**Selenium WebDriver overview**

All right. Now, we’ve gotta cover one more topic before we start coding.

[via GIPHY](https://giphy.com/gifs/kim-novak-tXL4FHPSnVJ0A)

I know, I know, be patient.

We’re going to get to write and run our first automated Selenium test. But first, we’ve gotta lay the foundation. We’ve gotta talk about, what exactly is Selenium.

**Introduction to Selenium WebDriver**

So, in this specific section, what you’re going to learn is a little of the history of Selenium.

Then, you’re going to understand that the Selenium WebDriver protocol and how it works.

Next, I’m going to teach you about the Selenium Script.

And then, finally, we’re going to talk about all the common Selenium commands that you can use.

Now, this isn’t meant to be a very comprehensive course in Selenium, right?

This is a getting started, introduction to Selenium WebDriver course. And so, the goal is to get you up and running as soon as possible.

People actually make careers, such as myself, doing test automation and using Selenium every single day.

There are actually courses out there that are hours and hours that cover this topic. And that will not be the case.

But I promise I’ll give you enough information so that you can proceed comfortably and then I’ll leave you with a bunch of resources to further expand your knowledge.

**History of Selenium**

Selenium was actually started back in 2004 by Jason Huggins at ThoughtWorks, such a long time ago, right?

The idea behind Selenium was that we needed a way to simulate user actions in our browser so that we can test.

We had unit tests, we had API tests … but we had no way of simulating browser rendering and browser interactions with our web applications.

So, Selenium was born.

Selenium has actually gone through many versions already.

Version one was the original one that was created. That was basically interacting with the browser, with JavaScript.

It had lots of problems because JavaScript is asynchronous. It’s really hard to sync the browser with random JavaScript commands.

In version two, Selenium introduced more object-oriented programming concepts. You could create stuff like pay jobs, you could have elements that contain a bunch of different locators.

That made the use of Selenium much easier. Also, they introduced JSON Wire Protocol. Instead of talking to through JavaScript, it was using HTTP requests, and I’ll cover that in a little bit.

Version three is the current version as of this date of Selenium.

This version basically removed Selenium RC, which is no longer really used, as we used to call Selenium Remote Control.

There are some API changes that improved the functionality of Selenium. This is the version that we’re going to be using this course.

**Selenium and the WebDriver Protocol**

In hopefully a not too distant future, Selenium 4 is coming.

Selenium 4 is going to bring with it the W3C protocol.

What the means is that Selenium is going to be a W3C standard and every browser now should support the Selenium API out of the box.

So, the problem with Selenium currently, up to version three, is that the Selenium community works on implementing their API on browsers.

However, they don’t have full internal knowledge of how the browsers operate. There are always a lot of problems that arise from that.

The beauty of integrating Selenium with the W3C protocol is that now the browser vendor such as Safari, Chrome, Firefox, will need to implement the Selenium API.

For example, if you want to do a browser click or we want to refresh your browser. Or, if we want to wait for a page to be loaded, the browser vendor will be responsible for making that implementation.

It will be a W3C recommendation, so it’ll kind of be a standard that every browser should follow.

That’s beautiful because now our Selenium code is going to be much faster.

[via GIPHY](https://giphy.com/gifs/latenightseth-lol-seth-meyers-lnsm-Pid08Kw1K8Qae9dAX2)

It’s going to be much more stable and it will allow browser automation to evolve.

**What is Selenium Webdriver?**

Selenium project actually has many different tools.

For example Selenium RC, which I already mentioned, Selenium IDE which actually used to exist but then it died, it was a record and replay tool.

We all know how record and replay tools are not the best.

Now, Selenium IDE is actually being brought back by Applitools and Dave Hefner.

They’re trying to make Selenium IDE awesome and more user-friendly.

However, the ultimate Selenium tool is Selenium WebDriver!

That’s exactly what we’re going to be using in this course and that’s exactly what’s used by most the world in order to be able to interact with the browsers.

All Selenium WebDriver is, is a language-specific API that allows us to use a programming language to communicate to a Selenium server that will talk to the browser and allow this to go back and forth.

**What are the Selenium Webdriver Language Bindings?**

Selenium WebDriver actually supports multiple language bindings, such as Java C#, Ruby, Python, and JavaScript.

Those are the ones officially supported.

There are a bunch of other ones that exist out on the web, but I don’t recommend you to use them.

The beauty of all these languages being supported is that if you’re comfortable coding in any of them, you simply have to learn the Selenium WebDriver API.

This is not so complicated, and once you learn it, you can start writing automated functional tests.

Really awesome!

