Automatic Operation of Browser using Selenium and Python

```
driver = webdriver.Firefox()

①driver.get("http://localhost/wordpress/wp-login.php")

②driver.find_element_by_id("user_login").send_keys("admin")

③driver.find_element_by_id("user_pass").send_keys("admin123")
```

①Open page

②Enter user name

③Enter password

SMTEBOOKS

kenpapa

Automatic operation is performed on the following sites using Selenium WebDriver

1. Demo site (Button, Link, Text box etc)
2. EC site (WordPress + WooCommerce)

These sites are built on virtual machine on PC so we can practice Selenium as much as we want

Finally we execute automated testing with test framework

Introduction

As the size of program grows it becomes more difficult to understand the code. Therefore, it is also more likely to accidentally embed bugs when changing code.

Ideally, it is desirable to test all existing functions each time we change source code, but it is not easy to test all existing functions frequently as testing takes time and cost. Therefore, it is necessary to consider automated testing.

There are many advantages to test automation. Most are related to the repeatability of the tests and the speed at which the tests can be executed. There are a number of commercial and open source tools available for assisting with the development of test automation.

One of test automation tools is Selenium. It is widely used around the world as a tool for automated testing for web application. In this book we use Selenium WebDriver to perform automatic operation of browser. As for browser, we use both Mozilla Firefox and Google Chrome.

Selenium is a very useful tool but it is not an easy tool which lets us use it as soon as we learn it. Practice is necessary but it is not desirable to use real website running in the internet for practice.

Therefore, in this book we prepare two sites, demo site and WordPress site for Selenium practice. Those sites are built on virtual machine on Windows PC and we perform automatic operation using Selenium WebDriver for those sites. Those are in our own environment so we can practice Selenium as much as we want.

SMTEBOOKS

Selenium official website says "What you do with that power is entirely up to you.".

Excerpt from Selenium official website

> What is Selenium?
> Selenium automates browsers. That's it! **What you do with that power is entirely up to you.** Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) be automated as well.

I hope you enjoy Selenium life.

2018.4 kenpapa

Who this book is for

This book is for those who are experienced in programming Python and are interested in automated testing of web application.

Since Linux (Ubuntu) is used as a working environment, if you don't have Linux development experience, it would be better for you to read this book after understanding Basic Linux Commands.

Basic Linux Commands is described in detail in the following site, for example.

Basic Linux Commands for Beginners
https://maker.pro/education/basic-linux-commands-for-beginners

SMTEBOOKS

Structure of this book

In Chapter 1, we will learn the knowledge necessary for automatic operation of browser using Selenium WebDriver. In particular, it is important to understand the methods for automatic operation and the way of identifying the elements to be operated.

In Chapter 2, we will prepare the environment for using Selenium WebDriver to perform automatic operation of browser. In this book we operate Firefox and Chrome browsers, so we will prepare the environment of both browsers. In addition, in order to understand the internal mechanism of Selenium WebDriver, we will communicate with browser driver directly using JsonWireProtocol and try to operate browser.

In Chapter 3, we will perform automatic operation of demo site as a basic practice of Selenium WebDriver. The demo site consists of "Button, Link, Text box, Radio Button, Checkbox, Dropdown, File Upload, Text, Text in iframe". The purpose of this chapter is to become accustomed to using Selenium WebDriver.

In Chapter 4, we will perform automatic operation of WordPress site as an advanced practice of Selenium WebDriver. First of all, we will install WordPress and WooCommerce plugin to prepare a site to be automatically operated, then we will perform automatic operation for "posting blog article", "registering product on EC site" and "purchasing product at EC site" using Selenium WebDriver. It is important to understand the difference of behavior between Firefox and Chrome.

In Chapter 5, we will execute automated testing of web application using Selenium WebDriver and Test Framework. As an example of automated testing, we will execute two tests, Google Search Test and WordPress Login Test.

In Chapter 6, I will mention some notes on the use of Selenium WebDriver based on my experience of Selenium.

Execution environment

The execution environment in this book is as follows.

Ubuntu is running on the virtual machine on Windows machine. WordPress is built on Ubuntu and we use Selenium WebDriver to operate WordPress automatically.

■Version of softwares used in this book
Ubuntu 16.04.4 LTS
(Installation using ubuntu-16.04.4-desktop-amd64.iso)

Mozilla Firefox 59.0.1 (Ubuntu update is necessary)
Mozilla GeckoDriver 0.20.0

Google Chrome 65.0.3325.181
Google Chrome Driver 2.36

WordPress 4.9.4
WooCommerce 3.3.4

Python 2.7.12 (preinstalled on Ubuntu)
Selenium Client 3.11.0

※For details on how to install VirtualBox and Ubuntu, please refer to Appendix.

Table of contents

Introduction
Who this book is for
Structure of this book
Execution environment
Chapter 1 Knowledge necessary for Selenium WebDriver
- 1-1 Selenium Overview
  - 1-1-1 What is Selenium IDE ?
  - Note: Selenium IDE alternative tool
  - 1-1-2 What is Selenium WebDriver ?
  - 1-1-3 What is Selenium Grid ?
- 1-2 Automatic operation method
  - Note: Selenium Client API document
- 1-3 How to investigate elements
  - 1-3-1 How to investigate elements on Mozilla Firefox
  - 1-3-2 How to investigate elements on Google Chrome
- 1-4 How to identify operation target element
  - 1-4-1 Locators in Selenium WebDriver
  - 1-4-2 Identify element by id attribute/name attribute/link text
  - 1-4-3 Identify element by xpath
  - 1-4-4 How to check xpath

Chapter 2 Prepare automatic operation environment
- 2-1 Prepare browser and driver
  - 2-1-1 Prepare Mozilla Firefox
  - 2-1-2 Prepare Google Chrome
- 2-2 Browser operation using JsonWireProtocol
  - 2-2-1 Operate Mozilla Firefox using JsonWireProtocol
  - 2-2-2 Operate Google Chrome using JsonWireProtocol
- 2-3 Prepare Selenium Client
- 2-4 Automatic operation with simple code

Chapter 3 Basic practice of automatic operation
- 3-1 Prepare demo site
- 3-2 Create automatic operation code for operation target element
- 3-3 Complete and execute automatic operation code

Chapter 4 Advanced practice of automatic operation
- 4-1 Prepare WordPress site
  - 4-1-1 What is WordPress ?

- 4-1-2 Install WordPress
- 4-1-3 Install WooCommerce plugin
- 4-2 Post blog article
  - 4-2-1 Flow of posting blog article
  - 4-2-2 Create automatic operation code for operation target element
  - 4-2-3 Complete and execute automatic operation code
- 4-3 Register product on EC site
  - 4-3-1 Flow of product registration
  - 4-3-2 Create automatic operation code for operation target element
  - 4-3-3 Complete and execute automatic operation code
- 4-4 Purchase product at EC site
  - 4-4-1 Flow of product purchase
  - 4-4-2 Create automatic operation code for operation target element
  - 4-4-3 Complete and execute automatic operation code

Chapter 5 Automated testing with Test Framework
- 5-1 What is unittest ?
- 5-2 Google Search Test
- 5-3 WordPress Login Test

Chapter 6 Some notes on the use of Selenium WebDriver
- 6-1 Changes of selenium related softwares
- 6-2 Behavior difference between browsers
  - 6-2-1 Difference in default window size
  - 6-2-2 Difference in scroll processing
  - 6-2-3 Processing timing difference
- 6-3 Implementation differences between programming languages
  - 6-3-1 ExpectedCondition
  - 6-3-2 Node.js is asynchronous
  - 6-3-3 How to take a screenshot

Appendix
- Appendix A: About VirtualBox
- Appendix B: About Ubuntu

Chapter 1 Knowledge necessary for Selenium WebDriver

In this chapter, we learn the knowledge necessary for automatic operation of browser using Selenium WebDriver.

1-1 Selenium Overview

Selenium is an open source automated testing suite for web applications across different browsers and platforms. It is a set of different software tools each with a different approach to supporting test automation.

Selenium consists of three tools as below.

a) Selenium IDE (Integrated Development Environment) b) Selenium WebDriver

c) Selenium Grid

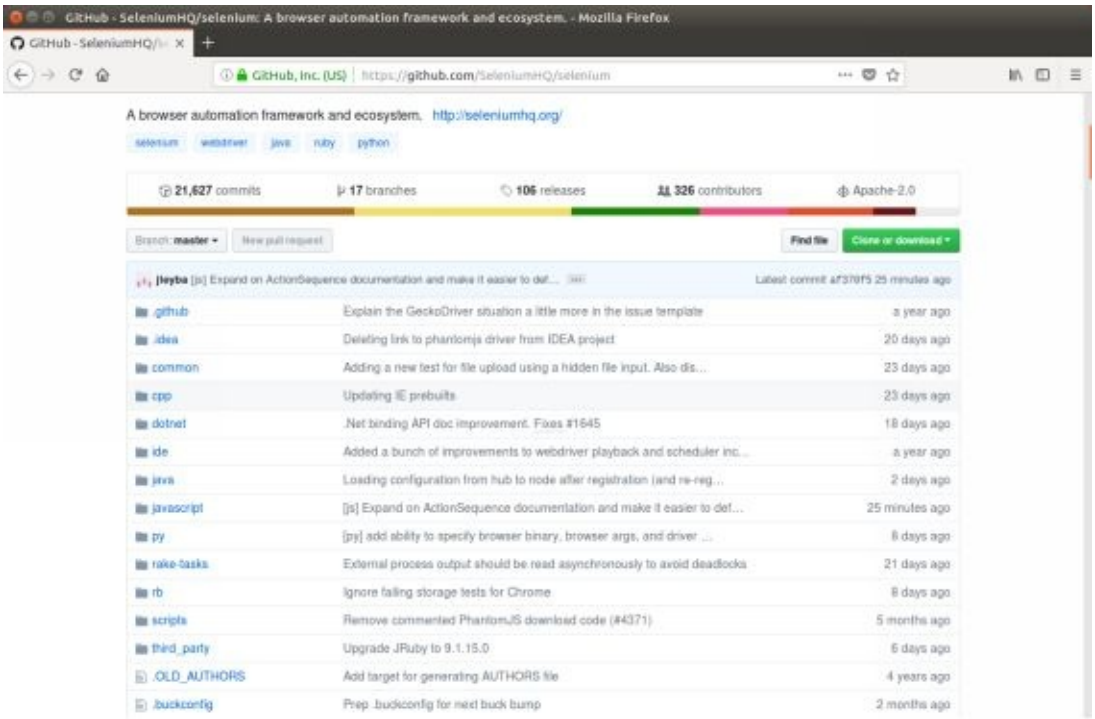
The following Selenium official website is very useful when using automatic operation with Selenium. We can see Selenium related documents on this site and we can also download Selenium related softwares from this site.

http://www.seleniumhq.org/

Selenium has been developed as open source software and anyone can use it free of charge under the Apache 2.0 License.

The latest source code can be checked on the site shown below.

GitHub - SeleniumHQ/selenium: A browser automation framework and ecosystem.



https://github.com/SeleniumHQ/selenium

1-1-1 What is Selenium IDE ?

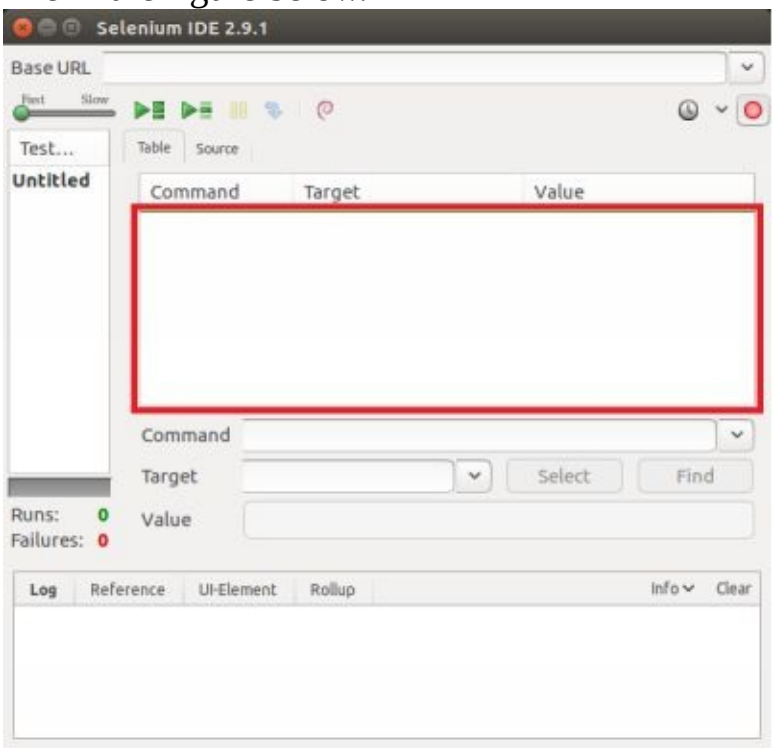
Selenium IDE (Integrated Development Environment) is a prototyping tool for building test scripts. It is a Firefox plugin and provides an easy-to-use interface for developing automated tests.

(Advantages)
Easy to use and install
No programming knowledge is necessary
It has recording feature which records user actions

(Disadvantages)
Support only Firefox browser
It doesn't work on Firefox version 55 or later

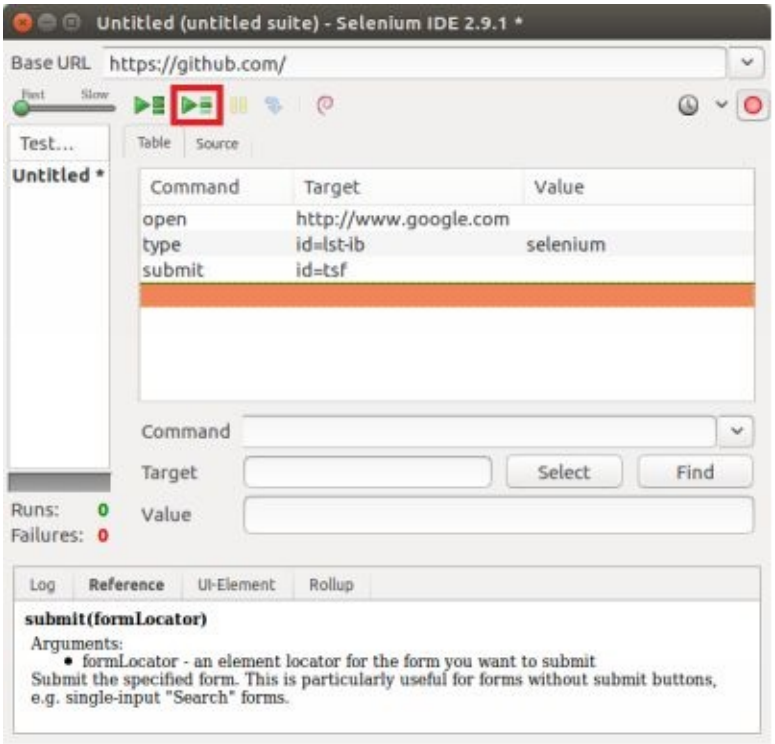
The flow of automatic operation of Firefox browser with Selenium IDE is as follows.

Enter the commands of automatic operation at the place surrounded by the red line in the figure below.



When finishing entering the automatic operation commands, click the execute

button.

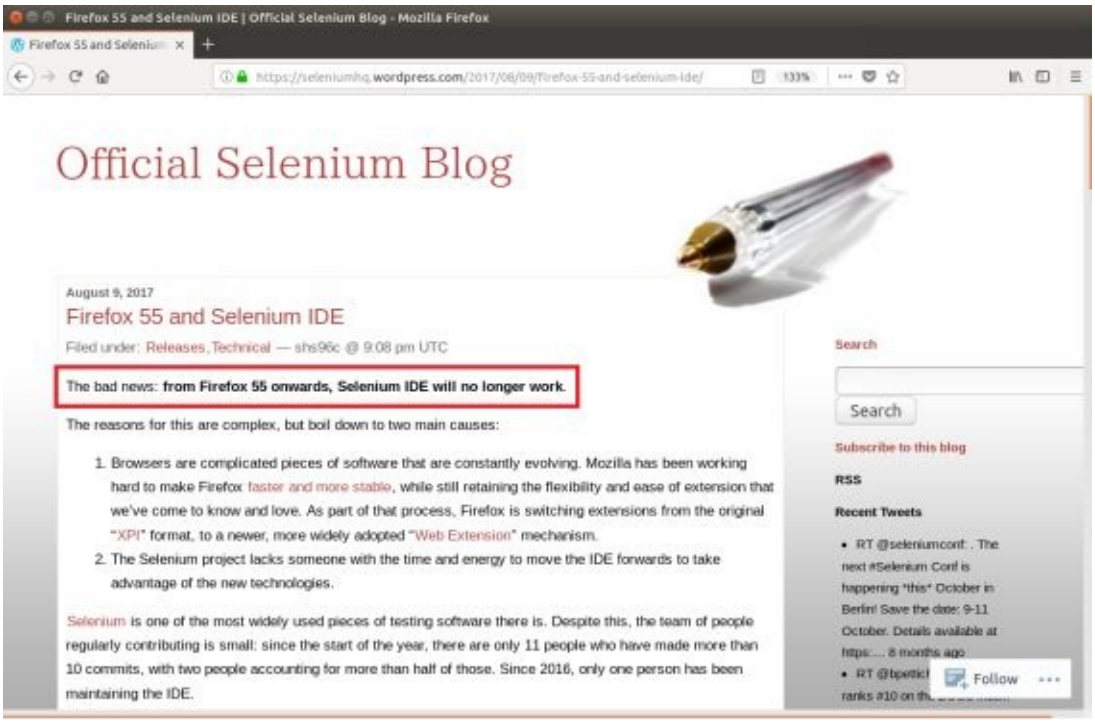


As a result, the following operations are performed automatically in Firefox.
"http://www.google.com" is displayed
"selenium" is entered in the search text box
The google search is executed and the search result is displayed

The figure below shows the search result screen after automatic operation is completed.

Selenium IDE is a convenient tool which can easily perform automatic browser operation by ordering the commands but unfortunately Selenium IDE doesn’t work on Firefox 55 or later.

The following is an announcement on Official Selenium Blog about Selenium IDE.



https://seleniumhq.wordpress.com/2017/08/09/firefox-55-and-selenium-ide/

Note: Selenium IDE alternative tool

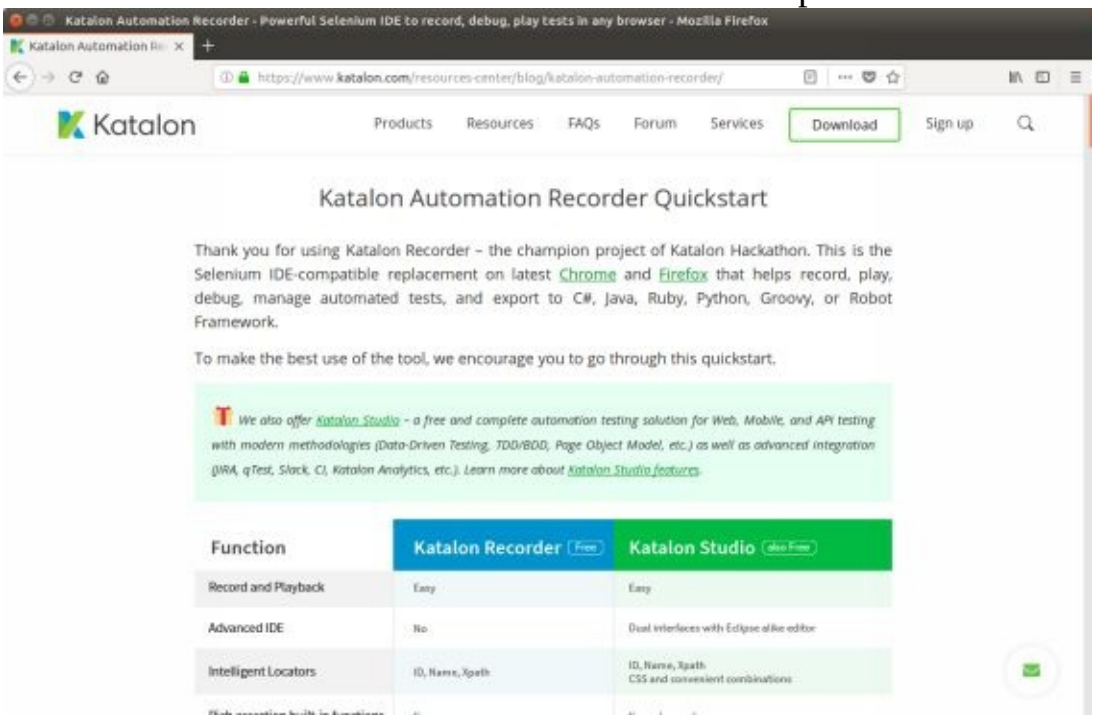
I introduce two tools like Selenium IDE which do not require programming knowledge and are easy to use.

Katalon Recorder

Robot Framework + Selenium2Library

■Katalon Recorder

Katalon Recorder is the Selenium IDE-compatible replacement on latest Chrome and Firefox. It can be used with the same sense of operation as Selenium IDE.



https://www.katalon.com/resources-center/blog/katalon-automation-recorder/

The features of Katalon Recorder are as follows.

