Beautiful Soup e Seletores

PUCRS
Pontifícia Universidade Católica do Rio Grande do Sul

– Aula 06 –
Coleta, Preparação e Análise de Dados

Prof. Me. Lucas R. C. Pessutto

Slides adaptados do material do Prof. Lucas Silveira Kupssinskü e do Prof. Luan Fonseca Garcia

Beautiful Soup 4

- É uma biblioteca disponível em Python para extrair dados de arquivos html e xml
- Pode ser usada com diversos parsers html
  - https://www.crummy.com/software/BeautifulSoup/bs4/doc/

Beautiful Soup 4

Parsers mais utilizados

Parser	Uso típico	Vantagens	Desvantagens
Python html.parser	`BeautifulSoup(markup, "html.parser")`	Batteries included Velocidade razoável Leniente	Não tão rápido como o lxml. Menos leniente que html5lib.
lxml's HTML parser	`BeautifulSoup(markup, "lxml")`	Muito rápido Leniente	Dependência externa código C
lxml's XML parser	`BeautifulSoup(markup, "lxml-xml")` `BeautifulSoup(markup, "xml")`	Muito rápido Único parser de xml suportado	Dependência externa código C
html5lib	`BeautifulSoup(markup, "html5lib")`	Extremamente leniente Similar a um navegador Cria HTML5 válido	Lento Dependência externa de Python

Beautiful Soup 4

Criar um ambiente virtual

```
conda create --name ambiente-cpa-p3 python 3.12
```

Ativar o ambiente

```
conda activate ambiente-cpa-p3
```

Instalando Beautiful Soup:

```
conda install bs4 html5lib
```

Beautiful Soup 4 – Tipos de Objetos

- BeautifulSoup
  - É o objeto que representa o documento como um todo;
  - Para todos as questões práticas podemos “tratá-lo como uma tag”;
  - Para iniciar o uso do BeautifulSoup criamos uma instância desse objeto

```
with open('html_exemplo.html', 'r') as f:
    soup = BeautifulSoup(f, 'html5lib')
print(soup.prettify())
```

```
<html>
 <head>
  <title>
   The Dormouse's story
  </title>
 </head>
 <body>
  <p class="title">
```

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

A hierarquia do dom é preservada no BeautifulSoup, podemos acessar diferentes tags pelo nome. Porém no caso de termos mais de um filho, acessamos apenas o primeiro.

```
with open('html_exemplo.html', 'r') as f:
    soup = BeautifulSoup(f, 'html5lib')
soup.body.p.b.text
```

```
"\n The Dormouse's story\n "
```

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

Podemos acessar os atributos de forma semelhante com a qual acessamos os valores de um dicionário.

```
soup = BeautifulSoup(html_doc,'html5lib')
soup.body.a['href']
```

```
['http://example.com/elsie']
```

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

Para acessar todos os filhos de uma tag, podemos utilizar o método .contents ou o gerador de listas .children

```
head_tag = soup.head
head_tag
```

```
<head><title>The Dormouse's story</title></head>
```

```
head_tag.contents
```

```
[<title>The Dormouse's story</title>]
```

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

Em alguns casos podemos querer também os *"filhos dos filhos "*, nesse caso podemos utilizar o método *.descendants*

```
for child in head_tag.descendants:
    print(child)
```

```
<title>The Dormouse's story</title>
The Dormouse's story
```

Beautiful Soup 4 – Tipos de Objetos

- NavigableString
  - Representa texto que está dentro de tags
  - Pode ser convertida para string usando a função *str()*
  - Não pode ser editada, mas pode ser substituída usando *.replace_with*

```
soup = BeautifulSoup('<b class="boldest">Extremely bold</b>', 'html5lib')
tag = soup.b
tag.string
```

```
'Extremely bold'
```

```
type(tag.string)
```

```
bs4.element.NavigableString
```

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

Quando a tag possui apenas uma NavigableString como filho, podemos acessar pelo *.string,* caso possua mais de um podemos acessar via *.strings* e *.stripped_strings*

title_tag.string

The Dormouse's story

Beautiful Soup 4 – Tipos de Objetos

- Tag # Corresponde a uma tag do documento html ou xml
  - Name # Toda tag possui um nome
  - Attributes # Tags podem possuir um ou mais atributos

