Fundamentos Matemáticos e Computacionais de Machine Learning

Especialização em Machine Learning e Big Data

UEL

Profa. Dra. Juliana Felix

jufelix16@uel.br

Web Scraping

Introdução

- Quando realizamos tarefas de data science, é comum querermos utilizar dados encontrados na Internet. Você normalmente poderá acessar esses dados no formato .CSV ou por APIs.
- No mundo de hoje, temos toneladas de dados/informações não estruturadas (principalmente dados da web) disponíveis gratuitamente.

Introdução

- Existem momentos em que os dados que você quer somente podem ser acessados como parte de uma página web.
- Às vezes, os dados disponíveis gratuitamente são fáceis de ler e às vezes não.

Introdução

- Não importa como seus dados estejam disponíveis, o **web scraping** é uma ferramenta muito útil para transformar dados não estruturados em dados estruturados, mais fáceis de ler e analisar.
- Em outras palavras, uma forma de coletar, organizar e analisar essa enorme quantidade de dados é por meio do web scraping.

O que é web-scraping?

- Scraping (ou raspagem) é simplesmente um processo de extração (de vários meios), cópia e triagem de dados.
- Quando fazemos scraping ou extraímos dados ou feeds da web (como de páginas da web ou sites), isso é denominado *web scraping*.

O que é web scraping?

- Portanto, o web scraping é a extração de dados da web ou colheita na web.
- Resumindo, o web scraping fornece aos desenvolvedores uma maneira de coletar e analisar dados da Internet.

Por que realizar Web Scrapping?

- Webscraping fornece uma das grandes ferramentas para automatizar a maioria das coisas que um ser humano faz durante a navegação.
- O web-scraping é usado de várias maneiras
  - Dados para Pesquisa
    - O analista inteligente (como pesquisador ou jornalista) usa o web scraper em vez de coletar e limpar manualmente os dados dos sites.
  - Preços de produtos e comparação de popularidade
    - Atualmente, existem alguns serviços que usam web scrappers para coletar dados de vários sites online e usá-los para comparar a popularidade e os preços dos produtos.

Por que realizar Web Scraping?

- Motores de busca
  - Existem algumas grandes empresas de TI cujos negócios dependem exclusivamente da raspagem da web.
- Vendas e Marketing
  - Os dados coletados por meio do web scraping podem ser usados por profissionais de marketing para analisar diferentes nichos e concorrentes ou pelo especialista em vendas para vender marketing de conteúdo ou serviços de promoção de mídia social.

Python Selenium

Selenium

- O Selenium é uma ferramenta poderosa para controlar o navegador da Web por meio de um programa. É funcional para todos os navegadores, funciona em todos os principais sistemas operacionais e seus scripts são escritos em várias linguagens, como Python, Java, C#, etc.
    - Trabalharemos com Python.
- O Selenium tem quatro componentes principais – Selenium IDE, Selenium RC, Selenium Web driver, Selenium GRID.

Selenium IDE

- O Selenium IDE (Integrated Development Environment) é a principal ferramenta do Selenium Suite. É um ambiente de desenvolvimento integrado (IDE) completo para testes do Selenium.
- Ele é implementado como um complemento do Firefox e como uma extensão do Chrome.
- Ele permite a gravação, edição e depuração de testes funcionais. Anteriormente, era conhecido como Selenium Recorder.

Selenium RC

- O Selenium Remote Control (RC) é um servidor, escrito em Java, que aceita comandos para o navegador via HTTP.
- O RC possibilita escrever testes automatizados para um aplicativo da Web em qualquer linguagem de programação, o que permite uma melhor integração do Selenium em estruturas de teste de unidade existentes.

Selenium RC

- Para facilitar a escrita de testes, o projeto Selenium atualmente fornece drivers clientes para PHP, Python, Ruby, .NET, Perl e Java.
- O driver Java também pode ser usado com JavaScript (por meio do mecanismo Rhino).
- Uma instância do servidor Selenium RC é necessária para iniciar o caso de teste html – o que significa que a porta deve ser diferente para cada execução paralela.

