# **Exercise: Selenium WebDriver**

## Working with HTML Elements

### **Handling Form Input**

In this exercise, you will **use Selenium to automate the process of filling out a form**. The steps involve **launching a web application**, **navigating to a form**, filling out **various input fields** such as **text boxes**, **radio buttons**, and **date pickers**, and then **submitting the form**. This exercise demonstrates how to locate and interact with different types of HTML elements commonly found in **Register User Scenarios**.

**Note:** We won't explain how to create a test project, install Selenium WebDriver or ChromeDriver, or how to write setup and teardown methods because you are already familiar with these steps.

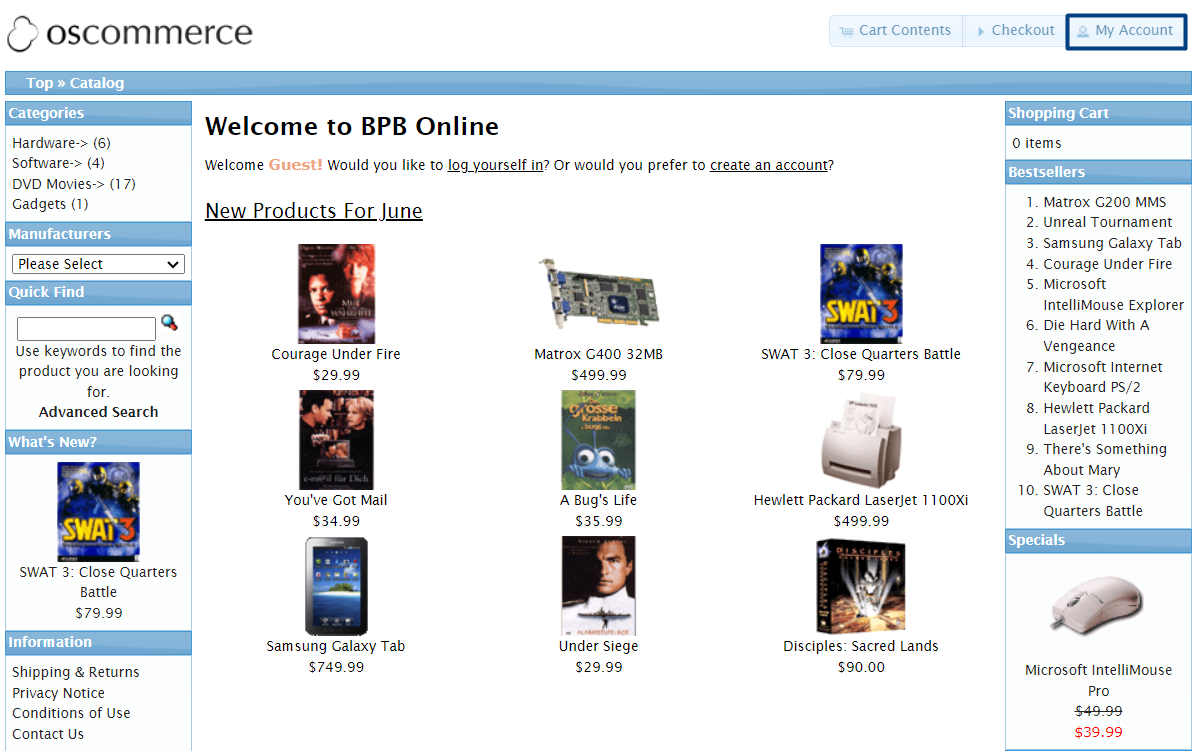
* Open the application using the URL: [**http://practice.bpbonline.com/**](http://practice.bpbonline.com/)**.** It's up to you which method you use to navigate to the URL. Here are **two options:**
* **driver.Url Property** - Navigate to a specific URL. Immediately loads the given URL in the current browser window.



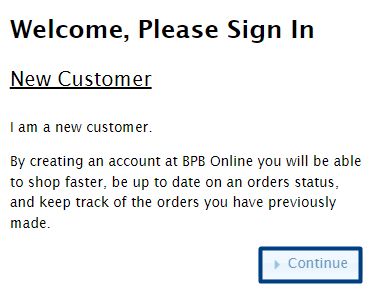
* **driver.Navigate().GoToUrl() Method** A more explicit way to navigate to a URL, part of a broader set of navigation commands. Performs the same action as setting the Url property, but also allows for more complex navigation actions like going back, forward, or refreshing the page.



