🕷️ Mastering Selenium in Python — Step-by-Step Guide for Beginners

🚀 Introduction

Selenium is one of the most powerful and widely used tools for **automating web browsers**. Whether you want to **scrape data**, **automate logins**, **click buttons**, or **fill forms**, Selenium does it all — just like a human interacting with a browser.

In this tutorial, you'll learn:

- How to install Selenium
- How to launch a browser and visit websites
- How to find and interact with elements
- How to scrape data from web pages
- Real-world automation examples

Features :

- Supports multiple browsers and programming languages.
- Enables cross-browser and cross-platform testing.
- Facilitates parallel test execution with Selenium Grid.
- Open-source with a strong community for support.
- Integrates with CI/CD tools for continuous testing.

Challenges in Selenium Testing :

- Steep learning curve for beginners.
- Limited support for desktop and mobile app testing.
- Requires external tools for reporting and advanced functionality.
- Debugging issues can be time-consuming.
- Maintenance of flaky tests due to dynamic web elements.

📚 Table of Contents

1. **Introduction** – Overview of Selenium, its features, and challenges.
2. **Install Selenium** – Setup instructions for Selenium and ChromeDriver.
3. **Basic Script** – Launch browser and visit a website.
4. **Find Elements** – Locate input fields and interact using XPath.
5. **Click & Scroll** – Perform button clicks and scroll the page.
6. **Scrape Saved HTML** – Save and parse a webpage using BeautifulSoup.
7. **Live Scraping** – Use Selenium + BeautifulSoup to extract live data.
8. **Create DataFrame** – Convert scraped data into a Pandas DataFrame.
9. **Selenium Functions** – Handy browser control methods like refresh and back.
10. **Project Ideas** – Real-world project suggestions from easy to advanced.

⚙️ Step 1: Install Selenium and ChromeDriver

First, you need to install Selenium:

--- **pip install selenium**

Then, download ChromeDriver based on your Chrome version:

1. Visit: https://googlechromelabs.github.io/chrome-for-testing/
2. Download the version that matches your Chrome.
3. Extract the .exe file and **copy the path**.

💡 **Tip** :

Prothomei Selenium ebong ChromeDriver setup ta bhalo moto korte hobe. Na hole kichu kaj korbe na!

A Sample Test Script Using Selenium WebDriver

Here's how to open a browser and visit a website using Selenium:

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

# Setup ChromeDriver path
service = Service("E:/chromedriver.exe")  # Replace with your actual path
driver = webdriver.Chrome(service=service)

# Open a website
driver.get("https://www.google.com")

# Wait a few seconds
time.sleep(3)

# Close browser
driver.quit()
```

✅ Step-by-Step Breakdown

1. Import Required Libraries

webdriver: controls the browser

Service: connects ChromeDriver to the script

By: used to find elements by name, ID, class, etc.

time: allows us to pause the program

Specifies the path to chromedriver.exe. Launches Chrome using that driver.

Note : You might need to change the Backslash(\) to Forward slash(/) after pasting the chromedriver.exe path!

In the code we are using **driver.get** function to enter in a website (https://www.google.com). And using a timer to load the window for 3 seconds!

driver.quit (Closes the entire browser window and ends the session)

💡 **Tip** :

Dekhtei pabe browser ta automatic open hoye gelo. Eta shudhu suru, aro onek moja ache!

🔍 Step 3: Find and Interact with Elements

You can use Selenium to find buttons, input fields, links, etc., and interact with them like a human.

Example: **Search on Google**

```
driver.get("https://www.google.com")
time.sleep(2)

search_box = driver.find_element(By.XPATH, "//input[@name='q']")  # Find search box using XPath
search_box.send_keys("Selenium Python")  # Type query
search_box.submit()  # Submit search

time.sleep(3)
driver.quit()  # Close browser
```

Explanation of the Code:

1. **Opens** Google Chrome and navigates to [google.com].

2. **Finds the search box using XPath**, types "Selenium Python", and submits the search.

3. **Waits 3 seconds**, then **closes the browser**.

4. You can **element.send_keys(Keys.ENTER)**

🔍 **How to Inspect and Copy XPath:**

1. **Right-click** on the element (e.g., search box) in the browser.

2. Click **"Inspect"** to open DevTools.

3. Right-click the highlighted HTML and choose **"Copy → Copy XPath"**.

💡 **Bangla Tip**:

Inspect diye element select kore **"Copy XPath"** nile shobcheye accurate path ta pawa jay — onek helpful dynamic site er jonno!

🎯 **Step 4: Click a Button**

```
button = driver.find_element(By.XPATH, "//button[@id='submit']")
button.click()
```

Finds the button by XPath and performs a click.

✅ **Step 5: Scroll the Page**