Selenium Web Driver

- O Selenium WebDriver é o sucessor do Selenium RC. O Selenium WebDriver aceita comandos (enviados em Selenese ou por meio de uma API de cliente) e os envia para um navegador.
- Isso é implementado por meio de um driver de navegador específico do navegador, que envia comandos para um navegador e recupera os resultados.

Selenium Web Driver

- A maioria dos drivers de navegador realmente inicia e acessa um aplicativo de navegador (como Firefox, Google Chrome, Internet Explorer, Safari ou Microsoft Edge)
- O Selenium WebDriver não precisa de um servidor especial para executar os testes.
- Em vez disso, o WebDriver inicia diretamente uma instância do navegador e a controla.

Selenium Grid

- O Selenium Grid é um servidor que permite testes para usar instâncias do navegador da Web em execução em máquinas remotas.
- Com o Selenium Grid, um servidor atua como hub. Os testes entram em contato com o hub para obter acesso às instâncias do navegador.

Selenium Grid

- O hub possui uma lista de servidores que fornecem acesso às instâncias do navegador (nós WebDriver) e permite que os testes usem essas instâncias.
- O Selenium Grid permite executar testes em paralelo em várias máquinas e gerenciar diferentes versões e configurações do navegador centralmente (em vez de em cada teste individual).

Selenium - Características

- Código aberto e portátil
  - Selenium é um framework de teste da Web, portátil, e de código aberto.
- Combinação de ferramenta e DSL
  - O Selenium é a combinação de ferramentas e DSL (Domain Specific Language) para realizar diversos tipos de testes.
- Mais fácil de entender e implementar
  - Os comandos do Selenium são categorizados em termos de classes diferentes, o que os torna mais fáceis de entender e implementar.

Selenium - Características

- Reduz o tempo de execução do teste
  - O Selenium oferece suporte à execução de teste paralelo, o que reduz o tempo gasto na execução de testes paralelos.
- Menos recursos necessários
  - O Selenium requer menos recursos quando comparado a seus concorrentes como UFT, RFT, etc.
- Suporta várias linguagens de programação
  - C#, Java, Python, PHP, Ruby, Perl e JavaScript.

Selenium - Características

- Suporta vários sistemas operacionais
  - Android, iOS, Windows, Linux, Mac, Solaris.
- Suporta vários navegadores
  - Google Chrome, Mozilla Firefox, Internet Explorer, Edge, Opera, Safari, etc.
- Execução de Testes Paralelos
  - Também oferece suporte à execução de testes paralelos, o que reduz o tempo e aumenta a eficiência dos testes.

Selenium - Aplicações

- O Selenium WebDriver é usado para interagir com aplicativos web (automatizar e verificar funcionamento), e para extração de dados (scraping).
- Ele suporta muitos navegadores, como Firefox, Chrome, IE e Safari.

Instalando o Selenium

Instalando o Selenium

- O Selenium é uma biblioteca que não é instalada por padrão, assim, precisamos instalá-la primeiro.

- Todavia, é importante isolar nosso ambiente de trabalho para não atrapalhar a configuração existente, assim, é importante criar um novo ambiente virtual.

Instalando o Selenium

- Para criar um novo ambiente no conda você pode usar o Anaconda-Navigator
- Ou pode usar a linha de comando:
  - Conda create -name <nome do ambiente>

2 hifens (sinais -)

conda create --name *ws* python=*3.10 anaconda*

  - Este comando criará um ambiente com base no python corrente da sua distribuição conda.
- Não se esqueça de ativar seu novo ambiente ou as bibliotecas não serão encontradas

Instalando o Selenium

- O Selenium nada mais é do que uma biblioteca que permite com que o Python abra o seu navegador para executar os comandos desejados.
- Para isto basta digitar:
  - Com o PIP

    pip install selenium
  - Com o Conda

    conda install selenium

Instalando o Selenium

- Agora para que seja possível fazer essa interação do Python com o navegador, o Selenium Python precisa de um webdriver.
- O webdriver, que nada mais é do que um arquivo que permite essa interação do Python com o navegador.
- No entanto, navegadores diferentes vão precisar de webdrivers diferentes.