Easy to use and install

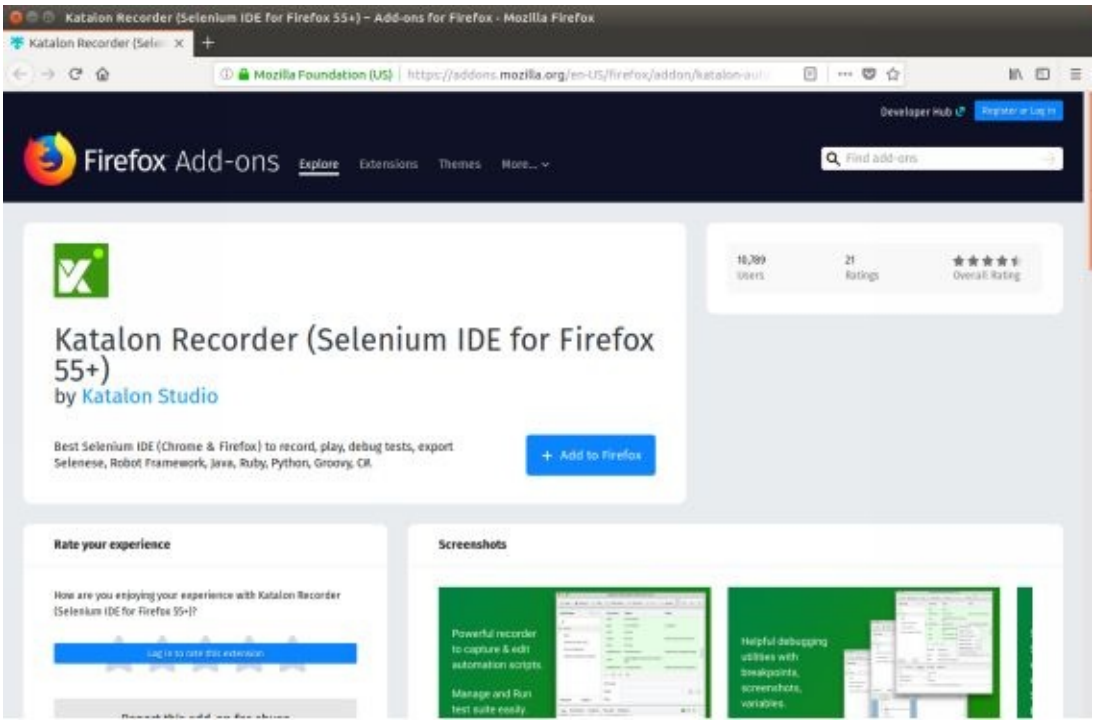
No programming knowledge is necessary

It has recording feature which records user actions

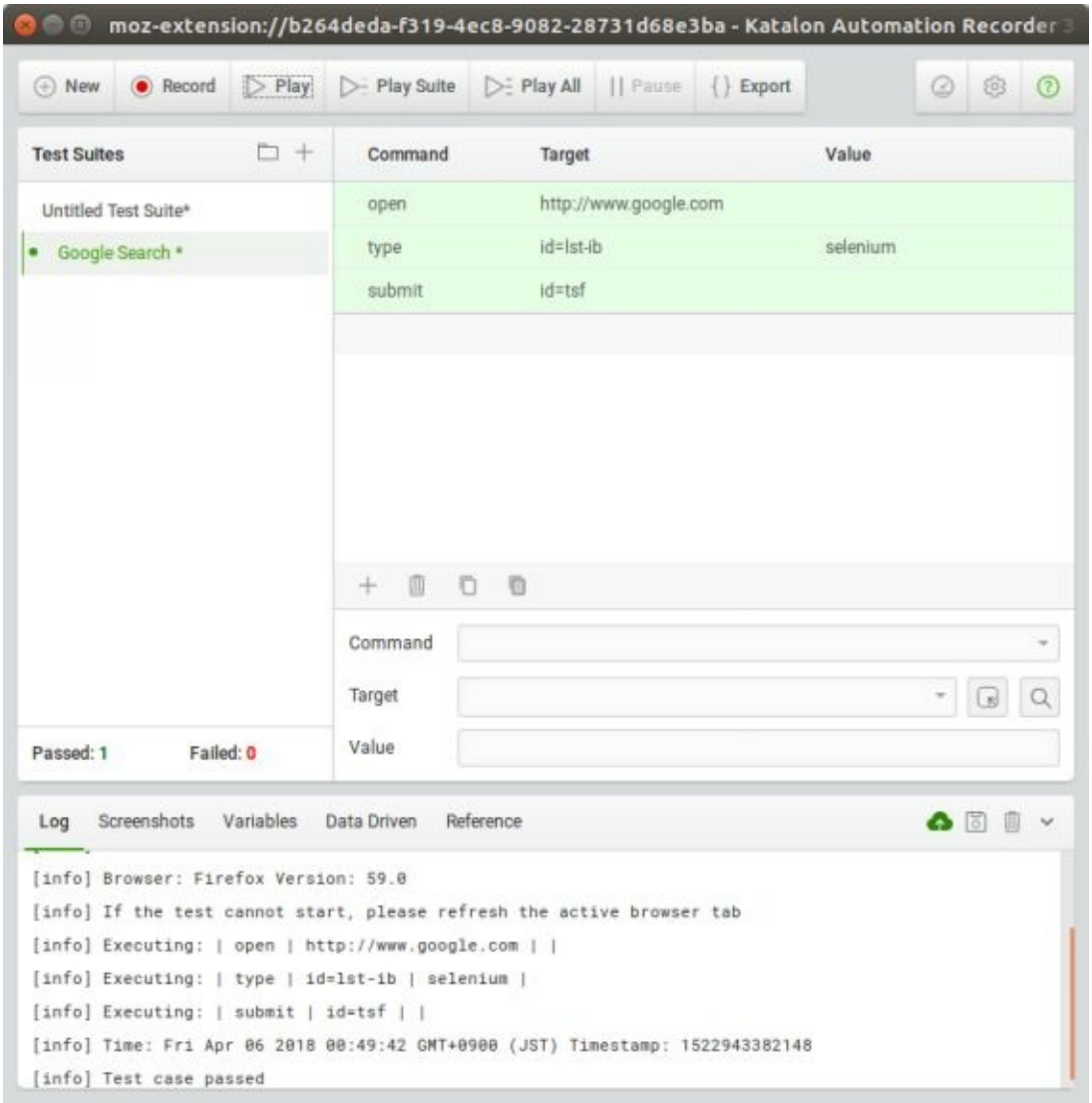
Support both Chrome and Firefox

It works on latest Chrome and Firefox

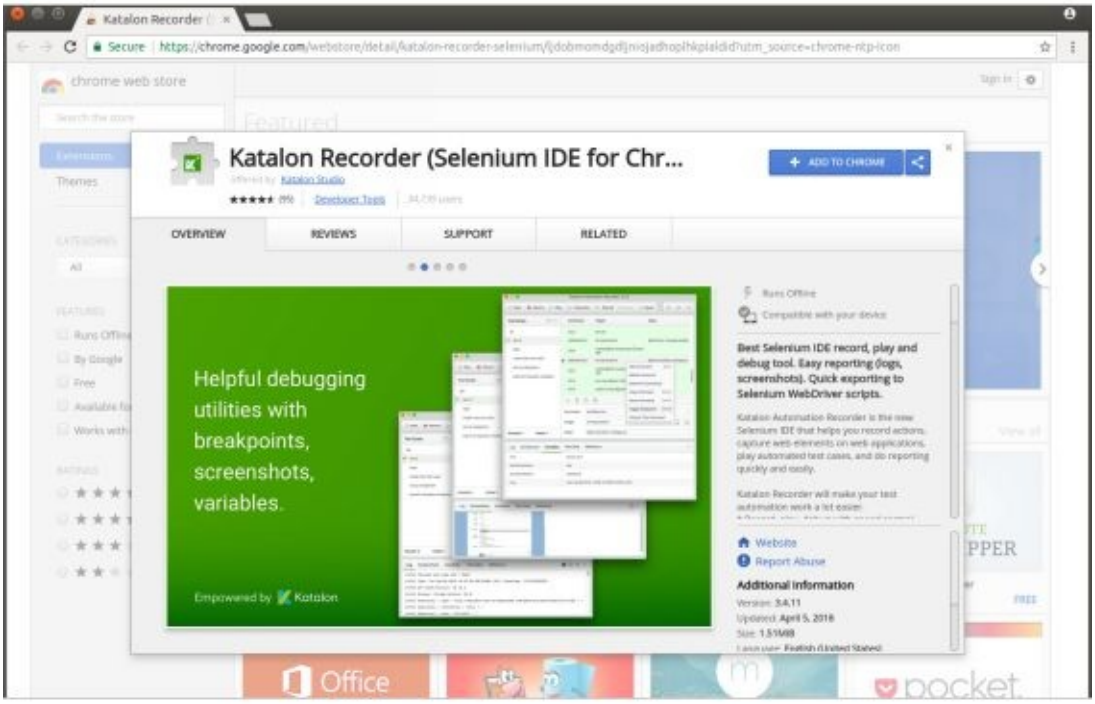
Katalon Recorder (Firefox)



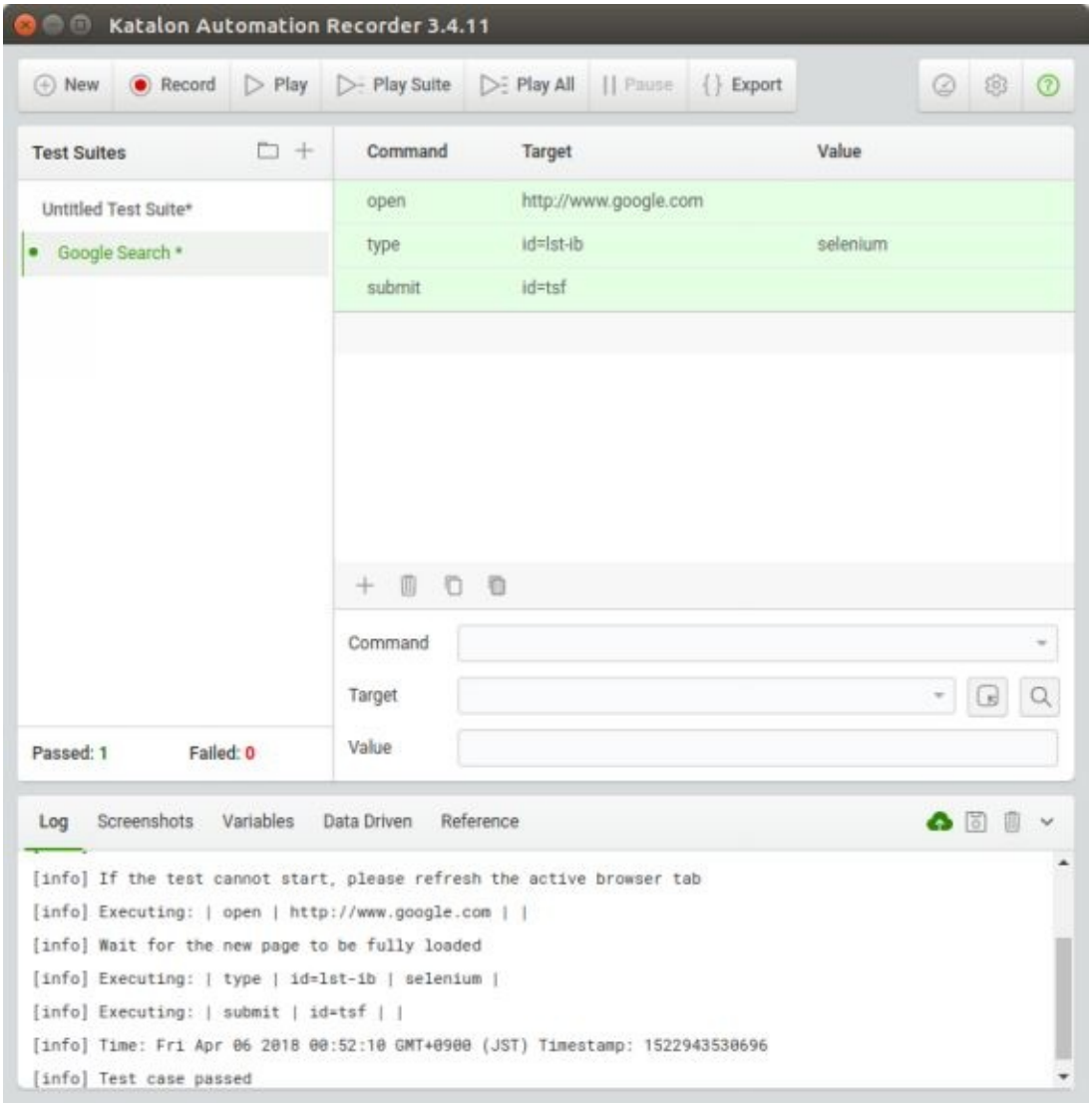
Execution screen of Katalon Recorder (Firefox)



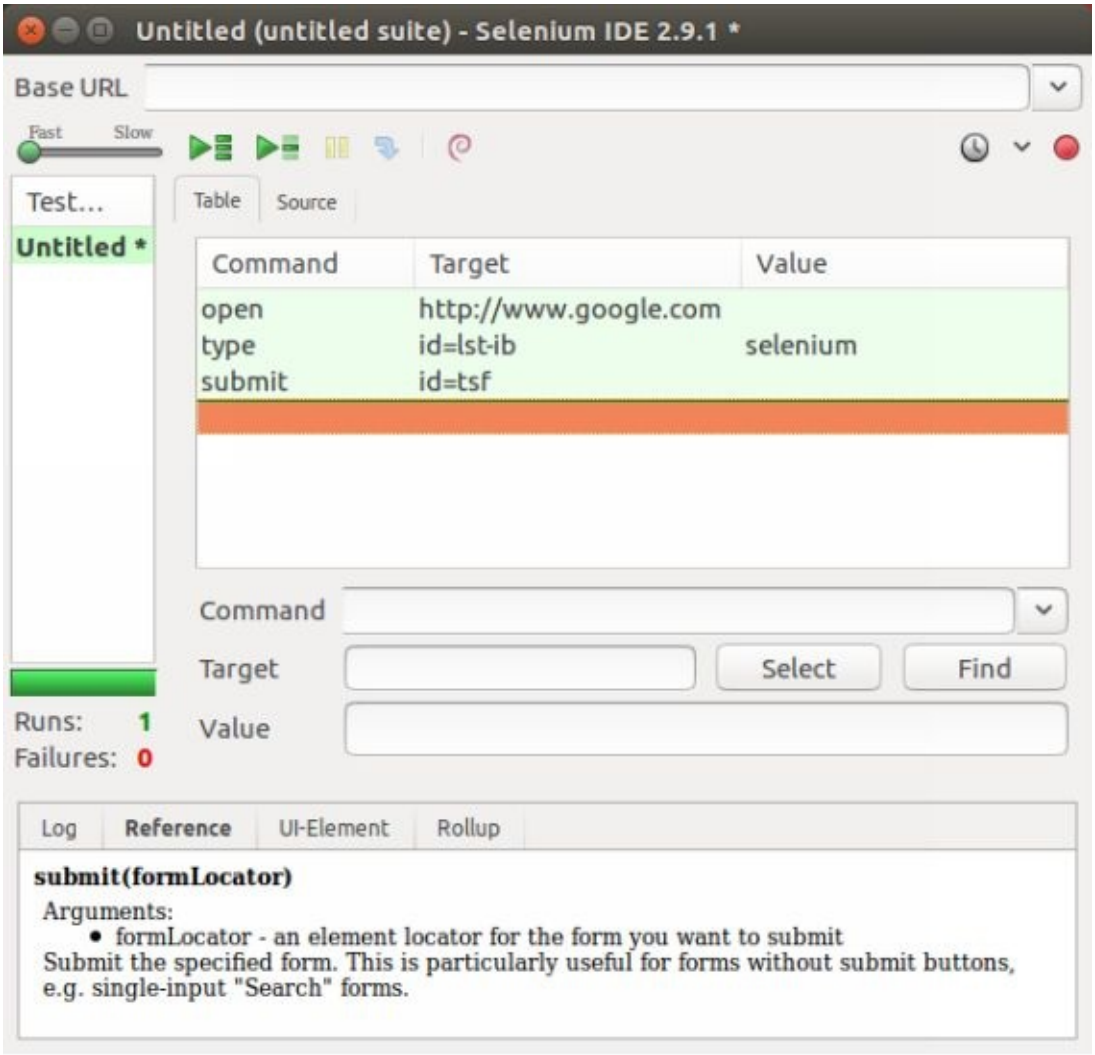
Katalon Recorder (Chrome)



Execution screen of Katalon Recorder (Chrome)



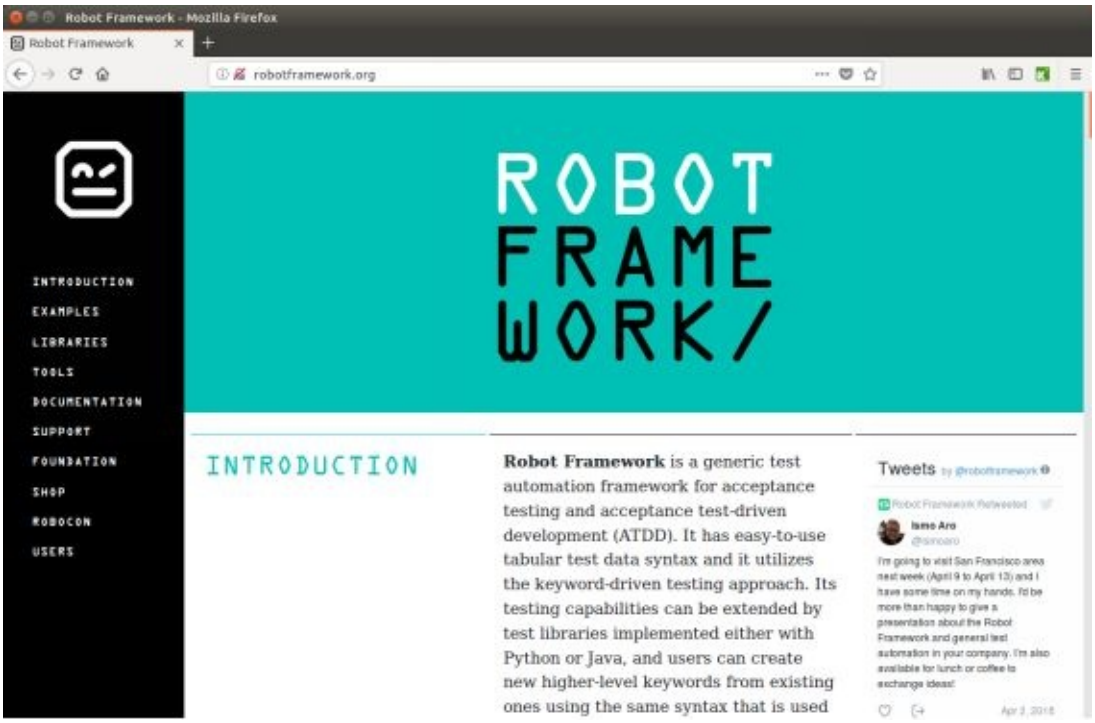
Execution screen of Selenium IDE



As you can see above, Katalon Recorder and Selenium IDE are using the same commands for automatic operation.

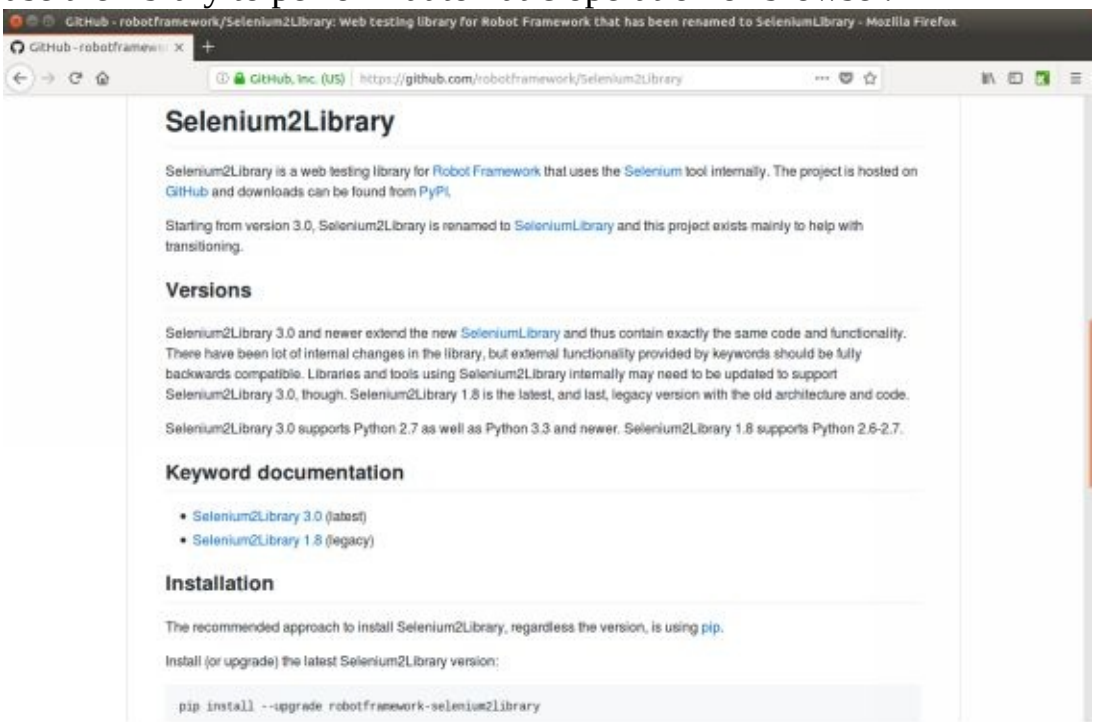
■Robot Framework + Selenium2Library
Robot Framework is a generic test automation framework for acceptance testing and acceptance test-driven development (ATDD).

