

# código.py

## GALÁXIA 8

Web Scraping



## Introdução

Olá, seja bem-vindo à Galáxia 8 de Web Scraping no Python! Nesta galáxia exploraremos o fascinante mundo do Web Scraping e descobrir como extrair informações valiosas da web usando a linguagem de programação Python.

Ao longo das aulas, aprenderemos desde os conceitos básicos até técnicas avançadas de Web Scraping, utilizando bibliotecas populares como BeautifulSoup e Selenium. Vamos te mostrar como coletar dados de sites de forma estruturada e automatizada, abrindo portas para uma infinidade de aplicações práticas.



## Mundo 1

Um site é uma coleção de scripts que se misturam. Cada página do site é composta por um conjunto de arquivos, sendo os mais comuns o HTML, o CSS e o JavaScript.

### 1.1. O que é um site?

**HTML (HyperText Markup Language):** é a linguagem de marcação utilizada para estruturar e organizar o conteúdo de uma página web. Ela define a hierarquia dos elementos, como cabeçalhos, parágrafos, imagens, links e muitos outros. O HTML utiliza tags para envolver e identificar cada elemento, permitindo que os navegadores interpretem e exibam o conteúdo corretamente.

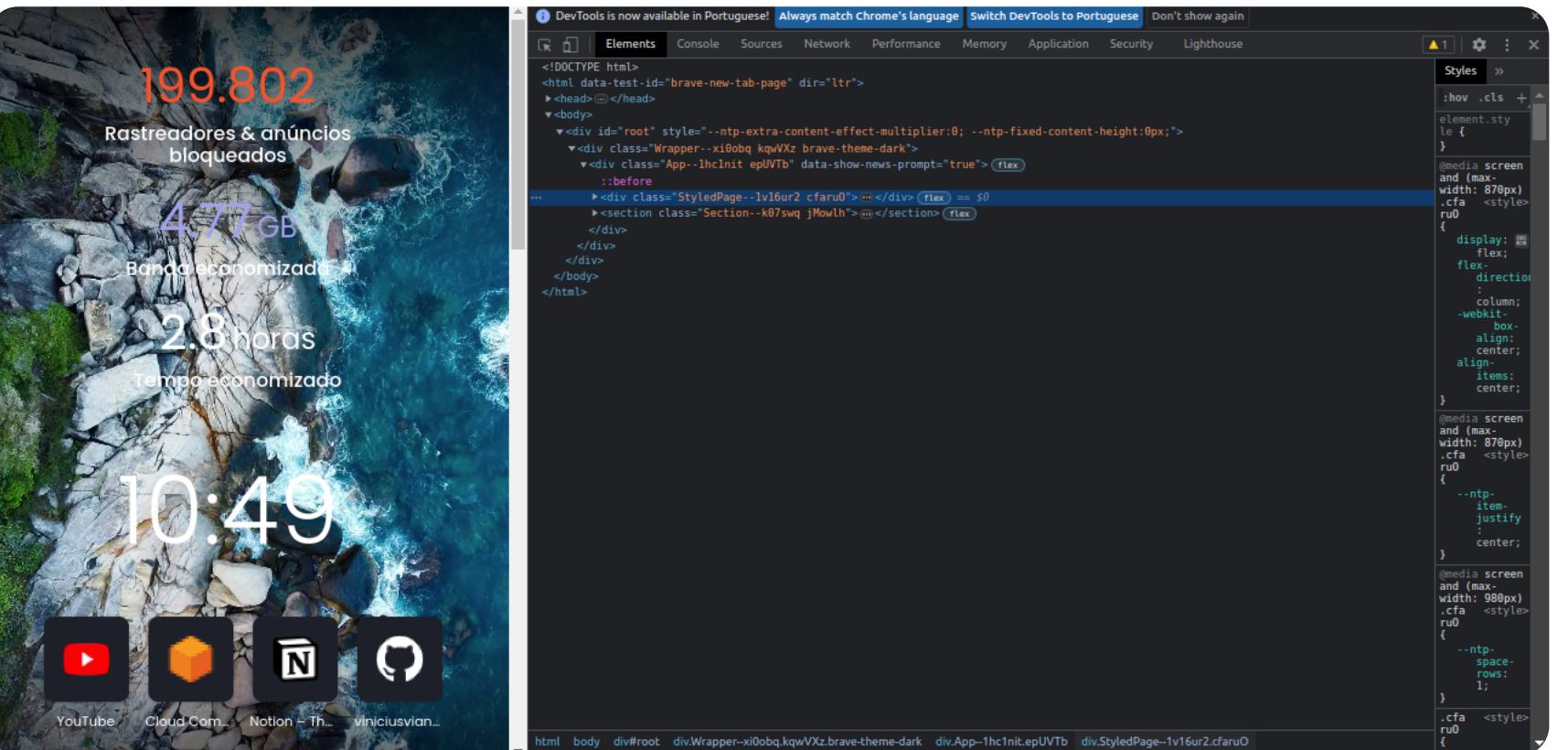
**CSS (Cascading Style Sheets):** é uma linguagem de estilo utilizada para controlar a apresentação e o layout dos elementos em uma página web. Com o CSS, é possível definir cores, fontes, tamanhos, margens, posicionamento e outros aspectos visuais. Ele separa a estrutura (HTML) do estilo (CSS), permitindo uma maior flexibilidade e controle sobre o design do site.

**JavaScript:** é uma linguagem de programação que permite adicionar interatividade e dinamismo às páginas web. Com o JavaScript, é possível realizar manipulações de elementos, responder a eventos, validar formulários, criar animações e até mesmo efetuar requisições para servidores, possibilitando a criação de aplicações web mais complexas e interativas.

Em resumo, o HTML define a estrutura e o conteúdo de uma página web, o CSS controla o estilo e a aparência, enquanto o JavaScript adiciona interatividade e funcionalidades dinâmicas. Essas três tecnologias trabalham em conjunto para criar a experiência visual e interativa que encontramos ao navegar em um site.

### 1.1.1. Inspecionando o HTML de uma página

Se você clicar com o botão direito do mouse e depois clicar em “inspecionar”. Será redirecionado para esta página, que contém todo HTML do site.



Cada elemento no HTML direciona para alguma parte da página.

Se você passar o mouse em cada palavra no HTML, ela direcionará a uma parte do site que corresponde a este HTML. É exatamente desta forma que acessaremos as informações utilizando Python.

## Mundo 2

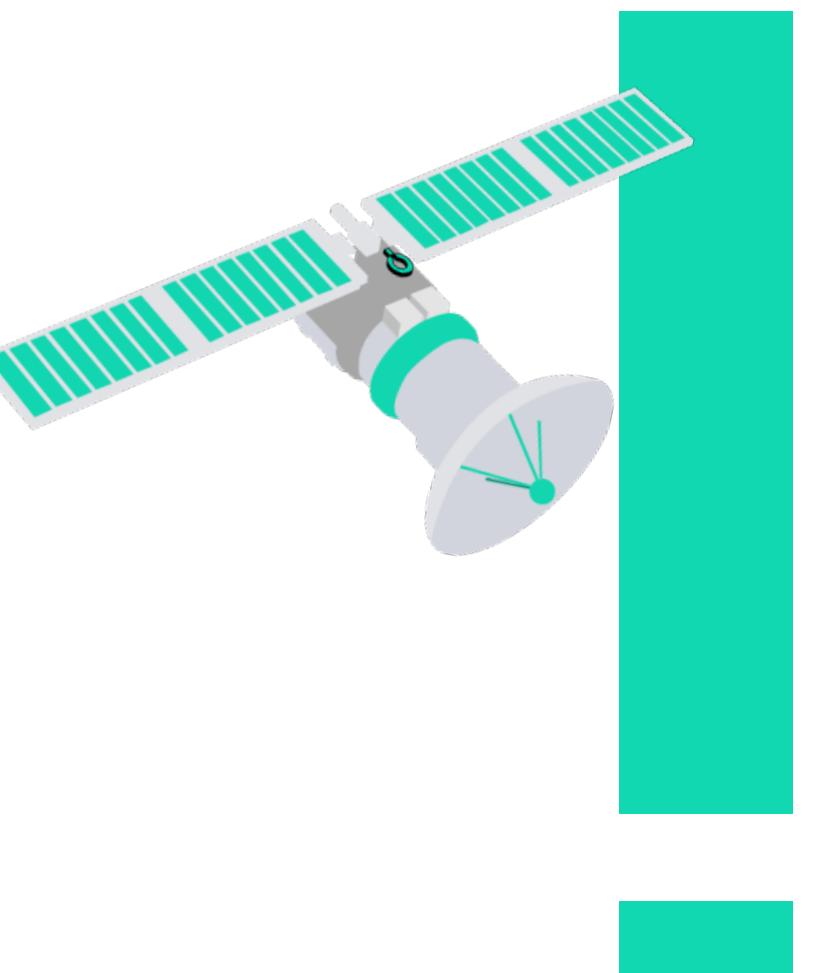
### 2.1. Estrutura HTML

HTML (HyperText Markup Language) é uma linguagem de marcação usada para criar a estrutura e o conteúdo de uma página web. Ela consiste em uma série de elementos (também conhecidos como tags) utilizados para identificar e formatar diferentes partes da página.