Instalando o Selenium

- Isso quer dizer que dependendo do navegador que estiver utilizando, ou queira utilizar, você terá que baixar um webdriver diferente.
- Neste caso, vamos dar exemplos dos dois mais utilizados:
  - Chromedriver para Google Chrome
  - Geckodriver para Firefox.
- Caso você utilize outro navegador, o procedimento é o mesmo, você pode ir ao buscador e pesquisar por webdriver + o nome do seu navegador ou olhar em:
  - https://github.com/SergeyPirogov/webdriver_manager

Instalando o Selenium

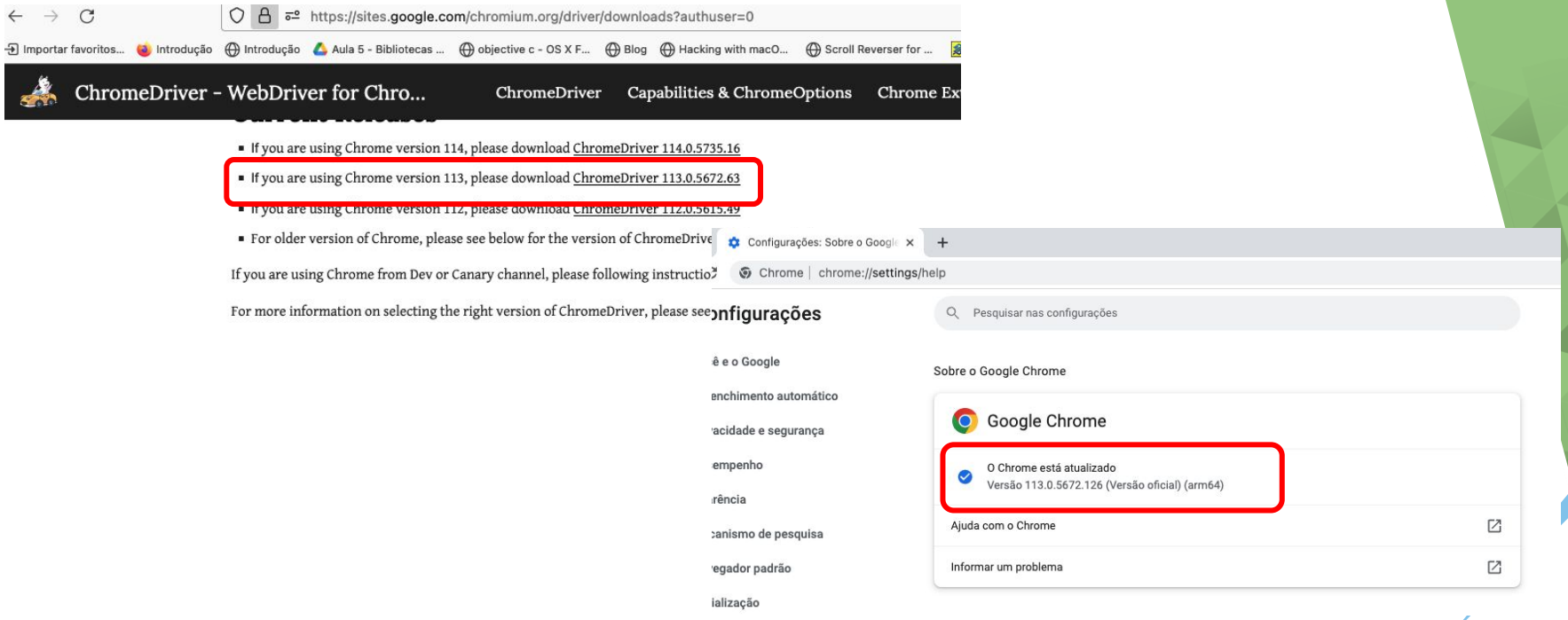
- Ao fazer o download do webdriver é MUITO IMPORTANTE verificar se:
  - Está fazendo o download da versão correta para
    - o seu sistema operacional
    - a versão do navegador