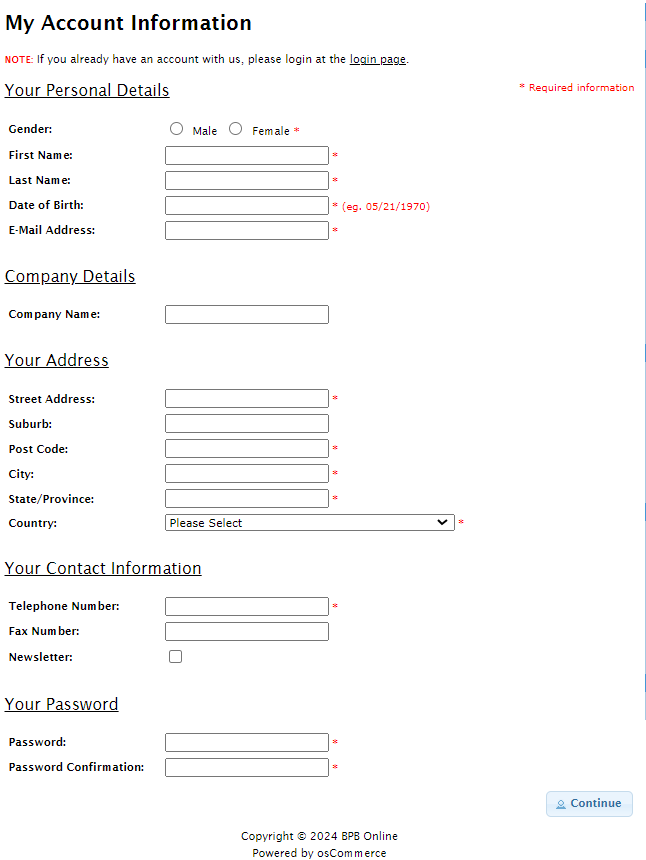
* + In this case, since we are performing a straightforward navigation to the URL, the more appropriate method is to use **driver.Url** for its simplicity and readability. However, you can choose to use **driver.Navigate().GoToUrl()** if you prefer a more explicit approach.
* Click on the **My Account link**



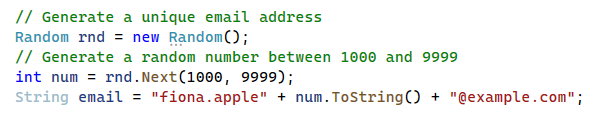
* In the **New Customer section**, click on the **Continue button**.



* This will take you to the **Account Creation page**.



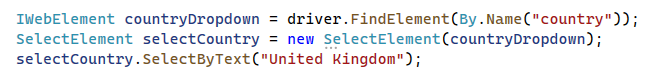
* For the **My Account Information page**, you will need to fill in all the mandatory information for account creation.
* **Select Gender**.
* **Enter First Name**.
* Enter **Last Name.**
* Enter **Date of Birth.**
* Since the registration will work only once per email address**, generate a unique email address.**



* Enter **Email Address.**
* Enter **Company Name.**
* Enter **Street Address.**
* Enter **Suburb.**
* Enter **Postcode.**
* Enter **City.**
* Enter **State.**
* Select **Country from dropdown.**

When dealing with dropdown elements in web forms, Selenium provides a convenient class called **SelectElement** which is part of the **OpenQA.Selenium.Support.UI namespace**. This class contains methods specifically designed for interacting with dropdowns.