Robot Framework official website



http://robotframework.org/

Robot Framework has an extension library called Selenium2Library and we can use the library to perform automatic operation of browser.



https://github.com/robotframework/Selenium2Library

The features of Robot Framework + Selenium2Library are as follows.

(Advantages)
No programming knowledge is necessary
Support a lot of browsers such as Chrome, IE, Edge, Firefox, Opera, Safari
Generate test result report

(Disadvantages)
Installation is more complicated than Selenium IDE

The following file is an example of automatic operation for Firefox using Robot Framework + Selenium2Library. The red character part is the content of automatic operation.

google_search_firefox.robot

```
*** Settings ***
Library Selenium2Library

*** Test Cases ***
Robotframework demo
    Open Browser http://www.google.com firefox
    Input Text id=lst-ib selenium
    Submit Form id=tsf
```

Below are the commands in Selenium IDE when doing the same operation but we can see that they are very similar.

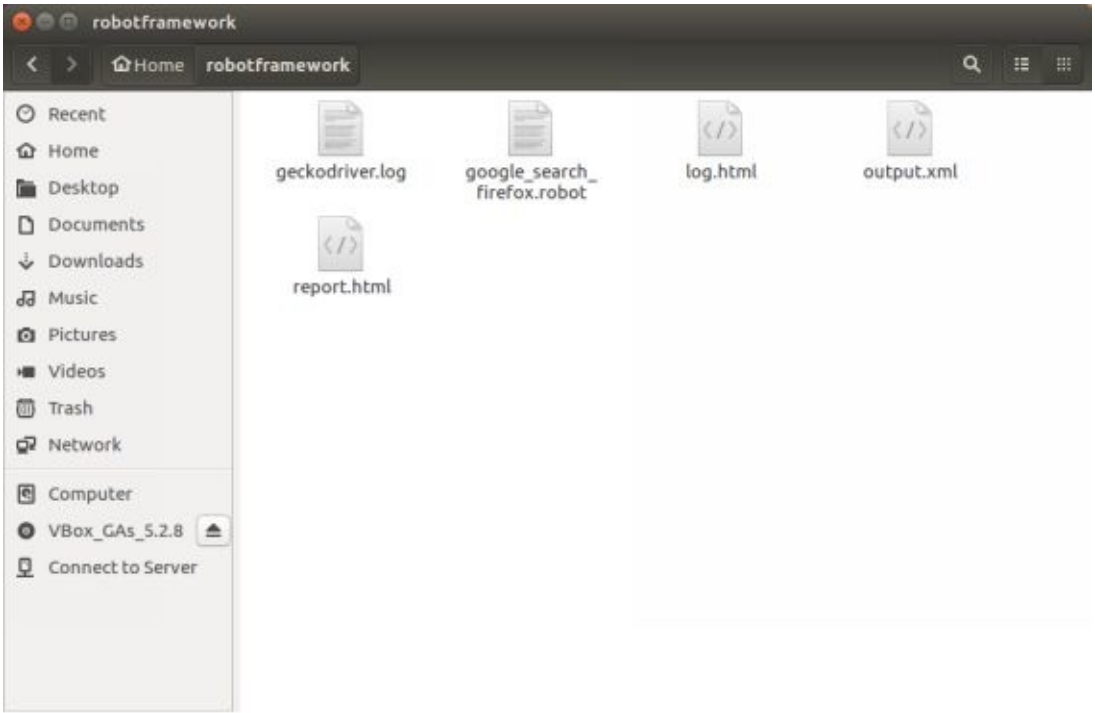
```
open http://www.google.com
type id=lst-ib selenium
submit id=tsf
```

Execute "google_search_firefox.robot" with robot command, so automatic operation is started.

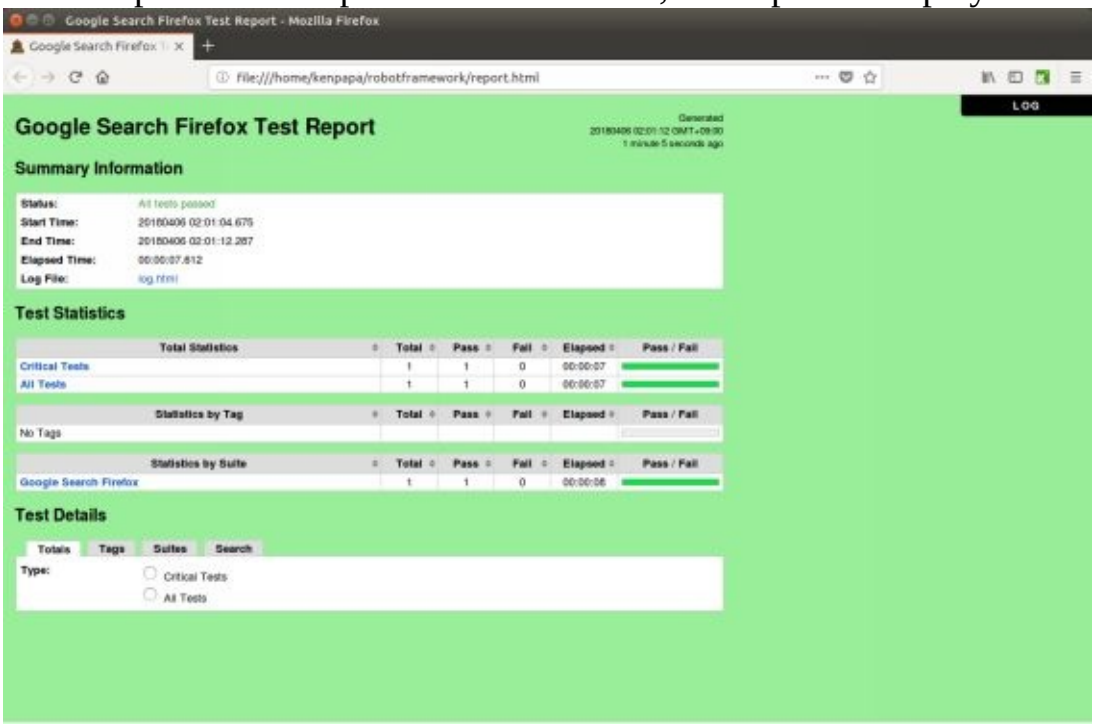
```
kenpapa@kenpapa-VirtualBox: ~/robotframework
kenpapa@kenpapa-VirtualBox:~/robotframework$ ls
google_search_firefox.robot
kenpapa@kenpapa-VirtualBox:~/robotframework$ robot google_search_firefox.robot
==============================================================================
Google Search Firefox
==============================================================================
Robotframework demo                                                   | PASS |
------------------------------------------------------------------------------
Google Search Firefox                                                 | PASS |
1 critical test, 1 passed, 0 failed
1 test total, 1 passed, 0 failed
==============================================================================
Output:  /home/kenpapa/robotframework/output.xml
Log:     /home/kenpapa/robotframework/log.html
Report:  /home/kenpapa/robotframework/report.html
kenpapa@kenpapa-VirtualBox:~/robotframework$
```

The figure below shows the search result screen after automatic operation is completed.

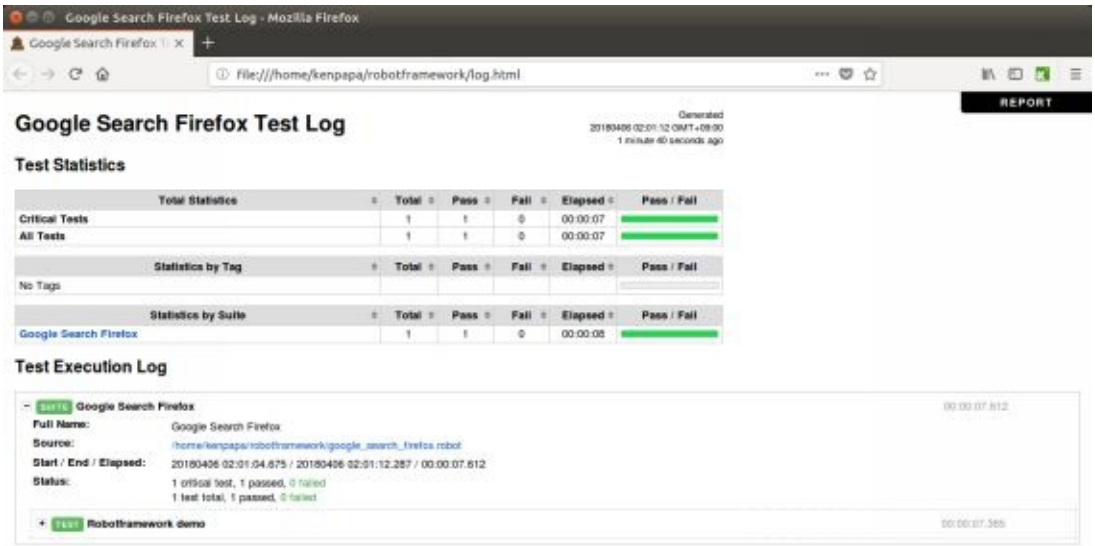
The following files are created after automatic operation is completed.



When report.html is opened in the browser, test report is displayed.



When log.html is opened in the browser, test log is displayed.



Selenium WebDriver (Python) is running inside Robot Framework + Selenium2Library but since we can perform automatic operation of browser without being conscious of Python programming, this tool would be easy to use for Selenium IDE users.

1-1-2 What is Selenium WebDriver ?

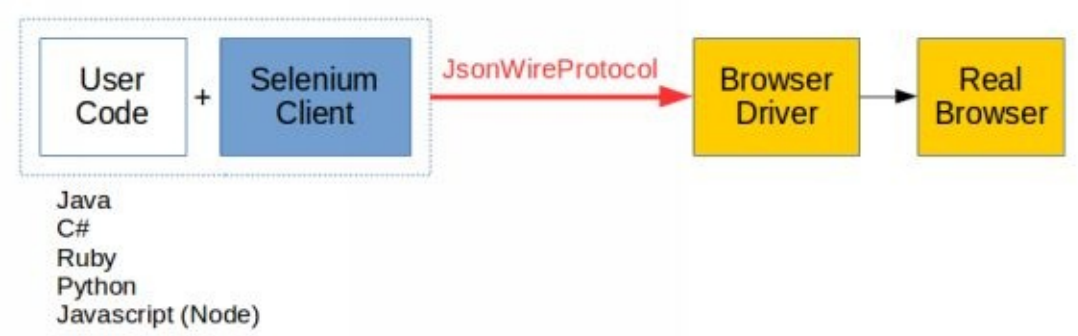
Selenium WebDriver is programming interface for automatic operation of browser. It was developed to better support dynamic web pages where elements of a page may change without the page itself being reloaded. Its goal is to supply a well-designed object-oriented API that provides improved support for modern advanced web-app testing problems.

(Advantages)
Support a lot of browsers such as Chrome, IE, Edge, Firefox, Opera, Safari
Support a lot of programming languages such as Java, C#, Ruby, Python, JavaScript(Node), PHP

(Disadvantages)
Installation is more complicated than Selenium IDE
Programming knowledge is necessary

There are two system configurations of Selenium WebDriver.
a) Only Selenium Client
b) Selenium Client + Selenium Server

■Only Selenium Client



The five programming languages (Java, C#, Ruby, Python, Javascript(Node)) officially supported by Selenium HQ can perform automatic operation of browser with this system configuration.

The code example of automatic operation in each language is shown below.

GoogleSearchChrome.java

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.chrome.ChromeDriver;

public class GoogleSearchChrome {

   public static void main(String[] args) {
   WebDriver driver = new ChromeDriver();

   driver.get("http://www.google.com");
   WebElement element = driver.findElement(By.id("lst-ib"));
   element.sendKeys("selenium");
   element.submit();
   }

}
```

GoogleSearchChrome.cs

```
using System;
using OpenQA.Selenium.Chrome;
using OpenQA.Selenium;

public class GoogleSearchChrome
{
   public static void Main(string[] args)
   {
   IWebDriver driver = new ChromeDriver();

   driver.Navigate().GoToUrl("http://www.google.com");
   IWebElement element = driver.FindElement (By.Id("lst-ib"));
   element.SendKeys("selenium");
   element.Submit();
   }
}
```

google_search_chrome.rb

```
require 'selenium-webdriver'
```

```
driver = Selenium::WebDriver.for :chrome

driver.get("http://www.google.com")
element = driver.find_element(:id => "lst-ib")
element.send_keys("selenium")
element.submit
```

google_search_chrome.py

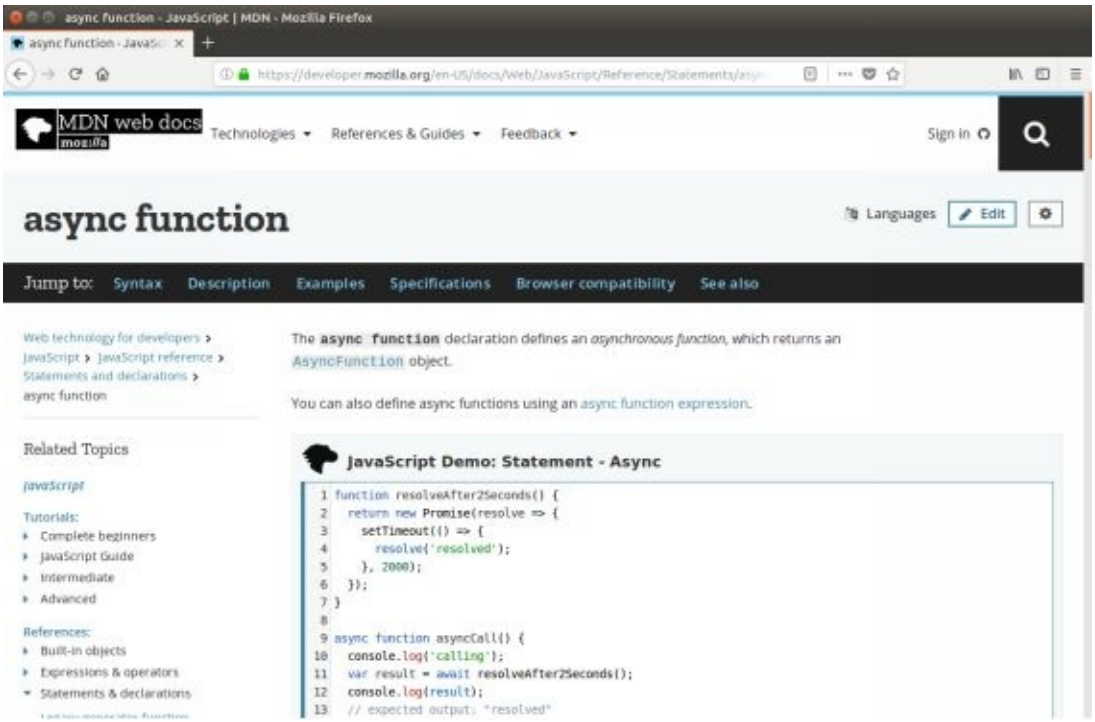
```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
```

google_search_chrome.js

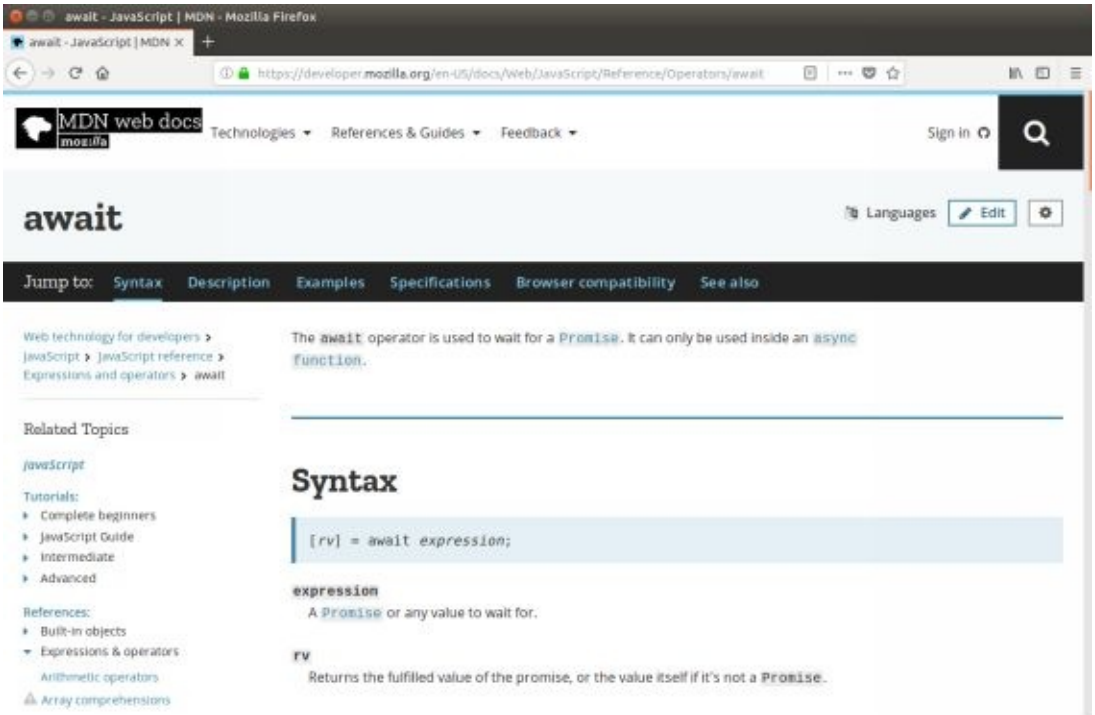
```
var webdriver = require("selenium-webdriver");
var By = webdriver.By;

var driver = new webdriver.Builder().forBrowser("chrome").build();

async function main() {
await driver.get("http://www.google.com");
element = await driver.findElement(By.id("lst-ib"));
await element.sendKeys("selenium");
await element.submit();
}