Podemos visitar também tags irmãs acessando os métodos *.next_sibling**s*** e *.previous_sibling**s***

```
link = soup.a
link
```

```
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>
```

```
link.next_sibling
```

```
',\n'
```

Beautiful Soup 4 – Movimentos

- Para acessar os filhos:
  - Pelo objeto diretamente: `soup.body.b`
  - Pela lista .contents: `soup.head.contents`
  - Pelo iterator: `soup.head.children`
  - Pelos decendentes usando: `soup.head.descendants`

- Para acessar os irmãos:
  - Próximo irmão(s): `soup.body.a.next_sibling(s)`
  - Irmão anterior(es): `soup.body.a.previous_sibling(s)`

- Para acessar os pais:
  - Pelo objeto parent: `soup.head.parent`
  - Pelo iterador parents: `soup.head.parents`

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;

```
soup.find_all('b')
[<b>The Dormouse's story</b>]
```

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;
  - Regex: Match com a regex;

```
import re
for tag in soup.find_all(re.compile("^b")):
    print(tag.name)
```

```
body
b
```

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;
  - Regex: Match com a regex;
  - Lista: Match com um elemento qualquer da lista;

```
soup.find_all(["a", "b"])
```

```
[<b>The Dormouse's story</b>,
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;
  - Regex: Match com a regex;
  - Lista: Match com um elemento qualquer da lista;
  - True

```
for tag in soup.find_all(True):
    print(tag.name)
```

```
html
head
title
body
p
b
p
a
a
a
p
```

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;
  - Regex: Match com a regex;
  - Lista: Match com um elemento qualquer da lista;
  - True
  - Função: Match com elementos que retornam true quando passados para a função.

```
def has_class_but_no_id(tag):
   return tag.has_attr('class') and not tag.has_attr('id')
soup.find_all(has_class_but_no_id)
```

```
[<p class="title"><b>The Dormouse's story</b></p>,
 <p class="story">Once upon a time there were three little sisters; and their names were
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>;
 and they lived at the bottom of a well.</p>,
 <p class="story">...</p>]
```

Beautiful Soup - Buscando tags

- Diversas funções de busca
  - find e find_all
- Tipos de filtros
  - String: Match com a string inteira;
  - Regex: Match com a regex;
  - Lista: Match com um elemento qualquer da lista;
  - True
  - Função: Match com elementos que retornam true quando passados para a função.

```
from bs4 import NavigableString
def surrounded_by_strings(tag):
   return (isinstance(tag.next_element, NavigableString)
           and isinstance(tag.previous_element, NavigableString))

for tag in soup.find_all(surrounded_by_strings):
   print(tag.name)
```

```
body
p
a
a
a
p
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name: para filtrar apenas tags com o nome específico;*
  - *attrs:*
  - *recursive:*
  - *string:*
  - *limit:*
  - ***kwargs.*

```
soup.find_all("title")
```

[<title>The Dormouse's story</title>]

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs: utilizado para realizar filtros de atributos;*
  - *recursive:*
  - *string:*
  - *limit:*
  - ***kwargs.*

```
soup.find_all(attrs={"class": "sister"})
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs:*
  - *recursive: pesquisar apenas na tag ou em seus descendentes*
  - *string:*
  - *limit:*
  - ***kwargs.*

```
soup.html.find_all("title")
```

```
[<title>The Dormouse's story</title>]
```

```
soup.html.find_all("title", recursive=False)
```

```
[]
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs:*
  - *recursive:*
  - *string: pesquisar pelo conteúdo/string das tags*
  - *limit:*
  - ***kwargs.*

```
soup.find_all(string="Elsie")
```

```
['Elsie']
```

```
soup.find_all(string=["Tillie", "Elsie", "Lacie"])
```

```
['Elsie', 'Lacie', 'Tillie']
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs:*
  - *recursive:*
  - *string: pesquisar pelo conteúdo/string das tags*
  - *limit:*
  - ***kwargs.*

```
import re
soup.find_all(string=re.compile("Dormouse"))
```

["The Dormouse's story", "The Dormouse's story"]

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs:*
  - *recursive:*
  - *string: pesquisar pelo conteúdo/string das tags*
  - *limit:*
  - ***kwargs.*

```
def is_the_only_string_within_a_tag(s):
    """“Retorna True se a string for o filho único da tag pai."""
    return (s == s.parent.string)

soup.find_all(string=is_the_only_string_within_a_tag)
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name:*
  - *attrs:*
  - *recursive:*
  - *string:*
  - *limit: limita o número de retornos do find_all*
  - ***kwargs.*

```
soup.find_all("a", limit=2)
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name: para filtrar apenas tags com o nome específico;*
  - *attrs:*
  - *recursive:*
  - *string:*
  - *limit:*
  - ***kwargs: Todos os parâmetros nomeados não conhecidos são convertidos para filtros de atributos.*

```
soup.find_all(id='link2')
```

```
[<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name: para filtrar apenas tags com o nome específico;*
  - *attrs:*
  - *recursive:*
  - *string:*
  - *limit:*
  - ***kwargs: Todos os parâmetros nomeados não conhecidos são convertidos para filtros de atributos.*

```
soup.find_all(href=re.compile("elsie"))
```

[Elsie]

Beautiful Soup - Buscando tags

- *find_all(name, attrs, recursive, string, limit, **kwargs)*
  - *name: para filtrar apenas tags com o nome específico;*
  - *attrs:*
  - *recursive:*
  - *string:*
  - *limit:*
  - ***kwargs: Todos os parâmetros nomeados não conhecidos são convertidos para filtros de atributos.*

```
soup.find_all(class_="sister")
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Beautiful Soup - Buscando tags

Variações de busca:

- find e find_all
- find_parent e find_parents
- find_next_sibling e find_next_siblings
- find_previous_sibling e find_previous_siblings
- find_previous e find_all_previous
- find_next e find_all_next

Exercícios – Usando find_all

Usando find_all selecione:

a) Todas as tags da classe story
b) Tag com id link2 e as suas irmãs subsequentes

```
<html>
  <head><title>The Dormouse's story</title></head>
  <body>
    <p class="title"><b>The Dormouse's story</b></p>
    <p class="story">Once upon a time there were three little sisters; and their names were
    <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
    <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
    <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
    and they lived at the bottom of a well.</p>
    <p class="story">...</p>
  </body>
</html>
```

Busca usando seletores CSS

- *.select* e *.select_one*
- https://facelessuser.github.io/soupsieve/
- https://www.w3.org/TR/selectors-3

Seletores CSS

Seletores são padrões textuais que casam com algum elemento de uma árvore (de um arquivo XML ou HTML). Fonte: W3C

Podemos ter os seguintes tipos de seletores:

- **Seletores Simples:** Por tipo, universal, por id, por classe, por atributo ou pseudo-classe;
- **Seletores Compostos:** É uma sequência de seletores simples (por exemplo por tipo e por classe);
- **Lista de Seletores:** É formada por uma lista de seletores separadas por vírgula
- **Seletores Complexos**: É formado por múltiplos seletores simples ou compostos junto de **combinadores**.

Cascading Style Sheets (CSS)

- Seletores na prática apontam para qual elemento HTML queremos definir o estilo.
- Podem ser definidos para todos elementos de um tipo, para uma classe, para um id e de muitas outras formas.

```
p {
  text-align: center;
  color: red;
}
```

Estilo vale para todos elementos do tipo parágrafo
<p>

```
p.center {
  text-align: center;
  color: red;
}
```

Estilo vale para todos parágrafos com a classe “center”
<p class=“center”>

```
.center {
  text-align: center;
  color: red;
}
```

Estilo vale para qualquer elemento com com a classe “center”
<p class=“center”>
<h1 class=“center”>

```
#para1 {
  text-align: center;
  color: red;
}
```

Estilo só vale para o elemento com o id único “para1”
<p id=“para1”>

Seletores CSS

Os exemplos serão apresentados considerando as funções *select* e *select_one* do Beautiful Soup, porém esses mesmos seletores podem ser usados com Selenium ou mesmo diretamente no navegador.

Seletores Simples

Seletor por tipo

- Define qual a tag será buscada;
- Exemplo: title

```
soup.select('title')
```

```
[<title>The Dormouse's story</title>]
```

Seletores Simples

Seletor universal

- Seleciona todas as tags;
- *

```
soup.select('*')
```

```
[<html><head><title>The Dormouse's story</title></head> <body>
<p class="title"><b>The Dormouse's story</b></p>
<p class="story">Once upon a time there were three little sisters; and their names were
<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
..
```

Seletores Simples

Seletor de atributo

- Seleciona todas as tags cujos atributos casarem com o seletor
- Exemplos:
  - [attr] Casa com todas tags que possuírem attr independente do valor;
  - [attr=value] Casa com todas as tags que possuírem attr com valor exatamente igual a value;
  - [attr~=value] Casa com todas as tags que possuírem como conteúdo de attr uma lista separada por espaços que contém value.