Instalando o Selenium

- Nos links a seguir você pode encontrar o Selenium para os navegadores mais populares
  - **Chrome**:
    - https://sites.google.com/chromium.org/driver/
  - **Edge**:
    - https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
  - **Firefox**:
    - https://github.com/mozilla/geckodriver/releases
  - **Safari**:
    - https://webkit.org/blog/6900/webdriver-support-in-safari-10/

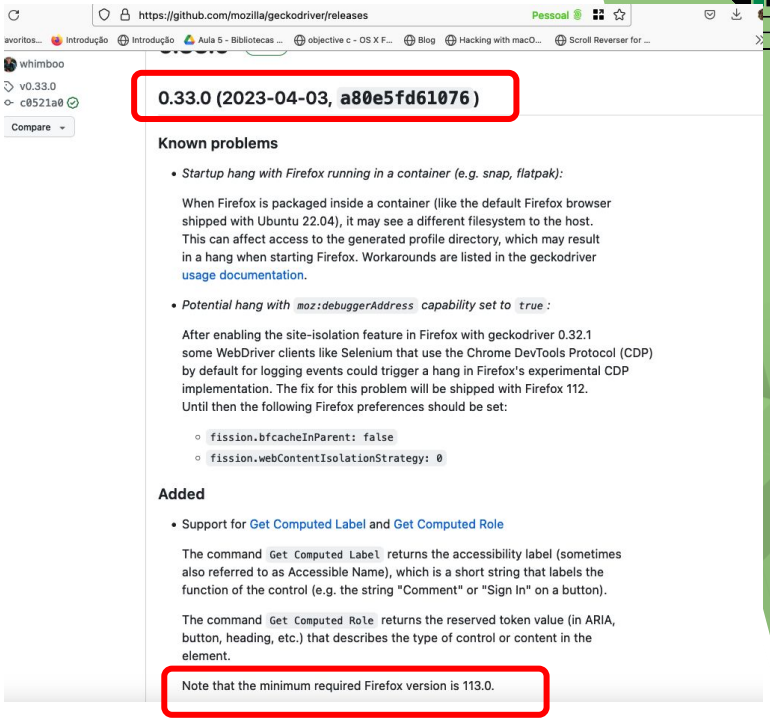
Instalando o Selenium

- Baixando o webdriver para o Chrome



Instalando o Selenium

- Baixando o webdriver para o Firefox



Instalando o Selenium

- Após baixar o webdriver, é importante descompactar o arquivo zip e copiar o *geckodriver* ou *chromedriver* para um diretório que esteja em seu PATH
  - Como cada sistema operacional tem suas características, não vamos concentrar esforços em como definir ou usar diretórios no PATH do sistema operacional,
  - Sugiro pesquisar como fazer isso no seu sistema operacional (Windows, MacOS, Linux, etc.)
    - Pesquise por <seu sistema operacional> + selenium + webdriver + path + install, ou algo equivalente
    - Ou olhar em
      - https://www.selenium.dev/documentation/webdriver/getting_started/install_drivers/

Selenium - Aplicações

- O Selenium WebDriver é usado para automatizar o teste de aplicativos da Web para verificar se funciona conforme o esperado.
- Ele suporta muitos navegadores, como Firefox, Chrome, IE e Safari.
- No entanto, usando o Selenium WebDriver, podemos automatizar o teste apenas para aplicativos da web.

Instalando e dando os primeiros passos no Selenium

Navegando por ligações

- A primeira coisa que alguém deseja fazer com o WebDriver é navegar até um link.
- A maneira mais comum de fazer isso é chamando o método get:

```
from selenium import webdriver

driver = webdriver.Chrome(executable_path="/usr/local/bin/chromedriver")

#driver = webdriver.Firefox(executable_path="/usr/local/bin/geckodriver")


driver.get("http://www.google.com")
```

Navegando por ligações

- O WebDriver aguardará até que a página seja totalmente carregada (ou seja, o evento onload foi acionado) antes de retornar o controle ao seu teste ou script.
- Vale a pena notar que, se a sua página usar muito AJAX no carregamento, o WebDriver pode não saber quando foi totalmente carregado.
- Se você precisar garantir que essas páginas sejam totalmente carregadas, poderá usar waits.