[via GIPHY](https://giphy.com/gifs/masonramsey-mason-ramsey-cdNSp4L5vCU7aQrYnV)

The way the Selenium WebDriver protocol actually works is…

1. You start with your Selenium code.
2. The Selenium code makes an HTTP request to the Selenium server, which is the Selenium WebDriver.
3. That will then send a JSON WebDriver request to the browser.
4. The browser will respond back with a WebDriver response.
5. Then that will all go all the way back to the code, and they talk back and forth.

It’s basically just a little HTTP server that allows communication between our code written in one of those five language bindings and the browser.

**The Selenium Project**

The Selenium Project is an opensource project.

It’s supported by members that don’t get paid. They’re always looking for support.

If you want to contribute, there are multiple different ways to contribute. You can get involved [clicking here](https://www.seleniumhq.org/about/getting-involved.jsp).

Also, if you even want to help, you can go to [the Slack channel](https://seleniumhq.slack.com/join/shared_invite/enQtNTE4MTc4MzYwMjkxLWEyNTZkMDgzNWIwZmY1ZTlmMzg4ZjM1YzZkNGUwZGFlMWE2OTYxMDYxODA1ZWJlMzZjYjc3MmE3ODA1OGZmZTk) or the [IRC channel](https://webchat.freenode.net/#selenium).

In those channels, you can come and ask questions and the maintainers of Selenium are there and can help you to answer them.

**What are the Seven Actions Of A Selenium Script?**

A Selenium Script really boils down to seven actions.

The very first action is you need to instantiate a WebDriver.

This basically allows you to interact with some kind of browser that you want.

The next step is, you have to open up a webpage.

After you open up a webpage you want to identify some elements that you want to interact with.

Then, you need to ensure if that element and the page are in the state that you want them to be in, before interacting.

Once you figure out it’s in the right state, then you perform an action on that element, then you get some kind of a result.

Here is where you perform your assertion using that assert class from MS Test.

Finally, you clean up and close the browser

[Find common Selenium errors and their fixes here…](https://ultimateqa.com/common-selenium-webdriver-errors-fix/)

**1-Start the session**

So, the very first action is creating an object that allows you to interface with one of the browsers.

There are multiple different drivers. You can have a Chrome driver, a Safari driver, an Edge driver … each of those allow you to communicate with the appropriate browser.

The very first step is you instantiate that object that will allow you to talk to that specific browser.

**2-Navigate to a page**

After you instantiated that object, the second step is to navigate to some specific URL.

**3-Locate the Element**

The third step is locating an element that you want to interact with.

In this case, we are using the Selenium by class and we’re locating an element using an ID.

**4-Ensure the Browser is in the Correct State**

After we’ve located our element, we actually want to make sure that the element is in a correct state before interacting with it.

In this case, we’re using the Selenium WebDriver wait, to wait until a specific condition.

In this example, the expected condition that we’re waiting for is that the element is present on the page, and once we’ve figured out that it is, at that point we can proceed with the rest of our actions. Interact with that element.

**5-Perform Action**

In this case. we can interact with that element, now that we know it’s present.

We can perform an action such as a click.

**6-Obtain and Record Result**

After we’ve interacted with some elements, at the end of the day, this is an automated test that’s meant to represent some kind of end-user scenario.

We want to assert that some condition is true or false.

So, we can retrieve some results whether it’s checking that an element is present, or checking that some text exists on some page. Or checking that some page was loaded.

At some point, we obtain and record that result.

**7-Terminate Session**

Finally, we just quit the driver session.

That will close the browser, shut down Selenium WebDriver, and clean everything up.

I know I flew through all of that.

[via GIPHY](https://giphy.com/gifs/Airbus-aircraft-airbus-a220-ekvTq3x8Uk9Ug1eJzv)

**Locating Elements to Test**

Let’s dive into a little bit more detail.

