Navegando na internet automaticamente com Selenium

Asimov Academy

ASIMOV

Conteúdo

01. O que vamos aprender neste curso

Se você está aqui é por que sabe que automatizar a sua rotina é fundamental pra aumentar sua produtividade. Quando analisamos o nosso trabalho, percebemos que a maior parte da nossa interação com o computador se dá através dos Web Browser (ainda mais com o advento dos aplicativos web que cada vez mais substituem os aplicativos que instalávamos no pc). Então, se queremos aumentar nossa produtividade com automações, é fundamental sabermos automatizar web browsers, correto? E é a isso que esse curso se propõem.

Você irá aprender a analisar uma página web, identificar e selecionar através de códigos os elementos dela e atuar sobre esss elementos de forma automatizada utilizando a biblioteca Selenium. No final, é esperado que você desenvolva todo o conhecimento necessário para automatizar qualquer atividade que você faça no browser (seja ele google chrome, edge, firexox, safari…).

Desenvolveremos projetos como automatização do WhatsApp web, automatização do Google maps para otimização de rotas, extração de informações das páginas de Amazon e Mercado livre para análise de tendências e muito mais.

O curso necessita apenas de conhecimento prévio de Python. Conhecer a estrutura html é importante, mas o básico será passado durante o curso.

Esperamos que você tenha uma ótima experiência de aprendizado, lembrando que estamos sempre felizes em receber suas opiniões e feedbacks :)

Nos vemos na primeira aula!

02. Breve introdução ao Selenium

Mas afinal, o que é Selenium?

O Selenium é uma ferramenta que permite automatizar a interação com os principais navegadores da web usando algo chamado WebDriver. O WebDriver é como uma ponte que nos permite controlar os navegadores de maneira uniforme, independentemente do navegador específico que estamos usando.

No final, o que temos com a utilização de Selenium é o poder de realizar qualquer operação, independente da plataforma, seja ela Chrome, Firefox, Edge e outros. Tudo que você faz com o mouse e teclado no seu browser, você poderá fazer através de código ao utilizar Selenium!

Instalando Selenium

A biblioteca pode ser instalada via pip, utilizando o seguinte código:

```
pip install selenium
```

Para este curso, usaremos Selenium na versão 4.13.0

Rodando nosso primeiro script

Utilizaremos o **Google Chrome** durante esse curso. Esse código é um exemplo de como fazer uma automação simples para esse navegador.

Primeira coisa que devemos fazer é iniciar o nosso driver:

```
from selenium import webdriver

driver = webdriver.Chrome()
```

Se você teve problema abrindo assim, é possível que você não tenha um driver vinculado ao PATH do seu computador. Para contornar isso, é possível utilizar a biblioteca webdriver_manager.

```
pip install webdriver-manager
```

E agora abrir o driver da seguinte forma:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)
```

Se por acaso essa solução ainda não funcionou, você terá que instalar o webdriver manualmente. O passo-a-passo se encontra no final desse capítulo!

Agora vamos abrir uma página com nosso driver recém aberto:

```
driver.get("https://www.selenium.dev/selenium/web/web-form.html")
```

Isso é realizado pela função `.get` do driver.

Agora digamos que queremos adicionar o texto 'Selenium' na caixa de pesquisa presente na página que possuí o título 'Text Input'. Analisando o html da página, podemos ver que a caixa possui o name de "my-text". Então primeiro localizamos ela:

```
text_box = driver.find_element(by=By.NAME, value="my-text")
```

E adicionamos o texto que gostaríamos:

```
text_box.send_keys("Selenium")
```

Agora, analisamos o html da página para encontrar o botão de "Submit". Podemos ver que ele é o único elemento da página com a tag 'button', portanto podemos localizá-lo da seguinte forma:

```
submit_button = driver.find_element(by=By.CSS_SELECTOR, value="button")
```

E depois podemos clicar:

```
submit_button.click()
```

Por fim, vamos mostrar o texto da mensagem que aparece na página seguinte:

```
message = driver.find_element(by=By.ID, value="message")
print('Mensagem final', message.text)
```

E fechamos o browser:

```
driver.quit()
```

É importante também adicionar uma espera, para que a automação não rode antes de uma página carregar. Para isso, adicionamos o seguinte logo após inicializar o driver:

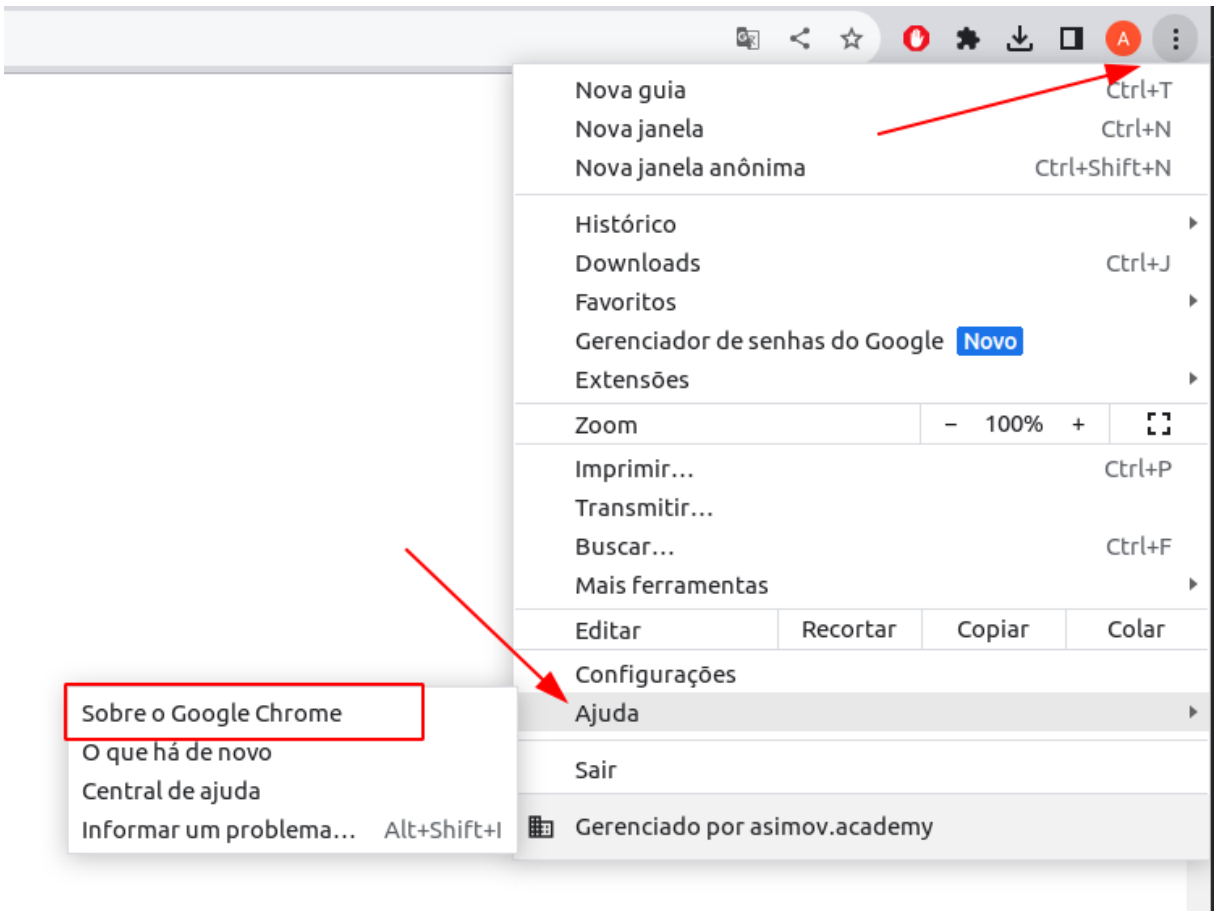
```
driver.implicitly_wait(1)
```

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
from time import sleep

service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)

driver.implicitly_wait(1)

driver.get("https://www.selenium.dev/selenium/web/web-form.html")

text_box = driver.find_element(by=By.NAME, value='my-text')
text_box.send_keys('Selenium')

submit_button = driver.find_element(by=By.CSS_SELECTOR, value='button')
submit_button.click()

message = driver.find_element(by=By.ID, value="message")
print('Mensagem final', message.text)

sleep(30)
driver.quit()
```

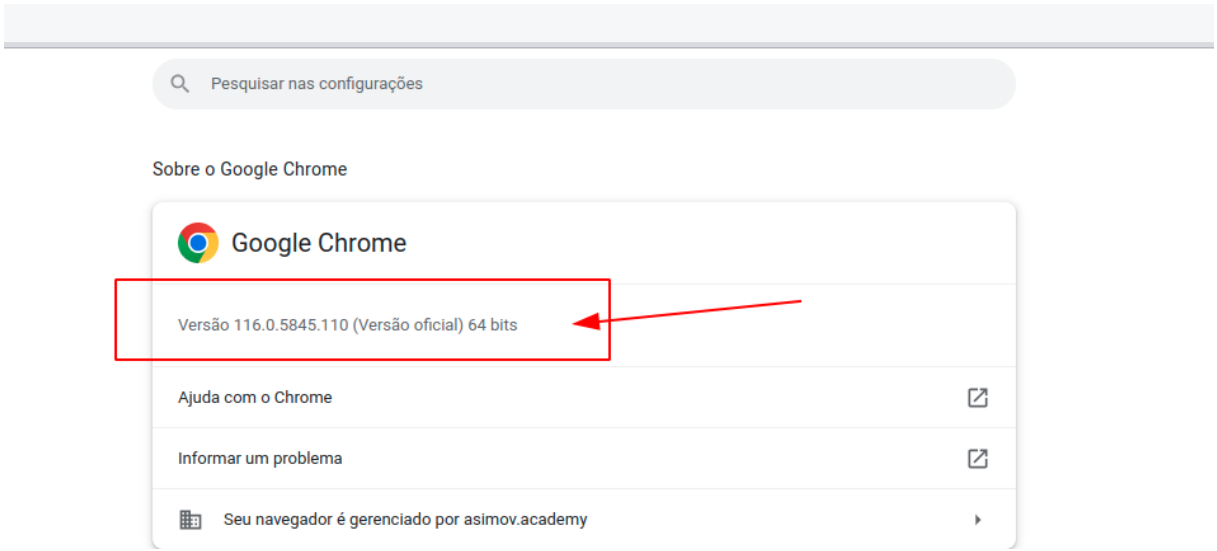
Instalando manualmente o driver (somente se necessário)

Outra forma, seria fazendo o download manual do driver. Este link apresenta como realizar o download do Driver, mas criamos abaixo um passo a passo para facilitar o download.

Primeiro verifique qual a versão do Chrome você está utilizando. Para isso, vá nos três pontos no canto superior do chrome, clique em **Ajuda** e depois em **Sobre o Google Chrome**.



E então você deve conseguir visualizar a versão do seu navegador.



No nosso caso, estamos com a versão **116.0.5845.110**!

Se sua versão for até a 114, o download é feito pelo seguinte link: https://chromedriver.chromium.org/downloads

Se for da versão 115 para cima, você pode utilizar o link:

- Para linux: https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/VERSAO_DO_CHROME/linux64/chr
  linux64.zip
- Para windows64: https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/VERSAO_DO_CHROME/win6
  win64.zip
- Para mac-arm64: https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/VERSAO_DO_CHROME/mac
  arm64/chromedriver-mac-arm64.zip
- Para mac-x64: https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/VERSAO_DO_CHROME/mac-x64/chromedriver-mac-x64.zip

> ATENÇÃO: cuidado para no link substituir o trecho de VERSAO_DO_CHROME pela versão do seu navegador

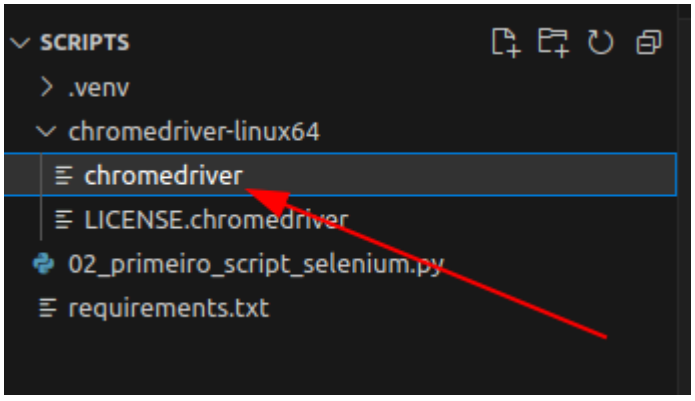
Como no nosso caso a versão é 116.0.5845.110 e estou utilizando o computador linux, o link final que eu utilizaria seria:

https://edgedl.me.gvt1.com/edgedl/chrome/chrome-for-testing/116.0.5845.110/linux64/chromedriver-linux64.zip

Clicando nesse link, o Download do driver já ocorre automaticamente. O driver virá em um arquivo compactado, basta descompactar e conseguimos nosso driver!