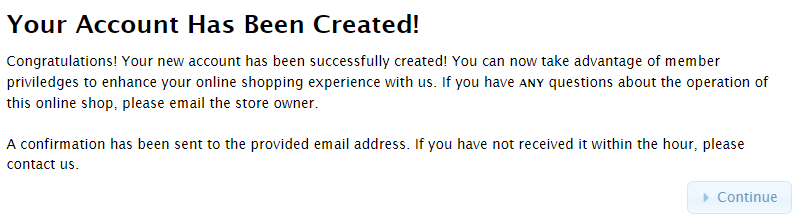
* Install Selenium Support NuGet package.
* Import the necessary namespace.
* Identify the dropdown element using a locator (e.g., by name).
* Create a SelectElement object by passing the identified dropdown element to its constructor.
* Select an option from the dropdown by its visible text.

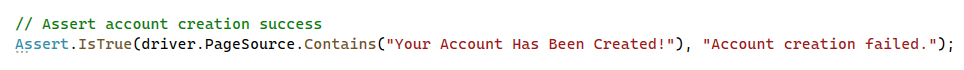


* You can also chain method calls to simplify the selection of an option from a dropdown list. Instead of breaking down the steps into multiple lines, chaining combines them into a single, streamlined statement.

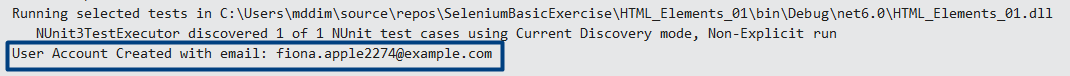


* Enter **Telephone Number**.
* Opt for **Newsletter Subscription**.
* Enter **Password**.
* **Confirm Password**.
* **Submit the form**.
* Assert **Your Account Has Been Created Page**.



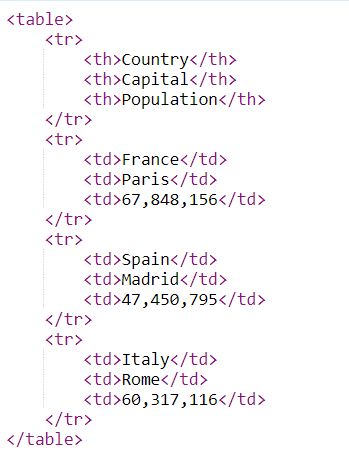


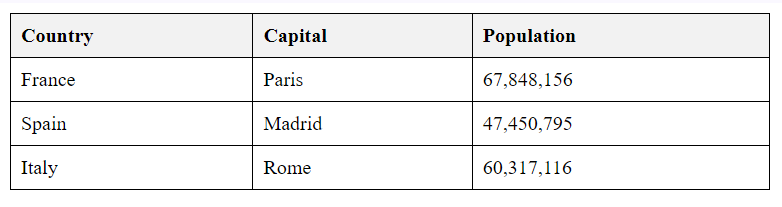
* Click on **Log Off link**.
* Click on **Continue button**.
* **Print success message to the console**.



### **Working with Web Tables**

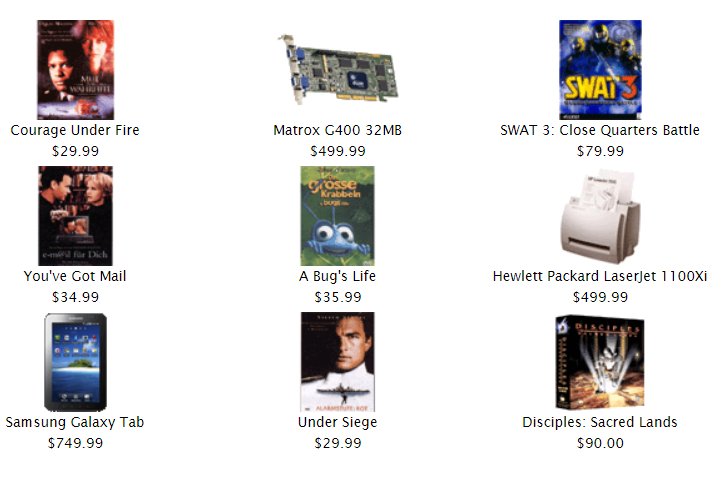
Many web pages need to show data in a clear, organized way. Usually, a web table element is used for this. A web table is an HTML element represented by the <table> tag. It contains other HTML elements, making it a container HTML element. Here is what a **<table> tag** can include:



The table created using the preceding elements will be displayed in the browser as a web table:

The scenario we will use to work with a web table is again from the [**http://practice.bpbonline.com/**](http://practice.bpbonline.com/) application. The home page of this website shows a list of products. In the background, this list is actually a web table.