**Types os Locators**

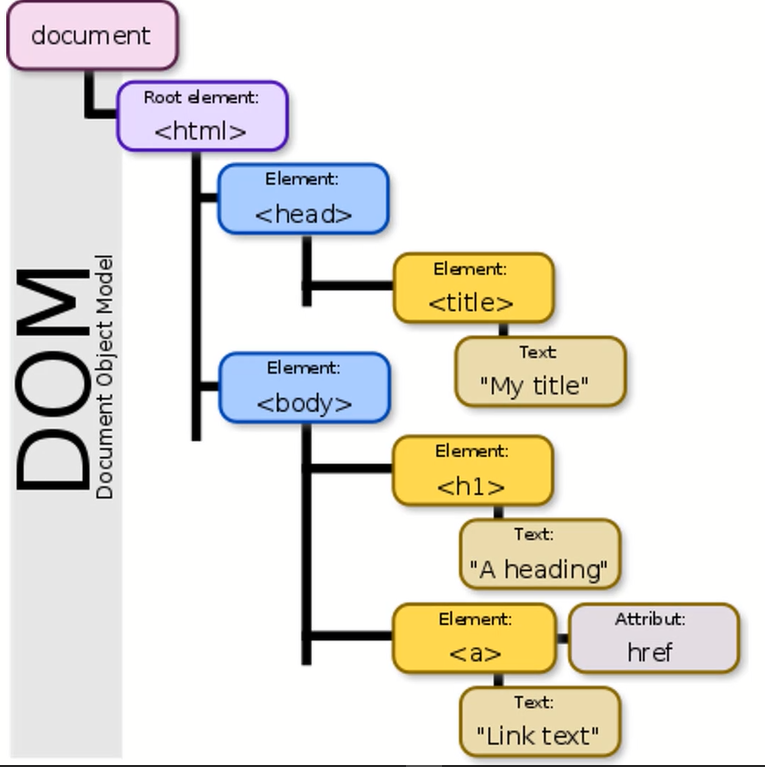
There are multiple types of locators that Selenium provides us.

To find an element on a page, you can use:

* ID
* Name
* CSS
* Class name
* Tag name
* Link text
* Partial link text
* XPath

**The DOM**

The Document Object Model (DOM) is basically used to find an element on a page.

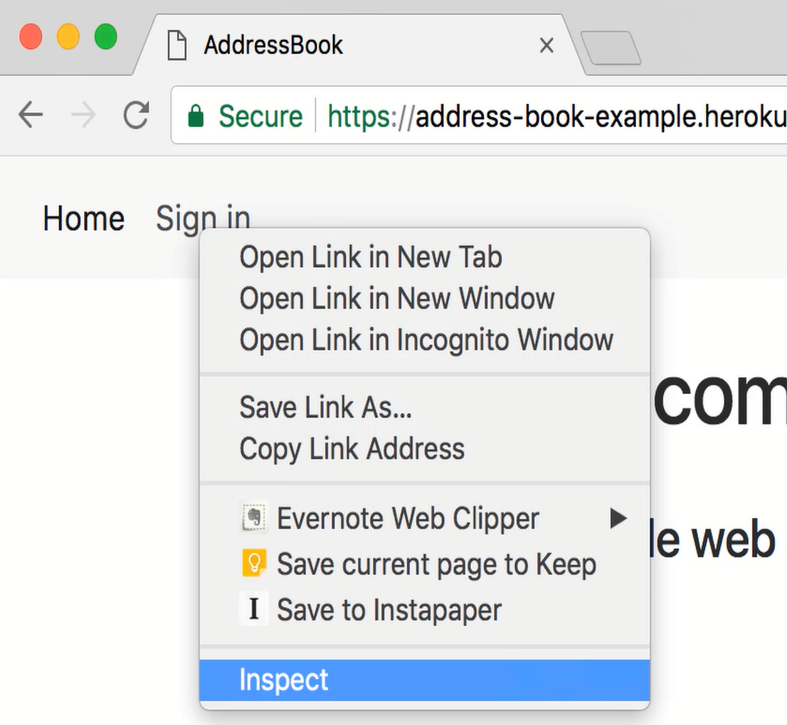


Based on the DOM, you will find an element in some location and you will want to interact with that specific element.

**How to use Developer Tools to locate an element?**

Normally, you find an element using Developer Tools.

All you have to do is right-click on your browser, open Developer Tools and find that element that you want to interact with.



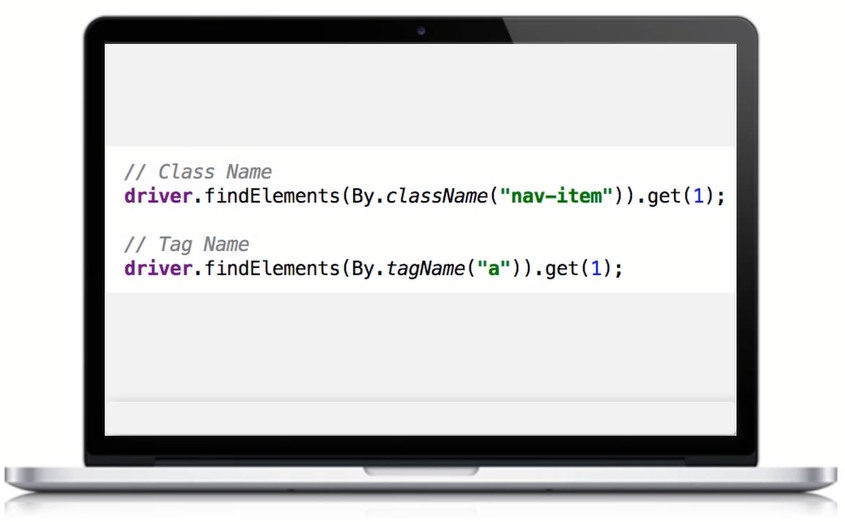
You find a unique element. You want to make sure that the element is the only element on a page that you want to interact with.