```
driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")
```

Scrolls the page to the bottom using JavaScript.

🔠 Step 6: Scrape from a Saved HTML File

```
# Get HTML source and write to file
html = driver.page_source
with open("page.html", "w", encoding="utf-8") as f:
    f.write(html)

driver.quit()
```

📜 What This Does:

- Visits the website
- Gets the **entire HTML content**
- Saves it into **page.html** in the same directory

```
from bs4 import BeautifulSoup

with open("page.html", "r", encoding="utf-8") as file:
    soup = BeautifulSoup(file, "html.parser")

titles = soup.find_all("h2")  # Example: get all <h2> tags
for title in titles:
    print(title.text)
```

📝 What's happening here?

- Selenium opens the browser and loads the webpage.
- **driver.page**_source captures all the HTML of the page.
- We write that content to a file named **page.html**.

✅ 2. Scrape Directly with Selenium + BeautifulSoup

🔸 **Step**:

- Load the page using **Selenium**.
- Use **.page_source** with **BeautifulSoup.**

```
driver = webdriver.Chrome(service=Service("E:/chromedriver.exe"))  # Update path
driver.get("https://example.com")
time.sleep(2)

soup = BeautifulSoup(driver.page_source, "html.parser")
items = soup.find_all("h2")
for item in items:
    print(item.text)

driver.quit()
```

⚙️ Step 7: Creating a dataframe

```
with open('smartprix.html', 'r', encoding='utf-8') as f:
    soup = BeautifulSoup(f.read(), 'lxml')

names, prices, ratings = [], [], []

for i in soup.find_all('div', {'class': 'sm-product has-tag has-features has-actions'}):
    try:
        names.append(i.find('h2').text.strip())
    except:
        names.append(np.nan)
    try:
        prices.append(i.find('span', {'class': 'price'}).text.strip())
    except:
        prices.append(np.nan)
    try:
        ratings.append(i.find('div', {'class': 'score rank-2-bg'}).find('b').text.strip())
    except:
        ratings.append(np.nan)

df = pd.DataFrame({
    'name': names,
    'price': prices,
    'rating': ratings
})
```

explanation of the code:

1. **Reads the saved HTML file** using BeautifulSoup with the lxml parser.
2. **Finds all product containers** using their class name.
3. **Extracts the name, price, and rating** for each product using .find() and appends them to lists.

4. **Handles missing values** with try-except, storing np.nan if data is unavailable.
5. **Creates a Pandas DataFrame** from the collected lists for structured data analysis.

🔡 Step 8: Useful Selenium Functions

Function	What It Does
driver.back()	Go back to the previous page
driver.forward()	Go forward to the next page
driver.refresh()	Refresh the current page
driver.current_url	Get the current URL
driver.title	Get the title of the page
driver.maximize_window()	Open browser in full-screen mode

💼 Real-World Selenium Project Ideas (Simple → Tough)

1. ✅ Automate a Google Search
   ➤ Open Google, search a keyword, and collect the result titles.
2. ✅ Scrape Product Titles & Prices from a Static Website
   ➤ Visit a basic e-commerce site (like books.toscrape.com), extract product info.
3. ✅ Scrape Dynamic Content (e.g., Smartprix or Flipkart)
   ➤ Handle content that loads with JavaScript using Selenium + BeautifulSoup.
4. ✅ Automate Login and Navigation on a Dashboard
   ➤ Login to a test site (or your own portal), navigate through pages, and extract data.
5. ✅ Fill and Submit Forms or Auto-Apply to Jobs
   ➤ Automatically fill forms (e.g., Google Forms or job applications) with user data.
6. ✅ Scrape stock/crypto data

🧠 Final Thoughts

In this guide, you've learned the complete basics of Selenium — from installing it properly to controlling your browser like a pro. You now know how to open any website, locate elements like buttons or search boxes, and automate interactions just like a real user would. You've also seen how to extract data and even save it in a structured format like CSV using BeautifulSoup and Pandas.

More importantly, we explored some real-world use cases — starting from automating a simple Google search to scraping dynamic websites like Smartprix and even auto-filling online forms. These examples should give you a strong foundation to build powerful scraping and automation projects with confidence.

✨ Selenium is more than just a tool — it's your **personal automation bot** for the browser!

(Selenium is not just for scraping — it's a full-fledged automation beast.)

🔗 **Connect With Me**

💼 **LinkedIn** → linkedin.com/in/ekrajhridoy

🐙 **GitHub** → github.com/ekrajhridoy

📸 **Instagram** → instagram.com/*ekraaaaaj*

📬 **Email** → ekrajhridoy850@gmail.com