On the home page products are listed as follows:



We can see **the <table> tag**, which includes a **<tbody>.** Inside, there are three <tr> tags representing table rows. Each row contains three <td> tags representing columns. The <td> tags include two anchor elements for the product image and product page, as well as text displaying the product's price.



By the end of this exercise, you will be able to:

* Traverse a web table and extract information.
* Save the extracted information to a CSV file.
* Open the application using the URL: [**http://practice.bpbonline.com/**](http://practice.bpbonline.com/)**.** Use either driver.Url or driver.Navigate().GoToUrl() to navigate to the URL.
* Identify the web table on the home page. Use XPath to locate the web table on the home page.



* Find all rows in the web table:



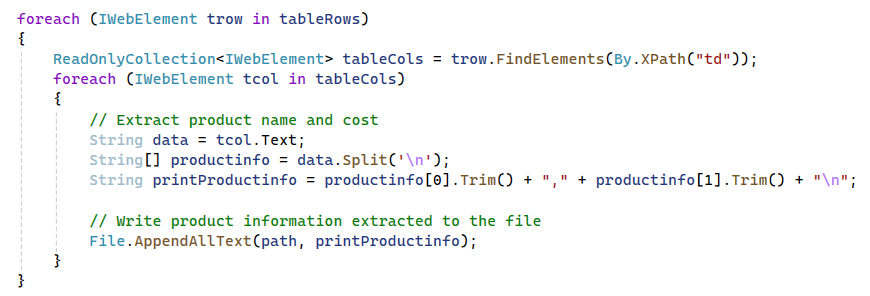
* Create a CSV file to save product information:



* This optional step ensures that if a CSV file with the same name already exists in the specified location, it will be deleted before creating a new one. This prevents appending data to an old file and ensures that each test run generates a fresh CSV file with only the current data.



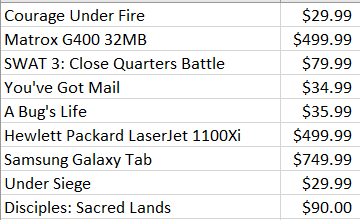
* Traverse through table rows to find the table columns:
  + Loop through each row and then each column within the row.
  + Extract the text from each cell, split the text to separate product name and cost, and format it.
  + Append the formatted text to the CSV file.



* Use assertions to check that the CSV file was created and is not empty.



* Check how the file looks like in the bin 🡪 Debug 🡪 net{version} directory



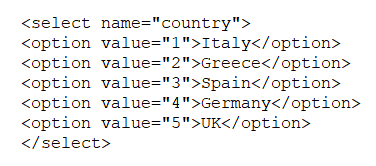
**Note:** Keep in mind that the products might be slightly different, as they change each month.

### **Drop-down Practice**

In this exercise, you will use Selenium to automate the interaction with dropdown elements. A dropdown web element is represented by the <select> tag in HTML and contains <option> tags, each representing a selectable item in the dropdown list.

To work with dropdown elements in Selenium, you use the SelectElement class from the OpenQA.Selenium.Support.UI namespace. This class provides several methods to select options by text, index, or value.

Here is a basic example of a dropdown in HTML:

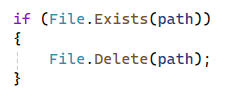


To work with dropdown elements in Selenium, you use the **SelectElement** class from the OpenQA.Selenium.Support.UI namespace. This class provides several methods to select options by text, index, or value.

By the end of this exercise, you will be able to:

* Traverse a dropdown and extract information.
* Save the extracted information to a text file.
* Open the application using the URL: [**http://practice.bpbonline.com/**](http://practice.bpbonline.com/). Use either driver.Url or driver.Navigate().GoToUrl() to navigate to the URL.
* Determine the path to save the text file and delete any existing file with the same name.





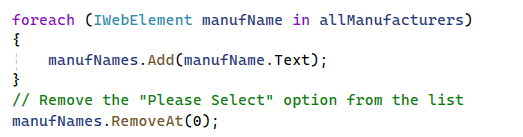
* Identify the dropdown element using name property.