**Toda linguagem possui suas características, o Python tem a tabulação, o JavaScript os { } e o HTML é por meio de <>. Cada comando, obrigatoriamente, deve ser diferenciado por <>.**

A estrutura básica de um documento HTML é composta por três principais elementos: `<head>`, `<body>`, e `<title>`.

**<head>**: O elemento `<head>` é usado para definir informações sobre o documento, como o título da página, as meta tags (usadas para fornecer informações sobre o documento para os mecanismos de busca) e referências a arquivos CSS e JavaScript externos.



**<body>**: O elemento `<body>` é onde todo o conteúdo visível da página é colocado. Isso inclui textos, imagens, vídeos, links e outros elementos visuais que os usuários podem interagir. É dentro do `<body>` que você estrutura o layout e o conteúdo da sua página.

**<title>**: O elemento `<title>` define o título da página, sendo geralmente exibido na barra de título do navegador e nos resultados de busca. É uma parte importante para a identificação e indexação da página pelos mecanismos de busca.

Além desses elementos básicos, existem várias outras tags HTML usadas para formatar o conteúdo:

**<h1>**: Os elementos de cabeçalho `<h1>` a `<h6>` são usados para criar títulos e subtítulos em uma página. O `<h1>` é o cabeçalho mais importante, geralmente usado para o título principal da página.

**<p>**: O elemento `<p>` é usado para criar parágrafos de texto na página. É utilizado para separar e estruturar o conteúdo em blocos de texto.

**<div>**: O elemento <div> é uma divisão genérica que permite agrupar outros elementos e criar seções ou áreas específicas da página.

**<table>**: O elemento <table> é usado para criar uma tabela na página. É composto por linhas (<tr>) e células (<td>) que permitem organizar dados em formato tabular.

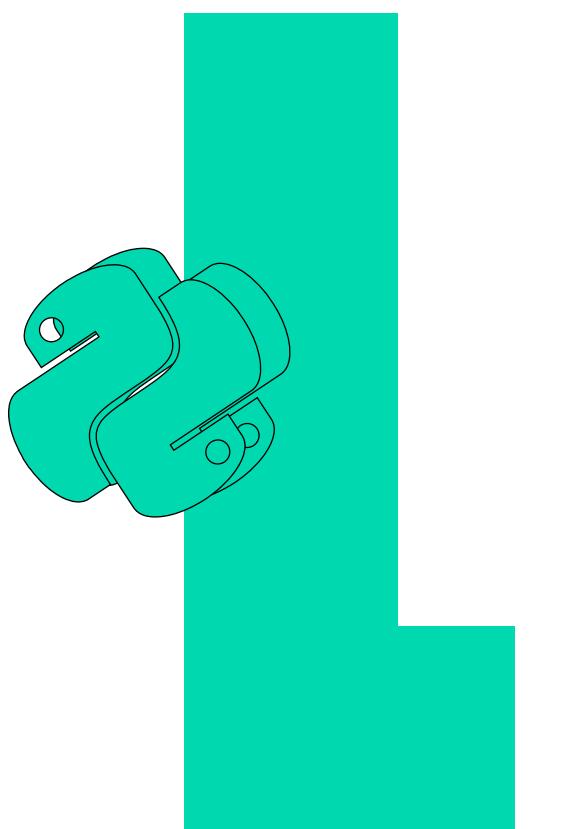
**<a>**: O elemento <a> é usado para criar um link (âncora) para outras páginas, arquivos ou locais na mesma página. Ele é essencial para a navegação e para direcionar os usuários para diferentes partes do seu site ou para outros sites.

**<iframe>**: O elemento <iframe> é usado para incorporar outro documento HTML ou página web em uma página. É frequentemente utilizado para exibir conteúdo de outras fontes, como vídeos do YouTube ou mapas do Google.

Esses são apenas alguns exemplos dos elementos HTML mais comuns. Existem muitos outros disponíveis para criar e formatar diferentes tipos de conteúdo em uma página web. A combinação desses elementos e suas configurações adequadas permite criar uma estrutura sólida e organizada para o seu site.

## 2.2. Criando um arquivo HTML

Assim como qualquer linguagem, ela possui também um próprio sufixo para especificar seus arquivos.



Por padrão, é comum que você crie esse arquivo como sendo index.html. Porém, isso só pode ser feito em um IDE como VSCode ou PyCharm. O Jupyter Notebook já vem configurado com a interpretação do HTML.

Então, para o Jupyter, a etapa anterior de criar um arquivo .html não será necessária.

## 2.3. Criando HTML na prática

Perceba que todo o HTML estará dentro de `<html> </html>` e assim por diante. O cabeçalho é o `<head> </head>` e possui apenas o título `<title> </title>` dentro.

Depois que você criar o `<head> </head>` você precisa criar o corpo do texto `<body> </body>`, sendo o local onde estará todo conteúdo e funcionalidade da sua página.

Dentro deste `<body> </body>` você pode incluir outras funcionalidades como `<p> </p>`, `<h1> </h1>` e `<button> </button>`.

O exemplo abaixo contempla esses parâmetros:

### Exemplo:

```
<html>
<head>
<title>Aprendendo HTML</title>
</head>
<body>

<h1>Meu primeiro H1</h1>
<h2>Que tal um subtítulo só de brincadeira?</h2>
<p style = "color:red;"> Vamos começar a escrever daqui.</p>

<p title = "qualquer coisa, isso é um identificador">Aqui podemos começar outro paragrafo </p>
<a href="https://www.varos.com.br/codigopy">Texto que leva a algum link</a>

<button>Que tal um botão inútil?</button>

</body>
</html>

<table>
<tr>
<th>Empresa</th>
<th>Cotação</th>
<th>Volume</th>
</tr>
<tr>
<td>Weg</td>
<td>10</td>
<td>2000</td>
</tr>
<tr>
<td>Petrobras</td>
<td>20</td>
<td>5000</td>
</tr>
</table>

<br>

<div style="background-color:black;color:white;padding:10px;">
<h2>Codigopy</h2>
<p>Aqui você vai aprender Python pro mercado financeiro.</p>
</div>
```

Resultado:

Meu primeiro H1

Que tal um subtítulo só de brincadeira?

Vamos começar a escrever daqui.

Aqui podemos começar outro paragrafo

Texto que leva a algum link

Que tal um botão inútil?

Empresa	Cotação	Volume
Weg	10	2000
Petrobras	20	5000

Codigopy

Aqui você vai aprender Python pro mercado financeiro.

## Mundo 3

### 3.1. O que é BeautifulSoup

BeautifulSoup é uma biblioteca Python muito utilizada e poderosa usada para extrair dados de documentos HTML e XML.



BeautifulSoup é uma ferramenta fundamental quando se trata de Web Scraping, pois permite analisar e extrair informações de forma estruturada a partir do código HTML de uma página. Com ela, você pode buscar por elementos específicos, como tags, classes, IDs, entre outros, e extrair o texto, atributos e conteúdo dentro desses elementos.

Ela não consegue acessar páginas com tanta complexidade, que utilizam JavaScript, por exemplo, o que a biblioteca Selenium permite (veremos a frente), porém ela é exemplar para analisar páginas em HTML, o que faz se tornar comum a utilização de BeautifulSoup com Selenium.

Existem várias situações em que você pode utilizar esta biblioteca:

**Web Scraping:** Se você precisa extrair dados de um site, como informações de produtos, notícias, preços ou qualquer outro tipo de conteúdo estruturado em uma página HTML, a BeautifulSoup pode ser usada para percorrer o código HTML, encontrar os elementos desejados e extrair os dados relevantes.

**Análise de dados:** A BeautifulSoup pode ser usada para analisar e extrair informações de arquivos HTML/XML em geral, não apenas de páginas da web. Isso pode ser útil para processar e extrair dados de documentos armazenados localmente.

**Limpeza de dados:** Às vezes, o código HTML de uma página pode conter informações indesejadas, como tags ou conteúdo desnecessário. A BeautifulSoup permite limpar e filtrar esses dados, removendo elementos ou conteúdo não desejado e fornecendo uma versão limpa e estruturada do HTML.

