# Lab Exercises: Selenium Basics

This document defines the homework assignments from the [QA Automation" Course @ Software University](https://softuni.bg/trainings/2550/qa-automation-may-2020).

Please submit as homework a single zip / rar / 7z archive holding the source code or whatever approach is needed to finish your tasks.

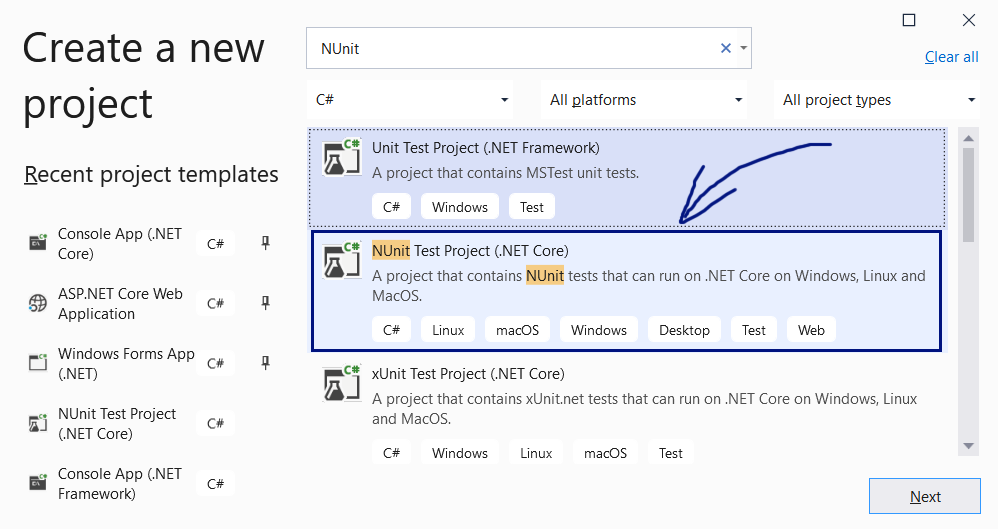
## Search for “QA” in Wikipedia

* Write a **test** that **opens a browser** and navigates to <https://wikipedia.org>.
* Then type “**QA**” in the search box and click **[Enter]**.
* **Assert** that the browser opens the following URL: <https://en.wikipedia.org/wiki/QA>.