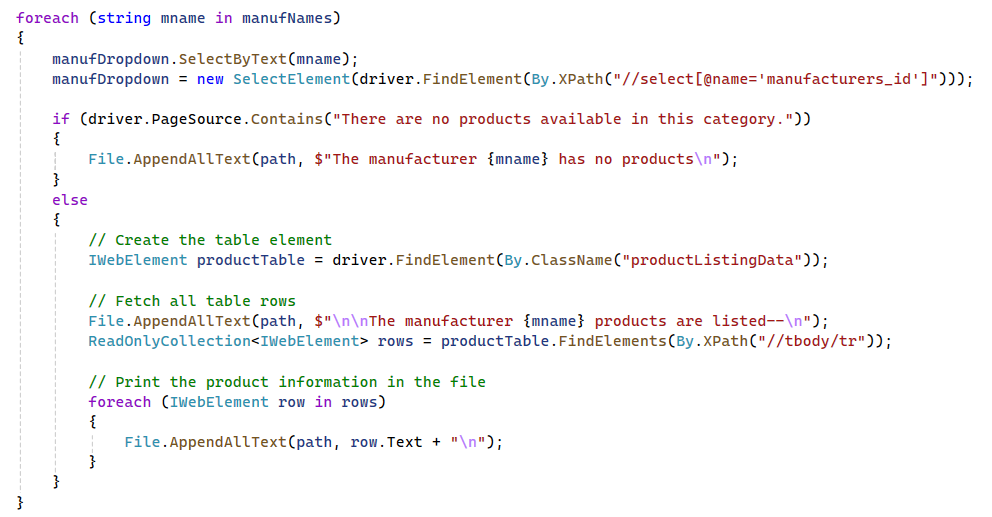
* Retrieve all options from the dropdown and store them in a list.



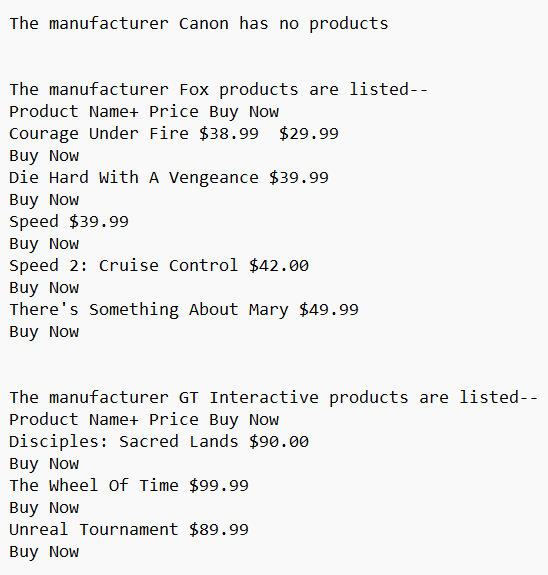
* Create a string list to fill in all manufacturers and remove the "Please Select" option.



* Iterate through the manufacturers to fetch the product information related to each:
  + Loop through each manufacturer and select it from the dropdown.
  + Check if there are no products available for the selected manufacturer.
  + If products are available, fetch all rows from the product table and print the information to the text file.



* Close the browser and end the session.
* Check how the file looks like in the bin 🡪 Debug 🡪 net{version} directory.



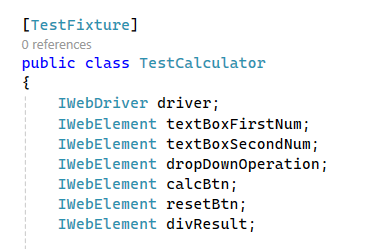
Note: Keep in mind that the products might be slightly different, as they change each month.