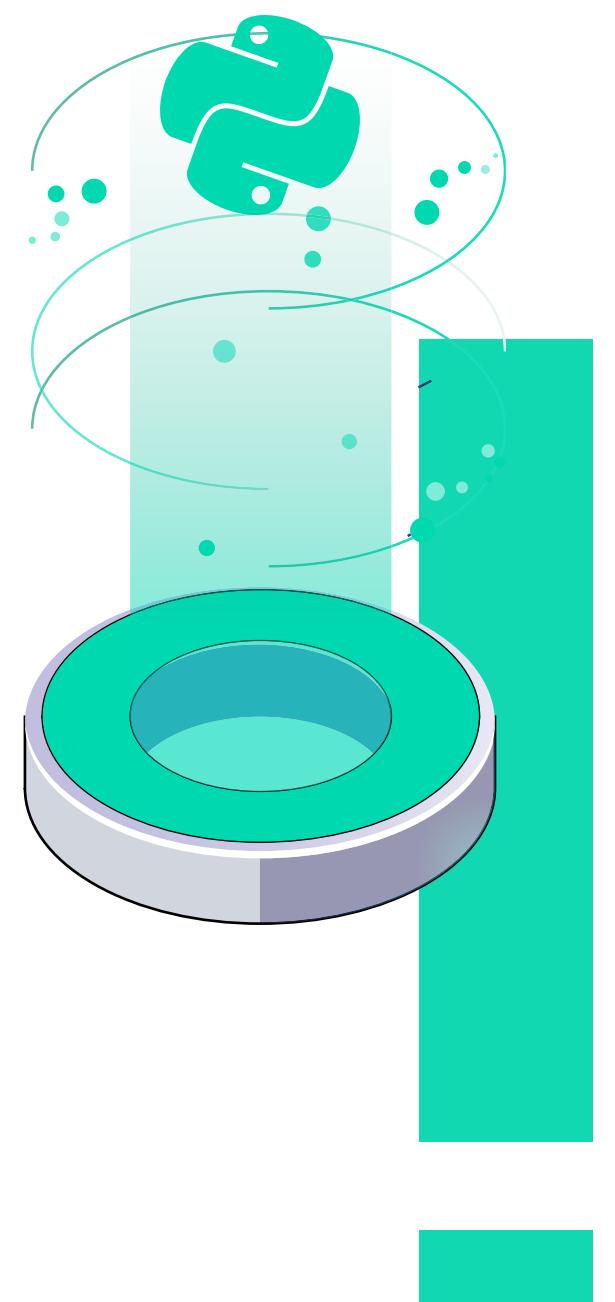
**Manipulação de dados:** Além de extrair dados, a BeautifulSoup também oferece recursos para manipular e modificar o conteúdo HTML. Você pode adicionar, remover ou modificar elementos e atributos de uma página da web, permitindo personalizar o código HTML de acordo com suas necessidades.

### 3.2. Utilizando o BeautifulSoup com Python

#### 3.2.1. Atributos do objeto obtido pelo request

Para poder utilizar o BeautifulSoup, você precisará instalar as dependências necessárias. Então digite os seguintes comandos:

```
!pip install beautifulsoup4  
!pip install lxml
```



### 3.2.2. Fazendo uma requisição básica com BS4

Como dito anteriormente, o Python é uma linguagem de orientação a objeto, por isso que para utilizar o BS4, você precisa utilizar o pacote requests para criar um objeto com todos os atributos de um site, inclusive o seu HTML.

#### Exemplo:

```
import requests
from bs4 import BeautifulSoup

response = requests.get('
https://pt.wikipedia.org/wiki/Popula%C3%A7%C3%A3o_mundial') #vai carregar a pagina

soup = BeautifulSoup(response.text, 'html.parser') #transformar o html da pagina em um objeto do soup
soup
```

#### Resposta:

```
<html class="client-nojs
vector-feature-language-in-header-enabled
vector-feature-language-in-main-page-header-disabled
vector-feature-sticky-header-disabled
vector-feature-page-tools-pinned-disabled
vector-feature-toc-pinned-enabled
vector-feature-main-menu-pinned-disabled
vector-feature-limited-width-enabled
vector-feature-limited-width-content-enabled
vector-feature-zebra-design-disabled" dir="ltr" lang="pt">
<head>
<meta charset="utf-8"/>
<title>População mundial - Wikipédia, a encyclopédia livre</title>
.
.
.
</body>
</html>
```

### 3.2.3. Procurando objetos dentro do HTML com FIND

Com o método FIND você consegue encontrar o tipo de tag que você está procurando, direto ao ponto. Só tome cuidado, pois ele retorna a primeira ocorrência.

obs: Estamos utilizando em conjunto, o método .read\_html() da biblioteca pandas que transforma HTMLs, no formato <table> em um dataframe.



#### Exemplo:

```
import requests
import pandas as pd
from bs4 import BeautifulSoup

response = requests.get('''
https://pt.wikipedia.org/wiki/Popula%C3%A7%C3%A3o_mundial''') #vai carregar a pagina

soup = BeautifulSoup(response.text, 'html.parser') #transformar o html da pagina em um objeto do soup

tabela = soup.find("table") #achar o que você precisa. acabou.

df = pd.read_html(str(tabela))[0]#n pode esquecer do STR!!

display(df)
```

Resposta:

Crescimento da população mundial		
	População	Ano
0	1 bilhão	1804
1	2 bilhões	1927
2	3 bilhões	1960
3	4 bilhões	1974
4	5 bilhões	1987
5	6 bilhões	1999
6	7 bilhões	2011
7	8 bilhões*	2022
8	9 bilhões**	2037
9	10 bilhões**	2057
10	11 bilhões**	2100+

### 3.2.4. Procurando objetos dentro do HTML com FINDALL

Com o método FINDALL você consegue encontrar todas as ocorrências do tipo de tag que você está procurando. Neste caso, será retornada uma lista para você com todas as ocorrências da sua busca.

**Exemplo:**

```
import requests
import pandas as pd
from bs4 import BeautifulSoup
url_vagas = '''https://www.timesjobs.com/candidate/job-search.html?
searchType=personalizedSearch&from=submit&txtKeywords=python&txtLocation='''

response = requests.get(url_vagas)

soup = BeautifulSoup(response.text, 'html.parser')

lista_vagas = soup.find_all("li", class_ = '''clearfix job-bx wht-shd-bx'''")
```

Resposta:

```
[ "HML1", "HMTL2", "HMTL3" ]
```



## Mundo 4

### 4.1. O que é Selenium

O Selenium é uma ferramenta que nos permite automatizar a interação com navegadores web. Você pode pensar nele como um "robô" que controla um navegador, fazendo tudo o que um usuário humano faria, como clicar em botões, preencher formulários e navegar por páginas da web.

O principal do Selenium é no Web Scraping avançado. Às vezes, a extração de dados de um site requer mais do que apenas a análise do código HTML estático. Alguns sites dependem de JavaScript para exibir informações ou carregar conteúdo dinamicamente. Nesses casos, a BeautifulSoup por si só não é suficiente. É aí que entra o Selenium! Ele pode carregar a página em um navegador real, permitindo que o JavaScript seja executado e o conteúdo seja totalmente carregado. Em seguida, você pode extrair os dados diretamente do HTML resultante. Isso é útil quando você precisa acessar informações geradas após interações do usuário, como carregamentos assíncronos ou resultados de pesquisa em tempo real.

Outra aplicação interessante do Selenium é em testes automatizados. Imagine que você tem um site e quer ter certeza de que tudo está funcionando corretamente. Em vez de testar manualmente cada funcionalidade, o Selenium permite que você escreva scripts de teste que executam automaticamente estas ações no site. Ele pode clicar em botões, preencher campos, verificar se os resultados estão corretos e até mesmo tirar screenshots para análise posterior. Isso ajuda a identificar problemas ou erros de forma rápida e eficiente.

A grande vantagem do Selenium é que ele é compatível com vários navegadores populares, como Chrome, Firefox, Safari e Edge.

No entanto, é importante observar que o Selenium pode ser um pouco mais complexo de usar do que a BeautifulSoup. Pois ele requer a instalação de drivers específicos para cada navegador que você deseja controlar e, por ser uma simulação da interação humana, é mais lento do que a extração estática oferecida pela BeautifulSoup. Portanto, o Selenium é mais adequado quando você precisa simular a interação do usuário com o site ou quando o conteúdo dinâmico não pode ser facilmente acessado apenas com a BeautifulSoup.

## 4.2. Utilizando o Selenium com Python

### 4.2.1. Instalando as dependências

Para poder utilizar o Selenium, você precisará instalar as dependências necessárias. Então digite os seguintes comandos:

```
!pip install selenium  
!pip install webdriver-manager
```

A primeira dependência é o Selenium, ele que é o pacote que utilizaremos para realizar o Web Scraping. Dentro deste pacote existem dezenas de outros submódulos.

A outra dependência é o WebDriver Manager, ele que vai gerenciar nossa conexão com os navegadores. Se não fosse ele, a gente precisaria sempre se preocupar com o tipo do navegador, com a versão do navegador e se o drive é compatível com nosso Selenium, em resumo, ele vai nos economizar uma chata dor de cabeça.

## 4.2.2. Utilizando o Selenium

### 4.2.2.1. Importando bibliotecas

É bom ter em sua cabeça que é normal não gravar tudo de primeira, muito desses códigos vocês irão reaproveitar. Para que refazer um trabalho né? Mas é importante que você saiba o que cada coisa faz, na hora de consertar um bug ou mudar uma funcionalidade você já sabe por onde começar.

