

BIG DATA



LÁBDATA



FUNDAÇÃO
INSTITUTO DE
ADMINISTRAÇÃO

Introdução ao Big Data

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

Prof.: **Dino Magri**

Coordenação:

Prof. Dr. Adolpho Walter
Pimazzi Canton

Profa. Dra. Alessandra
de Ávila Montini

Data: **08 de Agosto de 2016**

- **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>

Coordenação:

**Prof. Dr. Adolpho Walter
Pimazzi Canton**

**Profa. Dra. Alessandra
de Ávila Montini**

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:
 - <http://urls.dinomagri.com/posmba-turma2/>
 - Senha: fia2016
- Faça o download do arquivo **2016-08-08-py-aula7.zip**
- Salve na Área de Trabalho (Desktop)
- Depois que finalizar o download, acesse a pasta Área de trabalho e descompacte o arquivo 2016-08-08-py-aula7.zip.

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Web Scraping
- BeautifulSoup
- Expressão Regular

Conteúdo da Aula

- **Objetivo**
- Exercício de Revisão
- 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
- **Exercício de Revisão**
- Web Scraping
- BeautifulSoup
- Expressão Regular

Exercício de Revisão

- Recuperar as reações de um post específico do Facebook sem utilizar uma biblioteca específica de acesso a Graph API.
 - Entender como as URLs do Facebook são criadas
 - Criar o token de acesso
 - Recuperar o id do post
 - Definir os campos a serem recuperados (`reactions`)
 - Salvar o total de c

Abra o notebook "aula7-parte1-reacoes.ipynb"

Exercício para treino

- Escolha 5 páginas de empresas de mesmo segmento.
- Para cada empresa, recupere 25 posts e faça a contagem das reações desses posts.
- Ao final verifique qual das empresas tiveram maior quantidade de cada uma das reações.
- Exemplo de saída:
 - A empresa A teve mais LIKE
 - A empresa B teve mais WOW
 - A empresa C teve mais SAD
 - A empresa D teve mais HAHA
 - A empresa E teve mais ANGRY

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- **Web Scraping**
- BeautifulSoup
- Expressão Regular

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 **consulte** um servidor web, **solicite** os dados e **analise**-os para **extrair** as informações necessárias.

O que é Web Scraping?

- Transforma 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 a infraestrutura ou habilidade técnica para criar uma API.
- Mas mesmo que exista, ainda assim existem os limites de volume e velocidade das solicitações, tipos dos dados fornecidos entre outros.

Por que utilizar Web Scraping?

- Portanto 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 na aula passada, 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

- Além da **recuperação de dados HTML** a partir de um nome de domínio.
- Iremos **analisar os dados** buscando as informações desejadas.
- E também 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

- **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 ...
```

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- Web Scraping
- **BeautifulSoup**
- Expressão Regular

BeautifulSoup

- Para facilitar a análise e recuperação dos dados que estão inseridos dentro das marcações HTML, iremos utilizar outra biblioteca do Python.
- **BeautifulSoup** é uma biblioteca para extrair dados de arquivos HTML e XML.
- Fornece métodos que facilitam a navegação, pesquisa, e modificação na árvore de análise.

BeautifulSoup

- Para instalar essa biblioteca abra CMD ou Terminal e digite:

- `pip install beautifulsoup4`

- Agora já podemos importar em nosso notebook.

- `from bs4 import BeautifulSoup`

BeautifulSoup

- Para instalar essa biblioteca abra CMD ou Terminal e digite:

- `pip install beautifulsoup4`

- 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(req.text, "html.parser")
```

```
>>> print(bs.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 nos aprofundarmos mais na análise do HTML, precisamos tomar alguns cuidados com as requisições realizadas, pois algumas situações podem acontecer, como:
 - Os dados estarem mal formatados
 - Os sites estarem fora do ar
 - Faltar fechar alguma *tag*
 - Entre outros.

BeautifulSoup

- Quando realizamos a chamada do requests:

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

- É possível acontecer duas situações:

1. A página pode **não ser encontrada** no servidor (ou ocorrer algum erro na recuperação)

- Nesse caso uma mensagem HTTP (**404** Page Not Found, **500** Internal Server Error, etc.) será retornada e precisamos realizar esse tratamento de erro.

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)
```

- 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?!
```

BeautifulSoup

- Além disso, o BeautifulSoup irá retornar **None**, caso tente acessar uma tag HTML que **não exista**. Porém, se tentarmos acessar essa tag o BeautifulSoup irá lançar uma exceção de `AttributeError`.
- Desta forma, teríamos que explicitamente tratá-las:

```
try:  
    conteudo_ruim = bs.tag_inexistente.outra_tag  
except AttributeError as e:  
    print("Tag não encontrada")
```

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 podemos querer recuperar o título de outra página.
- **Nesse caso, funções como `recuperarHTML(url)` e `recuperarTitulo(url)` facilitam a execução rápida e confiável do Scraping na Web.**

Exercício de fixação

- Crie uma função chamada `recuperarTitulo(url)`, que deverá retornar o título da URL (página) passada por parâmetro. Lembre-se de tratar os erros necessários.
- Obs: Para o tratamento das exceções utilize:

```
from requests.exceptions import ConnectionError
```

Abra o notebook "aula7-parte2-web-scraping.ipynb"

BeautifulSoup

- Como vimos, o acesso as tags HTML é relativamente simples.
- No primeiro exemplo, recuperamos a tag H1 que tinha 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.

BeautifulSoup

```
>>> from bs4 import BeautifulSoup
>>> import requests
>>> req = requests.get("http://pythonscraping.com/pages/page3.html")
>>> bs = BeautifulSoup(req.text, "html.parser")
>>> bs.find({"span"})
<span class="excitingNote">Now with super-colorful bell peppers!</span>
>>> bs.findAll({"span"})
[<span class="excitingNote">Now with super-colorful bell peppers!</span>,
 <span class="excitingNote">8 entire dolls per set! Octuple the presents!</span>,
 <span class="excitingNote">Also hand-painted by trained monkeys!</span>,
 <span class="excitingNote">Or maybe he's only resting?</span>,
 <span class="excitingNote">Keep your friends guessing!</span>]
```

BeautifulSoup

- Além do parâmetro `attrs={}`, é possível utilizar os seguintes parâmetros:
 - `recursive` – Se a recursão for definida como `True`, a função descenderá 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 x 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 notebook "aula7-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 notebook "aula7-parte4-
wikipedia.ipynb"

Conteúdo da Aula

- Objetivo
- Exercício de Revisão
- 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)
 - **casa** (posição 1)
 - **brasa** (posição 2)
 - **atrasedo** (posição 3)
 - O padrão **123**
 - **123** (posição 0)
 - **0123** (posição 1)
 - **-1012345** (posição 2)

Expressões Regulares

- Quantificadores - Exemplos

- $a \rightarrow 'a'$
- $a^+ \rightarrow 'a', 'aa', 'aaa', \dots$
- $a^* \rightarrow '', 'a', 'aa', 'aaa', \dots$
- $a? \rightarrow '', 'a'$

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** \rightarrow 'abc'

- a**b**+c** \rightarrow 'a**b**c', 'a**bb**c', 'a**bbb**c', ...

- a*bc** \rightarrow ' bc', 'a**b**c', 'a**a**bc', 'a**aa**bc', ...

- ac?d** \rightarrow 'ad', 'a**cd**'

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 uteis 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 **início** 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
<code>a, X, 9, <</code>	Caracteres ordinários que correspondem apenas a si mesmo.
<code>.</code> (um ponto)	Corresponde a qualquer caractere único, exceto nova linha <code>\n</code> .
<code>[...]</code>	Corresponde a qualquer caractere incluído no conjunto.
<code>[^ ...]</code>	Corresponde a qualquer caractere não incluído no conjunto.
<code>\</code>	Utilizada para inibir a excepcionalidade de um caractere. Por exemplo, utilize <code>\.</code> para corresponder a um ponto ou <code>\\</code> para corresponder a uma barra invertida.
<code>\w</code>	Corresponde a um caractere alfanumérico <code>[a-zA-Z0-9_]</code> . Se for maiúscula <code>(\W)</code> , corresponde a um caractere não-alfanumérico.
<code>\s</code>	Corresponde a um único caractere de espaço em branco (nova linha, retorno, tabulação e forma <code>[\n, \r, \t, \f]</code>). Se for maiúscula <code>(\S)</code> , corresponde a um caractere não espaço em branco.
<code>\t, \n, \r</code>	Tabulação, Nova linha, Retorno.
<code>\d</code>	Dígito decimal <code>[0-9]</code> . Se for maiúscula <code>(\D)</code> , 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.
^ = início, \$ = 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|O|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 caiba em uma linha**
 - `\d\d\D` → `'00a', '45/', '34 ', '98a', ...`
 - `[^abc]+` → `'d', 'dcd', '1234', '[]{}()asd', ...`
 - `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(r'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 início.
- **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 notebook "aula7-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 notebook "aula7-parte5-er.ipynb"

Exercícios

Exercício 1 - Encontre todos os e-mails do arquivo emails.txt. Utilize a função `findall()` e imprima cada um dos e-mails.

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


Exercícios

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

```
Tue Feb 15 10:39:54 2028::xjkmxk@clt1l1sls.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>