**How to locate Element with the findElement Method?**

Here are some examples in Java of how we can locate elements.



You can see that there are IDs, CSS, partial link text, and so on.



What you see here is we’re using the findElements method, instead of the findElement which was the one from the original page to identify multiple elements.

This will allow us to get back a collection of elements.

**How to interact with Selenium Elements?**

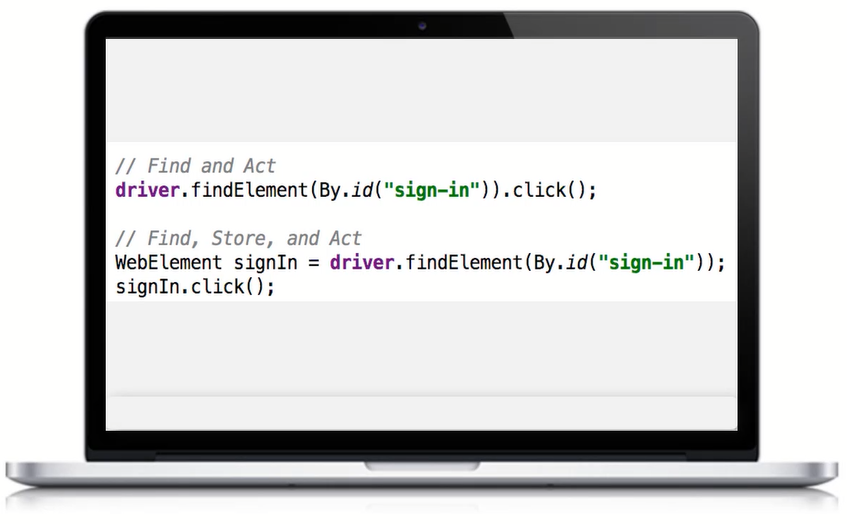
Once we’ve found an element using the findElement command or findElements, you want to start interacting with elements.

There are some basic actions. There are actually many more than this, but I’m just going to show you a couple.



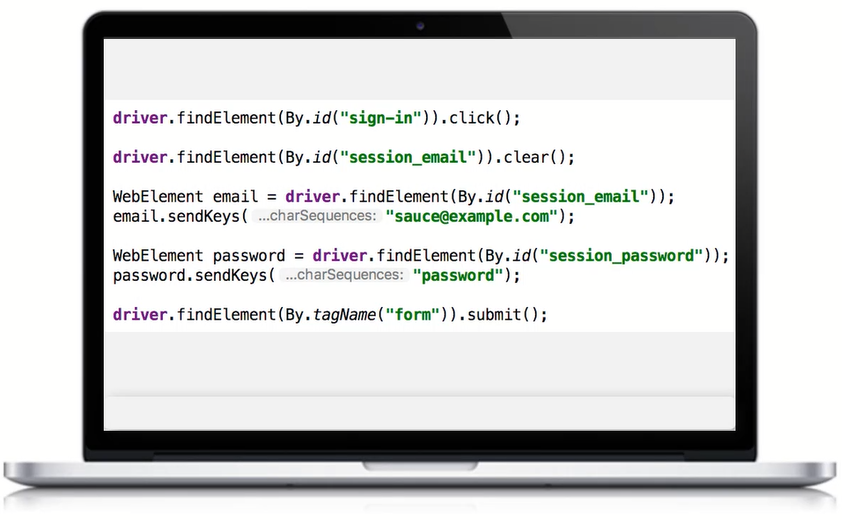
So, you can basically click on an element, you can sendKeys, or you can do something like a submit.

For example, here, you can see we are finding an element and then we are clicking on it.



There are two ways to do that, you can find and element and store it in a variable, or you can find an element and interact with it directly.

Here, you can see we’re taking actions on several elements such as clicking, clearing these elements, and sending keys.



Sending keys will type a string into some kind of a field.

Also, in the last line of code, we’re submitting a form.