Utilizando o Driver manualmente

Para utilizar o Driver recém baixado, primeiros descompactamos ele e colocamos no arquivo junto do nosso script:



Podemos ver que o driver está dentro de uma pasta chamada ‘chromedriver-linux64’ e tem o nome ‘chromedriver’!

Agora podemos utilizar o seguinte código para forçar a utilização desse driver recém baixado:

```
from pathlib import Path

from selenium import webdriver
from selenium.webdriver.chrome.service import Service

caminho = Path(__file__).parent / 'chromedriver-linux64' / 'chromedriver'
service = Service(str(caminho))

driver = webdriver.Chrome(service=service)
```

Nosso script final desse jeito, ficaria assim:

```
from time import sleep
from pathlib import Path

from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By


caminho = Path(__file__).parent / 'chromedriver-linux64' / 'chromedriver'
service = Service(str(caminho))

driver = webdriver.Chrome(service=service)

driver.implicitly_wait(1)

driver.get("https://www.selenium.dev/selenium/web/web-form.html")
print('Título da página', driver.title)
text_box = driver.find_element(by=By.NAME, value="my-text")
text_box.send_keys("Selenium")

submit_button = driver.find_element(by=By.CSS_SELECTOR, value="button")
submit_button.click()

message = driver.find_element(by=By.ID, value="message")
print('Mensagem final', message.text)

driver.quit()
```

03. Entendendo html e a estrutura de uma página web

Para conseguirmos trabalhar com automação de uma página web, primeiro temos entender o que ela é e do que é constituída. Vamos fazer aqui uma breve introdução dos seus principais elementos, focando apenas no essencial. Para mais explicações, é possível assistir nosso curso de WebScrapping.

Principais componentes da estrutura de uma página Web

1 - HTML (Hipertext Markup Language)

O HTML é a base de uma página da web. É como o esqueleto de um corpo humano. Ele define a estrutura básica do conteúdo e usa "tags" para marcar diferentes elementos na página. Por exemplo, você pode usar as tags `<head>`, `<title>`, `<body>`, `<h1>`, `<p>`, `<div>`, entre outras, para definir o título da página, parágrafos, cabeçalhos e divisões.

2 - CSS (Cascading Style Sheets)

O CSS é a pele, cabelo e os olhos do corpo humano, é o que dá sua aparência e senso estético. Ele é usado para estilizar a página e controlar a aparência de elementos HTML. Você pode definir cores, fontes, tamanhos, margens, espaçamento e muito mais usando regras CSS. Isso torna a página visualmente atraente e organizada.

3 - JavaScript

O JavaScript é como os músculos do corpo humano. Ele adiciona o movimento e a funcionalidade interativa à página. Com JavaScript, você pode criar recursos como menus deslizantes, formulários de validação, carrosséis de imagens e muito mais. Ele permite que a página responda às ações do usuário.

Quando tratamos de automações web, estamos principalmente preocupados com o bom entendimento do html da página, pois assim podemos fazer a localização dos elementos com os quais vamos interagir (botões, áreas de texto, links, etc.).

Por isso, vamos dar uma olhada mais a fundo em html.

Estrutura básica de um HTML

Vamos usar de exemplo um html simples como este:

```
<!DOCTYPE html>
<html>
    <head>
        <title>Aqui fica o meu título</title>
    </head>

    <body>
        <h1>Isto é uma frase em h1</h1>
        <p>Esse é um parágrafo</p>
        <button>Isso é um botão</button>
        <input type="text">
        <a href="https://www.google.com">Link para o Google</a>

    </body>
</html>
```

Tags de HTML

Você pode notar que a estrura básica de um html é feita a partir de tags, e cada tag adiciona uma funcionalidade específica a página web. Vamos listar aqui algumas tags principais:

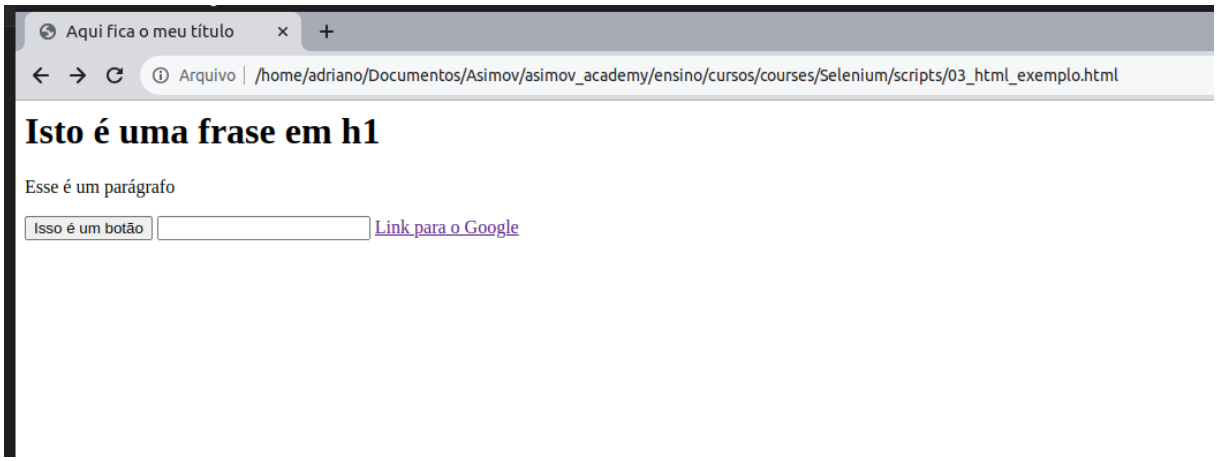
- `<!DOCTYPE html>` — Esta tag especifica o idioma que você escreverá na página. Neste caso, a linguagem é HTML 5.
- `<html>` - Esta tag sinaliza que daqui em diante vamos escrever em código HTML.
- `<head>` - É para onde vão todos os metadados da página - coisas destinadas principalmente a mecanismos de pesquisa e outros programas de computador.
- `<body>` - É para onde vai o conteúdo da página.