Navegando por ligações

- O que realmente gostaríamos de fazer é interagir com as páginas ou, mais especificamente, com os elementos HTML dentro de uma página.
- Primeiro de tudo, precisamos de uma maneira de localizar elementos dentro de uma página.

Navegando por ligações

- Para encontrar um elemento é preciso usar uma das estratégias de localização, por exemplo

```
element = driver.find_element(By.ID,"passwd-id")

element = driver.find_element(By.NAME, "passwd")

element = driver.find_element(By.XPATH, "//input[@id='passwd-id']")
```

Navegando por ligações

- Por exemplo, dado um elemento definido como:

```
<input type="text" name="passwd" id="passwd-id" />
```

- Para encontrar o elemento é preciso usar uma das estratégias de localização, por exemplo

```
elements = driver.find_elements(By.NAME, "passwd")
```

  - Se houver mais de um elemento que corresponda à consulta, apenas o primeiro será retornado.
  - Se nada for encontrado, uma NoSuchElementException será gerada.

Navegando por ligações

- Após encontrar um campo você pode, por exemplo, inserir texto no campo, por exemplo

  ```
  element.send_keys("some text")
  ```

- Pode-se simular o pressionamento das teclas de seta usando a classe “Keys”

  ```
  element.send_keys(" and some", Keys.ARROW_DOWN)
  ```

- Pode-se facilmente limpar o conteúdo de um campo de texto ou textarea com o método clear:

  ```
  element.clear()
  ```

Navegando por ligações

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys

#driver = webdriver.Chrome(executable_path="/usr/local/bin/chromedriver")
driver = webdriver.Firefox(executable_path="/usr/local/bin/geckodriver")


driver.get("https://www.google.com/")

# Obtém a caixa de pesquisa
element = driver.find_element(By.ID, "APjFqb")

element.send_keys("Selenium")
element.send_keys(Keys.RETURN)
```

Buscando elementos

- O Selenium Python segue diferentes estratégias de localização de elementos.
- Para localizar elementos, você deve importar By

```
from selenium.webdriver.common.by import By
```

Buscando elementos

Pode-se localizar um elemento de 8 maneiras diferentes.

- By.ID - O primeiro elemento com o valor do atributo id correspondente ao local será retornado.
- By.NAME - O primeiro elemento com o valor do atributo name correspondente ao local será retornado.
- By.XPATH - O primeiro elemento com a sintaxe xpath correspondente ao local será retornado.
- By.LINK_TEXT - Retorna o primeiro elemento com o valor de texto do link correspondente ao local.
- By.PARTIAL_LINK_TEXT - Retorna o primeiro elemento com o valor de texto do link parcial correspondente ao local
- By.TAG_NAME - O primeiro elemento com o nome de tag fornecido será retornado.
- By.CLASS_NAME - O primeiro elemento com o nome do atributo de classe correspondente será retornado.
- By.CSS_SELECTOR - O primeiro elemento com o seletor CSS correspondente será retornado.

By.ID

- Com esta estratégia, o primeiro elemento com o valor do atributo id correspondente será retornado.
- Se nenhum elemento tiver um atributo id correspondente, uma NoSuchElementException será gerada.

```
driver.find_element(By.ID, "id_of_element")
```

By.ID

- Exemplo

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
login_form = driver.find_element(By.ID, 'loginForm')
```

By.NAME

- Com esta estratégia, o primeiro elemento com o valor do atributo NAME correspondente ao digitado será retornado.
- Se nenhum elemento tiver um atributo name correspondente, uma NoSuchElementException será gerada.

```
driver.find_element(By.NAME, 'name_of_element')
```

By.NAME

- Exemplo

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
element = driver.find_element(By.NAME, 'username')
```

By.XPATH

- Com esta estratégia, o primeiro elemento com o padrão XPATH correspondente será retornado.
- Se nenhum elemento tiver o padrão correspondente, uma NoSuchElementException será gerada.