main();
```

Since Node.js is asynchronous, it is necessary to add wait processing to the code of automatic operation. In above code, async and await are used for wait processing.

For async and await, the following sites explains in detail.



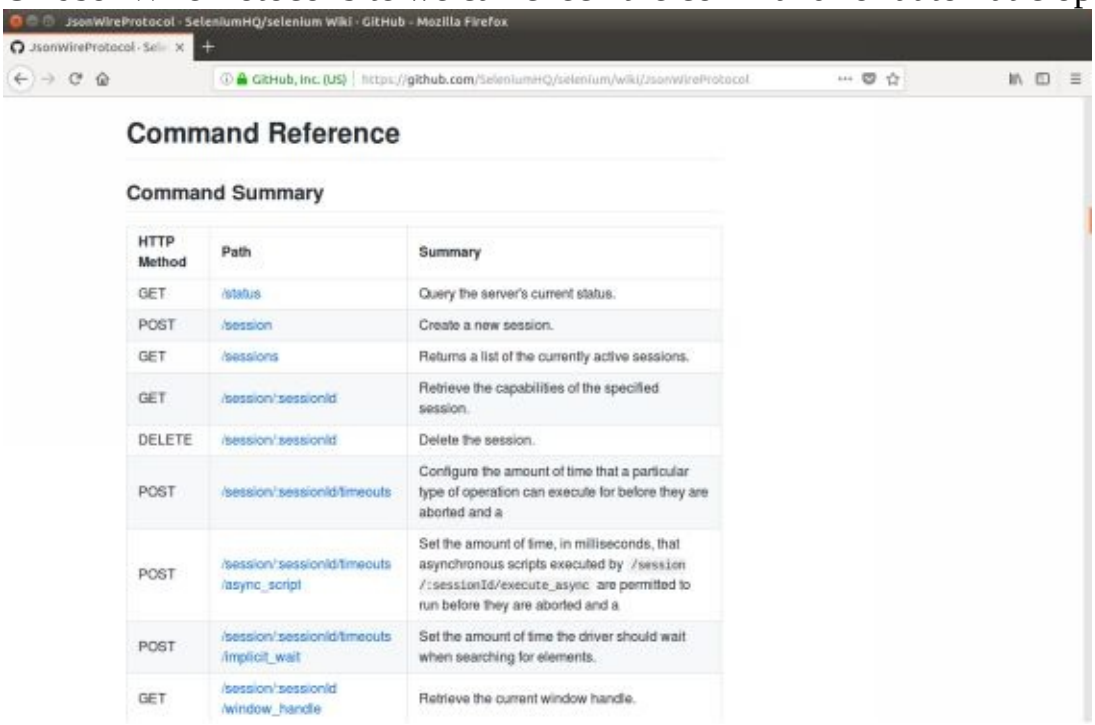
https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Statements/async_function



https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Operators/await

Selenium Client and browser driver communicate each other by HTTP method using JsonWireProtocol. When we execute the code of the automatic operation as described above, Selenium Client internally uses JsonWireProtocol to interact with browser driver and the browser operation is performed.

On JsonWireProtocol site we can check the command for automatic operation.



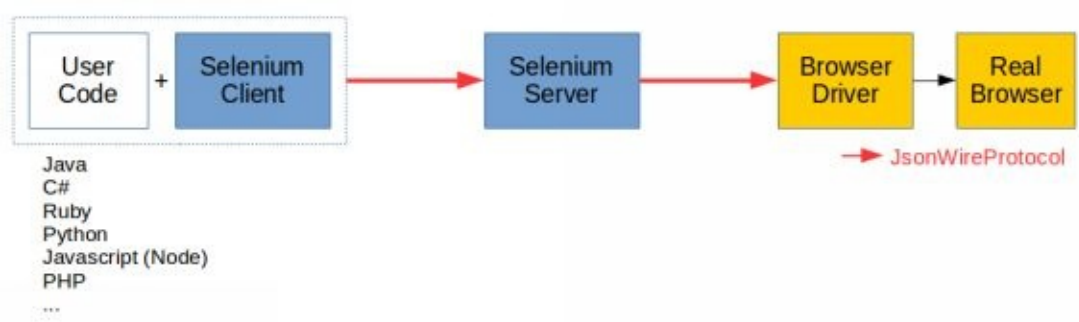
Command Reference

Command Summary

HTTP Method	Path	Summary
GET	/status	Query the server's current status.
POST	/session	Create a new session.
GET	/sessions	Returns a list of the currently active sessions.
GET	/session/:sessionId	Retrieve the capabilities of the specified session.
DELETE	/session/:sessionId	Delete the session.
POST	/session/:sessionId/timeouts	Configure the amount of time that a particular type of operation can execute for before they are aborted and a
POST	/session/:sessionId/timeouts /async_script	Set the amount of time, in milliseconds, that asynchronous scripts executed by /session /:sessionId/execute_async are permitted to run before they are aborted and a
POST	/session/:sessionId/timeouts /implicit_wait	Set the amount of time the driver should wait when searching for elements.
GET	/session/:sessionId /window_handle	Retrieve the current window handle.

https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol

■Selenium Client + Selenium Server



As for languages other than the five programming languages officially supported by Selenium HQ such as PHP, we can use this system configuration for automatic operation of browser.

For example, the code below is an example of automatic operation code using PHP's Selenium Client (by Facebook). The red character part indicates that this code is using Selenium Server.

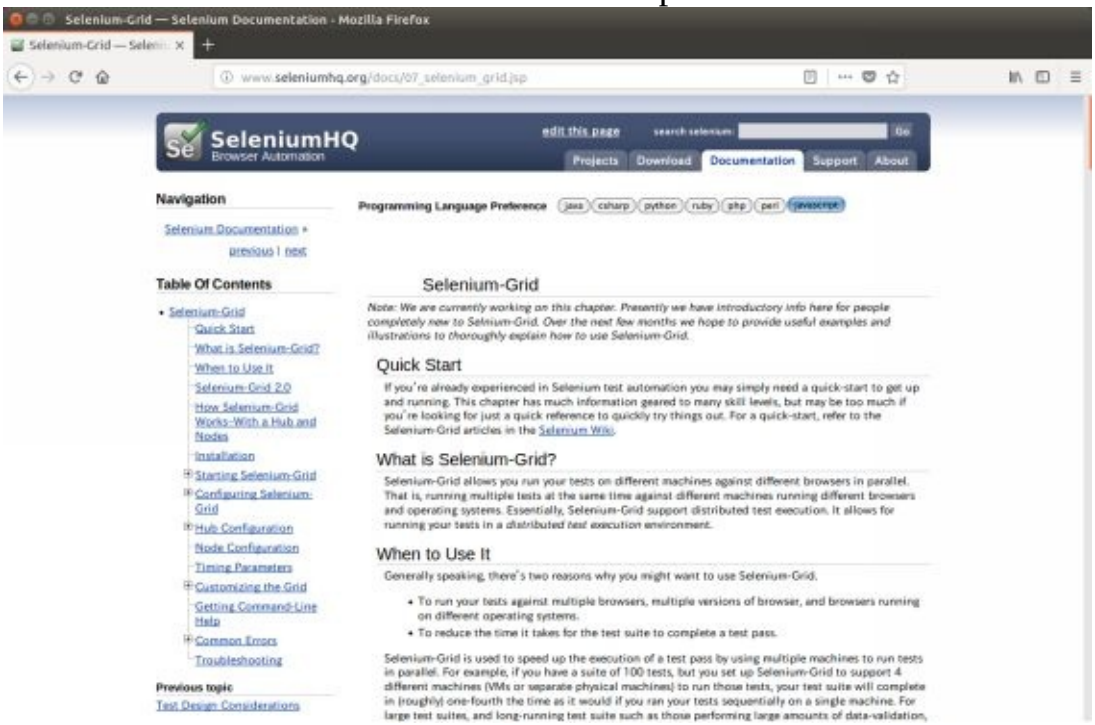
```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use Facebook¥WebDriver¥WebDriverBy;
use Facebook¥WebDriver¥Remote¥RemoteWebDriver;
use Facebook¥WebDriver¥Remote¥DesiredCapabilities;

$host = 'http://localhost:4444/wd/hub';
$driver = RemoteWebDriver::create($host, DesiredCapabilities::chrome());

$driver->get("http://www.google.com");
$element = $driver->findElement(WebDriverBy::id("lst-ib"));
$element->sendKeys("selenium");
$element->submit();
```

Besides that, Selenium Grid uses this system configuration.

1-1-3 What is Selenium Grid ?

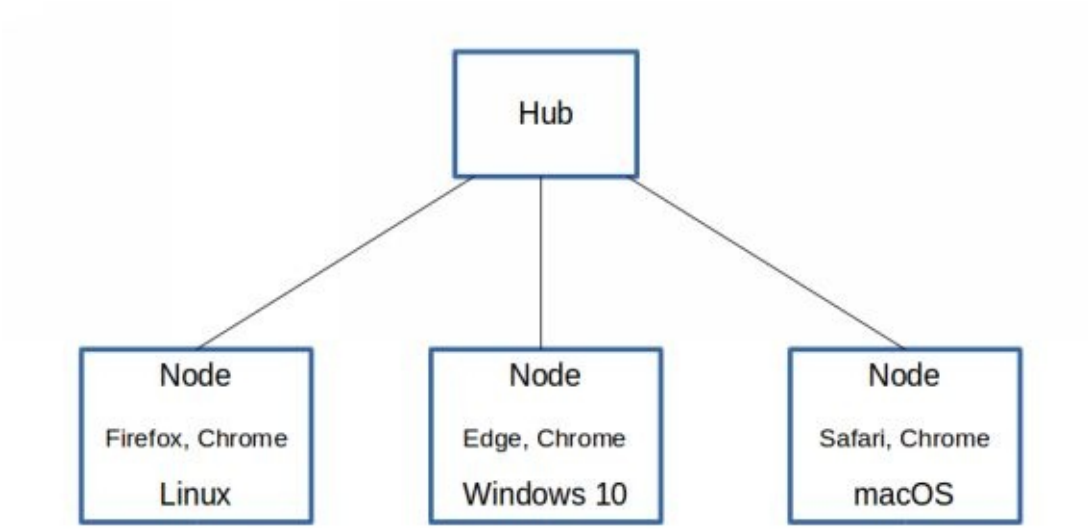
Selenium Grid allows us to run our tests on different machines against different browsers in parallel. That is, running multiple tests at the same time against different machines running different browsers and operating systems. Selenium Grid is used with Selenium WebDriver.

Selenium official website has a detailed explanation about Selenium Grid.



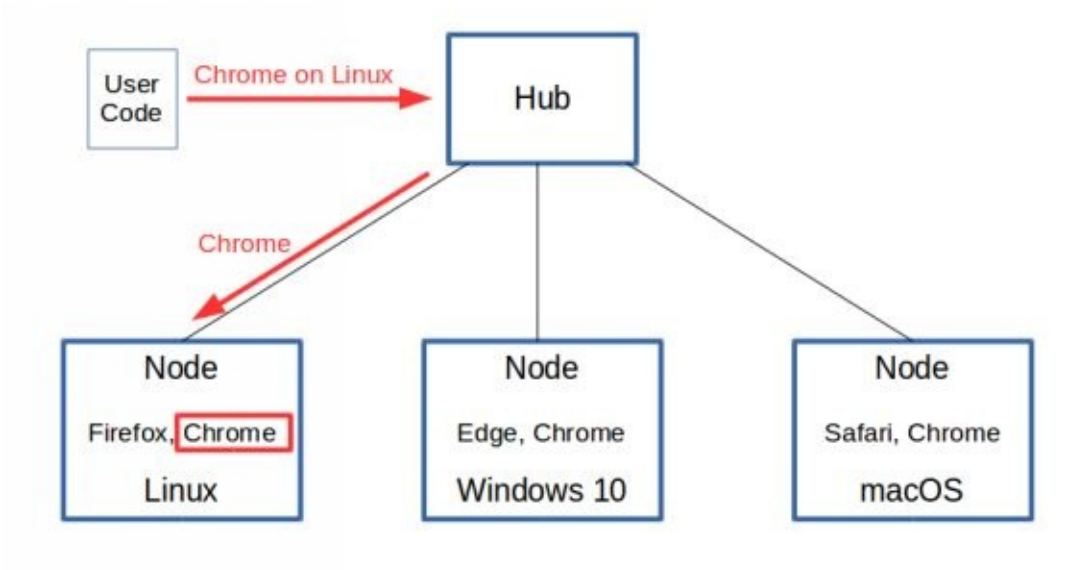
http://www.seleniumhq.org/docs/07_selenium_grid.jsp

Selenium Grid consists of Hub and Node. In the figure below, Node is composed of three machines (Linux, Windows10, macOS) and two browsers can be used for each Node.

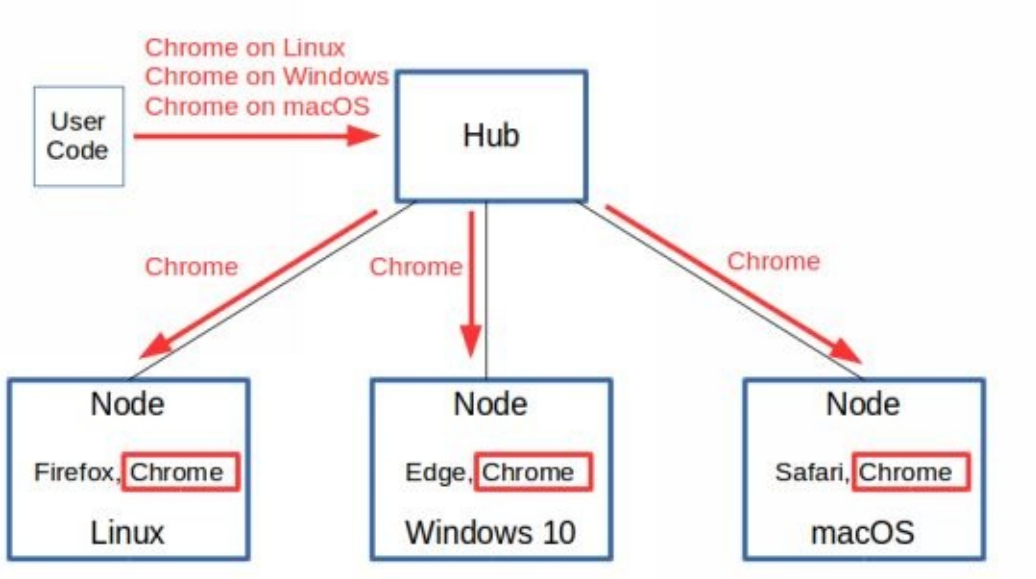


Hub knows the information of Node and when an execution request from User Code comes, the Hub issues an execution request to the corresponding Node and automatic operation is performed at the requested Node.

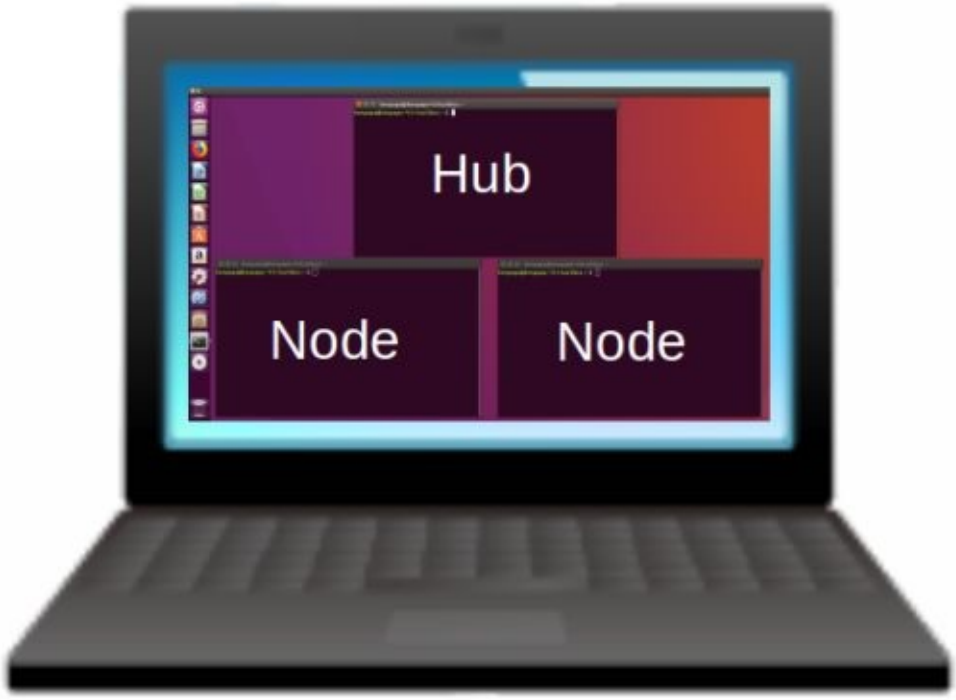
In the figure below, since a request "Chrome on Linux" comes, the Hub issues an execution request to the left Node.



Since Selenium Grid supports parallel execution, when Hub receives multiple execution requests from User Code at the same time, the Hub simultaneously issues multiple execution requests to the corresponding Nodes.



Now that we understand the overview of Selenium Grid, we'd like to experience automatic operation in Selenium Grid. However, it isn't easy to prepare multiple machines, so here we use three terminals on one virtual machine (Ubuntu) and construct a Selenium Grid environment with each terminal as Hub, Node and Node as shown below.



■Creation of Hub

Hub is created in the figure below. When the creation of the Hub is successful, the message "Selenium Grid hub is up and running" is displayed.

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ java -jar selenium-server-standalone-3.8.1.jar -role hub
22:09:25.389 INFO - Selenium build info: version: '3.8.1', revision: 'de93a0084b'
22:09:25.391 INFO - Launching Selenium Grid hub
2017-12-21 22:09:26.461:INFO::main: Logging initialized @1585ms to org.seleniumhq.jetty
9.util.log.StdErrLog
22:09:26.540 INFO - Will listen on 4444
2017-12-21 22:09:26.709:INFO:osjs.Server:main: jetty-9.4.7.v20170914
2017-12-21 22:09:26.805:INFO:osjs.session:main: DefaultSessionIdManager workerName=node
0
2017-12-21 22:09:26.806:INFO:osjs.session:main: No SessionScavenger set, using defaults
2017-12-21 22:09:26.811:INFO:osjs.session:main: Scavenging every 600000ms
2017-12-21 22:09:26.825:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextH
andler@709ba3fb{/,null,AVAILABLE}
2017-12-21 22:09:26.878:INFO:osjs.AbstractConnector:main: Started ServerConnector@7fd71
97f{HTTP/1.1,[http/1.1]}{0.0.0.0:4444}
2017-12-21 22:09:26.880:INFO:osjs.Server:main: Started @2005ms
22:09:26.880 INFO - Nodes should register to http://10.0.2.15:4444/grid/register/
22:09:26.880 INFO - Selenium Grid hub is up and running
```

Since Selenium Server is necessary to create Selenium Grid's Hub and Node, we need to download it beforehand. In addition, Java is required to start Selenium Server, so we need to install it beforehand.

■Creation of Node

The first Node is created in the figure below. We set up Firefox and Chrome so that we can start up to five at the same time.

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ java -jar selenium-server-standalone-3.8.1.jar -role node
 -hub http://10.0.2.15:4444/grid/register -port 5555 -browser browserName=firefox,maxIn
stances=5 -browser browserName=chrome,maxInstances=5
```

When the creation of the Node is successful, the message "The node is registered to the hub and ready to use" is displayed.

```
kenpapa@kenpapa-VirtualBox: ~
22:11:30.268 INFO - Using `new EdgeOptions()` is preferred to `DesiredCapabilities.edge
()`
22:11:30.268 INFO - Using `new FirefoxOptions()` is preferred to `DesiredCapabilities.f
irefox()`
22:11:30.272 INFO - Using `new OperaOptions()` is preferred to `DesiredCapabilities.ope
raBlink()`
22:11:30.274 INFO - Using `new SafariOptions()` is preferred to `DesiredCapabilities.sa
fari()`
22:11:30.304 INFO - Using the passthrough mode handler
2017-12-21 22:11:30.420:INFO:osjs.Server:main: jetty-9.4.7.v20170914
2017-12-21 22:11:30.532:WARN:osjs.SecurityHandler:main: ServletContext@o.s.j.s.ServletC
ontextHandler@45c7e403{/,null,STARTING} has uncovered http methods for path: /
2017-12-21 22:11:30.538:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextH
andler@45c7e403{/,null,AVAILABLE}
2017-12-21 22:11:30.576:INFO:osjs.AbstractConnector:main: Started ServerConnector@10063
2e{HTTP/1.1,[http/1.1]}{0.0.0.0:5555}
2017-12-21 22:11:30.577:INFO:osjs.Server:main: Started @2067ms
22:11:30.577 INFO - Selenium Grid node is up and ready to register to the hub
22:11:30.659 INFO - Starting auto registration thread. Will try to register every 5000
ms.
22:11:30.659 INFO - Registering the node to the hub: http://10.0.2.15:4444/grid/registe
r
22:11:30.785 INFO - The node is registered to the hub and ready to use
```

The message "Registerd a node …" is displayed on the Hub's terminal.

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ java -jar selenium-server-standalone-3.8.1.jar -role hub
22:09:25.389 INFO - Selenium build info: version: '3.8.1', revision: '6e95a6684b'
22:09:25.391 INFO - Launching Selenium Grid hub
2017-12-21 22:09:26.461:INFO::main: Logging initialized @1585ms to org.seleniumhq.jetty
9.util.log.StdErrLog
22:09:26.540 INFO - Will listen on 4444
2017-12-21 22:09:26.709:INFO:osjs.Server:main: jetty-9.4.7.v20170914
2017-12-21 22:09:26.805:INFO:osjs.session:main: DefaultSessionIdManager workerName=node
0
2017-12-21 22:09:26.806:INFO:osjs.session:main: No SessionScavenger set, using defaults
2017-12-21 22:09:26.811:INFO:osjs.session:main: Scavenging every 600000ms
2017-12-21 22:09:26.825:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextH
andler@709ba3fb{/,null,AVAILABLE}
2017-12-21 22:09:26.878:INFO:osjs.AbstractConnector:main: Started ServerConnector@7fd71
97f{HTTP/1.1,[http/1.1]}{0.0.0.0:4444}
2017-12-21 22:09:26.880:INFO:osjs.Server:main: Started @2005ms
22:09:26.880 INFO - Nodes should register to http://10.0.2.15:4444/grid/register/
22:09:26.880 INFO - Selenium Grid hub is up and running
22:11:30.785 INFO - Registered a node http://10.0.2.15:5555
```

The second Node is created in the figure below. We set up Firefox and Chrome so that we can start up to five at the same time.

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ java -jar selenium-server-standalone-3.8.1.jar -role node
 -hub http://10.0.2.15:4444/grid/register -port 5556 -browser browserName=firefox,maxIn
stances=5 -browser browserName=chrome,maxInstances=5
```

When the creation of the Node is successful, the message "The node is registered to the hub and ready to use" is displayed.