Essas quatro formam a base da estrutura. A partir de agora, listamos as tags mais específicas de conteúdo. Maior parte delas se encontrará posteriormente dentro da tag body do html.

- `<h1>` - Essa tag sinaliza um texto em destaque. Pode ser usada com outras numerações (h2, h3, h4, etc), quanto maior o número menor o destaque.
- `<p>` - Essa tag sinaliza um parágrafo de texto.
- `<button>` - Essa tag sinaliza um botão.
- `<input>` - Essa tag sinaliza uma entrada de valores.
- `<a>` - Essa tag sinaliza um texto com link.

Como visualizar um arquivo html

Para visualizar um arquivo, basta salvá-lo com a extensão .html (por exemplo, meu_arquivo.html), depois copiar o caminho completo do arquivo e colocar esse caminho na aba do navegador. Da seguinte forma:



Atributos de html

Os elementos em HTML possuem atributos; são valores adicionais que configuram os elementos ou ajustam seu comportamento de diversas maneiras para atender aos critérios desejados pelos usuários.

Id Deve ser um valor único no arquivo e é usado para identificar um elemento específico do html.

```
<h1 id='meu_titulo'>Isto é uma frase em h1</h1>
```

class É geralmente utilizada para estilização de um html, onde vários que receberão a mesma configuração de cores e formato podem ser referenciados de uma só vez.

```
<h1 class='titulos_grandes'>Isto é uma frase em h1</h1>
```

name É utilizado para atribuir um nome ao elemento

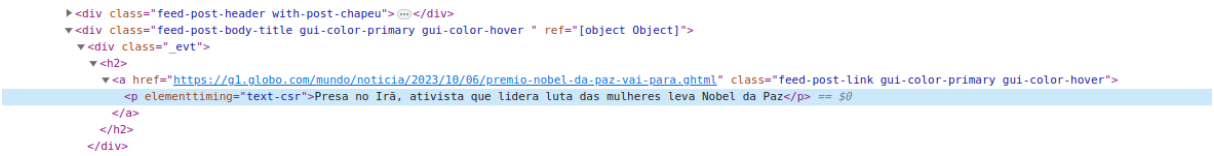
```
<button name="assunto" type="submit">HTML</button>
```

Inspecionando uma página

A forma mais simples de identificar e conseguir referenciar um elemento de uma página web e através da inspeção. Mostraremos agora como fazer isso.

Vamos começar utilizando a página do G1 de exemplo. Digamos que queremos saber o título de uma das notícias, para isso abrimos a página, clicamos com o botão direito sobre o título e clicamos em inspecionar:

Clicando no botão, o browser já vai mostrar exatamente o trecho do código fonte da página que está construindo aquele título:



Sabendo disso, podemos analisar o trecho do código para acharmos formas de referenciá-lo ao utilizarmos ferramentas como Selenium.

04. Localizando elementos por atributos

O desafio ao utilizar Selenium é de analisar corretamente a estrutura do htmlque compõe a página web para encontrar o elemento com o qual queremos interagir. Digamos que na minha página tenha um inpiut de texto, a qual preciso preencher com uma senha. Podemos dar um exemplo de um trecho de html que representaria essa input:

```
<input type="text" name="senha" id="senha-id" class="estilo_senha"/>
```

Podemos localizar esse elemento de diversas formas com Selenium:

Localizando elementos únicos

Localizando por ID

É sempre recomendado tentar localizar por id antes de tentar outras formas. Isto porque o Id é um atributo único, ou seja, apenas um elemento vai ter aquele determinado id. Outros elemenetos como name, class não são únicos, podendo ocorrer o erro de ao tentarmos localizar um elemento, encontrarmos outro que não era o de interesse.

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

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

Localizando por name

O atributo name não necessariamente é único, mas ele pode ser uma boa forma auxiliar de localizar um elemento também. Pode ser utilizado da seguinte forma:

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

element = driver.find_element(By.NAME,"senha")
```

Localizando por class

Localizar um elemento pela sua classe não é recomendado em geral, pois a classe faz parte da estilização dos elementos de um html e muitas vezes é usada indiscriminadamente dentro da estrutura da página. Mas de qualquer forma, fica aqui a forma de localizar:

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

element = driver.find_element(By.CLASS,"estilo_senha")
```

Localizando múltiplos elementos

Para localizar múltiplos elementos, podemos utilizar o método .find_elements:

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

element = driver.find_elements(By.CLASS,"estilo_senha")
```

Ele retorna uma lista de elementos. caso a lista esteja vazia, é porque nenhum elemento foi encontrado.

05. EXERCÍCIO 01 - Localizando elementos no Mercado Livre

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager

from time import sleep


service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)

driver.get('https://www.mercadolivre.com.br/ofertas')

# Encontra todos os carrosseis de promocao
itens_carrossel = driver.find_elements(By.CLASS_NAME, 'andes-carousel-snapped__slide')

resumo_itens_promo = []
for i in range(len(itens_carrossel)):
    carrossel = itens_carrossel[i]

    # Se o carrosel não está aparente, procura o botão de next e clica
    if not carrossel.is_displayed():
        botao_seguinte = driver.find_element(By.CLASS_NAME,
↪   'andes-carousel-snapped__control--next')
        botao_seguinte.click()

    # Clica no botao do carrossel de promocao
    carrossel.click()
    sleep(1)

    # Recarrega o elemento de carrossel após o click
    itens_carrossel = driver.find_elements(By.CLASS_NAME, 'andes-carousel-snapped__slide')
    carrossel = itens_carrossel[i]
    print(carrossel.text)

    # Procura todos os itens em promocao
    itens_promocao = driver.find_elements(By.CLASS_NAME, 'promotion-item__link-container')
    for iten_promo in itens_promocao:
        resumos_promo_item = {}

        # Procura se o item está com deconto
        desconto = iten_promo.find_elements(By.CLASS_NAME, 'andes-money-amount__discount')
        if len(desconto) != 0:
            desconto = float(desconto[0].text.split(' ')[0].replace('%', ''))
            resumos_promo_item['desconto'] = desconto

            # Salva o tipo do carrossel
            resumos_promo_item['tipo'] = carrossel.text

            # Salva a descricao do item
```

```
            descricao = iten_promo.find_element(By.TAG_NAME, 'p')
            resumos_promo_item['nome'] = descricao.text

            # Salva o preco do item
            preco_reais = iten_promo.find_element(By.CLASS_NAME,
↪  'andes-money-amount__fraction').text
            preco_centavos = iten_promo.find_elements(By.CLASS_NAME,
↪  'andes-money-amount__cents')
            if len(preco_centavos) == 1:
                resumos_promo_item['preco'] =  float(preco_reais) +
↪  float(preco_centavos[0].text) / 100
            else:
                resumos_promo_item['preco'] =  float(preco_reais)

            # Salva o link do item
            resumos_promo_item['link'] = iten_promo.get_attribute('href')

            # Adiciona a lista
            resumo_itens_promo.append(resumos_promo_item)
```

06. Utilizando XPATH para localizar elementos em um HTML

XPath usa expressões de caminho para selecionar um “nó” ou um conjunto de “nós” em um documento XML ou HTML.

Similar ao Path do Python que é utilizado para criar caminhos de documentos no seu computador, o XPATH é utilizado para gerar caminhos e facilitar a localização de elementos dentro de um arquivo XML ou HTML.

Vamos passar pelas principais funcionalidades e mostrar em aula através de exemplos o funcionamento do XPATH.

Seleção de nodos

XPath usa expressões de caminho para selecionar nós em um documento HTML. O nó é selecionado seguindo um caminho ou etapas. As expressões de caminho mais úteis estão listadas abaixo:

Expressão	Descrição
`nome_do_nodo`	Seleciona o nodo com nome = nome_do_nodo
`/`	Seleciona do nodo raiz
`'//'`	Seleciona nós no documento a partir do nó atual que correspondem à seleção, não importa onde estejam
`'.'`	Seleciona o nodo atual
`'..'`	Seleciona o pai do nodo atual
`'@'`	Seleciona atributos

Alguns exemplos de utilização:

Exemplo	Descrição
`/div`	Seleciona todas as divs que estão no elemento raiz
`//div`	Seleciona todas as divs do arquivo
`//button/..`	Seleciona os nodos pais de todos os elementos de tag button

Exemplo	Descrição
`'//div/span'`	Seleciona todos os spans que tem uma div como pai direto
`'//div//span'`	Seleciona todos os spans que tem uma div acima na hierarquia
`//@href`	Seleciona todos os elementos que tenham um atributo href

Predicados

Predicados são usados para localizar um nó específico ou um nó que contém um valor específico. Os predicados são sempre inseridos entre colchetes. Na tabela abaixo listamos algumas expressões de caminho com predicados e o resultado das expressões:

Expressão	Descrição
`//div[1]`	Seleciona a primeira div do documento
`//div[1]/div[last()]`	Seleciona a última div da primeira div
`//div[1]/div[last()-1]`	Seleciona a penúltima div da primeira div
`//div[position()<3]`	Seleciona as duas primeiras divs
`//button[@id]`	Seleciona todos buttons que tenham um id de atributo
`//button[@id='seguir']`	Seleciona o botão cujo id = ‘seguir’
`//button[text()='seguir']`	Seleciona o botão cujo texto = ‘seguir’
`//button[contains(text(), 'egui')]`	Seleciona o botão cujo texto contenha ‘egui’
`//button[contains(text(), 'egui') and @class='classe_botao']`	Seleciona o botão cujo texto contenha ‘egui’ e tenha um atributo classe com valor = ‘classe_botao’