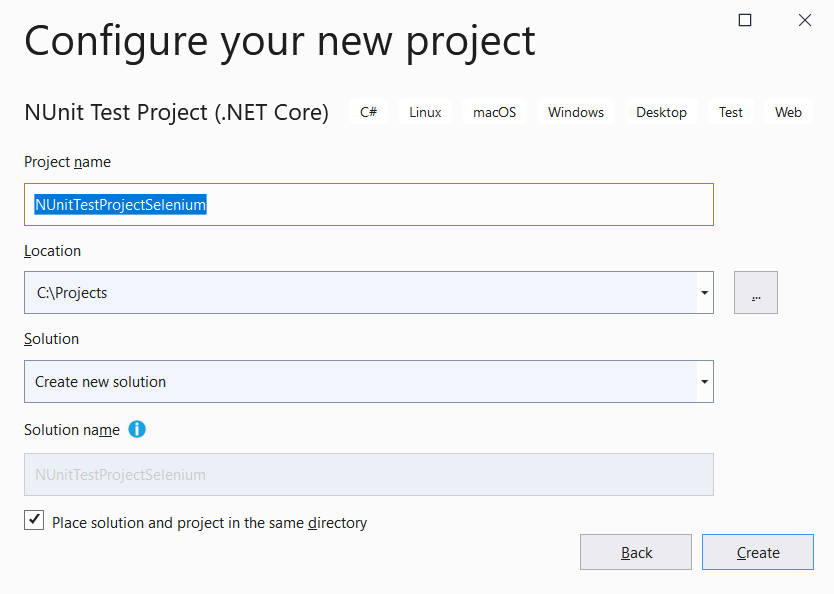
 

### Hints and Guidelines

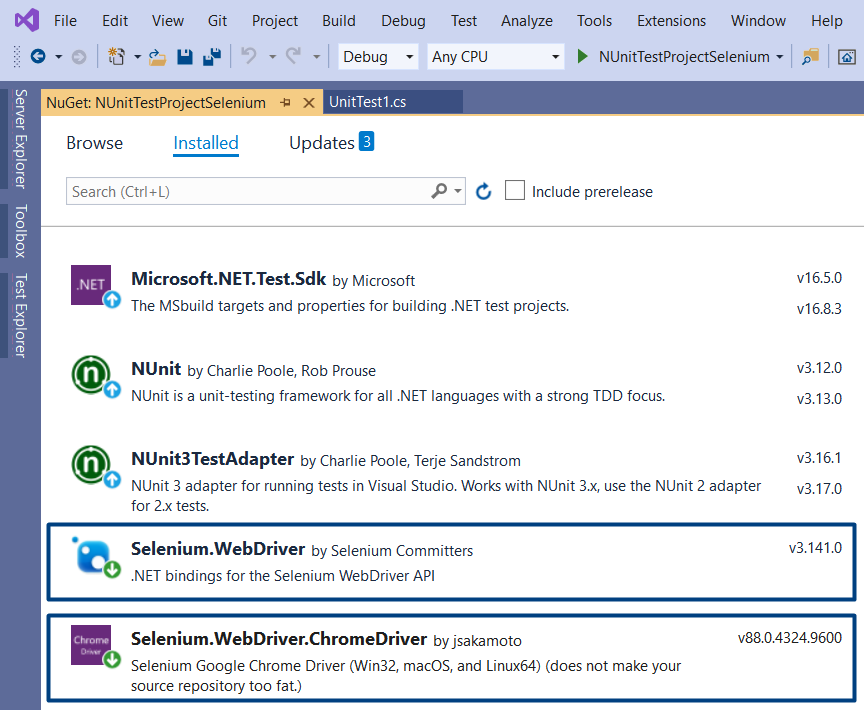
Create a new **C# NUnit Project** in Visual Studio:



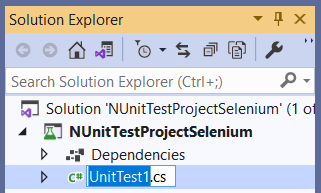
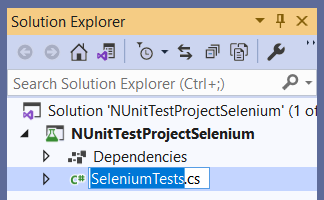
Give a meaningful name for your project. It will hold **Selenium tests**, so its name could be “SeleniumTests” or “NUnitProjectSelenium” or “SeleniumAutomatedTests”:

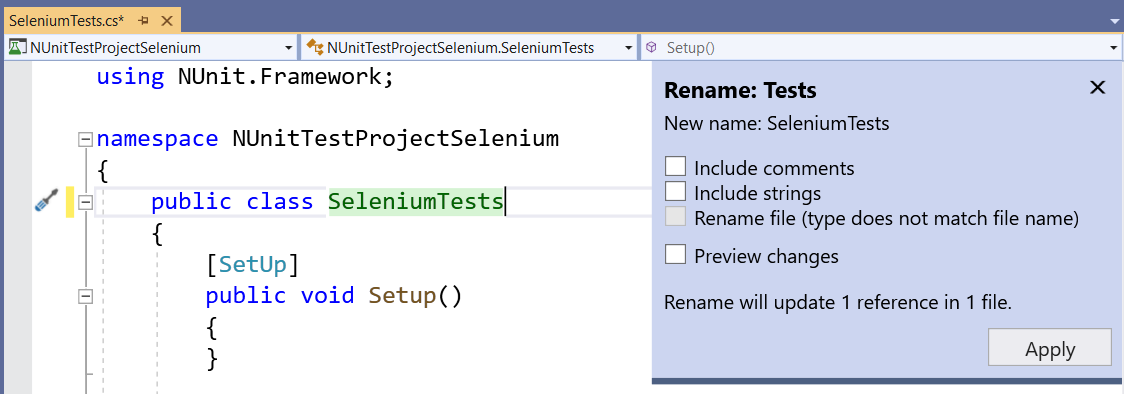


Install the NuGet packages “Selenium.WebDriver” and “Selenium.WebDriver.Chrome”:

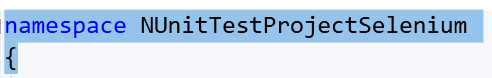


**Rename** the main C# class in your project. Use a meaningful name like “SeleniumTests”:



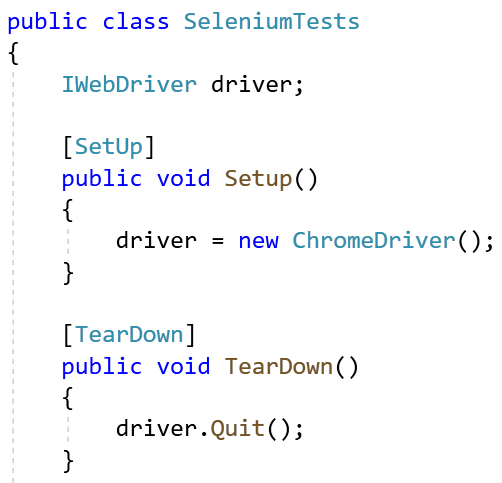
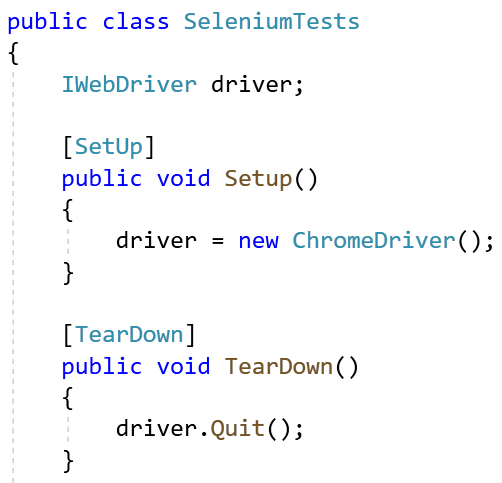
**Remove the namespace**. It is unnecessary in small projects.



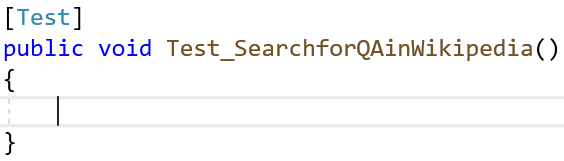
Import the required **namespaces for Selenium**:



Initialize the ChromeDriver in the SetUp() method and release the driver in the TearDown() method:

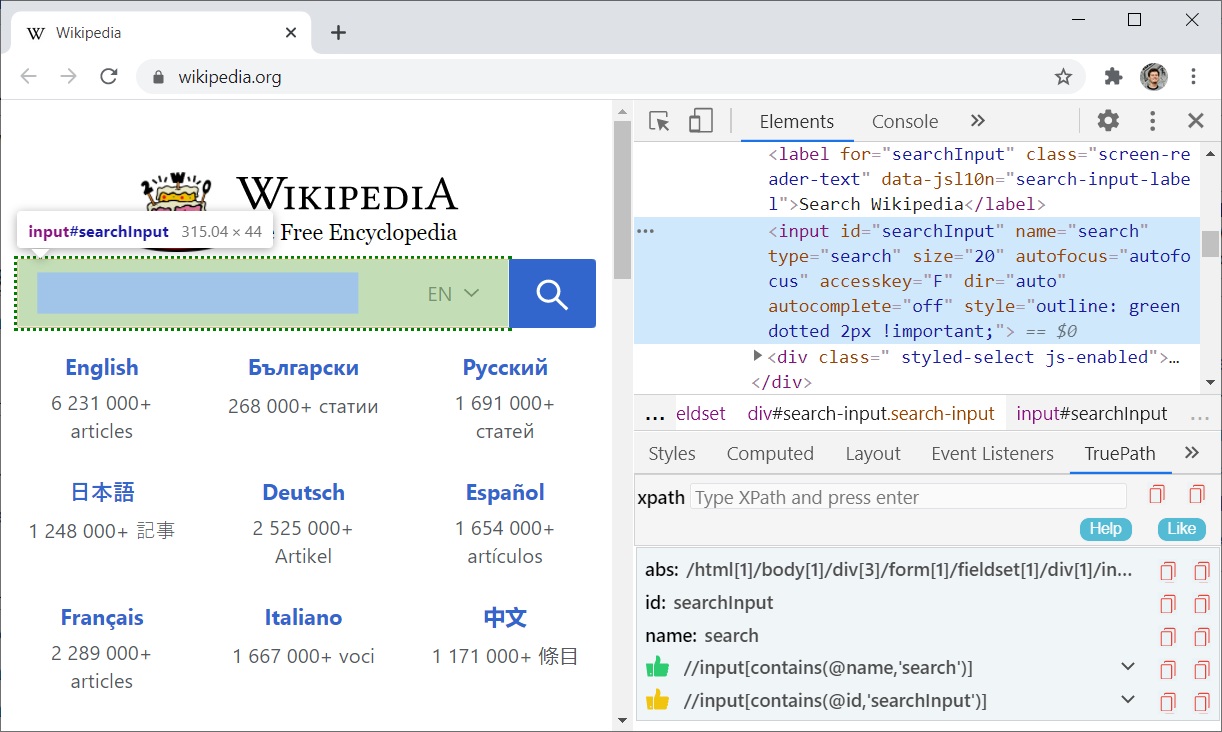
Now, it’s time to start **writing the Selenium test**. Define the test method. Choose a meaningful name:



Navigate to the target Web page: <https://wikipedia.org>:



Find the **locator** for the **search box**. You may use **XPath locator** or **CSS locator** or other way to locate the element. **Inspect the element** in the Web browser’s developer tools:

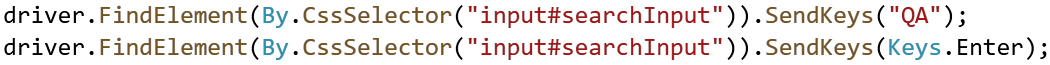


You may also use the [**TruePath** add-on](https://chrome.google.com/webstore/detail/truepath/mgjhkhhbkkldiihlajcnlfchfcmhipmn), to get some automatic suggestions for the element locator. This input text box is easy to locate. It has non-dynamic id="searchInput", name="search" and it is also the only input element of type="search" on the page.

We can **click on the search box** with the following Selenium command:



Then, we can **type “QA”** in the search box and click **[Enter]**:

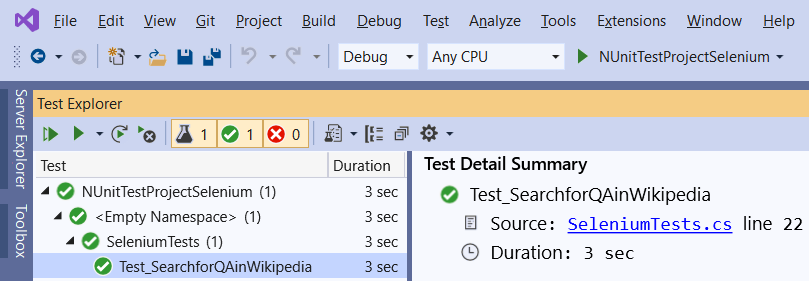


The browser should be **redirected** to the “search results” page: <https://en.wikipedia.org/wiki/QA>.

Finally, we need to **assert** that the search result page for “QA” is **loaded**:



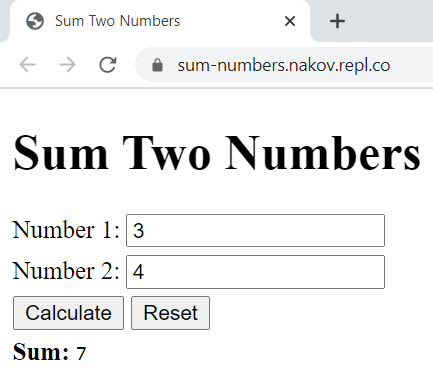
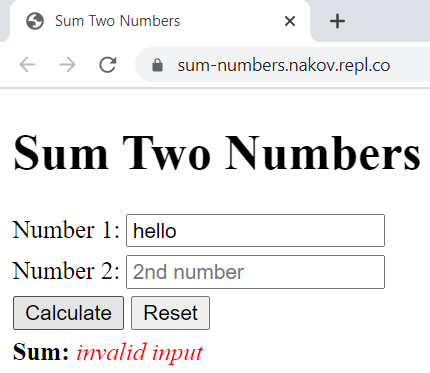
The last step is to **run the test** and make sure that it **passes correctly**, as expected:



## Automated Test for “Summator of Numbers” App

Write **automated Selenium IU tests** for the following app, which sums two numbers:

* Live app URL: <https://sum-numbers.nakov.repl.co>
* Source code: <https://repl.it/@nakov/sum-numbers>

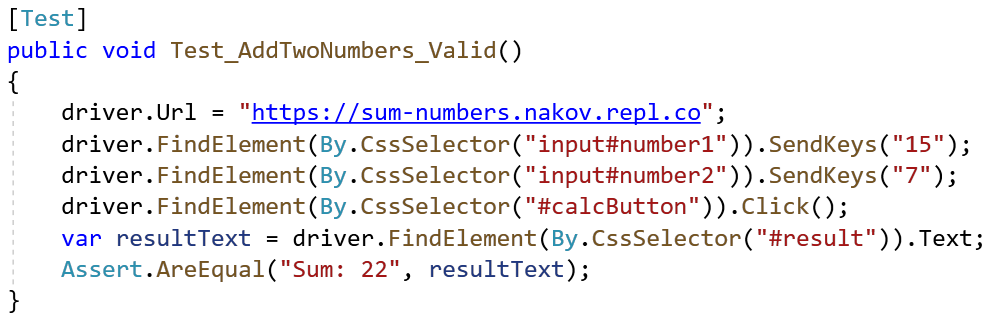
 

### Hints and Guidelines

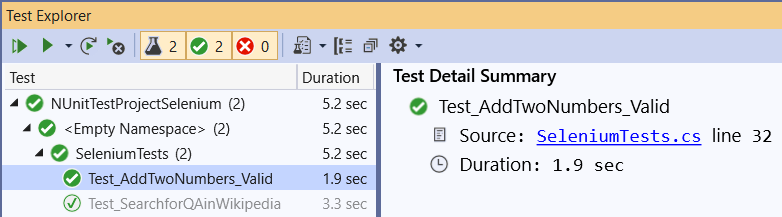
The **first test** should be the straightforward case (sum two valid integers). Use the following steps:

* Open the target web app URL with the Selenium Web driver.
* Find the **first text field** and type some number, e. g. “**15**”.
* Find the **second text field** and type some number, e. g. “**7**”.
* Find the **[Calculate]** button and **click** it.
* Find the result box and take the text from it.
* Assert that the text is as expected, e. g. “**Sum: 22**”.

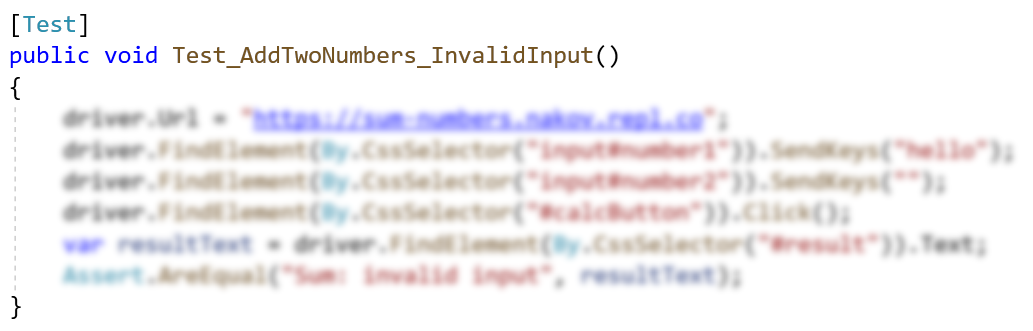
This is how the test source code may look like:



**Run the test** and make sure it passes correctly:



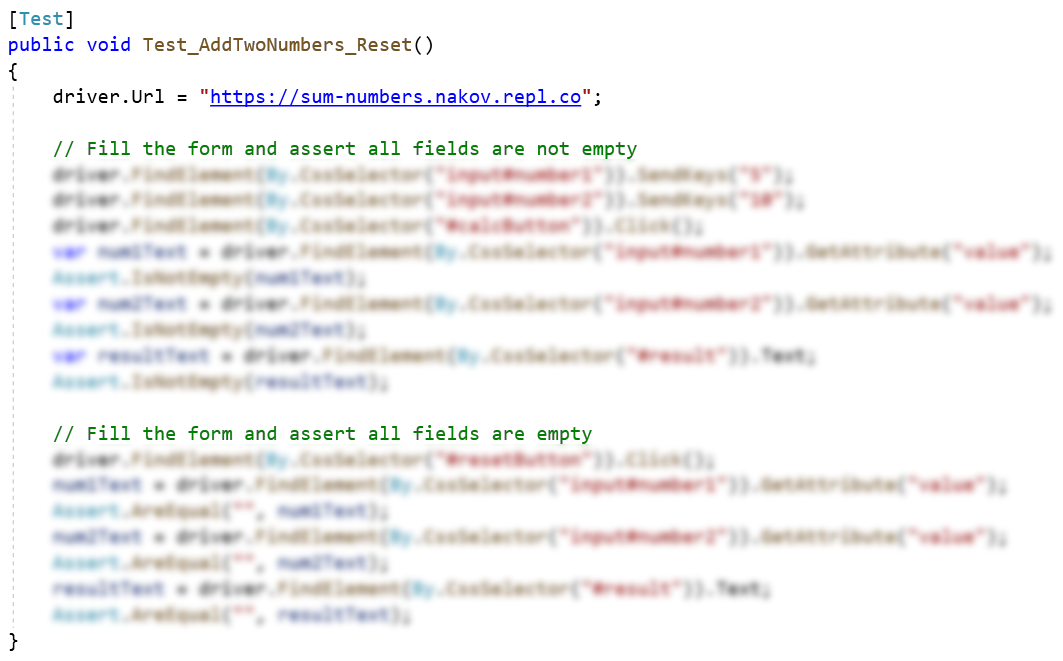
The **second test** should check that when we **sum invalid numbers** (like “**hello**” and “”), the result will be “*invalid input*”. It should be very similar to the previous test:



The **third test** should check whether the **[Reset] button** works correctly. This is how it can be tested:

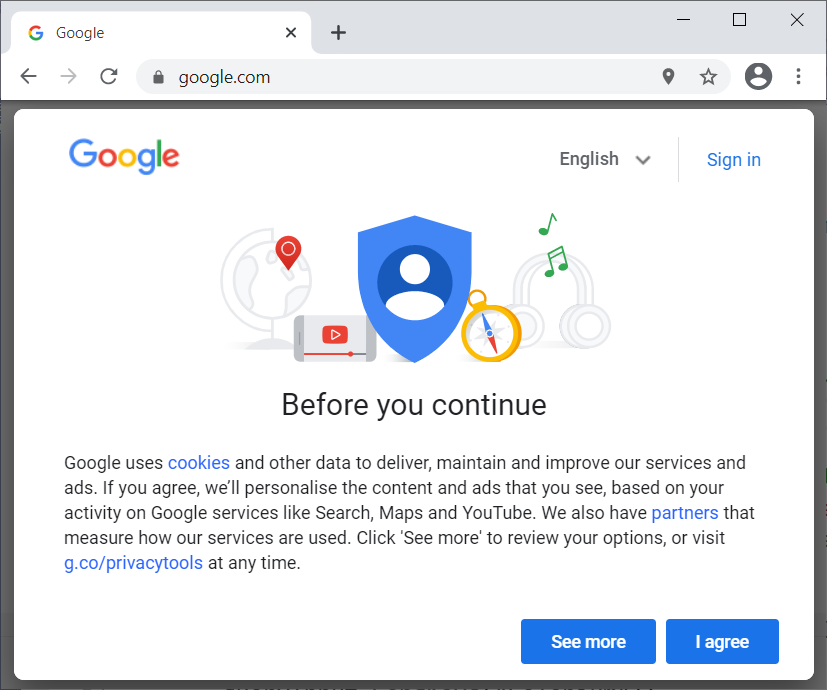
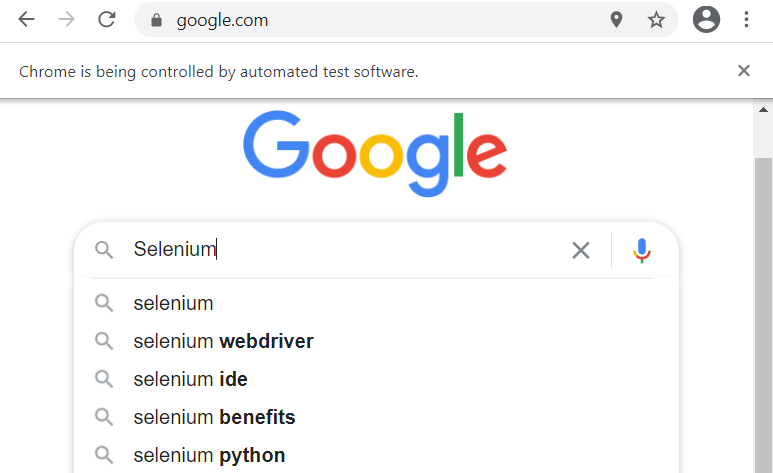
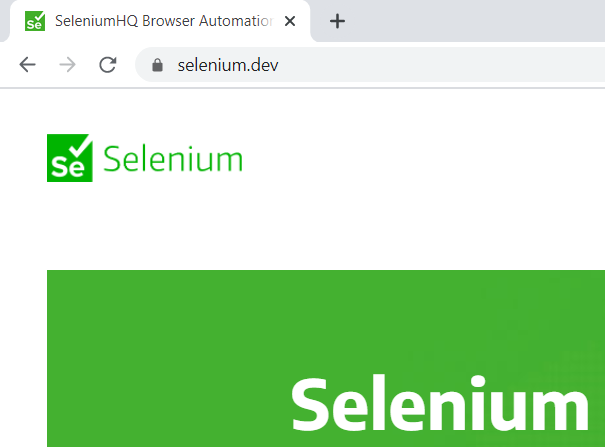
* **Open** the Summator app.
* Fill the form and **click [Calculate]**.
* Assert that all fields are **non-empty**.
* **Click [Reset]** to clear all fields.
* Assert that all fields are **empty**.

This is how the code of the test may look like:



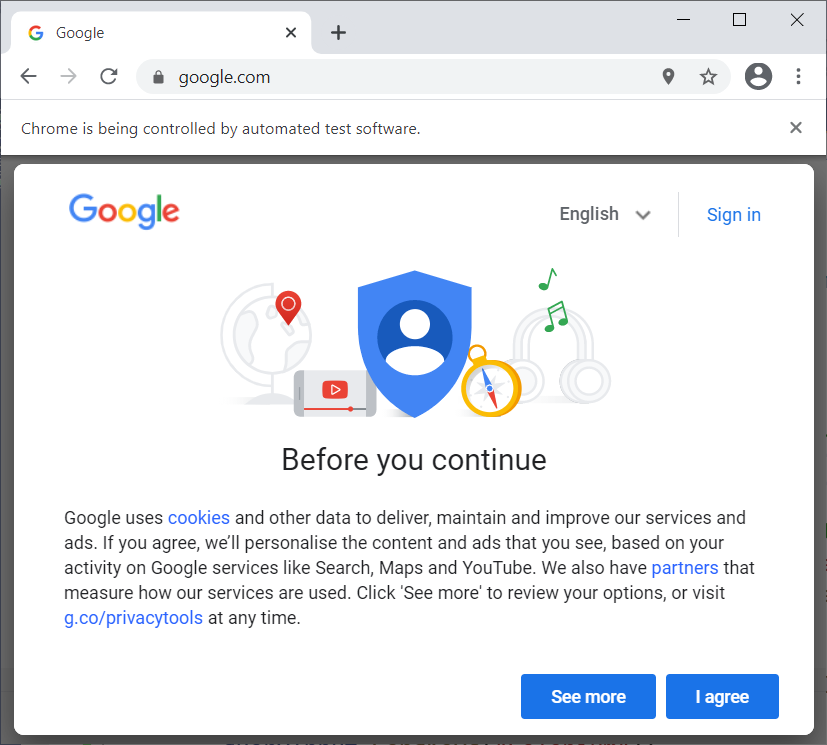
## Automated Google Search for “selenium”