```
kenpapa@kenpapa-VirtualBox: ~
22:12:16.291 INFO - Using `new EdgeOptions()` is preferred to `DesiredCapabilities.edge
()`
22:12:16.291 INFO - Using `new FirefoxOptions()` is preferred to `DesiredCapabilities.f
irefox()`
22:12:16.297 INFO - Using `new OperaOptions()` is preferred to `DesiredCapabilities.ope
raBlink()`
22:12:16.302 INFO - Using `new SafariOptions()` is preferred to `DesiredCapabilities.sa
fari()`
22:12:16.330 INFO - Using the passthrough mode handler
2017-12-21 22:12:16.440:INFO:osjs.Server:main: jetty-9.4.7.v20170914
2017-12-21 22:12:16.550:WARN:osjs.SecurityHandler:main: ServletContext@o.s.j.s.ServletC
ontextHandler@45c7e403{/,null,STARTING} has uncovered http methods for path: /
2017-12-21 22:12:16.561:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextH
andler@45c7e403{/,null,AVAILABLE}
2017-12-21 22:12:16.623:INFO:osjs.AbstractConnector:main: Started ServerConnector@10063
2e{HTTP/1.1,[http/1.1]}{0.0.0.0:5556}
2017-12-21 22:12:16.624:INFO:osjs.Server:main: Started @1899ms
22:12:16.624 INFO - Selenium Grid node is up and ready to register to the hub
22:12:16.713 INFO - Starting auto registration thread. Will try to register every 5000
ms.
22:12:16.713 INFO - Registering the node to the hub: http://10.0.2.15:4444/grid/registe
r
22:12:16.781 INFO - The node is registered to the hub and ready to use
```

The message "Registerd a node …" is displayed on the Hub's terminal.

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ java -jar selenium-server-standalone-3.8.1.jar -role hub
22:09:25.389 INFO - Selenium build info: version: '3.8.1', revision: '6e95a6684b'
22:09:25.391 INFO - Launching Selenium Grid hub
2017-12-21 22:09:26.461:INFO::main: Logging initialized @1585ms to org.seleniumhq.jetty
9.util.log.StdErrLog
22:09:26.540 INFO - Will listen on 4444
2017-12-21 22:09:26.709:INFO:osjs.Server:main: jetty-9.4.7.v20170914
2017-12-21 22:09:26.805:INFO:osjs.session:main: DefaultSessionIdManager workerName=node
0
2017-12-21 22:09:26.806:INFO:osjs.session:main: No SessionScavenger set, using defaults
2017-12-21 22:09:26.811:INFO:osjs.session:main: Scavenging every 600000ms
2017-12-21 22:09:26.825:INFO:osjsh.ContextHandler:main: Started o.s.j.s.ServletContextH
andler@709ba3fb{/,null,AVAILABLE}
2017-12-21 22:09:26.878:INFO:osjs.AbstractConnector:main: Started ServerConnector@7fd71
97f{HTTP/1.1,[http/1.1]}{0.0.0.0:4444}
2017-12-21 22:09:26.880:INFO:osjs.Server:main: Started @2005ms
22:09:26.880 INFO - Nodes should register to http://10.0.2.15:4444/grid/register/
22:09:26.880 INFO - Selenium Grid hub is up and running
22:11:30.785 INFO - Registered a node http://10.0.2.15:5555
22:12:16.782 INFO - Registered a node http://10.0.2.15:5556
```

The environment of Selenium Grid has been constructed with the above. When we check the console of Selenium Grid, we can confirm that two Nodes are registered.

Now we prepare the code for automatic operation. Let’s change the red character part of the following code so that the code can be used in Selenium Grid environment.

google_search.py

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
driver.quit()
```

The following code is the code modified for Selenium Grid. Since "firefox" is specified as the browser, the Hub is requested to execute in Firefox.

google_search_grid.py

```
from selenium import webdriver

driver = webdriver.Remote(
   command_executor="http://10.0.2.15:4444/wd/hub",
   desired_capabilities={
   "browserName": "firefox",
   })

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
driver.quit()
```

When we execute the above code, the Selenium Grid console changes as below.

We can see that the first Firefox icon on the left Node is thin. This means that it is in use. We can see that the execution request was issued to the left Node by Hub.

As described above, in Selenium Grid, when an execution request is issued to Hub, the Hub decides the Node to be executed and the automatic operation is performed.

Then, let’s try parallel execution next.

In order to execute in parallel, it is necessary to issue multiple execution requests to Hub at the same time. For example, since there is a test framework called TestNG in Java which supports parallel execution, we can use such a framework but here we use Python's threading module for parallel execution.

The following code is for Selenium Grid which performs parallel execution. firefox() is a function that performs Google Search in Firefox and chrome() is a function that performs Google Search in Chrome. In the code, firefox() and chrome() are executed twice, so total 4 browsers are launched when the code is executed.

google_search_grid_parallel.py

```
from selenium import webdriver
import threading
import time

def firefox():
   driver = webdriver.Remote(
   command_executor="http://10.0.2.15:4444/wd/hub",
   desired_capabilities={
   "browserName": "firefox",
   })
   driver.get("http://www.google.com")
   element = driver.find_element_by_id("lst-ib")
   element.send_keys("selenium")
   element.submit()
   time.sleep(10)
   driver.quit()

def chrome():
   driver = webdriver.Remote(
   command_executor="http://10.0.2.15:4444/wd/hub",
   desired_capabilities={
   "browserName": "chrome",
   })
   driver.get("http://www.google.com")
   element = driver.find_element_by_id("lst-ib")
   element.send_keys("selenium")
   element.submit()
   time.sleep(10)
   driver.quit()


th_f1 = threading.Thread(target=firefox)
th_c1 = threading.Thread(target=chrome)
th_f2 = threading.Thread(target=firefox)
th_c2 = threading.Thread(target=chrome)
th_f1.start()
th_c1.start()
th_f2.start()
th_c2.start()
```

time.sleep(10) is added to ensure that four browsers are opened at the same time for a while.

The following figure shows the Selenium Grid console during code execution. We can see that the four browser icons are thin.

As described above, since Selenium Grid can perform parallel execution, it is possible to shorten the test time.

1-2 Automatic operation method

Selenium WebDriver uses the methods provided by Selenium Client to perform automatic operation of browser. The methods are colored in red and blue in the following code.

google_search_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("http://www.google.com") element = driver.find_element_by_id("lst-ib") element.send_keys("selenium") element.submit()
```

The methods consist of WebDriver methods and WebElement methods. The main ones are shown below.

Commonly used WebDriver methods

get(self, url)	Loads a web page in the current browser session
back(self)	Goes one step backward in the browser history
forward(self)	Goes one step forward in the browser history
refresh(self)	Refreshes the current page
set_window_size(self, width, height, windowHandle='current')	Sets the width and height of the current window
maximize_window(self)	Maximizes the current window that webdriver is using
close(self)	Closes the current window
quit(self)	Quits the driver and closes every associated window
current_url(self)	Gets the URL of the current page
title(self)	Returns the title of the current page
page_source(self)	Gets the source of the current page
implicitly_wait(self, time_to_wait)	Sets a sticky timeout to implicitly

	wait for an element to be found, or a command to complete
save_screenshot(self, filename)	Saves a screenshot of the current window to a PNG image file
find_element_by_id(self, id_) find_element_by_xpath(self, xpath) find_element_by_link_text(self, link_text) find_element_by_partial_link_text(self, link_text) find_element_by_name(self, name) find_element_by_tag_name(self, name) find_element_by_class_name(self, name) find_element_by_css_selector(self, css_selector)	Search the page and return the first found element (WebElement)
driver.switch_to.frame(self, frame_reference)	Switches focus to the specified frame
driver.switch_to.default_content(self)	Switch focus to the default frame
execute_script(self, script, *args)	Executes JavaScript in the current window/frame. (In this book, it is used by changing text and scrolling)

Commonly used WebElement methods

click(self)	Clicks the element
send_keys(self, *value)	Simulates typing into the element
clear(self)	Clears the text if it's a text entry element
submit(self)	Submits a form
is_selected(self)	Returns whether the element is selected
is_enabled(self)	Returns whether the element is enabled
is_displayed(self)	Whether the element is visible to a user
text(self)	The text of the element

For details on the methods, please check API document.

WebDriver methods API document

selenium.webdriver.remote.webdriver — Selenium 3.8 documentation - Mozilla Firefox

https://seleniumhq.github.io/selenium/docs/api/py/webdriver_remote/

add_cookie(*cookie_dict*) [source]

Adds a cookie to your current session.

Args : • cookie_dict: A dictionary object, with required keys - "name" and "value"; optional keys - "path", "domain", "secure", "expiry"

Usage:

driver.add_cookie({'name' : 'foo', 'value' : 'bar'}) driver.add_cookie({'name' : 'foo', 'value' : 'bar', 'path' : '/'})
driver.add_cookie({'name' : 'foo', 'value' : 'bar', 'path' : '/', 'secure':True})

application_cache [source]

Returns a ApplicationCache Object to interact with the browser app cache

back() [source]

Goes one step backward in the browser history.

Usage : driver.back()

close() [source]

Closes the current window.

Usage : driver.close()

create_web_element(*element_id*) [source]

Creates a web element with the specified *element_id*.

current_url [source]

Gets the URL of the current page.

Usage : driver.current_url

current_window_handle [source]

Returns the handle of the current window.

Usage : driver.current_window_handle

WebElement methods API document

selenium.webdriver.remote.webelement — Selenium 3.8 documentation - Mozilla Firefox

https://seleniumhq.github.io/selenium/docs/api/py/webdriver_rem

Selenium 3.8 documentation » modules | index

This Page

Show Source

Quick search

Go

Enter search terms or a module, class or function name.

selenium.webdriver.remote.webelement

class selenium.webdriver.remote.webelement.**WebElement**(*parent*, *id_*, *w3c=False*) [source]

Represents a DOM element.

Generally, all interesting operations that interact with a document will be performed through this interface.

All method calls will do a freshness check to ensure that the element reference is still valid. This essentially determines whether or not the element is still attached to the DOM. If this test fails, then an `StaleElementReferenceException` is thrown, and all future calls to this instance will fail.

clear() [source]

Clears the text if it's a text entry element.

click() [source]

Clicks the element.

find_element(*by='id'*, *value=None*) [source]

find_element_by_class_name(*name*) [source]

Finds element within this element's children by class name.

Args : • name - class name to search for.

find_element_by_css_selector(*css_selector*) [source]

Finds element within this element's children by CSS selector.

Args : • css_selector - CSS selctor string, ex: 'a.nav#home'

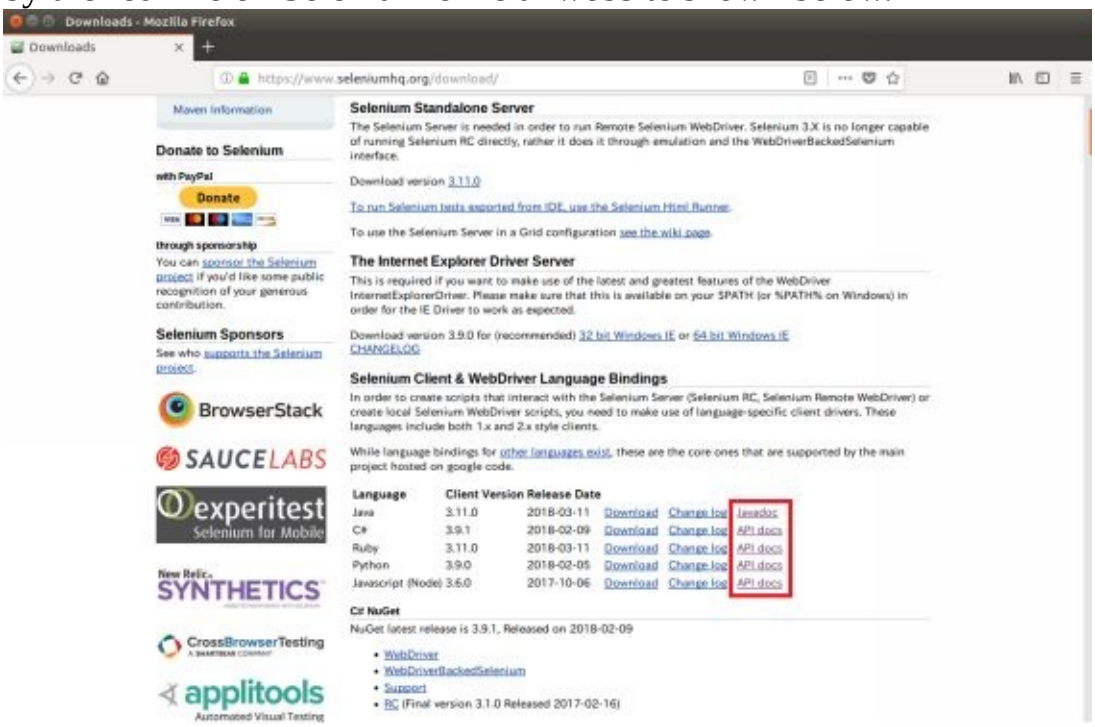
find_element_by_id(*id_*) [source]

Finds element within this element's children by ID.

Args : • id_ - ID of child element to locate.

Note: Selenium Client API document

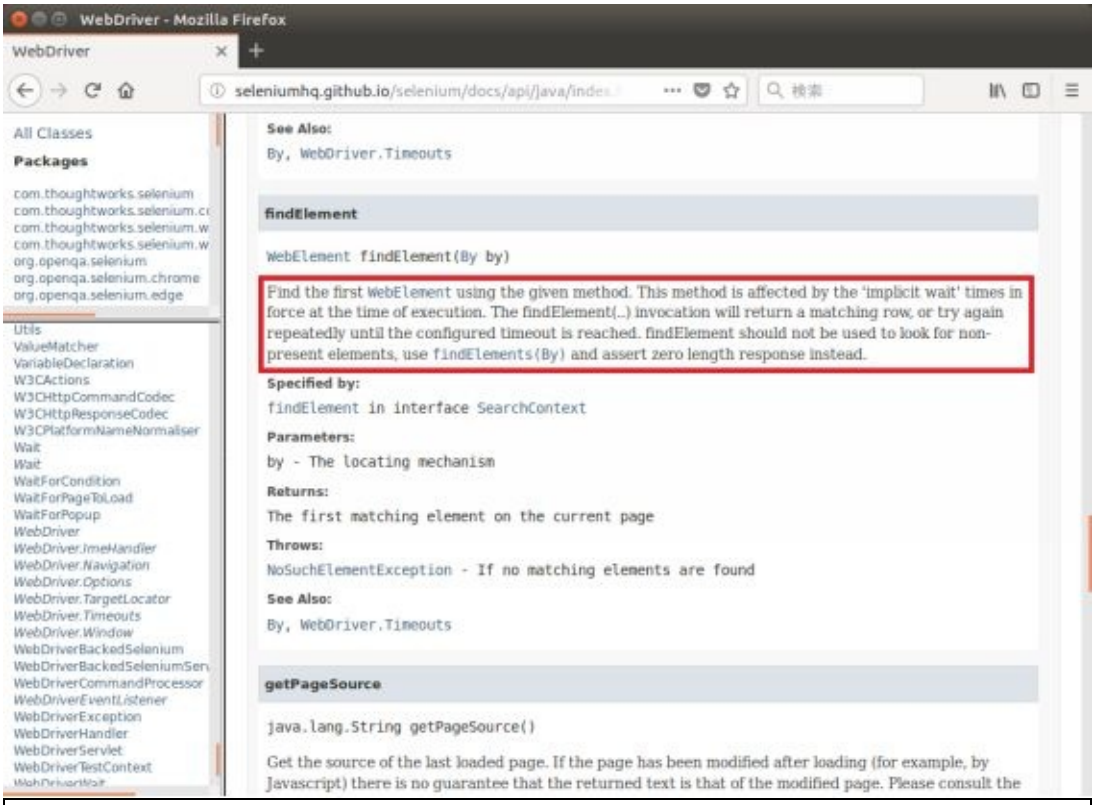
The Selenium Client's API document can be checked from the link surrounded by the red line on Selenium official website shown below.



API document is prepared for each language but the written information is not the same.

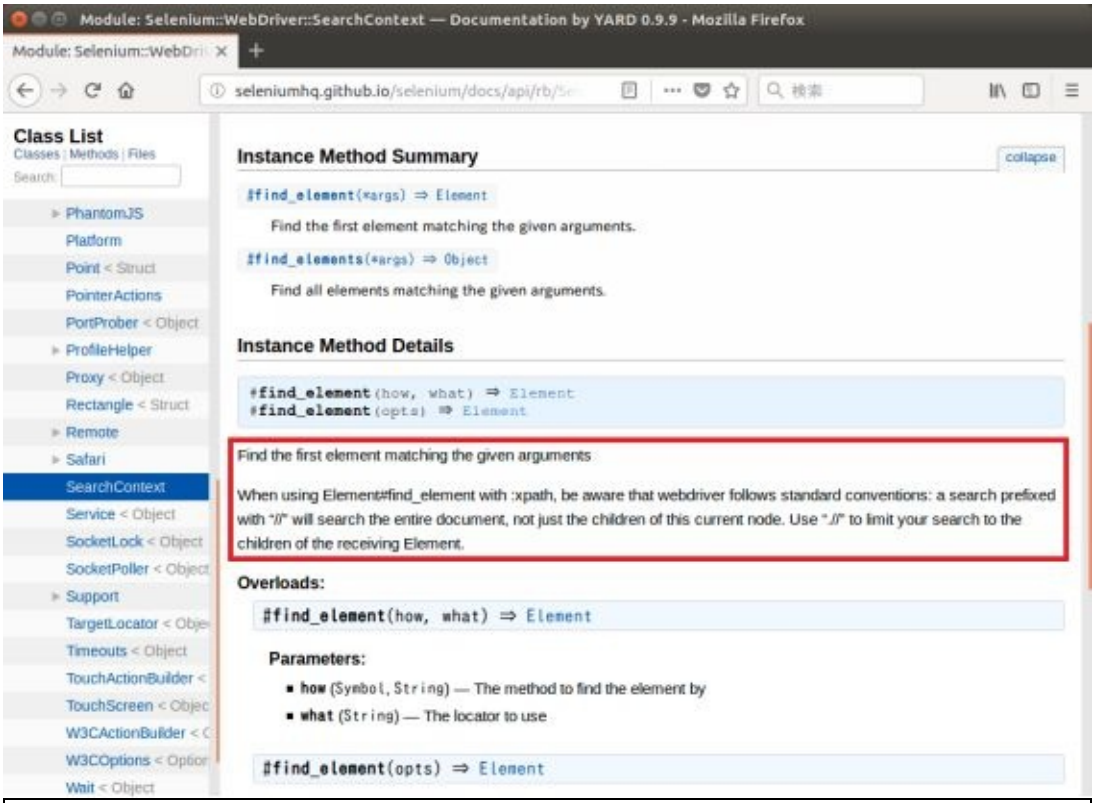
For example, let's check the description of the method that we use most in Selenium WebDriver called "findElement" in Java and Ruby.

Description of the findElement method in Java API document



Find the first WebElement using the given method. See the note in findElements(By) about finding via XPath. This method is affected by the 'implicit wait' times in force at the time of execution. The findElement(..) invocation will return a matching row, or try again repeatedly until the configured timeout is reached. findElement should not be used to look for non-present elements, use findElements(By) and assert zero length response instead.

Description of find_element method in Ruby API document



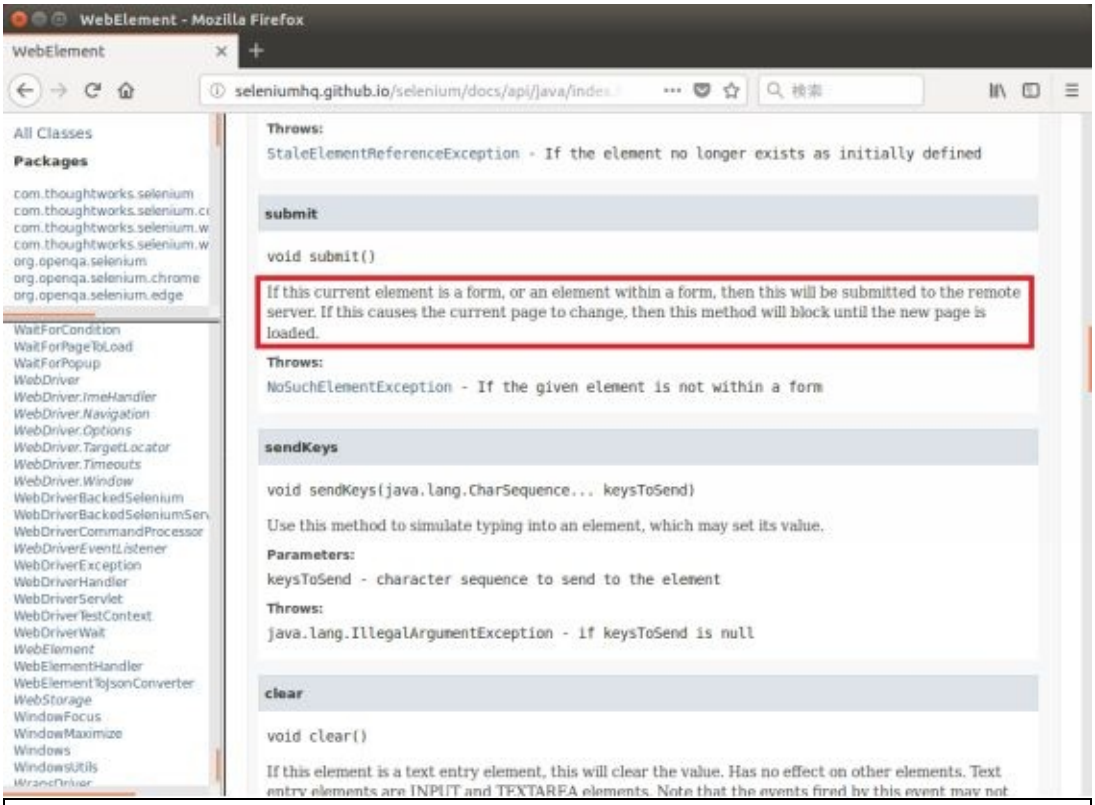
Find the first element matching the given arguments

When using Element#find_element with :xpath, be aware that webdriver follows standard conventions: a search prefixed with "//" will search the entire document, not just the children of this current node. Use ".//" to limit your search to the children of the receiving Element.

