

PYTHON – AULA 3

WEB SCRAPING

AGENDA

Aula 1

Introdução – conceitos, arquitetura cliente – servidor, Introdução a HTML, CSS e JavaScript, URLLIB e Requests.

Aula 2

Introdução ao BeautifulSoup, Expressões Lambdas, Expressões regulares (REGEX)

Aula 3

Beautiful Soup, List Comprehension, Revisitando NUMPY e PANDAS, Introdução ao Selenium

Aula 4

Selenium 4, Scraping de Imagens, Trabalhando com inputs em pesquisas, Introdução ao Docker.

Aula 5

Selenium 4, Banco de dados SQLite, projeto final, considerações finais.

WEB SCRAPING

BEAUTIFUL SOUP

LIST COMPREHENSION

REVISITANDO NUMPY E PANDAS

INTRODUÇÃO AO SELENIUM

BEAUTIFUL SOUP

- É uma biblioteca Python para extrair dados de arquivos HTML e XML.
- Vamos instalar a biblioteca **beautifulsoup4** pelo gerenciador de pacotes
- Após a instalação, vamos criar um novo arquivo `scraping_beautifulsoup.py`
- Devemos importar a biblioteca **BeautifulSoup** do módulo **bs4**.
- Enviamos o conteúdo da requisição e o parâmetro **html.parser**

```
import requests
from bs4 import BeautifulSoup

response = requests.get("https://books.toscrape.com/")
soup = BeautifulSoup(response.content, "html.parser")
```


• BEAUTIFUL SOUP - Parse tree

- Páginas web são estruturadas de forma hierárquica. Desse modo, temos uma árvore com seus nós e ramificações.
- Por meio de **tags** podemos organizar sua estrutura, e neste caso denominamos por pais, filhos e descendentes (parents, children e descendants)

```
<ul class="breadcrumb">
  <li>
    <a href="index.html">
      Home
    </a>
  </li>
  <li class="active">
    All products
  </li>
</ul>
```

• BEAUTIFUL SOUP - Parse tree

- + • Utilize a biblioteca **BeautifulSoup** para iterar com as tags. Experimente com **div**, conforme mostrado abaixo.

```
primeira_div = soup.div  
print(primeira_div.div.div.prettify())
```

- Faça o teste com outras **tags** que você conheceu.
- Para acessar o **CSS**, utilize o parâmetro **attrs**.

```
print(primeira_div.div.attrs)
```

• BEAUTIFUL SOUP - Parse tree

- + •
- + •
- Agora que podemos iterar pelas **tags** das páginas, vamos converter em uma lista para o Python

```
print(list(soup.ul))
```

- Perceba que os espaços vazios da página foram adicionados a nossa lista

```
['\n', <div class="row">
  <div class="col-sm-8 h1"><a href="index.html">Books to Scrape</a>
    <small>We love being scraped!</small>
  </div>
</div>, '\n']
```


BEAUTIFUL SOUP - Parse tree

- Para resolver, vamos aplicar um filtro e a biblioteca `NavigableString`.

```
print(list(filter(lambda x: type(x) != NavigableString, soup.ul)))
```

- A `NavigableString` retorna strings que estão dentro das **tags**
- Podemos aplicar este filtro para outras **tags**. Assim, crie uma função que recebe qualquer outro elemento para ser filtrado.

BEAUTIFUL SOUP - Parse tree

```
def filtra_espacos(elemento):  
    return list(filter(lambda x: type(x) != NavigableString,  
                        elemento))
```

```
lista_filtrada = filtra_espacos(soup.ul)  
print(lista_filtrada)
```

BEAUTIFUL SOUP - Parse tree

- Algumas vezes não queremos iterar de forma vertical (hierárquica) mas de forma horizontal (Siblings). Vamos analisar um item da lista

```
print(soup.ul.li)
```

```
<li>  
  <a href="index.html">Home</a>  
</li>
```

- Podemos utilizar **next_sibling** para navegar de forma horizontal

```
print(soup.ul.li.next_sibling.next_sibling)
```

```
<li class="active">All products</li>
```

BEAUTIFUL SOUP - Parse tree

Para retornar, vamos utilizar `previous_sibling`.

```
print(soup.ul.li.next_sibling.next_sibling.previous_sibling.previous_sibling)
```

```
<li>  
  <a href="index.html">Home</a>  
</li>
```

BEAUTIFUL SOUP - Parse tree

Para obter textos das **tags**, podemos simplesmente utilizar o atributo **text**.

```
print(soup.ul.text)
```

- Ou a função **get_text**, mais versátil e que possibilita a passagem de parâmetros para separar o texto e eliminar espaços vazios.

```
print(soup.ul.get_text(separator=", ", strip=True))
```


BEAUTIFUL SOUP - Parse tree

Se a intenção for extrair todo o texto da página e converter em uma lista de **strings** para o python, podemos utilizar **stripped_strings**

```
strings = list(soup.stripped_strings)
print("Numero de strings: ", len(strings))
print(strings)
```

BEAUTIFUL SOUP - Parse tree

- Quando fazemos **scraping** da web, normalmente queremos algo específico, desse modo, precisamos refinar nossas buscas, por meio de pesquisas.
- Vamos utilizar a função **find_all** para filtrarmos os preços dos livros.

```
print(soup.find_all("p", attrs={"class":"price_color"}))
```

- Podemos encontrar os parâmetros, acessando o site no modo de inspeção.

```
[<p class="price_color">£51.77</p>, <p class="price_color">£53.74</p>,  
<p class="price_color">£50.10</p>, <p class="price_color">£47.82</p>, <p  
class="price_color">£54.23</p>, <p class="price_color">£22.65</p>, ...]
```

BEAUTIFUL SOUP - Parse tree

- O resultado é uma lista com as **tags** contendo os preços. Vamos refinar iterando por meio de um **FOR** e utilizando a conhecida função **get_text**.

```
precos = soup.find_all("p", attrs={"class":"price_color"})
for preco in precos:
    print(preco.get_text())
```

- Perceba a diferença.
- Nota: dentro de uma lista podemos utilizar a função **find** para iterar com elementos aninhados.

BEAUTIFUL SOUP

- Conseguimos extrair os dados do site e transforma-los em uma lista de **strings** no Python.
- Entretanto, pode acontecer de precisarmos limpar alguns dados e converte-los em valores numéricos.
- Veja o preço dos livros acompanhados da representação monetária (libras)

'£37.59'

BEAUTIFUL SOUP

- Uma solução interessante é utilizar um padrão denominado Regular Expression (Regex).
- As expressões regulares são padrões para selecionar caracteres em uma string.
- Desse modo, podemos emprega-las para remover caracteres inválidos.

BEAUTIFUL SOUP

Algumas classes

- [0-9] todos os números devem ser destacados
- [a-z0-9] busca letras e números
- [A-z] todas as letras do alfabeto;
- [a-z] busca todas as letras minúsculas;
- [A-Z] busca todas as letras maiúsculas;
- [^as] ignora todas as combinações que tenham "as"
- W reduz a busca em caracteres que não sejam alfanuméricos, como espaços e símbolos.

BEAUTIFUL SOUP

- Vamos utilizar a biblioteca Regex no Python.

```
import re
```

- Utilizamos a expressão regular para filtrar apenas números e pontos.

```
def limpa_preco(preco):  
    return float(re.sub("[^0-9.]", "", preco))
```

```
livro = extração_livros(livros[2])  
print(limpa_preco(livro[1]))
```

List Comprehension

- Oferece uma sintaxe mais curta quando você deseja criar uma nova lista com base nos valores de uma lista existente.

```
frutas = ["banana", "goiaba", "manga", "maracujá"]
nova_lista = []
for i in frutas:
    if "m" in i:
        nova_lista.append(i)
print(frutas)
print(nova_lista)
```

List Comprehension

- Podemos substituir essa sintaxe por uma mais enxuta

```
nova_lista = [i for i in frutas if "m" in i]  
print(frutas)  
print(nova_lista)
```

List Comprehension – Exercícios

- Vamos praticar algumas análises por meio de List Comprehension.
- Crie um novo arquivo `scraping.py` e aloque as funções `limpa preço`, `converte avaliação` e `extração livros dict`.
- Crie um novo arquivo `scraping_analises.py`. Neste arquivo faça o scraping e gere a lista de livros; faça as análises conforme descrito no próximo slide.

List Comprehension - Exercícios

1. Crie uma lista ordenada com os preços da lista de livros (lista de dicionários)
2. Utilize SUM para exibir a soma dos valores dos livros contidos na lista
3. Faça a média aritmética simples dos preços da lista de livros.
4. Pesquise na lista de livros aqueles com valores inferiores ao valor médio; exibir os títulos.

List Comprehension - Exercícios

```
import requests
from bs4 import BeautifulSoup
from scraping import extracao_livros_dict

response = requests.get("https://books.toscrape.com/")
soup = BeautifulSoup(response.content, "html.parser")
livros = soup.find_all("article", attrs={"class": "product_pod"})
lista_livros = [extracao_livros_dict(livro) for livro in livros]
```

List Comprehension - Exercícios

#1

```
preco_livros = sorted([livro["preço"] for livro in lista_livros])  
print("Lista de preços: ", preco_livros)
```

#2

```
preco_soma_livros = sum([livro["preço"] for livro in lista_livros])  
print("Soma dos preços: ", preco_soma_livros)
```

#3

```
preco_medio_livros = sum([livro["preço"] for livro in lista_livros]) / len(lista_livros)  
print("Preço médio: ", preco_medio_livros)
```

#4

```
preco_menores_media_livros = [livro["titulo"] for livro in lista_livros if  
livro["preço"] < preco_medio_livros]  
print(preco_menores_media_livros)
```

Revisitando NUMPY e PANDAS

```
dataframe = pd.DataFrame(  
    {  
        "A": 1.0,  
        "B": pd.Timestamp("20210101"),  
        "C": pd.Series([1.2, 3.7, 5.5, 6], dtype="float32"),  
        "D": np.array([12, 5, 6, 9], dtype="int32"),  
        "E": pd.Categorical(["novo", "usado", "usado", "novo"])  
    }  
)  
print("Abaixo está nosso dataframe construído a partir de um dicionário")  
print(dataframe)  
  
print(dataframe.describe())  
  
print("Exportando dataframe para o formato XLSX")  
dataframe.to_excel("arquivo_excel.xlsx", sheet_name="Sheet1")  
  
print("Convertendo o dataframe para o formato JSON")  
print(dataframe.to_json())
```

Exercícios – NUMPY e PANDAS

1. Faça a conversão da lista de livros para um dataframe

2. Com o dataframe :

- Calcule a média dos preços
- Faça a estatística descritiva
- Exporte para o MS Excel
- Exporte para JSON

Web Scraping - praticando

- Vamos praticar os conhecimentos adquiridos em um novo site.
- Que tal um emprego para trabalhar com Python ?



The Free Python Job Board

for the global Python community

<https://pythonjobs.github.io/>

Web Scraping - praticando

1. Acesse o site e investigue sua estrutura
2. Faça o scraping com BeautifulSoup
3. Faça a extração dos Jobs (identifique a classe)
4. Imprima os Jobs
5. Crie uma função para filtrar os dados
6. Filtre os dados e salve em um dicionário
7. Faça buscas por empregos remotos

Scraping em sites com grande volume de dados

- Imagine fazer scraping do Wikipedia.
- Segundo a Wikimedia Foundation, a Wikipedia recebe mais de 2500 hits por segundo.
- Com frequência a Wikipedia é usada para demonstração de Scraping, por ser estável e simples; contudo são disponibilizadas API's com os mesmos dados de forma mais simples, rápida e eficaz.
- A bolsa de valores B3 também disponibiliza API's para scraping.
- Mas e o que dizer da **Amazon ou do Youtube?**

Scraping em sites com grande volume de dados

- Normalmente Amazon, Google, Youtube etc, bloqueiam scraoings, pois muitas aplicações abusam e se passam por pessoas reais.
- Vamos fazer um scraping da seção de livros do site da Amazon

```
import requests
from bs4 import BeautifulSoup

response =
requests.get("https://www.amazon.com.br/Livros/b/?ie=UTF8&node=6740748011&ref_=nav_cs_books")
soup = BeautifulSoup(response.content, "html.parser")

print(soup.prettify())
```

Scraping em sites com grande volume de dados

Note que não conseguimos extrair os dados sobre os livros e obtivemos um alerta: **“To discuss automated access to Amazon data please contact api-services-support@amazon.com.”**

Para resolver esse problema, precisamos “informar” que não temos intenções maliciosas.

Vamos utilizar um **HEADERS** informando que as requisições não estão vindo de um robô.

Acesse o site da amazon na seção de livros no modo de inspeção;
Clique em network.

Scraping em sites com grande volume de dados

The screenshot shows the Amazon Brazil homepage (amazon.com.br) with a search bar and navigation links. The developer tools are open, displaying the Network tab. The selected request is expanded, showing the following headers:

- accept-language:** pt-BR,pt;q=0.9,en-US;q=0.8,en;q=0.7
- cookie:** session-id=143-1247843-0696849; session-id-time=20827872011; i18n-prefs=BRL; ubid-acbbr=134-1546013-3535335; session-token=aL75087YBeYxI4rI48rPLh/TIU8AXbeKTGJ/zYH2eU/H0s0YYvJIw0zDZsVgQ9p77v3+gR1c1IFRJTJM7vxFI39zrLWnbhPnPFxxNZSRa7KqeyiHvJyaAUyQhBJQdTz09Nxl0Hb/FuF8DiCi4gQXcLSvC2oRNFmQ1ZPjNg7IESG18t2NfidV94qV+ewhWlyptbSE71958SsNg3Ax8aMYmeLrvfEmNYsm/7fs
- dnt:** 1
- referer:** https://www.amazon.com.br/
- sec-ch-ua:** "Google Chrome";v="111", "Not(A:Brand";v="8", "Chromium";v="111"
- sec-ch-ua-mobile:** ?1
- sec-ch-ua-platform:** "Android"
- sec-fetch-dest:** image
- sec-fetch-mode:** no-cors
- sec-fetch-site:** same-site
- user-agent:** Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36

The user-agent string is highlighted in blue. The console shows a message: "before a response was received".

Scraping em sites com grande volume de dados

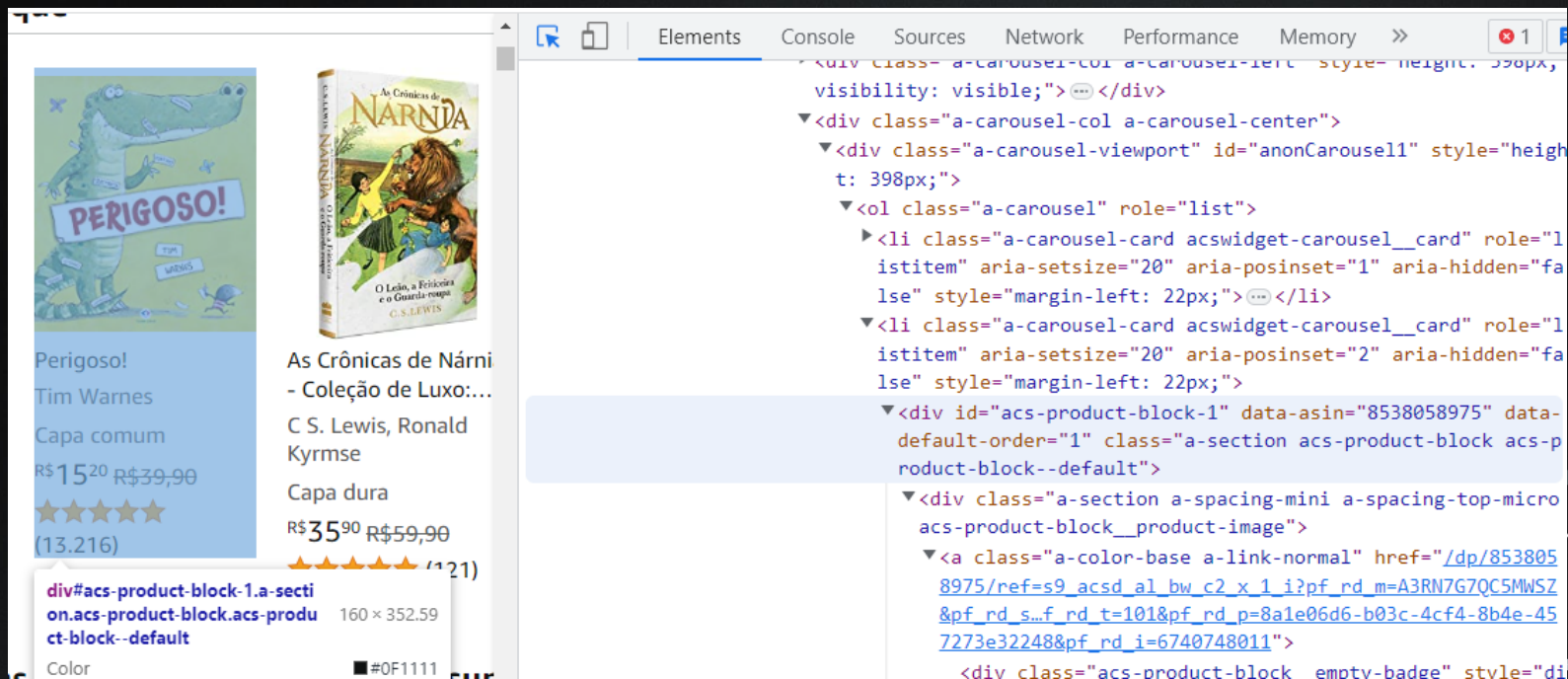
- Vamos adicionar o **Headers** na requisição.

```
HEADERS = {"user-agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5  
Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0  
Mobile Safari/537.36"})  
response =  
requests.get("https://www.amazon.com.br/Livros/b/?ie=UTF8&node=674074  
8011&ref_=topnav_storetab_b", headers=HEADERS)  
soup = BeautifulSoup(response.content, "html.parser")
```

- Nota: ao copiar e colar, fique atento em separar o **user-agent** do restante da string.

Scraping em sites com grande volume de dados

Ao inspecionar a div dentro do carousel, encontramos seu id = `acs-product-block-1`



The image shows a web browser window with two book listings on the left and the corresponding HTML code in the developer tools on the right.

Book Listing 1 (Left):

- Image: A green crocodile holding a sign that says "PERIGOSO!" (Dangerous!).
- Title: Perigoso!
- Author: Tim Warnes
- Price: R\$15²⁰ R\$39,90
- Rating: 5 stars (13.216 reviews)

Book Listing 2 (Right):

- Image: The cover of "As Crônicas de Nárnia: O Leão, a Feiticeira e o Guarda-roupa" by C.S. Lewis.
- Title: As Crônicas de Nárnia - Coleção de Luxo...
- Author: C S. Lewis, Ronald Kymrse
- Price: R\$35⁹⁰ R\$59,90
- Rating: 5 stars (121 reviews)

Developer Tools (Right):

The "Elements" panel shows the HTML structure of the carousel. The selected element is a `div` with the following attributes:

```
<div class="a-carousel-col a-carousel-center" style="height: 398px; visibility: visible;">
```

Inside this `div`, there is a `div` with the following attributes:

```
<div class="a-carousel-viewport" id="anonCarousel1" style="height: 398px;">
```

Inside this `div`, there is a `ol` with the following attributes:

```
<ol class="a-carousel" role="list">
```

Inside the `ol`, there are two `li` elements. The second `li` contains the `div` with the following attributes:

```
<div id="acs-product-block-1" data-asin="8538058975" data-default-order="1" class="a-section acs-product-block acs-product-block--default">
```

This `div` contains a `div` with the following attributes:

```
<div class="a-section a-spacing-mini a-spacing-top-micro acs-product-block__product-image">
```

Inside this `div`, there is an `a` element with the following attributes:

```
<a class="a-color-base a-link-normal" href="/dp/8538058975/ref=s9_acsd_al_bw_c2_x_1_i?pf_rd_m=A3RN7G7QC5MWSZ&pf_rd_s=pf_rd_t=101&pf_rd_p=8a1e06d6-b03c-4cf4-8b4e-457273e32248&pf_rd_i=6740748011">
```

The `div` with the following attributes is also visible:

```
<div class="acs-product-block_empty-badge" style="di
```

Scraping em sites com grande volume de dados

• Ao acessar o container conseguimos acessar os dados sobre o livro.

```
HEADERS = ({ "user-agent": "Mozilla/5.0 (Linux; Android 6.0; Nexus 5 Build/MRA58N) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/111.0.0.0 Mobile Safari/537.36" })
response =
requests.get("https://www.amazon.com.br/Livros/b/?ie=UTF8&node=6740748011&ref_=nav_cs_books", headers=HEADERS)
soup = BeautifulSoup(response.content, "html.parser")
livro = soup.find("div", id="acs-product-block-1")
print("Livro: ", livro.prettify())
```

Scraping em sites com grande volume de dados

```
titulo = livro.find(attrs={"class":"a-truncate-full"}).getText()
avaliacao = livro.find("span", attrs={"class":"acs-product-block__rating__review-
count"}).getText()
preco=livro.find("span", attrs={"class":"a-price a-text-price acs-product-block__price-
-strikethrough"}).find(attrs={"class":"a-offscreen"}).getText()
print("Título: ", titulo)
print("Preço: ", preco)
print("Numero de avaliações:", avaliacao)
```

Nota: para este exemplo, os dados foram extraídos do *carousel* da seção de destaques. Pode acontecer de o livro não estar disponível nessa seção em próximas consultas. Desse modo, você pode ir direto ao produto específico de consulta, exceto se quiser fazer scraping dos destaques.

Web Scraping – praticando na Amazon

1. Escolha um único produto com valor inferior a R\$ 50,00, no site da Amazon.
2. Faça o scraping
3. Inspecione a página e busque pelas variáveis título, preço, categoria, avaliação e disponibilidade.
4. Crie um alerta para verificar se o produto está disponível e se o preço é menor do que R\$50,00; caso seja, exiba os dados e informe que o produto é acessível.

Web Scraping – Selenium



Selenium

- O Selenium é uma biblioteca de testes para manipulação de páginas dinâmicas.
- Automatiza browsers
- Permite simular um usuário real utilizando um navegador
- Mais indicado para sites que contêm muito código Javascript, JQuery, Angular, Vue e React.

Para utilizar o Selenium no Python:

1. Instale a biblioteca pelo gerenciador de pacotes
2. Acesse: <https://pypi.org/project/selenium/>
3. Faça a escolha do driver para seu navegador; ao extrair direcione para a pasta com o python instalado.

Web Scraping – Selenium

Para localizar a instalação do Python na sua máquina, rode o código abaixo no Prompt de Comando

```
CA: Prompt de Comando - python
Microsoft Windows [versão 10.0.19044.2728]
(c) Microsoft Corporation. Todos os direitos reservados.

C:\Users\Emerson Abraham>python
Python 3.10.6 (tags/v3.10.6:9c7b4bd, Aug 1 2022, 21:53:49) [MSC v.1932 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> import sys
>>>
>>> print(os.path.dirname(sys.executable))
C:\Users\Emerson Abraham\AppData\Local\Programs\Python\Python310
>>> _
```

Web Scraping – Selenium

Vamos acessar a página de relação com os investidores.

```
from selenium import webdriver
```

```
navegador = webdriver.Edge()
```

```
navegador.get("https://ri.magazineluiza.com.br/#")
```

BIBLIOGRAFIA BÁSICA⁺

- • + BEAZLEY, David. *Python Essential Reference*, 2009.
- + • • BEK, ANDY. *The Ultimate Web Scraping With Python Bootcamp 2023*.
- BHARGAVA, ADITYA Y. *Entendendo Algoritmos. Um guia ilustrado para programadores e outros curiosos*. São Paulo: Ed. Novatec, 2017
- | • DOWNEY, ALLEN B. *Pense em Python. Pense como um cientista da computação*. São Paulo: Ed. Novatec, 2016
- + • KOPEC, DAVID. *Problemas clássicos de ciência da computação com Python*. São Paulo: Ed. Novatec, 2019
- MCKINNEY, WILLIAM WESLEY. *Python para análise de dados. Tratamento de dados com Pandas, Numpy e Ipython*. São Paulo: Ed. Novatec, 2018
- MITCHELL, RYAN. *Web Scraping com Python. Coletando mais dados na web moderna*. São Paulo: Ed. Novatec, 2019
- W3Schools. Disponível em: https://www.w3schools.com/python/python_lists_comprehension.asp, acesso: 03/2023

OBRIGADO



FIAP

Copyright © 2023 | Professor Dr. Emerson R. Abraham

Todos os direitos reservados. A reprodução ou divulgação total ou parcial deste documento é expressamente proibida sem o consentimento formal, por escrito, do professor/autor.