Seguiremos importando as bibliotecas:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
import pandas as pd
from datetime import datetime
from selenium.webdriver.chrome.options import Options
```

Primeiramente importaremos o webdriver do Selenium, é este o webdriver que será gerenciado pela biblioteca webdriver\_manager. Porém, este é um sub módulo da biblioteca Selenium, não confunda, é ele que fará a integração com o navegador.

Na segunda linha, estamos importando submódulos dentro de submódulos do Selenium, em resumo: estamos interagindo com o serviço do Chrome que ao importar a classe “Service”, você pode instanciá-la e usá-la para iniciar e parar o serviço do ChromeDriver(veremos a frente).

Já a terceira linha é a biblioteca que fará todo o gerenciamento do webdriver.

O quarta linha faz referência a algumas opções que podemos habilitar no Selenium como: aumentar a janela, ocultar janela e outras que veremos à frente.

Os outros módulos dispensam comentários.

#### 4.2.2.2. Se conectando ao navegador

Utilizaremos o método **Options()** para definir que toda a movimentação do Selenium seja feita em segundo plano, existem outras funções que podem ser definidas utilizando este método.

Abaixo, estamos criando um objeto no Python com todas as "funções" do navegador, tudo que você quiser fazer com o navegador será a partir do "driver".

Repare também que estou escolhendo o navegador do Google Chrome, cada navegador tem seus próprios métodos.

Esse comando que estamos fazendo abaixo(`service=Service(ChromeDriverManager().install())`), nos evita uma enorme dor de cabeça porque ele baixa todos os drivers necessários automaticamente, o que precisaria ser feito por você antes.

```
options = Options()
options.headless = True

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

#### 4.2.2.3. Abrindo a página do navegador

Como dito anteriormente, todos os métodos vão ser utilizados com o objeto "driver" que contém os atributos do navegador.

Assim como no módulo de APIs, daremos um **.get()** para entrar na URL que queremos, isso fará com que o navegador abra no seu computador (Caso a opção `headless()` esteja desabilitada).

O método `.implicitly_wait()` define quantos segundos você quer esperar a página carregar, é muito usado no Web Scraping pois é necessário que a página carregue totalmente antes de você acessar seus atributos.

O método `.maximize_window()` faz com que a janela do navegador fique expandida ao máximo, facilitando a visualização.

```
url = f'''  
https://www2.bmf.com.br/pages/portal/bmfbovespa/boletim1/SistemaPregao1.asp?pagetype=pop&caminho=Resumo%20  
Estat%EDstico%20-%20Sistema%20Preg%E3o&Data=03/02/2023&Mercadoria=DI1  
'''  
  
driver.get(url)  
driver.implicitly_wait(3)  
driver.maximize_window() #maximar a tela do navegador do bot
```

#### 4.2.2.4. Abrindo a página do navegador

Essa parte é bem parecida com a parte de BeautifulSoup, depois de ter carregado a página completamente, acesse os elementos dela pelo “Full Xpath”.[ Só clicar com botão direito no objeto “copy” >> “copy full xpath”].

Após ter copiado o caminho do elemento, vamos transformá-lo em um objeto “elemento” dentro do Python, com os seguintes comandos:

```
local_tabela = ''  
/html/body/div/div[2]/form[1]/table[3]/tbody/tr[3]/td[3]/table  
'''  
  
elemento = driver.find_element("xpath", local_tabela)
```

Após ter criado o objeto é só acessá-lo através do comando `get_attribute('outerHTML')` e transformar este objeto em uma tabela utilizando o método do pandas.

```
html_tabela = elemento.get_attribute('outerHTML')

tabela = pd.read_html(html_tabela)[0]
```

### Resultado:

0	1	2	3	4	5	6	7	8	9	10	
0	AJUSTE ANTER. (3)	AJUSTE CORRIG. (4)	PREÇO ABERT.	PREÇO MÍN.	PREÇO MÁX.	PREÇO MÉD.	ÚLT. PREÇO	AJUSTE	VAR. PTOS.	ÚLT. OF. COMPRA	ÚLT. OF. VENDA
1	99.190,76	99.190,76	13652	13650	13656	13651	13652	99.190,72	0,04-	13650	13652
2	98.037,60	98.037,60	13660	13658	13680	13668	13670	98.037,65	0,05+	13670	13672
3	97.142,56	97.142,56	13700	13674	13700	13679	13692	97.142,27	0,29-	13682	0000
4	96.055,26	96.055,26	13705	13705	13770	13727	13765	96.048,28	6,98-	13735	13770
5	95.027,78	95.027,78	13725	13715	13820	13763	13790	95.015,41	12,37-	13785	13790
6	94.006,29	94.006,29	13740	13740	13850	13768	13835	93.987,94	18,35-	13715	0000
7	92.912,59	92.912,59	13780	13775	13880	13803	13865	92.879,25	33,34-	13765	13860
8	91.975,18	91.975,18	13765	13730	13905	13794	13855	91.925,43	49,75-	13855	13870
9	91.001,77	91.001,77	13765	13740	13905	13861	13875	90.937,38	64,39-	13700	13885
10	90.091,18	90.091,18	13750	13750	13815	13798	13805	90.015,94	75,24-	0000	13865
11	89.209,65	89.209,65	13710	13660	13885	13744	13830	89.116,51	93,14-	13825	13835
12	88.249,78	88.249,78	13670	13665	13820	13685	13810	88.133,30	116,48-	13800	0000
13	86.582,36	86.582,36	13675	13570	13815	13658	13740	86.437,72	144,64-	13735	13745
14	84.065,75	84.065,75	13500	13415	13690	13514	13590	83.856,82	208,93-	13580	13590
15	81.601,58	81.601,58	13325	13240	13525	13350	13410	81.320,60	280,98-	13405	13410
16	79.309,22	79.309,22	13160	13080	13385	13191	13275	78.969,52	339,70-	13270	13280
17	77.111,75	77.111,75	13105	13015	13330	13128	13210	76.720,39	391,36-	13205	13235
18	74.959,67	74.959,67	13015	12970	13290	13089	13175	74.520,57	439,10-	13165	13185

Após ter alcançado este resultado faltarão apenas a formatação dos dados.



## Mundo 5

### 5.1. Iframe

Neste mundo, aprenderemos como resgatar informações do índice Ibovespa diretamente da B3, mas para isso você precisa entender um conceito antes.

No site da B3 existe um imprevisto que é bastante comum, principalmente em sites que abrangem o mercado financeiro no Brasil. Na composição do HTML existe um campo chamado <iframe> que impede você de acessar informações no HTML diretamente.

Este <iframe> é uma forma de você compartilhar, vídeos, imagens ou qualquer link por meio do seu site. E ele é muito utilizado para se colocar um site dentro de outro site, é uma forma de melhorar a estilização do site criando esta “casca”, então sempre que você se deparar com um <iframe> você precisará informar ao seu programa que se trata de um <iframe> e fazer as modificações necessárias.

### 5.2. Composição do Ibovespa

O objetivo deste código é fazer um download de uma planilha Excel no site da B3.

Para começar, acessemos o site do Ibovespa através do Python, utilizando o script a seguir:

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support.ui import Select

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

driver.get(''https://www.b3.com.br/pt_br/market-data-e-indices/indices/indices-amplos/indice-ibovespa-ibovespa-composicao-da-carteira.htm''')
```

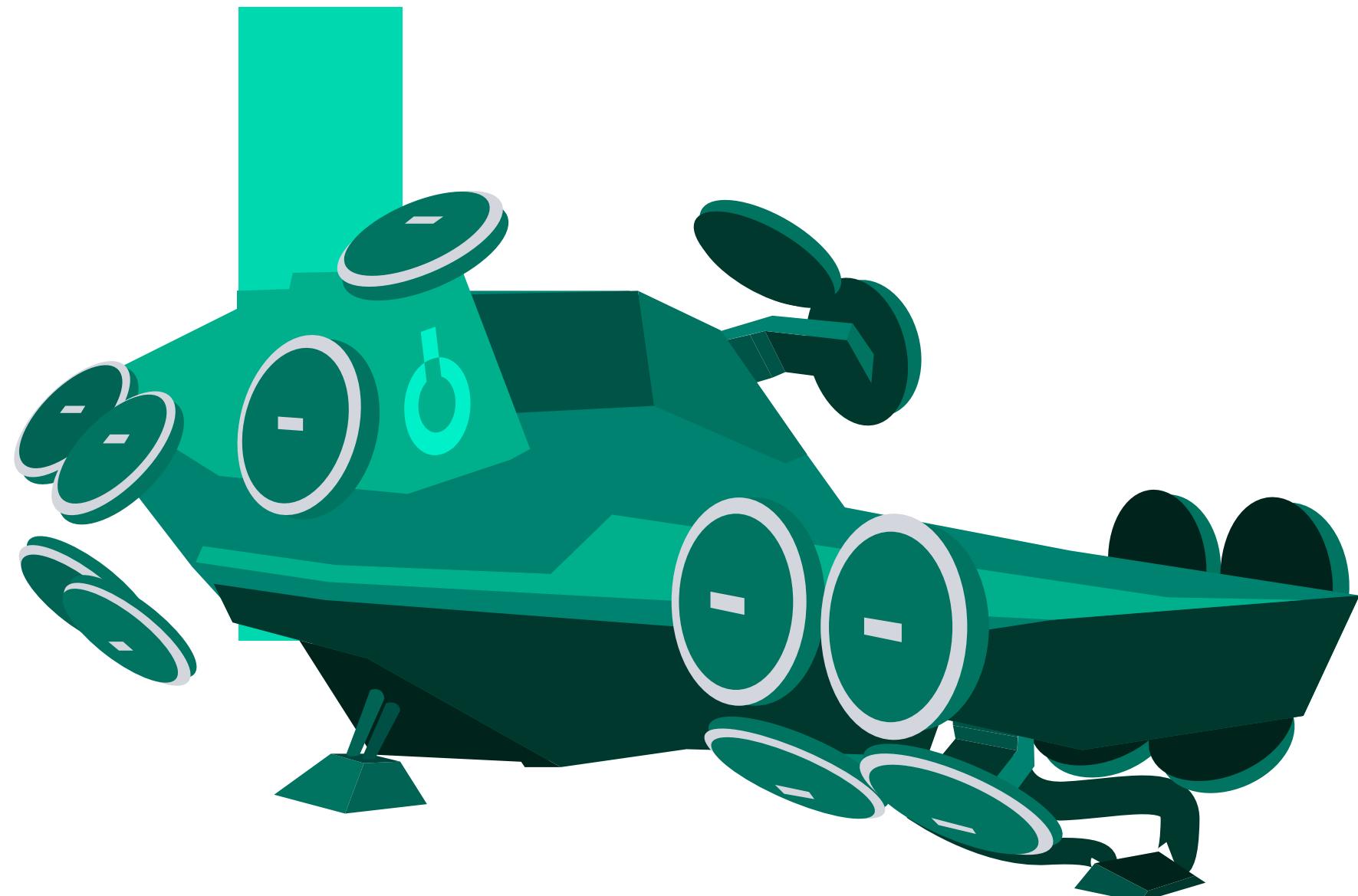
Como dito anteriormente, você não precisa gravar estes submódulos de primeira, mas é importante que você saiba do que se trata. Estes submódulos novos serão utilizados na hora de trocar o site para o iframe dele. Isso porquê como temos um site dentro de um site, ele precisará ser direcionado ao novo local de acesso:

```
WebDriverWait(driver, 10).until(EC.frame_to_be_available_and_switch_to_it((By.ID, "bvmf_iframe")))
```

O comando acima informa que faremos uma espera de 10 segundos, para o site responder, e fará a troca do iframe utilizando o id dele **"bvmf\_iframe"**.

Após informar ao programa que existe um iframe, só seguir normalmente o script.

Selecionamos a caixa de seleção, através do id, e procuraremos pelo valor “60” na caixa, é ele que informará a quantidade de dados que estarão disponíveis para gente.

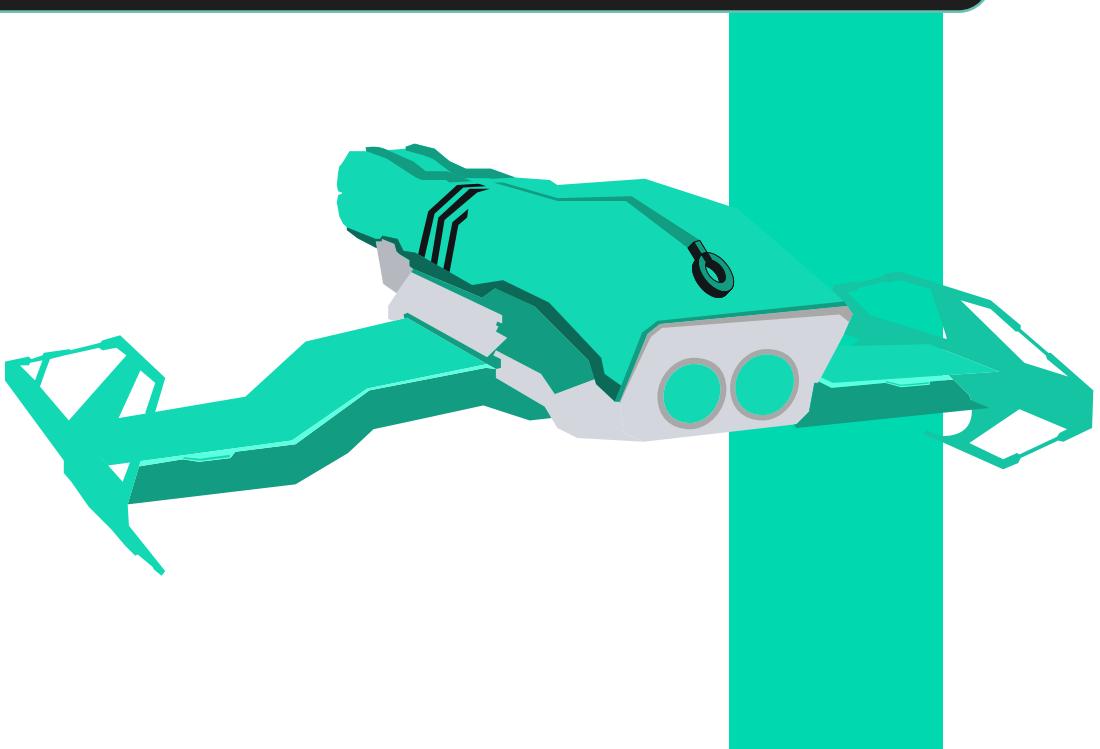


```
botao_de_selecao = driver.find_element("id", "selectPage")  
  
caixa_de_selecao = Select(botao_de_selecao).select_by_visible_text("60")
```

Com o intervalo de dados selecionado, poderemos clicar no botão de download através do comando “execute\_script()”

```
planilha = driver.find_element("xpath", '//*[@id="divContainerIframeB3"]/div/div[1]/form/div[2]/div/div[2]/div/div/div[1]/div[2]/p/a')  
driver.execute_script("arguments[0].click();", planilha)
```

E pronto, download concluído.



## Mundo 6

### 6.1. Projeto

Neste mundo, aprenderemos a extrair informações do histórico de apostas da bet365, este é um projeto bem legal que envolve muitos desafios como preenchimento de formulários.

### 6.2. Código da aula explicado

#### 6.2.1. Importando as dependências

Assim como feito anteriormente, nesta primeira parte importamos os módulos necessários para se conectar ao site da bet365.

Repare que nesta parte estamos importando as nossas variáveis de ambiente que serão usadas para autenticação do login no site.

Nada de novo, estamos criando um objeto “driver” com as funções do Selenium e abaixo estamos chamando o método dele que faz uma requisição http, esta parte é muito parecida com a do módulo “Requests”

## 6.2.2. Saindo do IFRAME

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import os

%load_ext dotenv

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

url = '''https://www.bet365.com/?#/ME/X8020'''

driver.get(url)

driver.maximize_window()

usuario = os.getenv('usuario')
senha = os.getenv('senha')
```

Precisaremos fazer os comandos necessários neste projeto para desativar o IFRAME e conseguirmos acessar o site, igual ao que realizamos no mundo anterior.

Só que para isso, precisaremos importar alguns submódulos da biblioteca selenium:

Basicamente, este comando indica ao objeto “driver”, criado com as funcionalidades do selenium, que: será feita uma espera, de no máximo 10 segundos, até que o objeto execute a tarefa.

Vale salientar mais uma vez que muito do que estamos fazendo, você não precisa gravar, você só precisa entender o que está fazendo no código. Porque provavelmente você utilizará em outras ocasiões.

```
from selenium import webdriver
from selenium.webdriver.chrome.service import Service
from webdriver_manager.chrome import ChromeDriverManager
from bs4 import BeautifulSoup
import numpy as np
import pandas as pd
import os
from selenium.webdriver.common.keys import Keys
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait

%load_ext dotenv

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

url = '''https://www.bet365.com/?#/ME/X8020'''

driver.get(url)

driver.maximize_window()

usuario = os.getenv('usuario')
senha = os.getenv('senha')

WebDriverWait(driver, 10).until(
EC.frame_to_be_available_and_switch_to_it((By.NAME, "undefined")))
```

### 6.2.3. Preenchendo formulários

Nesta etapa do projeto, preencheremos os campos de login e senha com as informações em nossas variáveis de ambientes exportadas corretamente. Se deve ter muito cuidado sobre informações sensíveis dentro dos seus programas, por isso sempre que tiver uma informação sensível como uma senha ou token, utilize a variável de ambiente.



Como feito em outros exercícios, você vai até o campo de senha e login, clique com botão esquerdo em “inspecionar”, clique em “copy” e “copy full XPATH”. Depois que você fizer isso, você terá o caminho necessário dos objetos para utilizar os métodos do selenium.

Repare que fizemos de formas diferentes, para o Login e a Senha, o primeiro jeito é mais simples, mais direto.

O segundo jeito é um pouco mais específico e mais cuidadoso na hora de achar um elemento. Este jeito é melhor por definir um tempo de espera e por especificar que um elemento é clicável, isto tudo minimiza o erro.

