seguintes parâmetros:
  - `attrs={}` – Caso o nome a ser procurado seja uma palavra reservada do Python, utiliza-se o atributo `attrs`.
  - `recursive` – Se a recursão for definida como **`True`**, a função descerá aos filhos e aos filhos dos filhos procurando tags que coincidam com seus parâmetros.
  - `text` – procurar ocorrências de acordo com o conteúdo de texto das tags.
  - `limit` – é utilizado no findAll e recupera os **n primeiros** itens da página.

BeautifulSoup

- A função `findAll` encontra as tags de acordo com seu nome e atributo.
- Mas, caso seja necessário encontrar uma tag de acordo com sua localização em um documento?

- Podemos utilizar a **navegação em árvore**.

BeautifulSoup

```
html
        body
                div.wrapper
                        h1
                        div.contente
                        table#giftList
                                tr
                                        th
                                        th
                                        th
                                        th
                                tr.gift#giftList
                                        td
                                        td
```

BeautifulSoup

- Lidando com filhos e outros descendentes
  - As tags `tr` são filhas da tag `table`.
  - Para encontrar somente descendentes que sejam filhos, é possível utilizar a tag `.children`
- Lidando com irmãos
  - Podemos utilizar a função `next_siblings`.

</> Abra o arquivo **"aula6-parte3-tags.ipynb"**

Exemplo prático

- Iremos aplicar os conceitos aprendidos de Web Scraping e da biblioteca BeautifulSoup para recuperar uma página da Wikipédia.
  - https://pt.wikipedia.org/wiki/Unidades_federativas_do_Brasil
- Essa página contém informações sobre as unidades federativas do Brasil.
- O que iremos fazer:
  - Recuperar todos os links internos e externos da página
  - Recuperar a tabela que lista todos os estados
  - Realizar os tratamentos necessários
  - Salvar em um DataFrame

</> Abra o arquivo **"aula6-parte4-wikipedia.ipynb"**

Conteúdo da Aula

- Objetivo
- Web Scraping
- BeautifulSoup
- **Expressão Regular**

Expressões Regulares

- Expressões regulares permitem encontrar **padrões em texto**.
- Aplicações:
  - Validação de entradas
  - Aplicação de máscaras
  - Filtragem de resultados em consultas
  - Remover caracteres de texto
  - Entre outros ...

Expressões Regulares

- Como funcionam?
  - Casamento de padrões
  - Quantificadores
  - Classes de caracteres

Expressões Regulares

- Casamento de padrões
  - O padrão **asa**
    - **asa** (posição 0)
    - c**asa** (posição 1)
    - br**asa** (posição 2)
    - atr**asa**do (posição 3)
  - O padrão **123**
    - **123** (posição 0)
    - 0**123** (posição 1)
    - -10**123**45 (posição 2)

Expressões Regulares

- Quantificadores - Exemplos
  - a ⟶ 'a'
  - a+ ⟶ 'a', 'aa', 'aaa', ...
  - a* ⟶ '', 'a', 'aa', 'aaa', ...
  - a? ⟶ '', 'a'

Quantificador	Descrição
+	1 ou mais
*	0 ou mais
?	0 ou 1

Expressões Regulares

- Quantificadores - Exemplos
  - **abc** ⟶
  - a**b+**c ⟶
  - **a***bc ⟶
  - a**c?**d ⟶

Quantificador	Descrição
+	1 ou mais
*	0 ou mais
?	0 ou 1

Expressões Regulares

- Quantificadores - Exemplos
  - **abc** ⟶ 'abc'
  - a**b+**c ⟶ 'a**b**c', 'a**bb**c', 'a**bbb**c', ...
  - **a***bc ⟶ ' bc', '**a**bc', '**aa**bc', '**aaa**bc', ...
  - a**c?**d ⟶ 'ad', 'a**c**d'

Quantificador	Descrição
+	1 ou mais
*	0 ou mais
?	0 ou 1

Expressões Regulares

- Classes de caracteres
  - Dígito: **\d**
  - Letra: **[A-Z]**, **[a-z]**, **[a-zA-Z]**
  - Letra, dígito ou '_': **\w**
- O que casa com as seguintes expressões regulares
  - \d\d\d ⟶
  - [a-z]\w ⟶
  - [A-Z]\d\d ⟶

Expressões Regulares

- Classes de caracteres
  - Dígito: **\d**
  - Letra: **[A-Z]**, **[a-z]**, **[a-zA-Z]**
  - Letra, dígito ou '_': **\w**
- O que casa com as seguintes expressões regulares
  - \d\d\d ⟶ '000', '111', '104', '054', '545', ...
  - [a-z]\w ⟶ 'aa', 'ab', 'cd', 'a4', 'a1', 'zh', 'a_', ...
  - [A-Z]\d\d ⟶ 'A87', 'Y11', 'B32', 'X62', ...

Expressões Regulares

- **Grupos** permitem agrupar trechos de strings. São úteis para extrair partes de uma string e/ou substituir strings.
- Exemplos:
  - Horas e minutos (HH:MM)
  - \d\d:\d\d ⟶ Valida a string!
  - (\d\d):(\d\d) ⟶ Separa em grupos para serem acessados usando \1, \2

Expressões Regulares

- É possível determinar o **inicio** e **fim** da string através dos metacaracteres **^** e **$**, respectivamente.
- Exemplos:
  - `^inicio` ⟶ Casa com `'inicio'`, porém não casa com `'1inicio'`
  - `fim$` ⟶ Casa com `'2 fim'`, porém não casa com `'2 fim 2'`
  - `^qualquerPadrao$` ⟶ Casa somente quando o padrão ocupa a linha inteira
    - `^\d\d\d$`

Padrões básicos

Padrão	Significado
a, X, 9, <	Caracteres ordinários que correspondem apenas a si mesmo.
. (um ponto)	Corresponde a qualquer caractere único, exceto nova linha '\n'.
[...]	Corresponde a qualquer caractere incluído no conjunto.
[^...]	Corresponde a qualquer caractere não incluído no conjunto.
\	Utilizada para inibir a excepcionalidade de um caractere. Por exemplo, utilize **\.** para corresponder a um ponto ou **** para corresponder a uma barra invertida.
\w	Corresponde a um caractere alfanumérico [a-zA-Z0-9_]. Se for maiúscula (\W), corresponde a um caractere não-alfanumérico.
\s	Corresponde a um único caractere de espaço em branco (nova linha, retorno, tabulação e forma [\n, \r, \t, \f]. Se for maiúscula (\S), corresponde a um caractere não espaço em branco.
\t, \n, \r	Tabulação, Nova linha, Retorno.
\d	Digito decimal [0-9]. Se for maiúscula (\D), corresponde a um não-dígito.

Padrões básicos

Padrão	Significado
{n}	Exatamente n ocorrências.
{n,m}	No mínimo n ocorrências e no máximo m.
{n, }	No mínimo n ocorrências.
{, m}	No máximo m ocorrências.
?	0 ou 1 ocorrência; o mesmo que {0, 1}.
+	1 ou mais ocorrências; o mesmo que {,1}.
*	0 ou mais ocorrências.
^ = inicio, $ = fim	Corresponde com o início ou fim de uma string.
(...)	Define um grupo para posterior extração ou reuso
... | ...	Alternativa, corresponde tanto a expressão regular da esquerda, quanto da direita

Expressões Regulares

- Mais exemplos, o que casa/corresponde com:
  - a\s*b ⟶
  - python[XYZ] ⟶
  - 1. ⟶
  - .* ⟶
  - \d\d\D ⟶
  - [^abc]+ ⟶
  - py|thon ⟶
  - c|\d ⟶
  - P|y|t|h|0|n ⟶

Expressões Regulares

- Mais exemplos, o que casa/corresponde com:
  - a\s*b ⟶ **'a b', 'a b', 'a   b', 'a   b', ...**
  - python[XYZ] ⟶ **'pythonX', 'pythonY, 'pythonZ'**
  - 1. ⟶ **'1a', '1;', '1%', '1$', '1~', '11', '19', ...**
  - .* ⟶ **Qualquer coisa que preencha em uma linha**
  - \d\d\D ⟶ **'00a', '45/', '34 ', '98a', ...**
  - [^abc]+ ⟶ **'d', 'dcd', '1234', '[]{}()sd', ...**
  - py|thon ⟶ **'py', 'thon'**
  - c|\d ⟶ **'c', '0', '1', ... , '9'**
  - P|y|t|h|0|n ⟶ **'P', 'y', 't', 'h', '0', 'n' ⟶ mesmo que [Pyth0n]**

import re

- Em Python utilizamos o módulo `re` para realizar uma **pesquisa** de expressão regular. Utilizamos o método `search`:

```
>>> match = re.search(padrao, texto)
```

- O método `re.search()` recebe dois parâmetros: **padrão** e o **texto**.
- Esse método irá procurar pelo padrão no texto, se encontrar retorna o objeto correspondente, se não encontrar retorna `None`.
- Normalmente a busca é validada para verificar se o padrão foi encontrado.