```
driver.find_element(By.XPATH, "xpath")
```

By.XPATH

- Exemplo

```
<html>
<body>
<form id="loginForm">
<input name="username" type="text" />
<input name="password" type="password" />
<input name="continue" type="submit" value="Login" />
</form>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
login_form = driver.find_element(By.XPATH, "/html/body/form[1]")
```

```
login_form = driver.find_element(By.XPATH, "//form[1]")
```

By.LINK_TEXT

- Com esta estratégia, o primeiro elemento com o valor do texto do link correspondente ao local será retornado.
- Se nenhum elemento tiver um atributo de texto de link correspondente, uma NoSuchElementException será levantada.

```
driver.find_element(By.LINK_TEXT, 'Text of Link')
```

By.LINK_TEXT

- Exemplo

```
<html>
<body>
<p>Are you sure you want to do this?</p>
<a href="continue.html">Continue</a>
<a href="cancel.html">Cancel</a>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
login_form = driver.find_element(By.LINK_TEXT, 'Continue')
```

By.PARTIAL_LINK_TEXT

- Com essa estratégia, o primeiro elemento com o valor de texto do link parcial correspondente ao local será retornado.
- Se nenhum elemento tiver um atributo de texto de link parcial correspondente, uma NoSuchElementException será lançada.

```
driver.find_element(By.PARTIAL_LINK_TEXT, 'Text of Link')
```

By.PARTIAL_LINK_TEXT

- Exemplo

```
<html>
<body>
<p>Are you sure you want to do this?</p>
<a href="continue.html">Continue</a>
<a href="cancel.html">Cancel</a>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
login_form = driver.find_element(By.PARTIAL_LINK_TEXT, 'Conti')
```

By.TAG_NAME

- Com esta estratégia, será retornado o primeiro elemento com o nome da tag informado.
- Se nenhum elemento tiver um nome de tag correspondente, uma NoSuchElementException será lançada.

```
driver.find_element(By.TAG_NAME, 'Tag name')
```

By.TAG_NAME

- Exemplo

```
<html>
<body>
<h1>Welcome</h1>
<p>Site content goes here.</p>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
login_form = driver.find_element(By.TAG_NAME, 'h1')
```

By.CLASS_NAME

- Com esta estratégia, o primeiro elemento com o nome do atributo de classe correspondente será retornado.
- Se nenhum elemento tiver um nome de atributo de classe correspondente, uma NoSuchElementException será lançada.

```
driver.find_element(By.CLASS_NAME, 'class_of_element')
```

By.CLASS_NAME

- Exemplo

```
<html>
<body>
<p class="content">Site content goes here.</p>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
content = driver.find_element(By.CLASS_NAME, 'content')
```

By.CSS_SELECTOR

- Com esta estratégia, o primeiro elemento com o seletor CSS correspondente será retornado.
- Se nenhum elemento tiver um seletor CSS correspondente, uma NoSuchElementException será levantada.

```
driver.find_element(By.CSS_SELECTOR, 'CSS Selectors')
```

By.CSS_SELECTOR

- Exemplo

```
<html>
<body>
<p class="content">Site content goes here.</p>
</body>
<html>
```

- Agora, depois de criar um driver, você pode pegar um elemento usando

```
content = driver.find_element(By.CSS_SELECTOR, 'p.content')
```

Localizando múltiplos elementos

- Para se localizar vários elementos basta usar a variante no plural de find_element