Após inseridos senha e login, vá até o botão de entrar, copie sua localização e insira no método do selenium, com uma função “.click()”, ao final, que indica que este botão está sendo clicado.

```
caixa_usuario = driver.find_element("xpath", '/html/body/div[1]/div/div[1]/div/div/div/div[1]/div[2]/div/input')

caixa_usuario.send_keys(usuario)

WebDriverWait(driver, 10).until(EC.element_to_be_clickable(
(By.XPATH, '//*[@id="password"]')).send_keys(senha)

#clicando no botão de login de uma forma diferente

WebDriverWait(driver, 10).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div/div/div/div[2]/button')).click()
```

## 6.2.4. Preenchendo formulários

Após inserir as credenciais corretamente, você será redirecionado para esta tela, nesta parte teremos que definir o intervalo que queremos. Note no aviso abaixo que o intervalo máximo de informações é de 6 meses.

Para isso, neste script a seguir estamos primeiramente clicando na opção onde será possível selecionar o intervalo de datas.

Depois abriremos o calendário, voltaremos 5 meses e selecionaremos o dia ideal.

```
#selecionando intervalo de datas
WebDriverWait(driver, 10).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div[2]/div/div[2]/div[1]/div[3]')).click()

#abrindo calendário
WebDriverWait(driver, 10).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div[2]/div/div[2]/div/div[2]/div[1]/div[1]/div[2]')).click()

#voltando 5 meses
for i in range(5):
    WebDriverWait(driver, 10).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div[2]/div/div[2]/div/div/div[2]/div[1]/div[2]/div[1]')).click()

#selecionando o dia do mês
WebDriverWait(driver, 25).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div[2]/div/div[2]/div/div/div[2]/div[2]/div[1]/div[1]/div[2]/tbody/tr[1]/td[7]')).click()
```

### 6.2.5. Abrindo todo o histórico

Nesta parte do código, carregaremos todo o histórico possível clicando em “ver mais”, para na hora de a gente capturar o HTML, só precisar efetuar uma requisição, ao invés de 30.

Este comando é composto de uma estrutura de repetição que funcionará até que o comando dê erro, ou seja, quando não houver mais o botão “ver mais”, o que significa que todo o histórico foi carregado.

```
while True:
    try:
        WebDriverWait(driver, 25).until(EC.element_to_be_clickable(
(By.XPATH, '/html/body/div[1]/div/div[1]/div[2]/div/div[2]/div/div/div[3]')).click()
    except:
        break
```

## 6.2.6. Carregando o html com BeautifulSoup

O BeautifulSoup é um pacote mais leve e mais eficiente do que o selenium, só que como dito em outras aulas, ele tem suas limitações. Por isso é preciso utilizar os dois juntos.

O Selenium a gente utiliza para carregar todas as informações e tirar um “print” e o BeautifulSoup a gente usa para manusear as informações em HTML.

Primeiramente, estamos carregando todo o HTML para que ele seja acessível.

```
inspect = '/html/body/div[1]/div/div[1]/div[2]/div[2]/div/div[2]/div/div[2]'  
element = driver.find_element('xpath', f'{inspect}')  
html_element = element.get_attribute('outerHTML')  
soup = BeautifulSoup(html_element, 'html.parser')  
  
soup
```

Após carregar todo o HTML, vamos dividir as partes deles no que achamos necessário.

Óbvio que isso não é feito de forma fluída, tem que parar, ler e entender o HTML e isso leva tempo, então não se preocupe se você não entender.

Abra a página, leia o HTML e prossiga com a explicação.

Criamos um objeto “dados”, com dicionários vazios para armazenar as informações desejadas e criamos objetos com cada informação separada, para ficar mais fácil de percorrer:

```
dados = {'datas': [], 'apostas': [], 'odds': [],  
'valores': [], 'retornos': [], 'detalhes': []}  
  
datas = soup.find_all("div", {"class": "h-BetSummary_DateAndTime"})  
valores = soup.find_all(  
"div", {"class": "h-StakeReturnSection_StakeContainer"})  
retornos = soup.find_all("div", {"class": "h-StakeReturnSection_ReturnText"})  
apostas = soup.find_all("div", {"class": "h-BetSummary"})  
detalhes = soup.find_all("div", {"class": "h-StakeDescription_Text"})
```

Após criarmos cada objeto iterável, percorremos cada um, pegando apenas a parte que importa e jogando para o dicionário criado “dados”, transformando-o em um Dataframe no final.

```
i = 0

for data, valor, retorno, aposta, detalhe in zip(dados, valores, retornos, apostas, detalhes):
    tipos = aposta.find_all("div", {"class": "h-BetSelection_Name"})
    odds = aposta.find_all("div", {"class": "h-BetSelection_Odds"})
    #isso aqui pra conferir se é uma múltipla
    if tipos == []:
        tipos = aposta.find_all(
            "div", {"class": "h-BetBuilderSelection_SelectionLabel"})
        odds = aposta.find_all(
            "div", {"class": "h-BetBuilderMultipleSelections_OddsLabel"})

    aposta_string = ""
    for possibilidades in tipos:
        aposta_string = aposta_string + " & " + possibilidades.text
        aposta_string = aposta_string[2:] #pra não ter que fazer um exceção de primeiro elemento no loop

    valores_odds = []
    for odd in odds:
        spans = odd.find_all('span')
        for span in spans:
            if i == 8:
                print(span)
    #isso aqui serve pra não pegar coisa errada quando existe um boost na aposta
            if not 'h-BetOdds' in str(span):
                valores_odds.append(float(span.text))
    .replace(',', '.'))
    #nós usamos "," para decimal
    valor_final = np.prod(np.array(valores_odds)) #pra calcular a múltipla temos multiplicar as odds

    dados['apostas'].append(aposta_string)
    dados['datas'].append(data.text)
    dados['valores'].append(valor.text)
    dados['retornos'].append(retorno.text)
    dados['odds'].append(valor_final)
    dados['detalhes'].append(detalhe.text)
    i = i + 1

df = pd.DataFrame(dados)
display(df)
```

## Mundo 7

### 7.1. Projeto

Neste mundo, aprenderemos a utilizar o PyAutogui . Diferentemente do Selenium e do BeautifulSoup, ele é literalmente um robô que mexe seu mouse e teclado, é a forma menos programática de automatizar as tarefas.

O PyAutogui é geralmente utilizado para automações no computador, ou seja, criar pastas, excluir arquivos, etc. Porém, ele possui também a possibilidade de extração de dados na Web, aliás, ele está simulando um humano, tudo que você pode fazer ele também pode.

A parte ruim de usar o PyAutogui é que o computador fica inativo, pois ele está, literalmente, utilizando o seu mouse e o seu teclado.

## 7.2. Ponta pé inicial

### 7.2.1. Instalando as dependências

Para instalar o pacote do PyAutogui, é só utilizar o comando a seguir:

```
pip install pyautogui
```

### 7.2.2. Importando as dependências

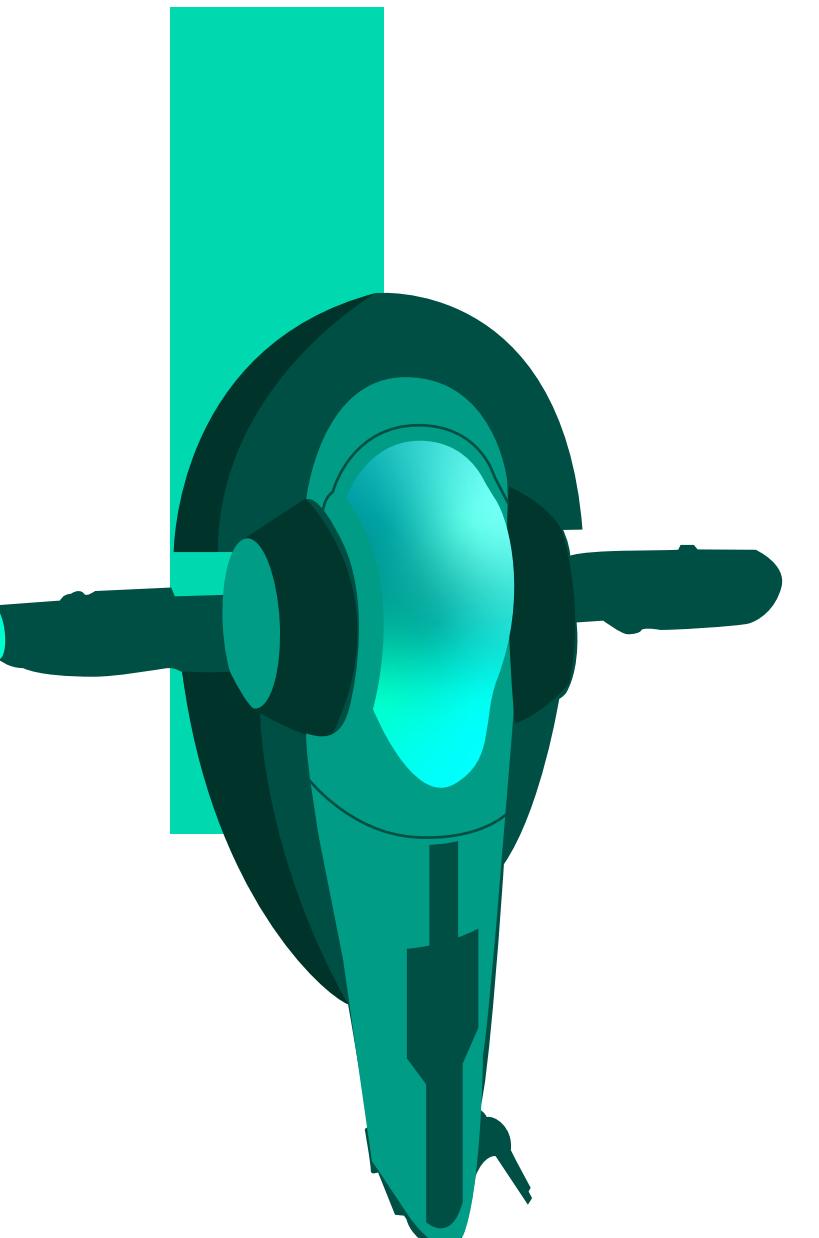
Importaremos as dependências necessárias. Diferentemente dos projetos anteriores, não precisaremos importar aquela quantidade enorme de módulos e submódulos.