In Java document, there are some information about implicit wait, but there is no information in Ruby document.

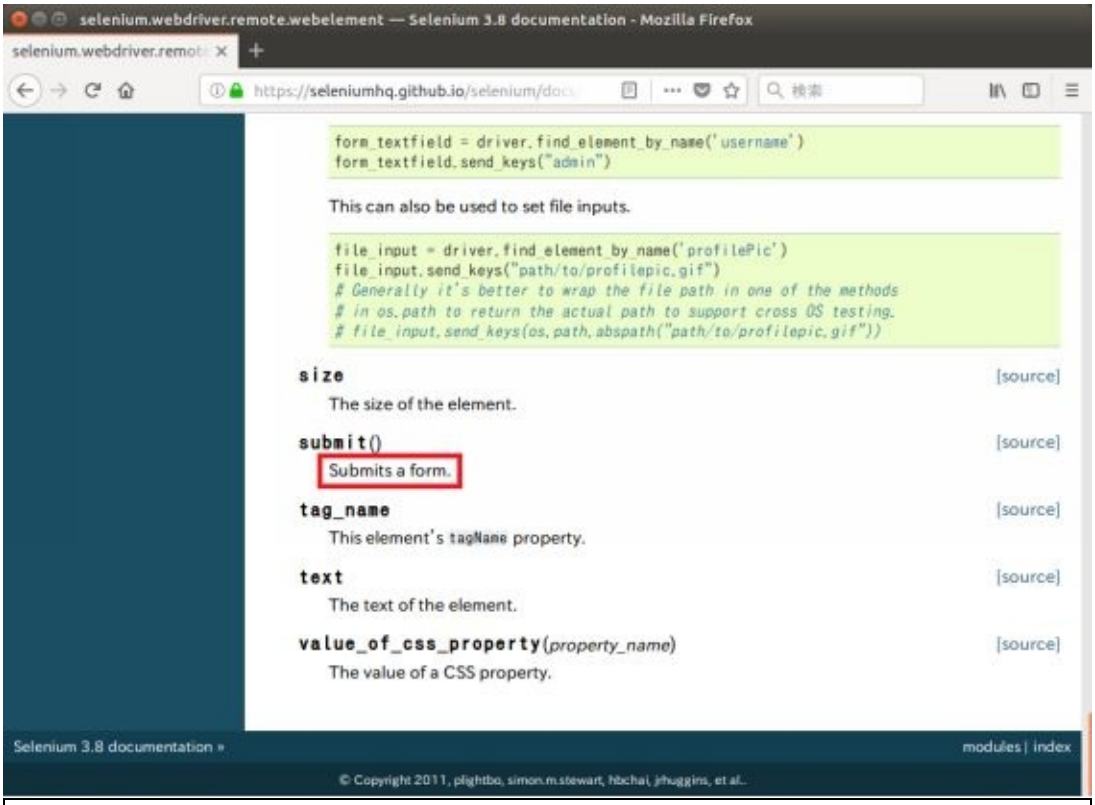
As another example, let's check the method "submit" in Java and Python.

Description of submit method in Java API document



If this current element is a form, or an element within a form, then this will be submitted to the remote server. If this causes the current page to change, then this method will block until the new page is loaded.

Description of submit method in Python API document



Submits a form.

Java document has more information about submit method than Python document.

As mentioned above, the information in the API document is not the same for each programming language.

1-3 How to investigate elements

The red character part of the following code represents the search text box on Google Site.

google_search_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
```

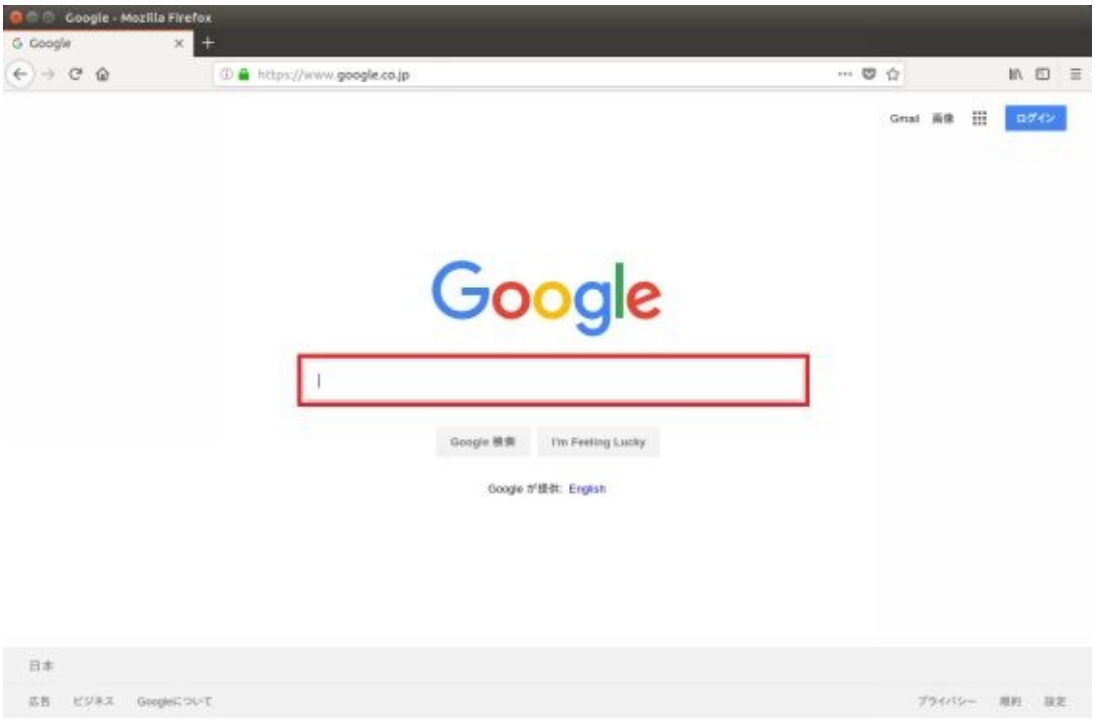
When we check the HTML of the search text box of the Google site, it looks like the following and the information of the red character part of this HTML corresponds to the red character part of the above code.