## Data-Driven Tests

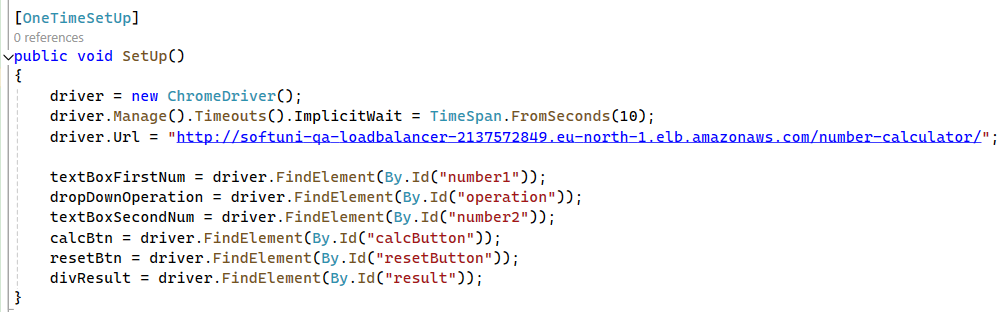
In this exercise, we will write automated Selenium tests for the "Number Calculator" web application. We will implement data-driven testing to make our tests more efficient and maintainable.

By the end of this exercise, you will be able to:

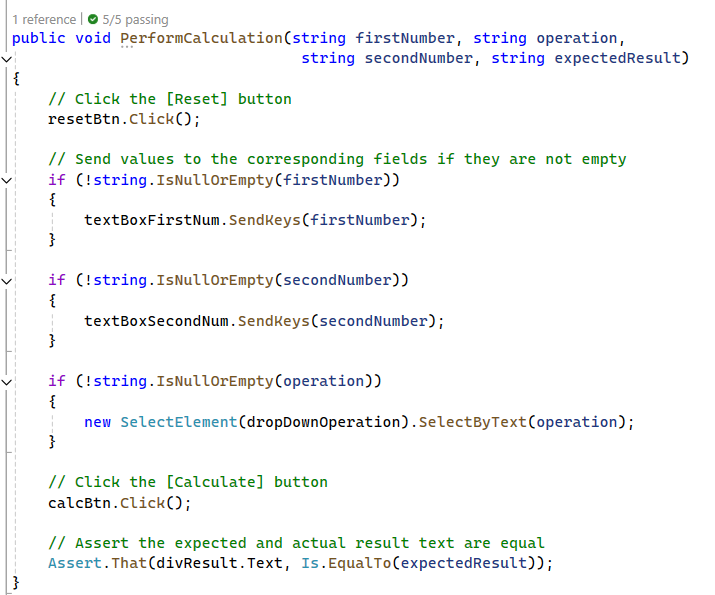
* Implement data-driven testing to reuse test logic for multiple sets of data, enhancing test clarity and efficiency.
* Set up your project.
* Install the necessary Selenium packages via NuGet.
* Create fields for the driver and web elements.



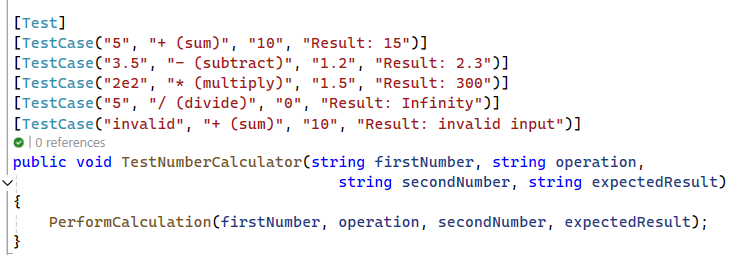
* Initialize the ChromeDriver and navigate to the application URL.
* Locate the necessary web elements on the page.



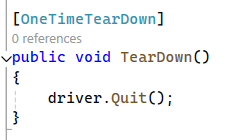
* Define a method that accepts strings for the first number, operator, second number, and expected result.
* Implement the steps to interact with the web elements.



* Use the [TestCase] attribute to write various test cases with different inputs.



* Create the TearDown method.



In data-driven testing the test data is separated from test logic. Note that each test case is executed as a separate test, but the code of the test method is not repeated, it is reused. Each test case runs independently with its own data set, ensuring that the logic is applied consistently across different scenarios.

* Write some more test cases.

At the end, your test cases may look like this 😊

Картина, която съдържа текст

Описанието е генерирано автоматично

Картина, която съдържа текст

Описанието е генерирано автоматично