Importaremos o módulo time, pois ele fará a espera entre uma página e outra, esperando uma página carregar, por exemplo.

```
import pyautogui  
import time
```

## 7.3. Pegando coordenadas do mouse

Este comando é utilizado para pegar as coordenadas do mouse. Elas podem ser necessárias, para você identificar a localização de um certo arquivo supondo que o seu computador é um plano cartesiano. Ou seja, para descobrir a localização de um arquivo, só colocar o mouse em cima dele.



Uma observação importante é que esta posição pode mudar de um computador para o outro, porque cada um tem uma quantidade de pixel, que basicamente determina as posições no plano cartesiano.

```
time.sleep(2) #pode ser usado pra esperar um sistema abrir  
  
posicao_mouse = pyautogui.position()  
  
print(posicao_mouse)
```

#### 7.4. Botão de clicar do mouse

Este comando é utilizado para pegar clicar com o mouse. É necessário passar as posições que você deseja clicar. Por isso o comando de cima, que determina as coordenadas do mouse, é tão importante.

Repare que você pode determinar:

- Com qual botão você quer clicar
- A quantidade de cliques
- O intervalo em segundos entre os cliques

OBS: Outro ponto importante de notar, é que as posições extraídas acima foram salvas em uma variável, o que as tornou referenciável posteriormente.

```
pyautogui.click(x=posicao_mouse[0], y=posicao_mouse[1],  
clicks=1, interval=0, button='left') #testar com 2 clicks
```

## 7.5. Movendo o mouse de posição

Este comando é pode ser utilizado, principalmente, para posicionar o seu mouse antes de um clique, por exemplo:

Rpare que além das posições finais do seu mouse, você pode determinar também em quantos segundos ele realiza esse movimento.

```
pyautogui.moveTo(x=posicao_mouse[0], y=posicao_mouse[1], duration=3)
```

## 7.6. Escrevendo com o teclado

Este comando é utilizado para simular o seu teclado. Você pode escrever aonde quiser, é só utilizar o seguinte comando:

```
pyautogui.click(x=posicao_mouse[0], y=posicao_mouse[1],  
clicks=1, interval=0, button='left')  
  
pyautogui.write("teste")
```

## 7.7. Pressionando uma tecla

Este comando é utilizado para simular o seu teclado também, então você pode utilizar em conjunto com o comando de cima. Escrever em um formulário e depois o enviar com a tecla enter.

```
pyautogui.press("enter")
```

## 7.8. Abrindo o navegador

Neste mini projeto, direcionamos o mouse até o botão iniciar. Vamos digitar pelo navegador desejado, esperar ele carregar, digitar a página que desejamos acessar e fim.

```
pyautogui.click(x=posicao_mouse[0], y=posicao_mouse[1],  
clicks=1, interval=0, button='left')  
  
pyautogui.write("brave")  
  
pyautogui.press("enter")  
  
time.sleep(1)  
  
pyautogui.write("www.globo.com")  
  
pyautogui.press("enter")
```

## 7.9. Segurando, pressionando e soltando a tela

Estes comandos vão simular uma pessoa segurando e soltando teclas. Eles serão muito utilizados para quando precisarmos usar atalhos.

A diferença entre segurar e pressionar, é que enquanto você não soltar a tecla ela considerará que a tecla está apertada. Ou seja, os comandos serão diferentes, já que, por exemplo, o "Ctrl" está pressionado.

### 7.9.1. Segurando uma tecla

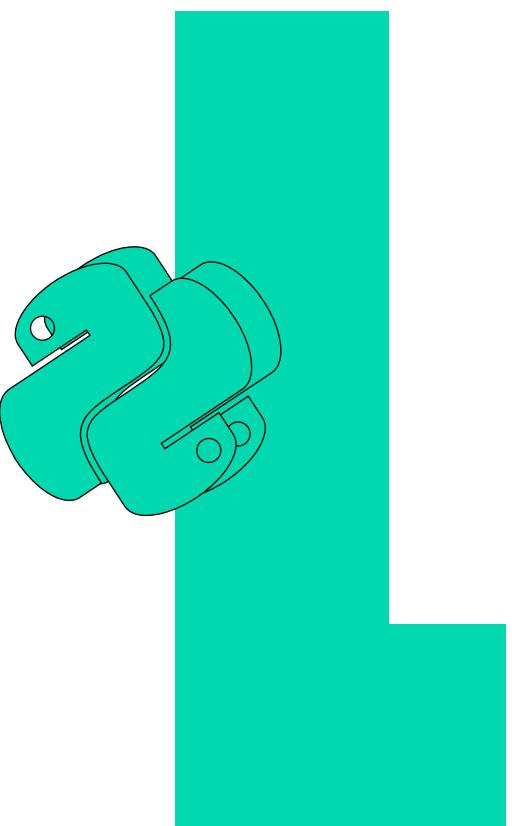
```
pyautogui.keyDown('ctrl')
```

### 7.9.2. Pressionando uma tecla

```
pyautogui.press('pgdn')  
pyautogui.press('pgdn')  
pyautogui.press('pgdn')
```

### 7.9.3. Soltando uma tecla

```
pyautogui.keyUp('ctrl')
```



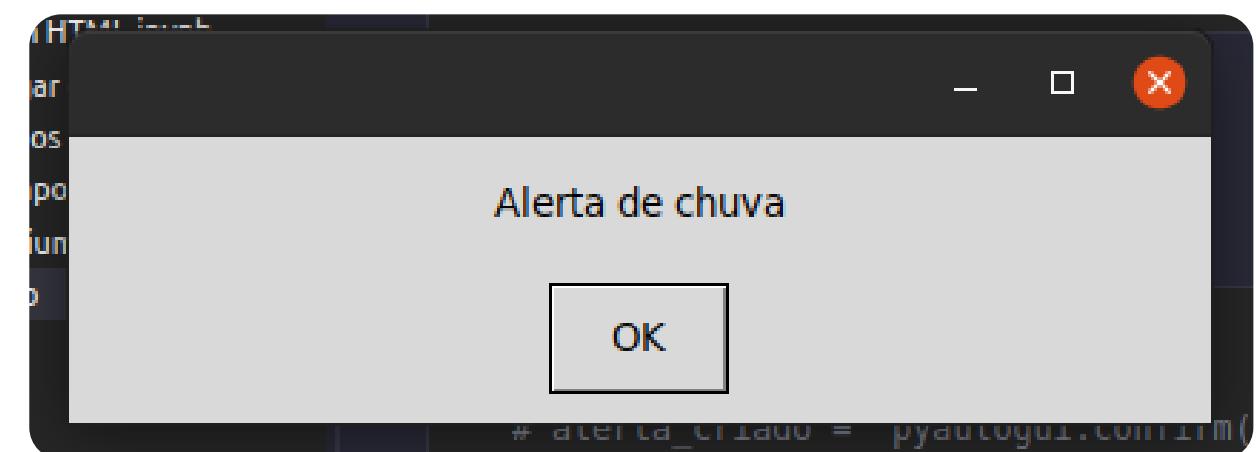
### 7.10. Pressionando uma tecla

Com PyAutogui você pode criar caixas, que servem como alertas ou que podem ser interativas.