```
<input class="gsfi" id="lst-ib" maxlength="2048" name="q" autocomplete
haspopup="false" role="combobox" aria-autocomplete="list" style="border:
background:
url(&quot;
transparent; position: absolute; z-index: 6; left: 0px; outline: none;" dir="ltr" spe
```

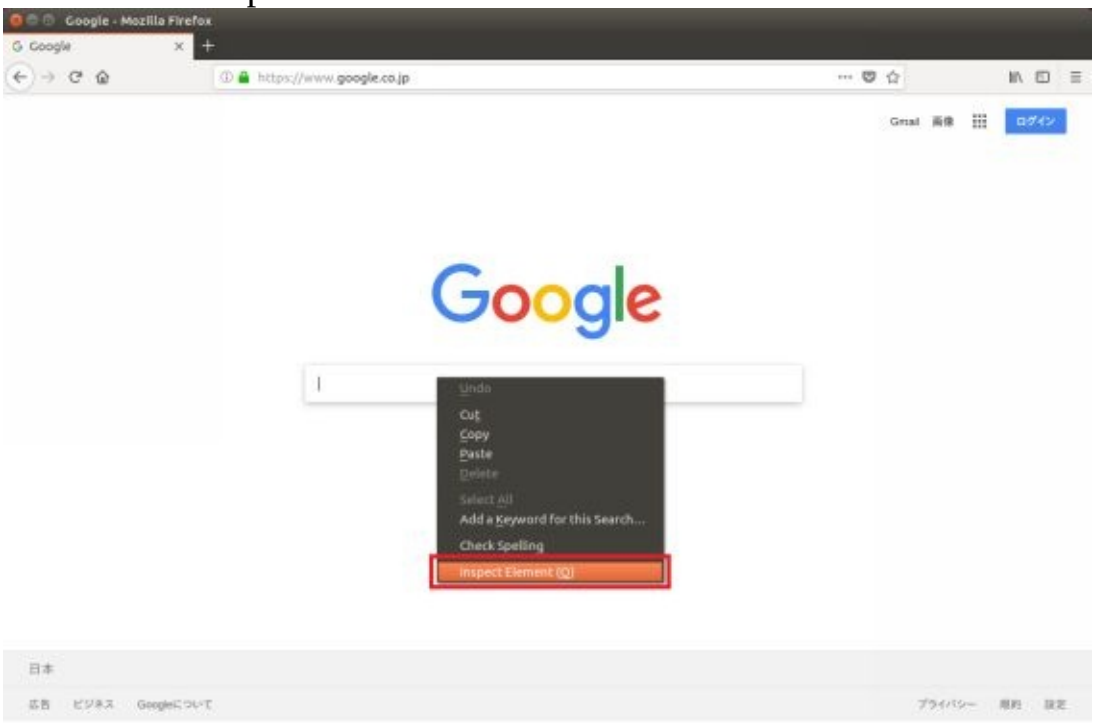
Here, we will learn how to investigate HTML of operation target element in Mozilla Firefox and Google Chrome.

1-3-1 How to investigate elements on Mozilla Firefox

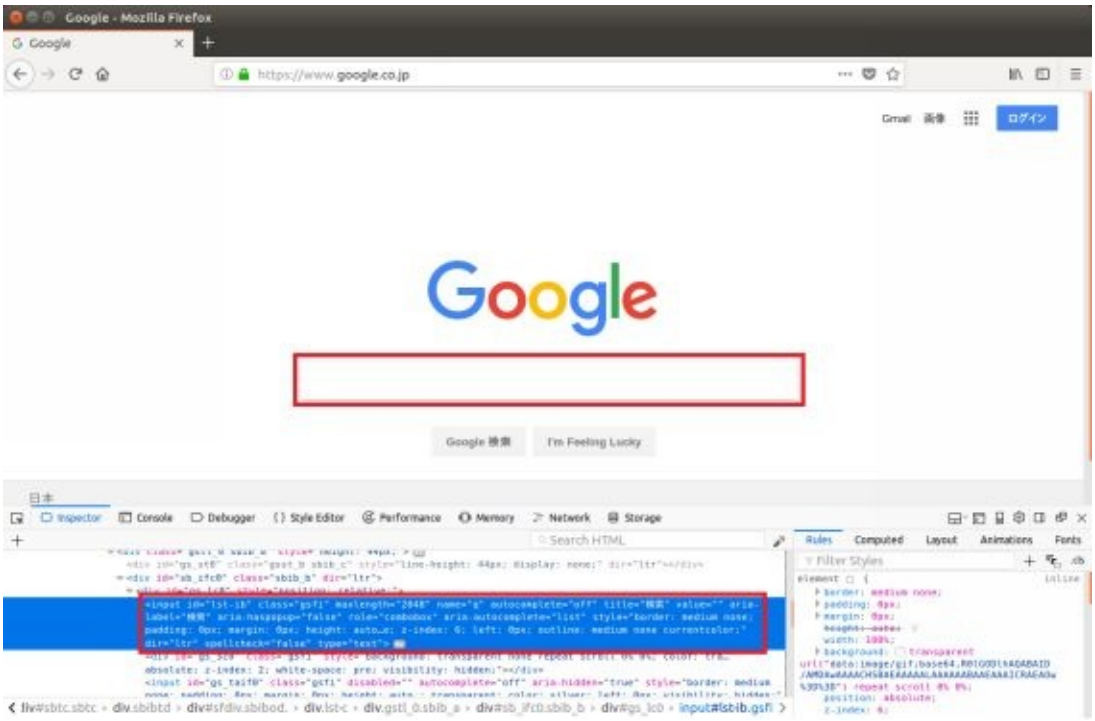
Suppose we want to investigate the element surrounded by the red line in the figure below.



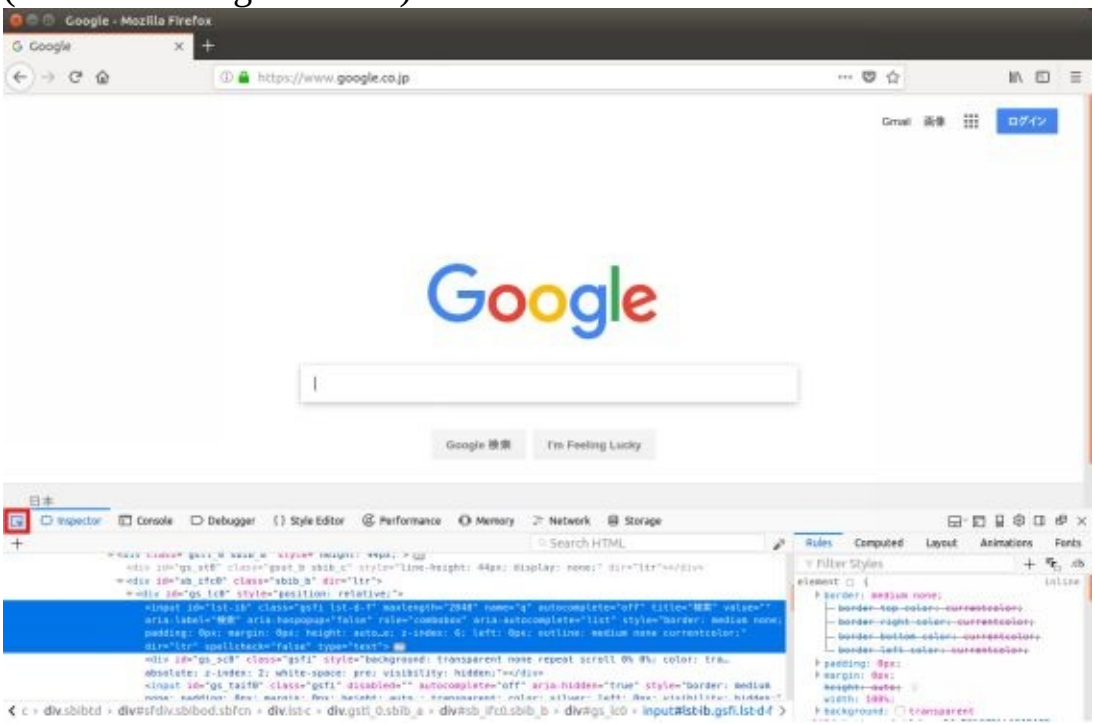
Mouse over the element we want to investigate and right click to open the menu and choose "Inspect Element".



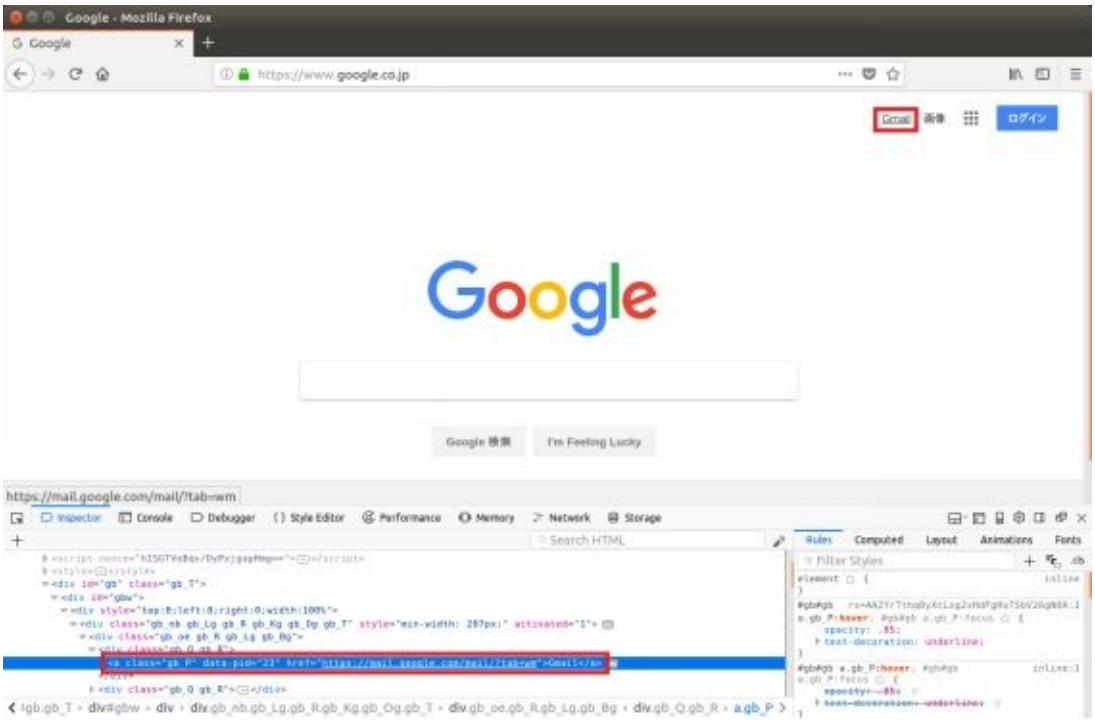
The development tool is displayed at the bottom and we can check the HTML of the surveyed element.



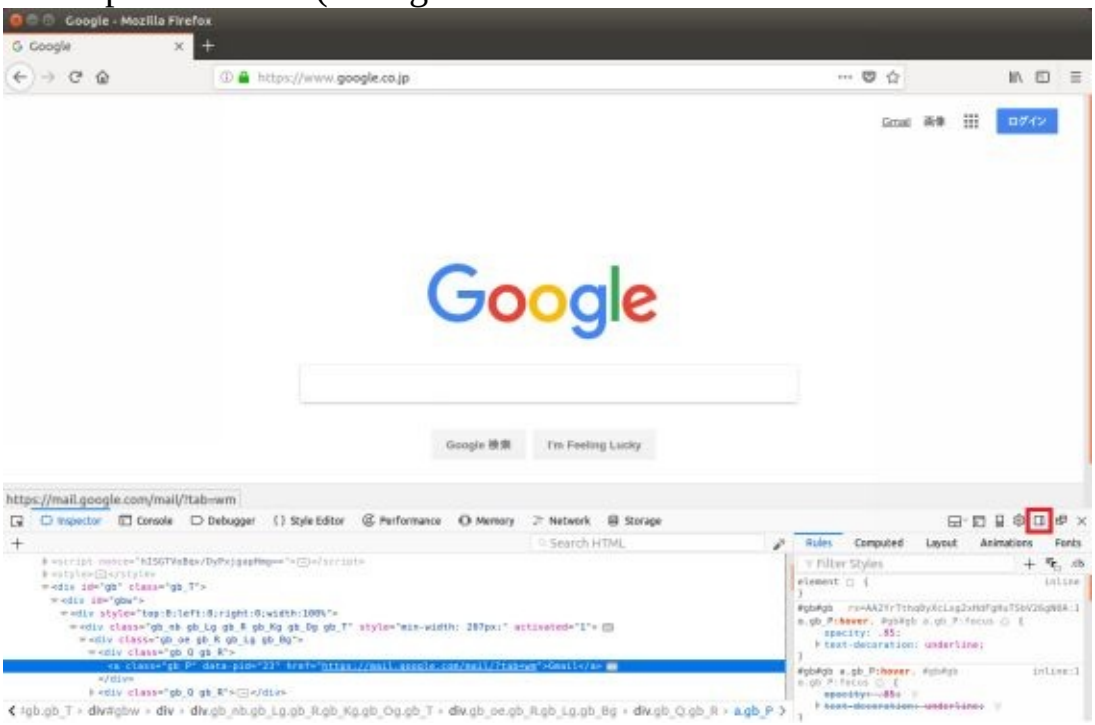
To continue investigating another element, click the "node picker" icon below. (The icon changes to blue)



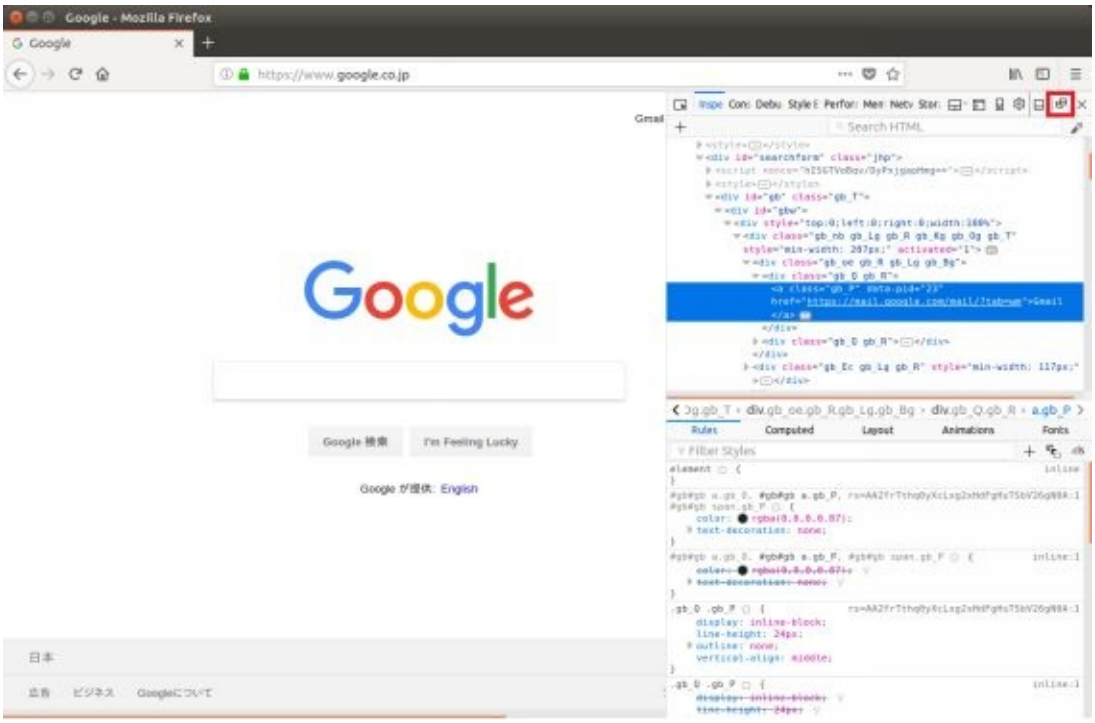
With the icon in blue, we can check the element's HTML by clicking on the element we want to investigate.



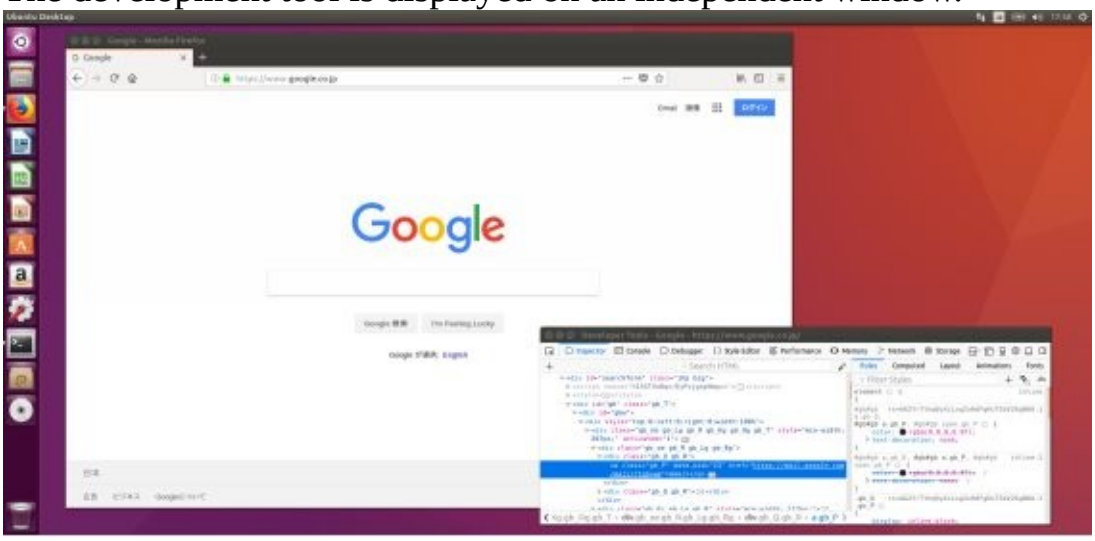
Click the icon in the following figure to change the display position of the development tool. (Change from the bottom of the window to the right)



Click the icon in the following figure to display the development tool in a separate window.

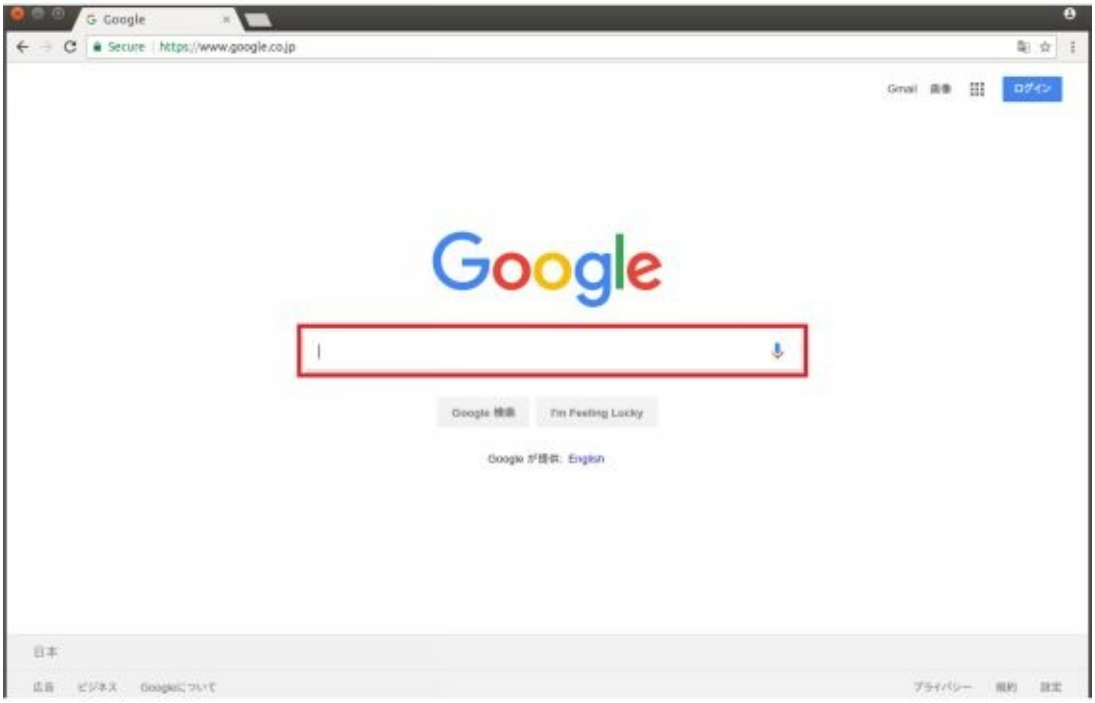


The development tool is displayed on an independent window.

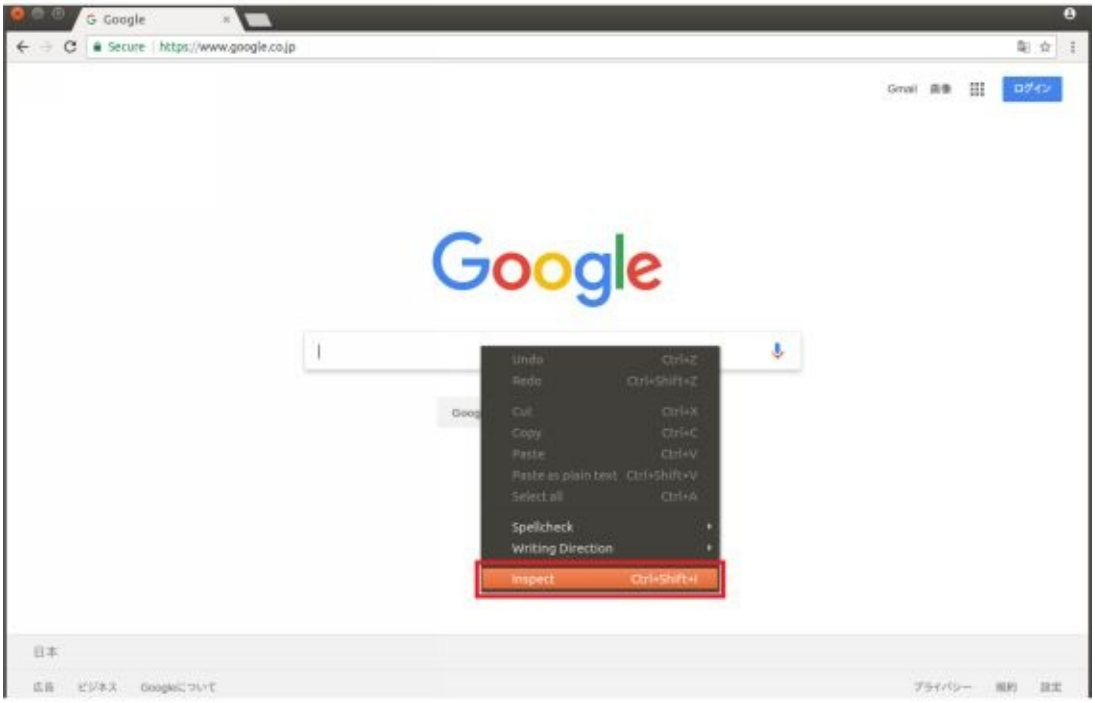


1-3-2 How to investigate elements on Google Chrome

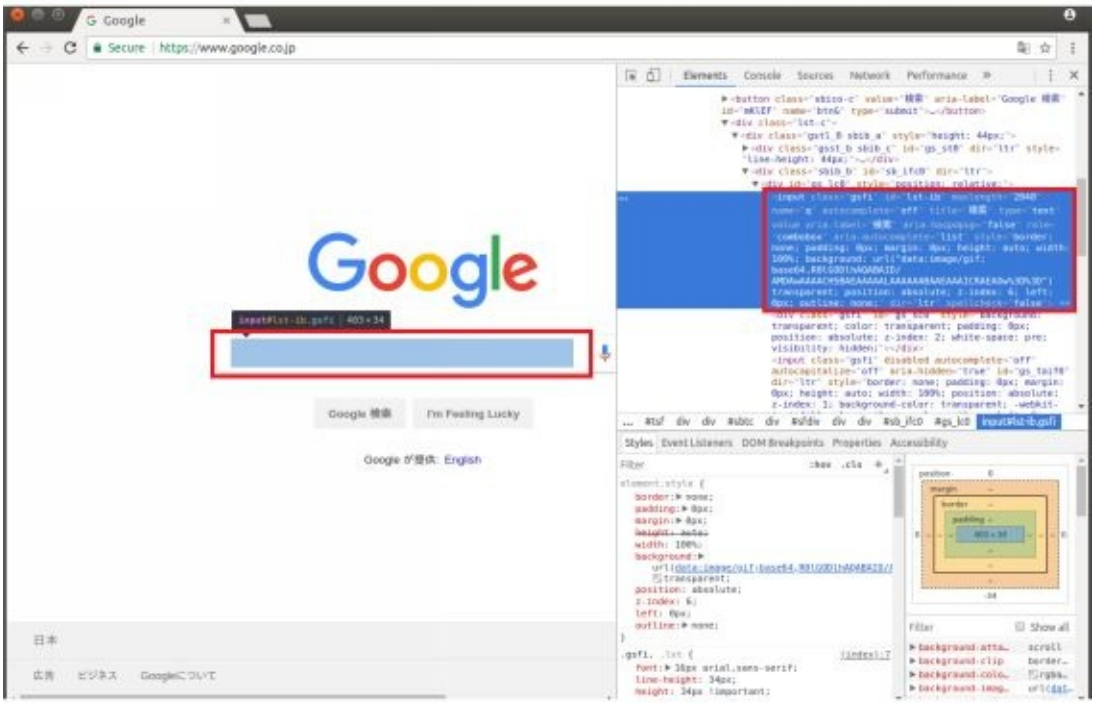
Suppose we want to investigate the element surrounded by the red line in the figure below.



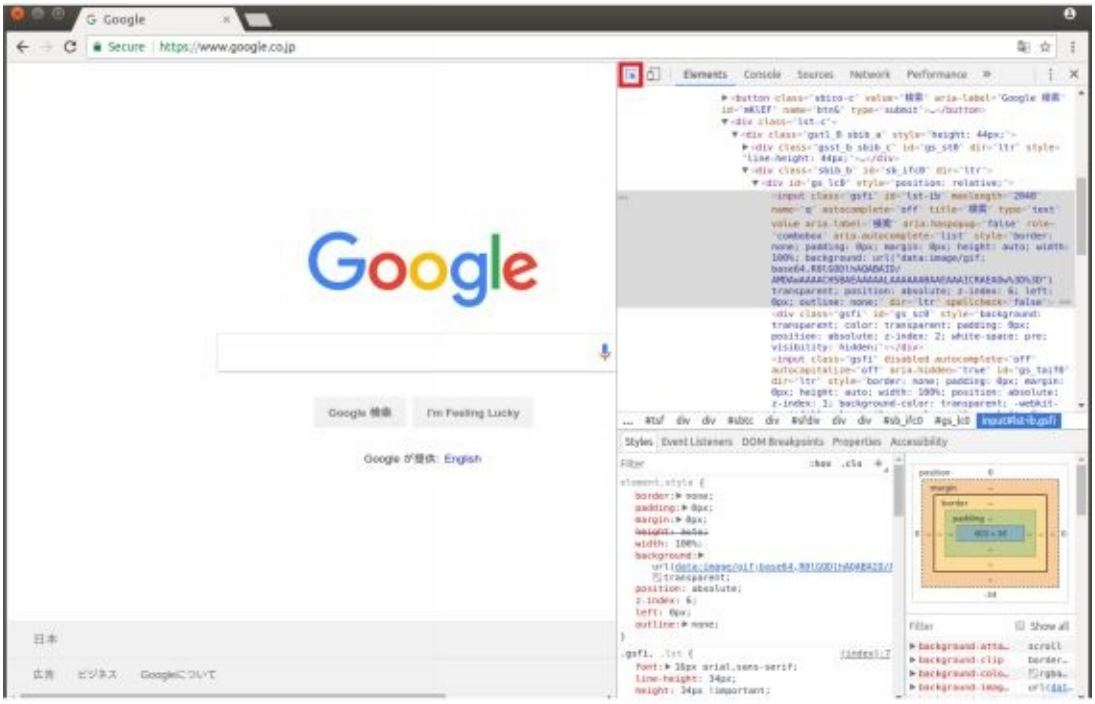
Mouse over the element we want to investigate and right click to open the menu and choose "Inspect".



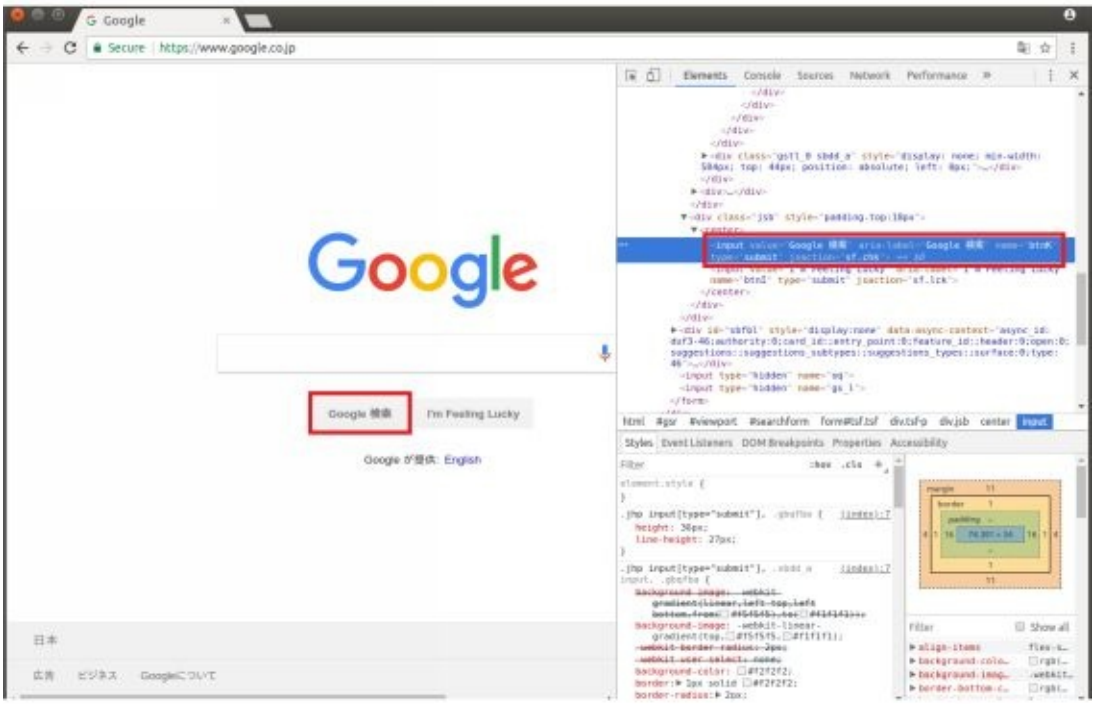
Developer tool is displayed on the right side of the window and we can check the HTML of the surveyed element.



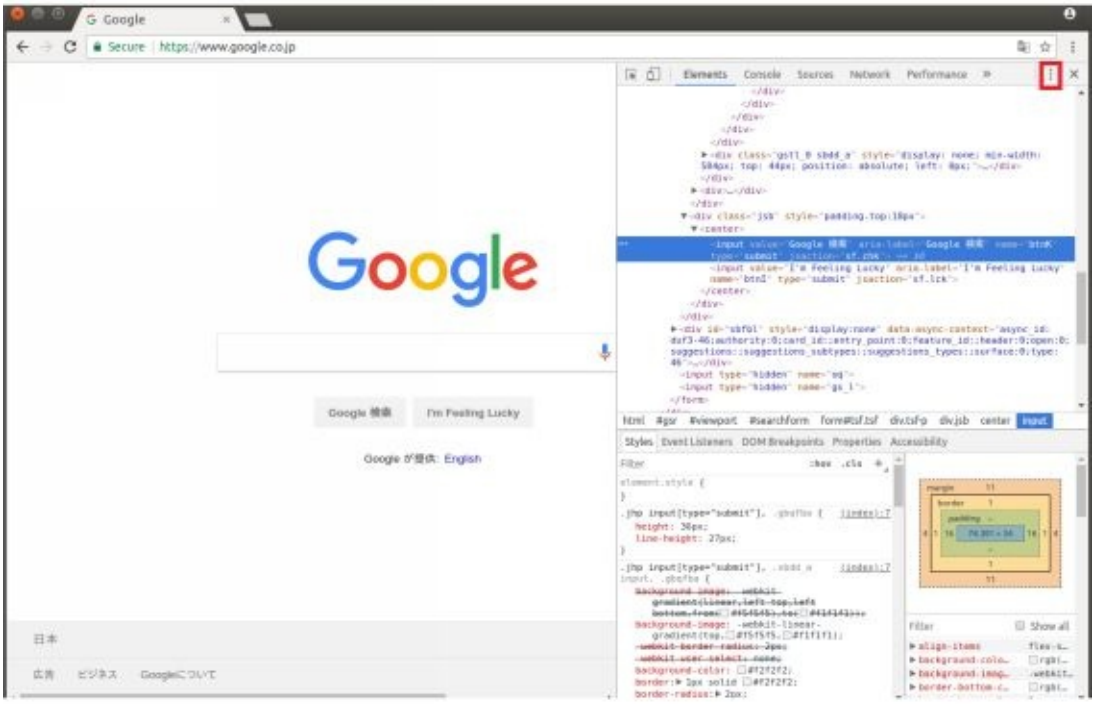
To continue investigating another element, click the "node picker" icon below. (The icon changes to blue)



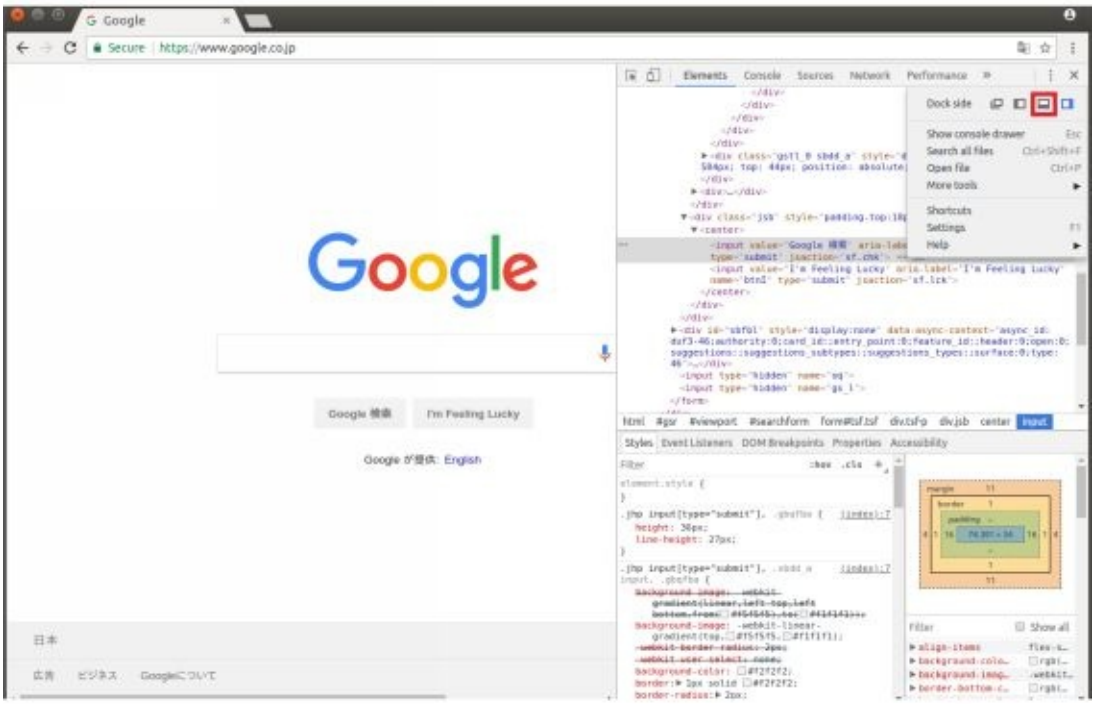
With the icon in blue, we can check the element's HTML by clicking on the element we want to investigate.



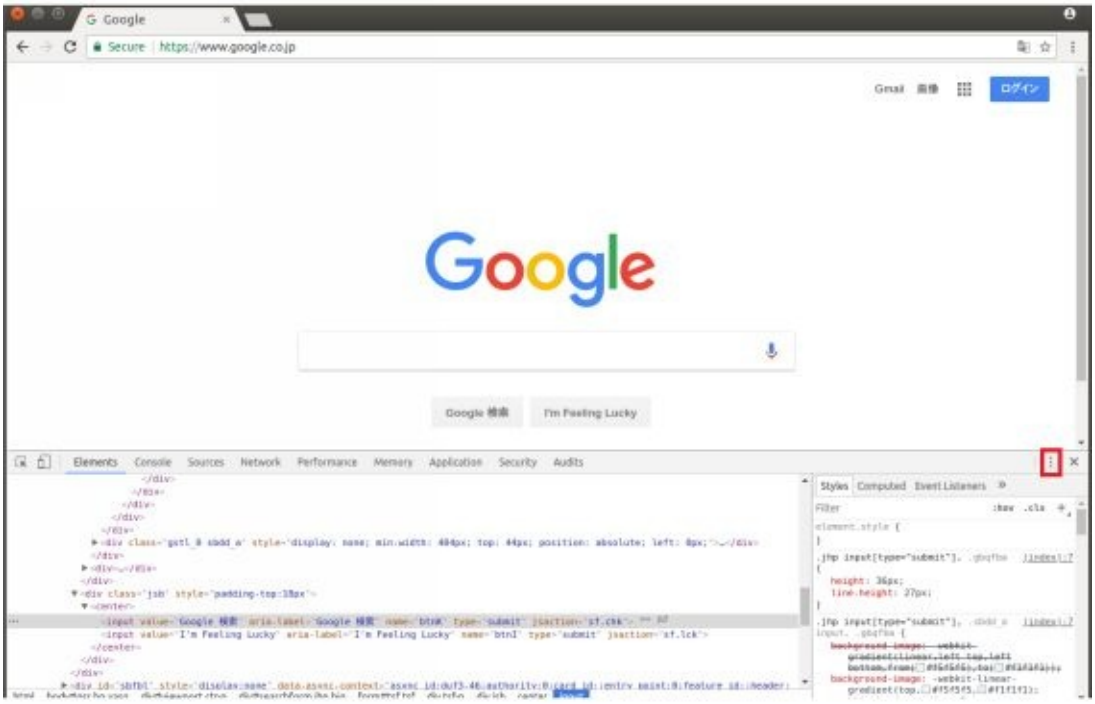
Click the icon shown below to display the menu.



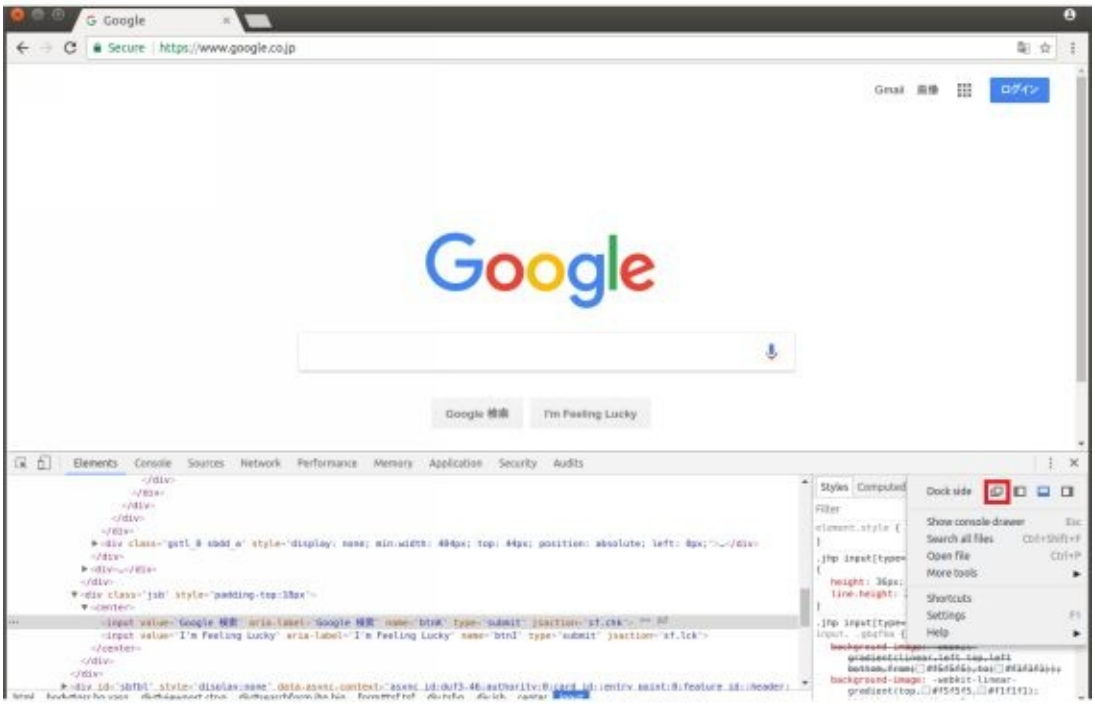
Click the icon shown below to change the display position of the developer tool. (Change from the right side of the window to the bottom)



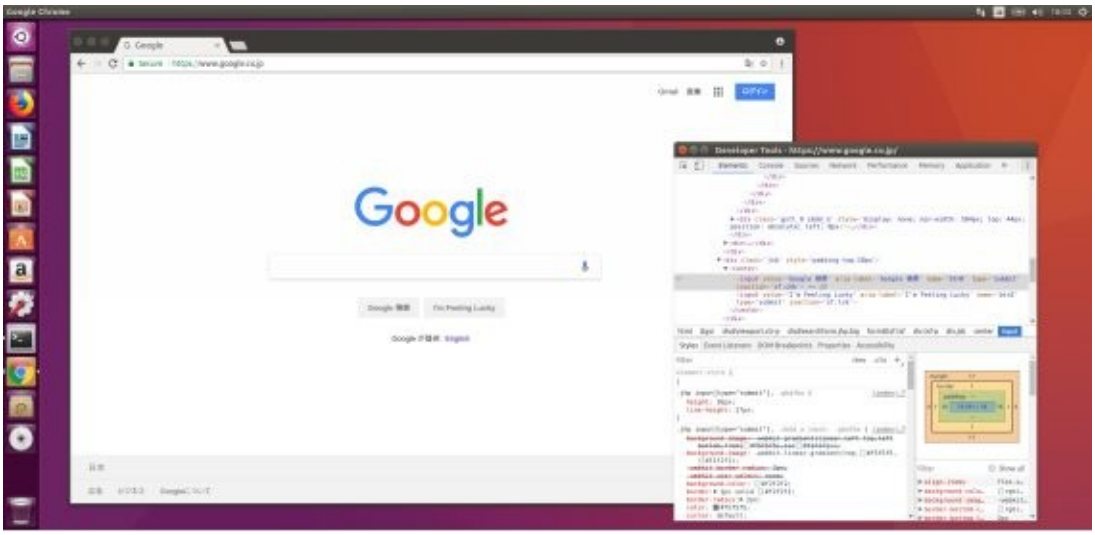
Developer tool is displayed at the bottom of the window. Click the icon in the following figure to display the menu.



Click the icon in the following figure to display the developer tool in a separate window.



Developer tool is displayed on a separate window.


Google Chrome
Google
Developer Tools - https://www.google.co.jp/

1-4 How to identify operation target element

The following code uses id attribute as a way to identify the element.

google_search_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
```