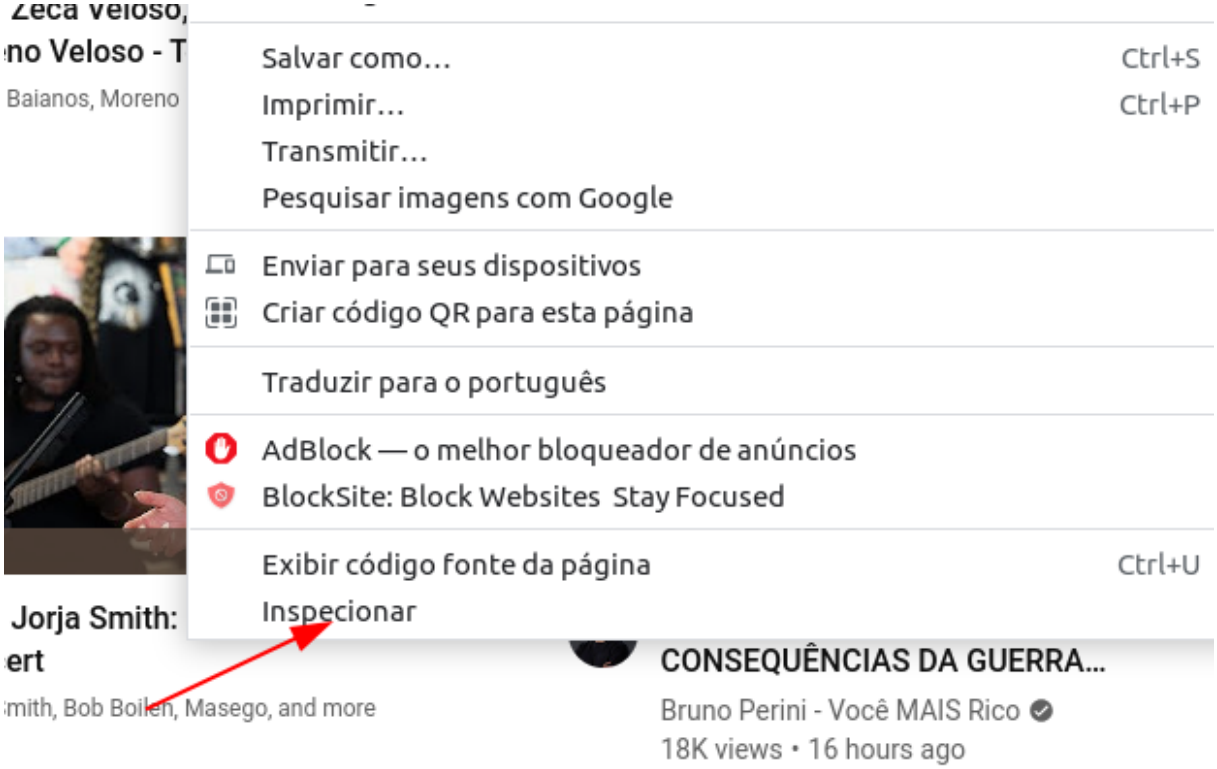
Selecionando nós desconhecidos

O asterisco no XPath pode ser usados para selecionar nós HTML desconhecidos.

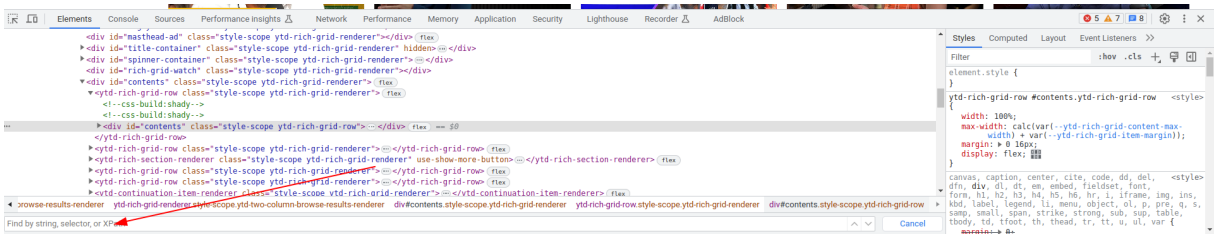
Expressão	Descrição
`//div/*`	Seleciona todos os filhos de todas as divs
`//a[@**]`	Seleciona todos os as que possuem ao menos um atributo

Testando XPATH no seu browser

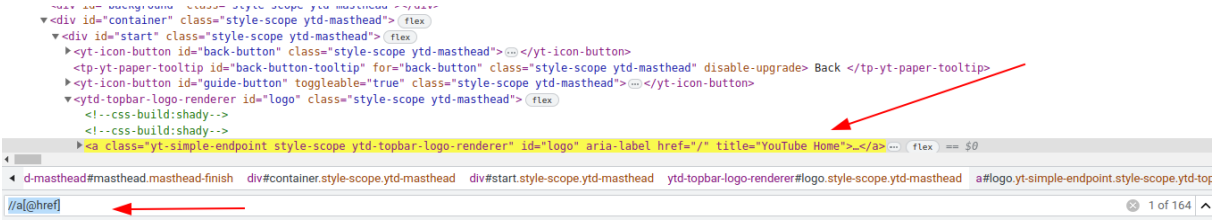
Para testar um xpath no seu browser basta abrir o inspecionar, clicando com o botão direito em qualquer lugar da página:



Depois aperte “ctrl + f” para abrir a aba de busca no Inspecionr:



E depois podemos testar os caminhos XPATHS que criamos para a página que estamos atualmente:



Você pode ver que no momento que coloco a expressão desejada, automáticamente o navegador já encontra no código fonte todos os elementos que correspondem aquele caminho.

07. Localizando elementos por XPATH

Da mesma forma que localizamos elementos por tags e atributos, podemos utilizar os caminhos XPATH para localização.

Localizando elementos únicos

Para localizar um elemento, utilizamos o método find_element. Caso não encontre o elemento, retornará um ero.

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

element = driver.find_element(By.XPATH,"//button[text()='seguir']")
```

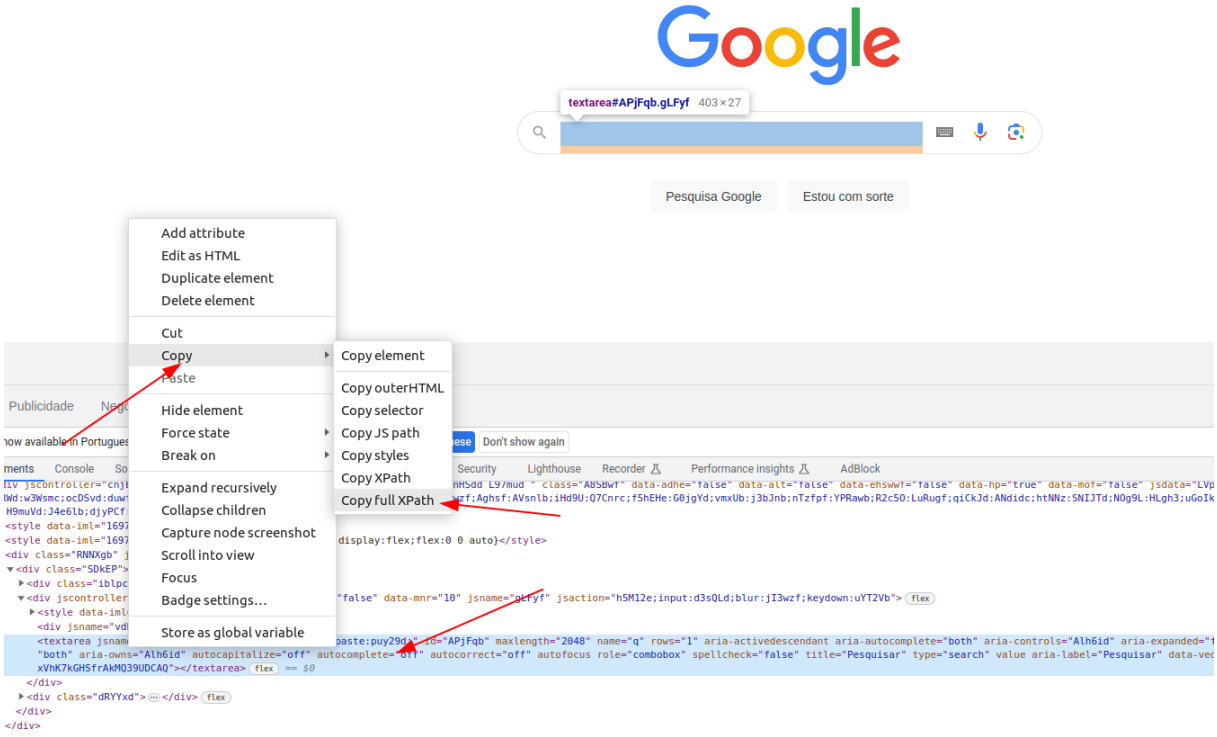
Localizando múltiplos elementos

Para localizar múltiplos elementos, utilizamos o método find_elements (com s no final). Ele retorna uma lista, e caso não encontr a lista será vazia.

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

elements = driver.find_elements(By.XPATH,"//button[text()='seguir']")
```