* Write a **test** that **opens a browser** and navigates to <https://google.com>.
* Then **searches** for the keyword "**selenium**".
* Click on the **first result link** and check if the browser will open <https://www.selenium.dev/>.
* **Assert** that the window title is "**Selenium - Web Browser Automation**".

** ** 

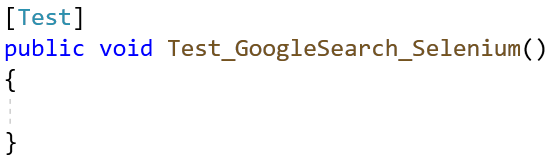
### Hints

Note that Google will show a popup box, holding **<iframe>**, which displays their **terms of service**. You will need to **click [Agree]**. This makes more complicated the automation of this Google search.



### Guidelines

Create a new **unit test** and give it a meaningful name:



Set the **maximum waiting time** for locating elements to 5 seconds:



Open <https://google.com> using the Selenium Web driver:



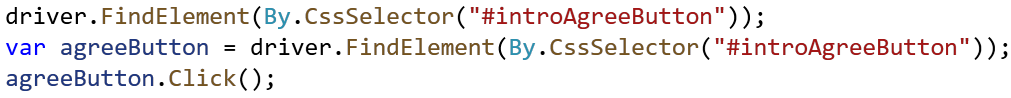
Now the above-mentioned popup box will be shown. We need to click the [Agree] button, but this is not trivial. It needs several steps:

Switch to the first open **<iframe>** element, which holds the popup box, along with the **[Agree]** button.



Otherwise, you will be unable to locate the [Agree] button.

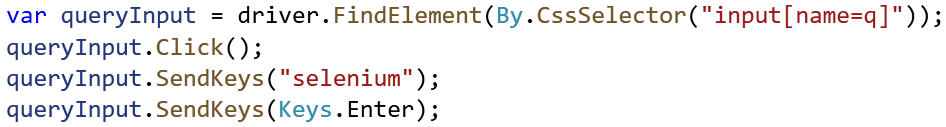
Now wait for up to 5 seconds (the implicit wait) for the **[Agree] button** to appear on the page and **click** it.



Now switch back to the **main browser window** (outside of the <frame> popup):

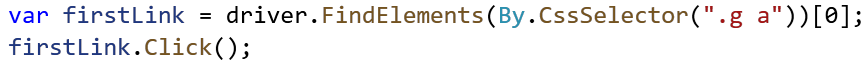


Now type “**selenium**” in the search box and press **[Enter]** to start searching:



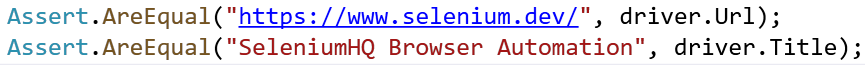
The **search box** is easy to be found. It is a simple text box element: <input name='q' …>.

Now **open the first search result**. Note that the implicit wait of 5 seconds will wait until the Google search server-side operation completes and the search results are shown. This may take a while. This is the code:



The search results are found by a CSS selector “.g a”. This selector returns a list of elements, and we take the first one (at position 0) and click it.

Now the target site “[**https://selenium.dev**](https://selenium.dev)” will be open. It takes time, but we have an explicit wait, that will handle it. Finally, we assert that the loaded page is as expected (we assert the page URL and its title text):



Run the unit test in from the Visual Studio test runner. It should pass successfully:

