

# BIG DATA

# 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

- Contatos:

- E-mail: [professor.dinomagri@gmail.com](mailto:professor.dinomagri@gmail.com)
- Twitter: [https://twitter.com/prof\\_dinomagri](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](http://www.fia.com.br)
- **(2018-Presente)** – Pesquisa e Desenvolvimento de Big Data e Machine Learning na Beholder (<http://beholder.tech> )
- **(2013-2018)** – Pesquisa e Desenvolvimento no Laboratório de Arquitetura e Redes de Computadores (LARC) na Universidade de São Paulo – [www.larc.usp.br](http://www.larc.usp.br)
- **(2012)** – Bacharel em Ciência da Computação pela Universidade do Estado de Santa Catarina (UDESC) – [www.cct.udesc.br](http://www.cct.udesc.br)
- **(2009/2010)** – Pesquisador e Desenvolvedor no Centro de Computação Gráfica – Guimarães – Portugal – [www.ccg.pt](http://www.ccg.pt)
- **Lattes:** <http://lattes.cnpq.br/5673884504184733>

## 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 (**InstalaçãoPython3v1.2.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**.



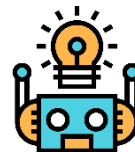
# Conteúdo da Aula

- 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?

- 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 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 **"aula7-parte1-web-scraping.ipynb"**



# Conteúdo da Aula

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

# BeautifulSoup

- Para facilitar a análise 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 "**aula7-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` (início 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 "**aula7-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 "[aula7-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 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 **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 "[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](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 "**aula7-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
  - Filtragem de resultados em consultas
  - Remoção de caracteres de texto
  - Entre outros.

# Expressões Regulares

- Como funcionam as expressões regulares?
  - 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

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

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



# Expressões Regulares

- Quantificadores - Exemplos

- **abc** →
- a**b+**c →
- **a\***bc →
- ac**?**d →

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

# Expressões Regulares

- Quantificadores - Exemplos

- **abc**  $\rightarrow$  'abc'
- a**b+c**  $\rightarrow$  'abc', 'abbc', 'abbbc', ...
- **a\***bc  $\rightarrow$  'bc', 'abc', 'aabc', 'aaaabc', ...
- a**c?**d  $\rightarrow$  'ad', 'acd'

| 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`  $\rightarrow$  '000', '111', '104', '054', '545', ...
  - `[a-z]\w`  $\rightarrow$  'aa', 'ab', 'cd', 'a4', 'a1', 'zh', 'a\_', ...
  - `[A-Z]\d\d`  $\rightarrow$  '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 **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, &lt;</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]` →
  - `l.` →
  - `.*` →
  - `\d\d\D` →
  - `[^abc]+` →
  - `py|thon` →
  - `c|\d` →
  - `P|y|t|h|0|n` →

# Expressões Regulares

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

# import re

- Como podemos utilizar expressões regulares no Python?

```
import re
```

- Basicamente existem 3 operações que podemos utilizar:
  - search
  - findall
  - sub

## re - search

- A função `search` procura pela primeira ocorrência da expressão regular dentro da string.
- A sintaxe básica é: `re.search(padrao, string)`.
  - `padrao` é a expressão regular que precisa ser correspondida.
  - `string` é o texto onde o padrão será correspondido.
- 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.

## re - search

```
>>> import re

>>> texto = 'Programando em python em 3 2 1!'

>>> r = re.search('python', texto)

>>> print(r)

<_sre.SRE_Match object; span=(15, 21), match='python'>

>>> print(r.group())

python
```



Abra o arquivo "**aula7-parte5-er.ipynb**"

## re - search

```
>>> texto = 'Dino Magri - 11-999999999'
>>> r = re.search('(\d\d)-(\d{9})', texto)
>>> print(r.group())
11-999999999
>>> print(r.group(1))
11
>>> print(r.group(2))
999999999
```



Abra o arquivo "**aula7-parte5-er.ipynb**"

## re - findall

- Podemos utilizar o método `findall` para encontrar todas as ocorrências de um padrão em um texto.
- O retorno será uma lista com todas as correspondências.
- A sintaxe básica é: `re.findall(padrão, string)`

## re - findall

```
>>> dados = """
```

 Abra o arquivo "aula7-parte5-er.ipynb"

```
01/02/2019 - IPAD 5 - R$ 1287,99
```

```
01/02/2019 - NOTE+ - R$ 987,56
```

```
"""
```

```
>>> datas = re.findall(\d\d/\d\d/\d{4}, dados)
```

```
>>> print(datas)
```

```
['01/02/2019', '01/02/2019']
```



## re – sub

- Por fim, podemos utilizar o **método sub** para substituir o padrão encontrado por um valor específico.
- Essa funcionalidade procura por todas as ocorrências de um padrão em um determinado texto e substitui.
- Utilizando a variável `dados` criada anteriormente, substitua a virgula por ponto.

re - sub

```
>>> resultado = re.sub(',', ' ', '.', dados)
```

```
>>> print(resultado)
```

```
"""
```

```
01/02/2019 - IPAD 5 - R$ 1287.99
```

```
01/02/2019 - NOTE+ - R$ 987.56
```

```
"""
```

## Exercícios

- Abra o notebook **aula7-parte5-er.ipynb** para resolver os 3 exercícios de Expressão Regular.

 Abra o arquivo "**aula7-parte5-er-gabarito.ipynb**"

# 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>