Identificando XPATH de um elemento

Podemos encontrar o XPATH completo de um elemento através do Inspecionar do browser, ispecionando o elemento de interesse, depois clicando com o botão direito e indo em copy e depois em Copy Full XPATH



No caso, o XPATH da área de texto do google é o seguinte:

```
/html/body/div[1]/div[3]/form/div[1]/div[1]/div[1]/div/div[2]/textarea
```

> Atenção: sugerimos nunca utilizar o XPATH copiado nos seus códigos. Esses códigos podem mudar facilmente, inclusive com a iteratividade dos sites e não funcionar como devido. Aconselhamos assistir as aulas para entender estratégias inteligentes de criação de XPATHs

08. EXERCÍCIO 02 - Localizando elementos na Amazon

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from selenium.webdriver.common.by import By
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.action_chains import ActionChains

from time import sleep


service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)

driver.get('https://www.amazon.com.br/')

# Abre menu hamburguer
menu_hamburguer = driver.find_element(By.XPATH, '//a[@aria-label="Abrir
↪  Menu"]/span[text()="Todos"]')
menu_hamburguer.click()
sleep(1)

# Abre clica na caixa de Produtos em alta
caixa_prod_alta = driver.find_element(By.XPATH, '//a[@class="hmenu-item" and text()="Produtos
↪  em alta"]')
caixa_prod_alta.click()
sleep(1)

# Scrola para fazer load de todos carroseis
carrosseis = driver.find_elements(By.XPATH, '//div[contains(@id, "anonCarousel")]')
i = 0
while i < len(carrosseis):
    carrosel = carrosseis[i]
    ActionChains(driver).scroll_to_element(carrosel).perform()
    sleep(1)
    carrosseis = driver.find_elements(By.XPATH, '//div[contains(@id, "anonCarousel")]')
    i+= 1

# Pega todos titulos
titulos = driver.find_elements(By.XPATH, '//div[contains(@id,
↪  "anonCarousel")]/../../../..//h2')
titulos = [t.text.replace('Produtos em alta em ', '') for t in titulos]
sleep(1)

for i in range(len(titulos)):
    maior_item = 0
    while True:
        # Pega todos os cards do carrossel
        carrosseis = driver.find_elements(By.XPATH, '//div[contains(@id, "anonCarousel")]')
        carrosel = carrosseis[i]
        itens_carrossel = carrosel.find_elements(By.XPATH, f'.//li[@class="a-carousel-card"]')
        # Verifica se card já foi contabilizado
```

```
        if int(itens_carrossel[0].get_attribute('aria-posinset')) > maior_item:
            maior_item = int(itens_carrossel[0].get_attribute('aria-posinset'))
        else:
            break
        for item in itens_carrossel:
            if not item.get_attribute('aria-hidden') == 'true':
                modificacao = item.find_element(By.XPATH, './/span[contains(@class,
↪  "carousel-sales-movement")]')
                print(titulos[i], modificacao.text)
        # Clica no botão de next
        botao_seguir = carrosel.find_element(By.XPATH, './../..//a[@class="a-button
↪  a-button-image a-carousel-button a-carousel-goto-nextpage"]')
        botao_seguir.click()
        sleep(2)
```

10. Esperando carregamento de elementos

Talvez o desafio mais comum para a automação do navegador seja garantir que a página esteja em condições de executar um comando específico do Selenium conforme desejado. Os processos geralmente terminam em uma condição de corrida onde às vezes o navegador entra no estado correto primeiro (as coisas funcionam como esperado) e às vezes o código Selenium é executado primeiro (as coisas não funcionam como esperado). Esta é uma das principais causas de scripts instáveis.

Em muitos aplicativos de página única, os elementos são adicionados dinamicamente a uma página ou alteram a visibilidade com base em um clique. Um elemento deve estar presente e exibido na página para que o Selenium interaja com ele.

Veja esta página, por exemplo: https://www.selenium.dev/selenium/web/dynamic.html Quando a mensagem “Adicionar uma caixa!” botão é clicado, um elemento “div” que não existe é criado. Quando o botão “Revelar uma nova entrada” é clicado, um elemento de campo de texto oculto é exibido. Em ambos os casos, a transição leva alguns segundos. Se o código Selenium clicar em um desses botões e interagir com o elemento resultante, isso será feito antes que o elemento esteja pronto e falhe.

A primeira solução que muitas pessoas recorrem é adicionar uma instrução *sleep* para pausar a execução do código por um determinado período de tempo. Como o código não consegue saber exatamente quanto tempo precisa esperar, isso pode falhar quando não espera o suficiente. Como alternativa, se o valor for definido muito alto e uma instrução *sleep* for adicionada em todos os locais necessários, a duração da sessão poderá se tornar proibitiva.

O Selenium fornece dois mecanismos diferentes de sincronização que são melhores.

Esperas implícitas

O Selenium possui uma maneira integrada de esperar automaticamente por elementos, chamada espera implícita.

Esta é uma configuração global que se aplica a cada chamada de localização de elemento durante toda a sessão. O valor padrão é 0, o que significa que se o elemento não for encontrado, retornará imediatamente um erro. Se uma espera implícita for definida, o driver aguardará a duração do valor fornecido antes de retornar o erro. Observe que assim que o elemento for localizado, o driver retornará a referência do elemento e o código continuará em execução, portanto, um valor de espera implícito maior não aumentará necessariamente a duração da sessão.