#### 7.10.1. Criando alerta

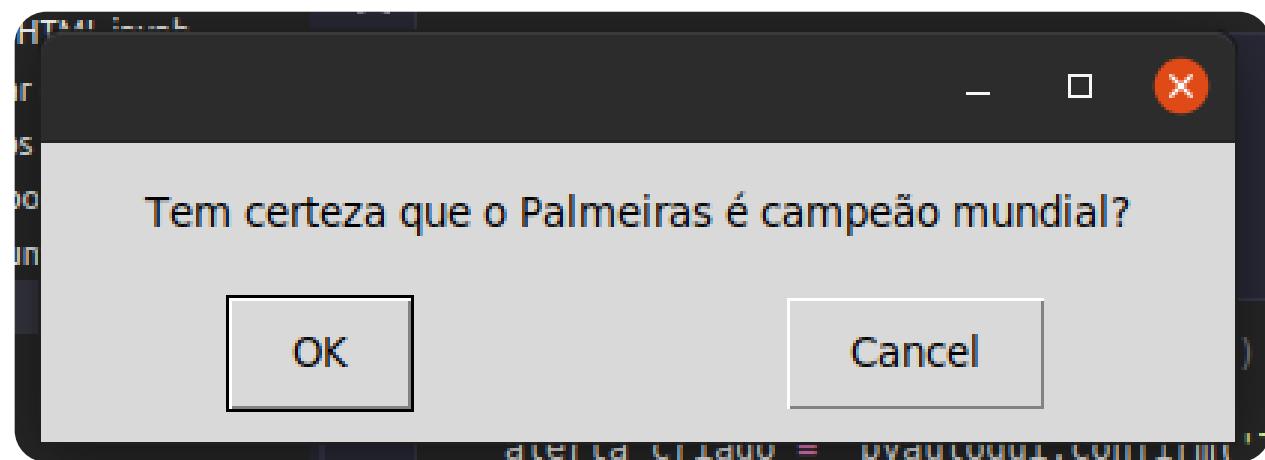
Com o comando a seguir podemos criar alertas que servem para informar um usuário.

```
pyautogui.alert('Alerta de chuva')
```



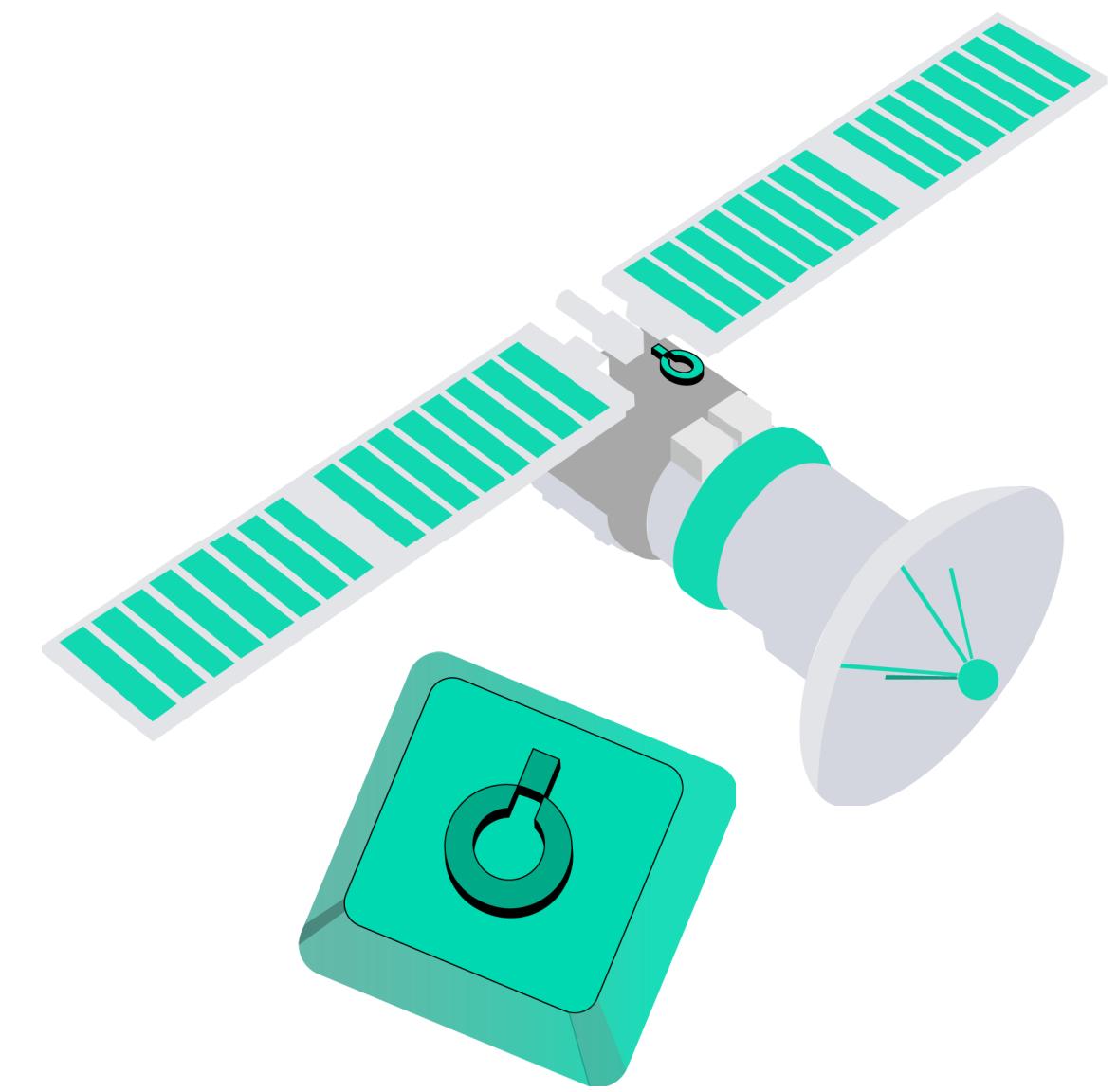
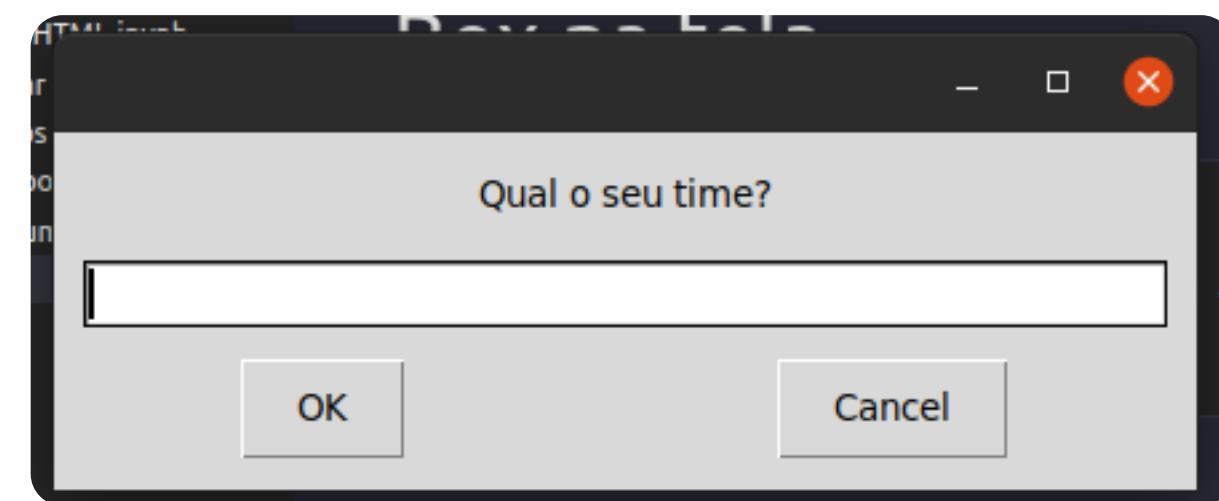
### 7.1O.2. Criando uma confirmação

Com o comando a seguir podemos criar confirmação e armazenar a resposta de um usuário, para utilizar diversos condicionais para guiar nosso código de acordo com que usuário escolher.



### 7.1O.3. Criando um prompt

Com o comando a seguir podemos criar um prompt de comando para interagir com um usuário. E assim como na confirmação, também será possível que você armazene a resposta do usuário.



## 7.11. Projeto: Abrindo uma planilha

Neste projeto utilizaremos todo conhecimento adquirido nesta aula para abrir uma planilha do excel no nosso computador e pegar uma informação.

Repare que neste projeto estamos clicando no menu iniciar, escrevendo o nome da planilha, selecionando a planilha desejada e retirando informações dela.

É um super projeto, que por mais que seja simples, tem potencial de te agilizar MUITO. Esse é o bom do Python, o simples, bem feito, é transformador.

```
nome_planilha = 'ValuationLojasQuero'

pyautogui.click(x=24, y=1072,
clicks=1, interval=0, button='left')

pyautogui.write(nome_planilha)

time.sleep(1)

pyautogui.press("enter")

time.sleep(7)

pyautogui.click(x=731, y=737,
clicks=1, interval=0, button='left')

time.sleep(1)

pyautogui.click(x=144, y=124,
clicks=1, interval=0, button='left')

time.sleep(1)

pyautogui.click(x=1788, y=7,
clicks=1, interval=0, button='left')

upside_da_acao = pyautogui.prompt('Qual o Upside?')

time.sleep(2)

pyautogui.press("enter")

print(upside_da_acao)
```