import re

```
import re

texto = 'um exemplo palavra:python!!'

match = re.search('python', texto)

if match:
      print('encontrou: ' + match.group())

else:
      print('não encontrou')
```

import re

- Utilizamos o **r** no início da string de padrão para definir que a string é uma string sem tratamento algum ("**raw string**").
- Desta forma a barra invertida não tem tratamento especial, o que é muito útil em expressões regulares.
- Sempre que definir um padrão, iremos utilizar o **r' '** no inicio.
- **O poder das expressões regulares é que podemos especificar padrões e não apenas utilizar caracteres fixos**.

import re

- Também é possível utilizar a flag IGNORECASE, para testar as expressões, facilitando identificar o padrão desejado.

```
texto = "GGATCGGAGCGGATGCC"

match = re.search(r'a[tg]c', texto, re.IGNORECASE)
```

import re

- As regras básicas para pesquisar por expressões regulares dentro de uma string são:
  - A pesquisa ocorre do início ao fim da string, parando quando encontrar a primeira ocorrência do padrão.
  - Todo o padrão deve ser correspondido, porém não toda a string.
  - Se o padrão for encontrado, o texto está no método `.group()`.

</> Abra o arquivo **"aula6-parte5-er.ipynb"**

Extração do Grupo

- É possível **extrair partes de uma expressão regular**. Suponha que queremos salvar separadamente o nome do usuário e do domínio. Para isso adicione parênteses entre o padrão do usuário e do domínio.

```
match = re.search('([\w.-]+)@([\w.-]+)', texto)
```

- Desta forma, se o padrão for encontrado, ele irá salvar a primeira correspondência no `group(1)` e a outra no `group(2)`.

Extração do Grupo

```
match = re.search('([\w.-]+)@([\w.-]+)', texto)

if match:

      print(match.group()) # Imprime tudo

      print(match.group(1)) # Imprime o nome de usuário

      print(match.group(2)) # Imprime o domínio.
```

Encontrando tudo!

- Até agora utilizamos o `re.search()` para procurar pela primeira ocorrência de um padrão.
- Para encontrar **todas as ocorrências** podemos utilizar o `re.findall()`, que irá salvar todas as ocorrências em uma lista, onde cada posição representa uma ocorrência.

```
>>> texto = 'teste teste@gmail.com teste123 teste-
123@gmail.com, python 123@123.com'
>>> emails = re.findall('[\w.-]+@[\w.-]+', texto)
```

Encontrando tudo!

```
emails = re.findall('[\w.-]+@[\w.-]+', texto)

for email in emails:

      print email
```

</> Abra o arquivo **"aula6-parte5-er.ipynb"**

Exercícios

Exercício 1 - Encontre todos os e-mails do arquivo er-emails.txt.

Utilize a função `findall()` e imprima cada um dos e-mails.

```
arquivo = open('er-emails.txt', 'r')
```

</> Abra o arquivo **"aula6-parte6-exercicios.ipynb"**

Exercícios

Exercício 2 - Considere o arquivo `er-dados.txt`. Esse arquivo contêm diversas strings no formato:

```
Tue Feb 15 10:39:54 2028::xjkmxk@cltllsls.com
```

1. Crie um expressão regular para encontrar todos os e-mails e salve-os em uma lista chamada `emails = []`.
2. Crie um expressão regular para recuperar o login e o domínio de cada item salvo na lista **`emails`**. Salve cada item em uma lista `logins = []` e `dominios = []`.
3. Crie um expressão regular para recuperar o ano, mês, dia, horário.
4. Por fim, imprima os valores no seguinte formato:

```
1 - email@completo.com | login | dominio | dia/mês/ano | hora:minuto
```

1. É importante pensar em como deve-se carregar o arquivo. Repare que o horário no arquivo considera os segundos e a impressão final não.

Referências Bibliográficas

- **Web Scraping with Python** – Ryan Mitchell – O´Reilly, 2015.
- **Mastering pandas** – Femi Anthony – Packt Publishing, 2015.
- **Data Science from Scratch** – Joel Grus – O'Reilly, 2015.
- **Python for Data Analysis** – Wes McKinney – USA: O'Reilly, 2013.
- Referência da BeautifulSoup - https://www.crummy.com/software/BeautifulSoup/

Referências Bibliográficas

- **Python for kids – A playful Introduction to programming** – Jason R. Briggs – San Francisco – CA: No Starch Press, 2013.

- **Python Cookbook** – David Beazley & Brian K. Jones – O'Reilly, 3th Edition, 2013.

- As referências de links utilizados podem ser visualizados em http://urls.dinomagri.com/refs