```
driver.implicitly_wait(2)
```

Esperas explícitas

Esperas explícitas são loops adicionados ao código que pesquisam o aplicativo em busca de uma condição específica para avaliar como verdadeira antes de sair do loop e continuar para o próximo comando no código. Se a condição não for atendida antes de um valor de tempo limite designado, o código apresentará um erro de tempo limite.

```
from selenium.webdriver.support.wait import WebDriverWait

driver.find_element(By.ID, "reveal").click()

revealed = driver.find_element(By.ID, "revealed")
wait = WebDriverWait(driver, timeout=2)
wait.until(lambda d : revealed.is_displayed())

revealed.send_keys("Displayed")
```

Esperas explícitas + Condições esperadas

Existem algumas condições comuns que são frequentemente usadas ao automatizar navegadores da web. Listados abaixo estão os nomes de cada um. A ligação Selenium Python fornece alguns métodos convenientes para que você não precise codificar uma classe expect_condition sozinho ou criar seu próprio pacote de utilitários para eles.

- title_is
- title_contains
- presence_of_element_located
- visibility_of_element_located
- presence_of_all_elements_located
- text_to_be_present_in_element
- element_to_be_clickable

```
from selenium.webdriver.support.wait import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver.find_element(By.ID, "reveal").click()

wait = WebDriverWait(driver, timeout=2)
wait.until(EC.visibility_of_element_located((By.ID, "revealed")))

revealed = driver.find_element(By.ID, "revealed")
revealed.send_keys("Displayed")
```

11. Explorando métodos de elemento

Neste momento, já esperado que tenhamos sólido em nós as formas de localizar os elementos de uma página web. Vamos passar a explorar mais a fundo os métodos do driver e dos elementos de Selenium. Alguns já vimos nos exercícios que realizamos, mas é importante explorarmos com mais calma, pois é através deles que a interação realmente acontece com o navegador.

Métodos do driver

close

Para fechar o navegador, podemos utilizar o método close:

```
driver.close()
```

back

Para retornar a página anterior no histórico de navegação, podemos utilizar o método back:

```
driver.back()
```

forward

Para avançar no histórico de navegação, podemos utilizar o método forward:

```
driver.forward()
```

fullscreen_window

Para colocar o browser em modo de tela cheia, podemos utilizar o método fullscreen_window:

```
driver.fullscreen_window()
```

maximize_window

Para maximizar a tela do browser, podemos utilizar o método maximize_window:

```
driver.maximize_window()
```

get_screenshot_as_file

Para tirar um print screen podemo utilizar o método fullscreen_window:

```
driver.get_screenshot_as_file('print.png')
```

current_url

Para pegar a url da página atual:

```
driver.current_url
```

title

Para pegar o título da página atual:

```
driver.title
```

Métodos dos elementos

is_displayed

Para saber se o elemento está visível:

```
element.is_displayed()
```

get_property

Para saber a propriedade de um elemento:

```
element.get_property("link")
```

get_attribute

Para saber um atributo de um elemento:

```
element.get_attribute("link")
```

click

Para clicar em um elemento:

```
element.click()
```

clear

Para limpar o conteúdo de um elemento :

```
element.clear()
```

tag_name

Para retornar o nome da tag de um elemento :

```
element.tag_name
```

text

Para retornar o texto de um elemento:

```
element.text
```

12. Ações em cadeia e Keys Especiais

Ações em cadeias

Ações em cadeia são formas mais avançadas de interação com o browser. Com elas, é possível armazenar ações em uma fila que são executadas ao chamarmos o método *perform()*. Vamos passar alguns métodos principais das ações em cadeias:

click

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

elemento = driver.find_element(By.XPATH, '//textarea[@aria-label="Pesquisar"]')

acao = ActionChains(driver)
acao.click(on_element=elemento)
acao.perform()
```

send_keys

```
acao = ActionChains(driver)
acao.click(on_element=elemento)
acao.send_keys('Asimov')
acao.perform()
```

key_down e key_up

```
acao = ActionChains(driver)
acao.click(on_element=elemento)
acao.key_down(Keys.SHIFT)
acao.send_keys('asimov')
acao.key_up(Keys.SHIFT)
acao.perform()
```

click_and_hold

```
driver.get('https://selenium.dev/selenium/web/mouse_interaction.html')

clickable = driver.find_element(By.ID, 'clickable')
acao = ActionChains(driver)
acao.click_and_hold(clickable)
acao.perform()
```

double_click

```
driver.get('https://selenium.dev/selenium/web/mouse_interaction.html')

clickable = driver.find_element(By.ID, 'clickable')
ActionChains(driver).double_click(clickable).perform()
```

drag_and_drop

```
driver.get('https://selenium.dev/selenium/web/mouse_interaction.html')

draggable = driver.find_element(By.ID, 'draggable')
droppable = driver.find_element(By.ID, 'droppable')

acao = ActionChains(driver)
acao.drag_and_drop(draggable, droppable)
acao.perform()
```

scroll_to_element

```
driver.get('https://www.globo.com')
elemento = driver.find_element(By.XPATH, '//h2[@aria-label="A gente explica"]')

acao = ActionChains(driver)
acao.scroll_to_element(elemento)
acao.perform()
```

Keys Especiais

Keys Especiais permitem pressionar teclas através do teclado como ctrl+f, ou shift+c+v, etc. A classe selenium.webdriver.common.keys.Keys lida com todas as chaves no Selenium Python. Ele contém um grande número de métodos principais que podem ser usados no Selenium Python.

```
action.key_down(Keys.CONTROL).send_keys('F').key_up(Keys.CONTROL).perform()
```

Algumas keys principais

- ARROW_DOWN: seta para baixo
- ARROW_LEFT: seta para esquerda
- ARROW_RIGHT: seta para direita
- ARROW_UP: seta para cima
- BACKSPACE: tecla de retornar
- CONTROL: tecla Ctrl

- `DELETE`: tecla del
- `ESCAPE`: tecla esc
- `RETURN`: tecla enter
- `SHIFT`: tecla shift
- `SPACE`: tecla espaço
- `TAB`: tecla tab