```
find_elements
```

- Find_elements suporta 7 estratégias By
  - By.NAME
  - By.XPATH
  - By.LINK_TEXT
  - By.PARTIAL_LINK_TEXT
  - By.TAG_NAME
  - By.CLASS_NAME
  - By.CSS_SELECTOR

Localizando múltiplos elementos

- A principal diferença do find_element para o find_elements é que
  - `find_element` traz a primeira ocorrência do elemento buscado
  - `find_elements` traz uma lista dos elementos e portanto é possível iterar na lista para manipular cada elemento.

Esperas e Cadeia de Ações

Esperando por elementos

- Muitas vezes quando queremos encontrar um elemento, a página ainda está sendo carregada, ou o elemento depende de alguma ação prévia para se tornar disponível.
- Assim, é necessário esperar pelo elemento.
- O Selenium provê dois tipos de esperas.
  - Espera explícita
  - Espera implícita

Espera Explícita

- Uma espera explícita é um código que você define para aguardar a ocorrência de uma determinada condição antes de prosseguir no código.
- Existem alguns métodos de conveniência fornecidos que o ajudam a escrever código que aguardará apenas o tempo necessário.
- As esperas explícitas são obtidas usando a classe WebDriverWait em combinação com as condições esperadas.

Espera Explícita

- Exemplo

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Firefox()

driver.get("http://dominio/url-lenta-para-carregar")

try:
      element = WebDriverWait(driver, 10).until(
      EC.presence_of_element_located((By.ID, "myDynamicElement"))
      )
finally:
      driver.quit()
```

Espera Explícita

- Isso espera até 10 segundos antes de lançar um TimeoutException, a menos que encontre o elemento para retornar em 10 segundos.
- Por padrão, WebDriverWait chama ExpectedCondition a cada 500 milissegundos até que retorne com êxito.

Condições para Espera Explícita

- title_is
- title_contains
- presence_of_element_located
- visibility_of_element_located
- visibility_of
- presence_of_all_elements_located
- text_to_be_present_in_element
- text_to_be_present_in_element_value
- frame_to_be_available_and_switch_to_it
- invisibility_of_element_located
- element_to_be_clickable
- staleness_of
- element_to_be_selected
- element_located_to_be_selected
- element_selection_state_to_be
- element_located_selection_state_to_be
- alert_is_present

Esperas Implícitas

- Uma espera implícita informa ao WebDriver para pesquisar por um determinado período de tempo antes de tentar localizar qualquer elemento (ou elementos) não imediatamente disponíveis.
- A configuração padrão é 0 (zero).

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://dominio/url-lenta-para-carregar")
myDynamicElement = driver.find_element_by_id("myDynamicElement")
```

Cadeias de Ações

- ActionChains são implementadas com a ajuda de um objeto de cadeia de ação que armazena as ações em uma fila e quando perform() é chamado, executa as operações enfileiradas.

Como criar um Action Chain Object?

- Para criar um objeto de Action Chain, importe a classe Action Chain e passe o driver como o argumento chave.
- Depois disso, pode-se usar esse objeto para executar todas as operações das cadeias de ação.

```
from selenium import webdriver
from selenium.webdriver.common.action_chains import ActionChains

driver = webdriver.Firefox()

actions = ActionChains(driver)
```

Como usar Action Chains no Selenium?

- Depois de criar um objeto da cadeia de ações, abra uma página da Web e execute vários outros métodos usando a sintaxe e os exemplos abaixo.
- As cadeias de ação podem ser usadas em sequência ou enfileiradas uma a uma e depois executadas, como abaixo.

```
menu = driver.find_element_by_css_selector(".nav")
hidden_submenu = driver.find_element_by_css_selector(".nav # submenu1")