```
soup.select('[id]')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Seletores Simples

Seletor de atributo por substring

- Seleciona todas as tags cujos atributos casarem com o seletor
- Exemplos:
  - [attr^=value]: Deve possuir attr começando com o valor value;
  - [attr$=value]: Deve possuir attr finalizando com o valor value;
  - [attr*=value]: Deve possuir attr com pelo menos uma ocorrência do valor value.

```
soup.select('[class^=st]')
```

```
[<p class="story">Once upon a time there were three little sisters; and their names were
 <a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a> and
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>; and they lived
 at the bottom of a well.</p>,
<p class="story">...</p>]
```

Seletores Simples

Seletor de atributo por classe

- É mais usado do que a notação class~=value
- Exemplos:
  - .sister: Deve possuir sister no atributo class;

```
soup.select('.sister')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Seletores Simples

Seletor de id

- Seleciona tags considerando um id específico
- Exemplos:
  - #submit: Deve possuir id=‘submit’;

```
soup.select('#link2')
```

```
[<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

Seletores Compostos

- Seletores que combinam dois ou mais seletores simples sem uso de combinadores.
- Sempre iniciam com um seletor universal ou com um seletor de tipo.
- Exemplo:
  - a[href$=tillie]: tags a com href terminando em “tillie”.

```
soup.select('a[href$=tillie]')
```

```
[<a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Lista (Grupo) de Seletores

- Lista de dois ou mais seletores simples e/ou compostos separados por vírgulas;
- Equivalente a um “ou”. Seleciona tags que casem com qualquer um dos seletores na lista.

- Exemplo:
  - #link1,#link2: tags id igual a link1 ou id igual a link2.

```
soup.select('#link1,#link2')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>]
```

Seletores com combinadores (complexos)

- Combina seletores Simples ou Compostos de forma a explorar sua hierarquia.

- Combinador espaço ‘ ‘
- Indica que o elemento a direita é descendente (não necessariamente direto) do elemento da esquerda
- Exemplo:
  - body a: seleciona todas as tags a dentro de body

```
soup.select('body a')
```

```
[<a class="sister" href="http://example.com/elsie" id="link1">Elsie</a>,
 <a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Seletores com combinadores (complexos)

- Combina seletores Simples ou Compostos de forma a explorar sua hierarquia.

- Combinador filho ‘>‘
- Indica que o elemento a direita é descendente direto (filho) do elemento da esquerda
- Exemplo:
  - head > title

```
soup.select('head > title')
```

```
[<title>The Dormouse's story</title>]
```

Seletores com combinadores (complexos)

- Combina seletores Simples ou Compostos de forma a explorar sua hierarquia.

- Combinador irmão subsequente ‘~‘
- Indica que o elemento a direita sucede o elemento da esquerda (podem haver elementos intermediários) e ambos tem o mesmo elemento pai
- Exemplo:
  - #link1 ~ .sister: seleciona o elemento que possui a classe sister e é irmão a direita do elemento com id=link1

```
soup.select('#link1 ~ .sister')
```

```
[<a class="sister" href="http://example.com/lacie" id="link2">Lacie</a>,
 <a class="sister" href="http://example.com/tillie" id="link3">Tillie</a>]
```

Exercícios – Usando seletores CSS

Crie um seletor CSS para cada item abaixo.

a) Todas as tags b que estão dentro de um p
b) Todas as tags da classe story
c) Tag com id link2 e as suas irmãs subsequentes

```
<html>
  <head><title>The Dormouse's story</title></head>
  <body>
    <p class="title"><b>The Dormouse's story</b></p>
    <p class="story">Once upon a time there were three little sisters; and their names were
    <a href="http://example.com/elsie" class="sister" id="link1">Elsie</a>,
    <a href="http://example.com/lacie" class="sister" id="link2">Lacie</a> and
    <a href="http://example.com/tillie" class="sister" id="link3">Tillie</a>;
    and they lived at the bottom of a well.</p>
    <p class="story">...</p>
  </body>
</html>
```