actions = ActionChains(driver)
actions.move_to_element(menu)
actions.click(hidden_submenu)
actions.perform()
```

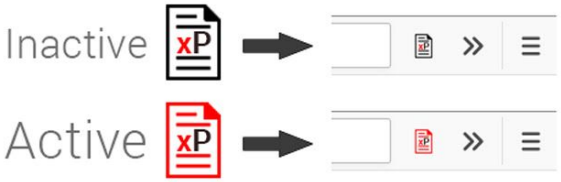
Ferramentas úteis e Exemplos mais complexos

XPath Finder

- Identificar o XPATH pode ser bastante complexo quando analisando um HTML longo.
- Para auxiliar neste processo você pode usar uma extensão para o navegar chamada XPATH Finder
    - https://chrome.google.com/webstore/detail/xpath-finder/ihnknokegkbpmofmafnkoadfjkhlogph
    - https://addons.mozilla.org/en-US/firefox/addon/xpath_finder/

XPath Finder

- Quando o XPATH Finder está ativo, você pode clicar em elementos na tela e visualizar e copiar o XPATH de qualquer elemento na tela.



Exemplo - Google Translator

- O exemplo a seguir acessa o Google Translator, preenche a caixa com um texto em inglês.
- Como o texto traduzido é quebrado em diversos elementos, ao invés de tentar copiar elemento a elemento, o script clica no botão de copiar para a área de trabalho.
- Com o auxílio da classe clipboard, o conteúdo da área de trabalho é copiado para uma variável e, então, é impresso na tela.
- O pacote clipboard pode ser instalado com

```
pip install clipboard
```

Exemplo (Google Tradutor)

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import clipboard as c                          # instalado com pip install clipboard
#driver = webdriver.Chrome(executable_path="/usr/local/bin/chromedriver")
driver = webdriver.Firefox(executable_path="/usr/local/bin/geckodriver")
driver.get("https://translate.google.com/")
print(driver.title)
elem = driver.find_element(By.XPATH,
               "/html/body/c-wiz/div/div[2]/c-wiz/div[2]/c-wiz/div[1]/div[2]/div[3]/c-wiz[1]/span/span/div/textarea")
elem.clear()
elem.send_keys("This is a test")
elem.send_keys(Keys.RETURN)
try:
    element = WebDriverWait(driver, 10).until( EC.presence_of_element_located((By.CLASS_NAME, "YJGJsb")) )
except:
    driver.quit()
div = driver.find_element(By.CLASS_NAME, "YJGJsb")
button = div.find_element(By.TAG_NAME, "button")
button.click()
driver.close()

texto = c.paste()
print(texto)
```

Exemplo - Google Scholar

- O exemplo a seguir acessa o Google Scholar e preenche a caixa de busca com um tópico.
- Quando a pesquisa está pronta, o script seleciona todos os resultados e imprime seus títulos no console.
- Foi inserida uma pausa para pressionar enter para dar continuidade.
- Isso é útil se você quiser, por exemplo, efetuar um login antes de dar continuidade.

Exemplo (Google Scholar)

```
from selenium import webdriver
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
#driver = webdriver.Chrome(executable_path="/usr/local/bin/chromedriver")
driver = webdriver.Firefox(executable_path="/usr/local/bin/geckodriver")
driver.get("https://scholar.google.com/")
print(driver.title)
elem = driver.find_element(By.ID, "gs_hdr_tsi")
elem.clear()
elem.send_keys("Web Scraping")
elem.send_keys(Keys.RETURN)
input('Pressione <ENTER> para continuar')                          # Faz uma pausa aguardando pressionar enter
try:
    corpo = WebDriverWait(driver, 10).until( EC.presence_of_element_located((By.ID, "gs_res_ccl")))
except:
    driver.quit()

resultados = corpo.find_elements(By.CLASS_NAME, "gs_ri")

for r in resultados:
        titulo = r.find_element(By.TAG_NAME, "a")
        print(titulo.text)

driver.close()
```

Tutorial e Outras ferramentas

- **Selenium**
  - Web Scraping Instagram with Selenium
  - Python Selenium Tutorial #1 - Web Scraping, Bots & Testing
- **Beautiful Soup**
  - Webscraping com Beautiful Soup | Raspando tudo!
  - Web Scraping with Beautiful Soup - Make Databases from Scratch
  - Beautiful Soup 4 Tutorial #1 - Web Scraping With Python
- **Mechanical Soup**
  - Better Web Scraping with Mechanical Soup - Build a Database of Cat and Dog Photos in 10 minutes
  - Web Scraping Databases with Mechanical Soup and SQlite
- **Pandas**
  - Much Better Web Scraping with Pandas - Automatically Extract All Table Elements From a Web Page!