Here, we will learn how to identify operation target element using locators.

1-4-1 Locators in Selenium WebDriver

In Selenium WebDriver we can use the following 8 locators to identify element but in this book we use 4 locators colored in red.

Locate element by id attribute
Locate element by name attribute
Locate element by link text
Locate element by partial link text
Locate element by xpath
Locate element by tag
Locate element by class attribute
Locate element by css selector

4 locators used in this book

id attribute	The way to locate element with a value of id attribute. Since id attribute value is unique within the HTML document, it is best for specifying the element. However, when id attribute changes dynamically (ex: id = "edit-quantity-3"), we can't use id attribute to locate the element, so we need to use other way such as locating by name attribute or xpath.
name	The way to locate element by the value of name

attribute	attribute. The name attribute is used if id attribute does not exist but name exists.
link text	The way to locate element by the value of link text.
xpath	The xpath is used if we can't use id attribute, name attribute and link text to locate element. Although there are various ways of writing xpath, first of all, it would be better to master the following basic 6 patterns and advanced 3 patterns.

Basic 6 patterns of xpath

	//tag[@attribute='value'] e.g. **//input[@name='lastname']**	Locate all input tag element which has name = 'lastname' attribute.
	//tag[text()='value'] e.g. **//option[text()='Jul']**	Locate all option tag element which has text value of 'Jul'.
	//tag[contains(@attribute,'value')] e.g. **//input[contains(@name,'passwd')]**	Locate all input tag element which has name attribute containing 'passwd'.
	//tag[contains(text(),'value']) e.g. **//a[contains(text(),'Facebook')]**	Locate all a tag element which has text value containing 'Facebook'.
	//tag[starts-with(@attribute,'value')] e.g. **//input[starts-with(@id,'html5_')]**	Locate all input tag element which has id attribute starting with 'html5_'.
	//tag[starts-with(text(),'value']) e.g. **//a[starts-with(text(),'Facebook')]**	Locate all a tag element which has text value starting with 'Facebook'.

If an element can't be identified with basic 6 patterns of xpath, we try to identify element using one of the following advanced 3 patterns. Since find_element method returns the first element if element can't be identified as one, we need to write xpath so that the target element is located first.

Advanced 3 patterns of xpath

	and	Identify element by using multiple attribute values or text values.
	hierarchy	Identify element by using parent element's

		information.
	following	Identify element by using the relation between target element and another element which exists before target element.

Example of 1

//a[text()='Add New' and @class='page-title-action']

Locate all elements which meet the following two conditions.

1) a tag element which has text value of 'Add New'

2) a tag element which has class = 'page-title-action' attribute

Example of 2

//select[@id='month']/option[@value='7']

Locate all elements which meet the following two conditions.

1) option tag element which has value = '7' attribute

2) parent element's tag is select and the parent element has id = 'month' attribute

Example of 3

//label[text()='File Upload']//following::a[text()='Google']

Locate all elements which meet the following two conditions.

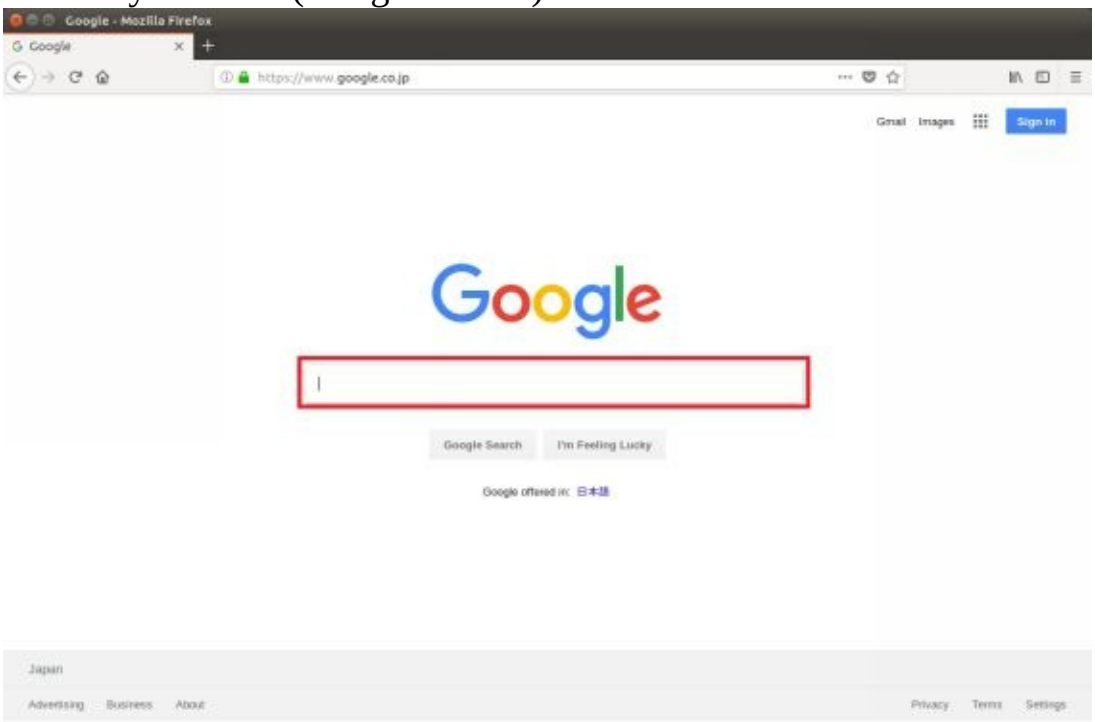
1) a tag element which exists after label tag element which has text value of 'File Upload'

2) a tag element which has text value of 'Google'

1-4-2 Identify element by id attribute/name attribute/link text

Here let's identify text box (Google Search) and link (Gmail) on Google site using id attribute, name attribute and link text.

■Identify text box (Google Search)



HTML of element

```
<input class="gsfi" id="lst-ib" maxlength="2048" name="q" autocom
haspopup="false" role="combobox" aria-autocomplete="list" style="border: me
background:
url(&quot;
repeat scroll 0% 0%; position: absolute; z-index: 6; left: 0px; outline: medium n
```

Since there is an id attribute and a name attribute, the element can be identified as follows.

Locate element by id attribute

```
lst-ib
```

find_element method is used to return an element in the form of WebElement

```
find_element_by_id("lst-ib")
```

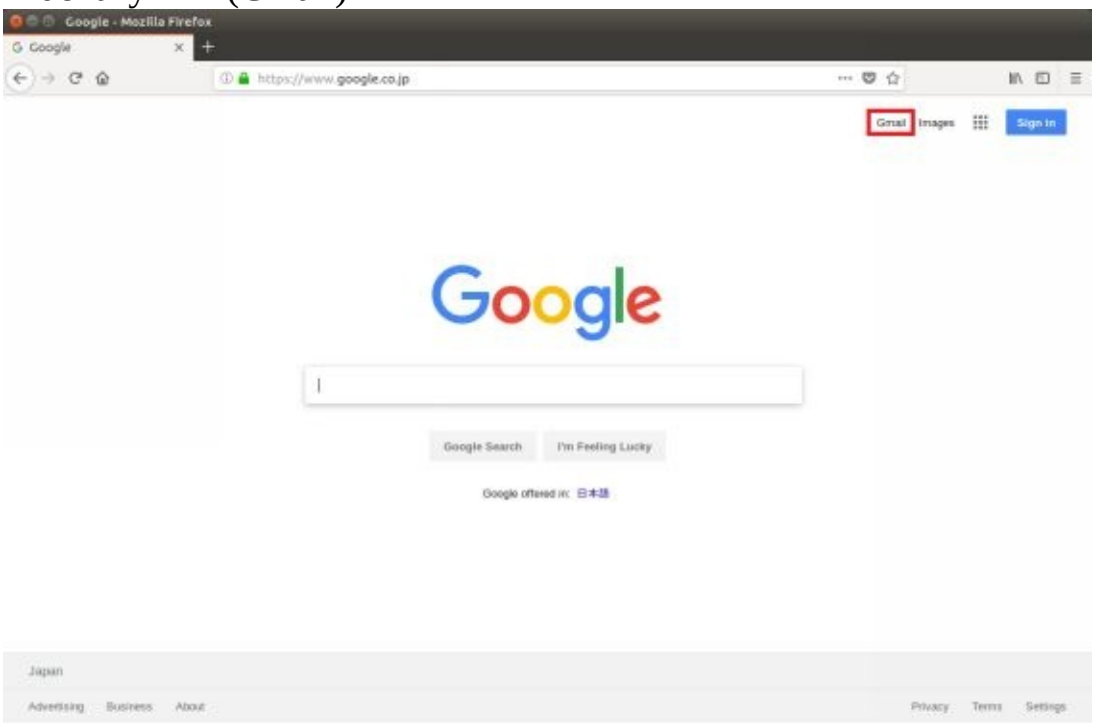
Locate element by name attribute

```
q
```

find_element method is used to return an element in the form of WebElement

```
find_element_by_name("q")
```

■Identify link (Gmail)



HTML of element

```
<a class="gb_P" data-pid="23" href="https://mail.google.com/mail/?tab=wm">Gmail</a>
```

Since there is a text of the link, the element can be identified as follows.

Locate element by link text

```
Gmail
```

find_element method is used to return an element in the form of WebElement

```
find_element_by_link_text("Gmail")
```

1-4-3 Identify element by xpath

Here let's identify each element on Facebook site using xpath.

After determing xpath, find_element method is used to return an element in the form of WebElement.

```
find_element_by_xpath("write xpath here")
```

■Identify text box (First name)

HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="firstname" aria-required="1" placeholder="" aria-label="First name" id="u_0_e" type="text">
```

We can identify the element as follows.

Locate element by xpath

```
//input[@name='firstname']
```

Locate element by xpath (Another xpath)

```
//input[@aria-label='First name']
```

■Identify text box (Last name)

HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="lastname" aria-required="1" placeholder="" aria-label="Last name" id="u_0_g" type="text">
```

We can identify the element as follows.

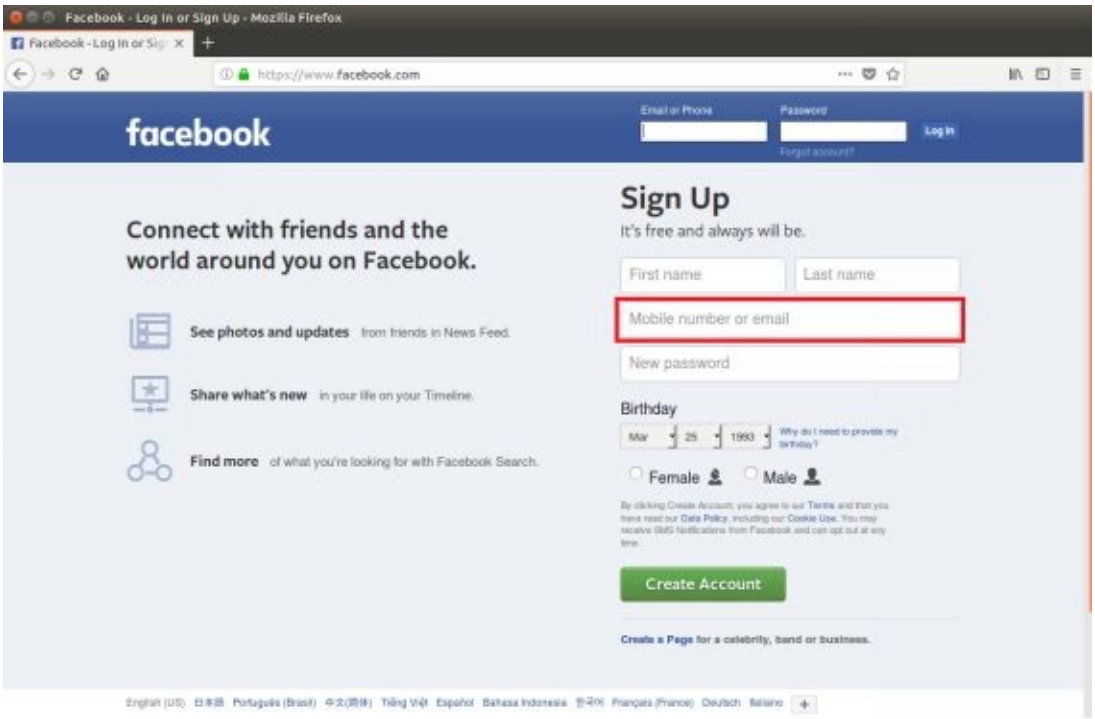
Locate element by xpath

```
//input[@name='lastname']
```

Locate element by xpath (Another xpath)

```
//input[@aria-label='Last name']
```

■Identify text box (Mobile number or email)



HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="reg_email__" aria-required="1" placeholder="" aria-label="Mobile number or email" id="u_0_j" type="text">
```

We can identify the element as follows.

Locate element by xpath

```
//input[@aria-label='Mobile number or email']
```

■Identify text box (New password)

HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="password" autocomplete="new-password" name="reg_passwd__" aria-required="1" placeholder="" aria-label="New password" id="u_0_q" type="password">
```

We can identify the element as follows.

Locate element by xpath

```
//input[@autocomplete='new-password']
```

Locate element by xpath (Another xpath)

```
//input[contains(@name,'passwd')]
```

Locate element by xpath (Another xpath)

```
//input[@aria-label='New password']
```

■Identify dropdown menu option (Jul)

HTML of element

```
<select aria-label="Month" name="birthday_month" id="month"
title="Month" class="_5dba"> <option value="0">Month</option>
<option value="1">Jan</option>
<option value="2">Feb</option>
<option value="3" selected="1">Mar</option>
<option value="4">Apr</option>
<option value="5">May</option>
<option value="6">Jun</option>
<option value="7">Jul</option> <option value="8">Aug</option>
<option value="9">Sep</option>
<option value="10">Oct</option>
<option value="11">Nov</option>
<option value="12">Dec</option>
</select>
```

We can identify the element as follows.

Locate element by xpath

```
//option[text()='Jul']
```

Locate element by xpath (Another xpath)

```
//select[@id='month']/option[@value='7']
```

■Identify radio button (Female)

HTML of element

```
<input name="sex" value="1" id="u_0_6" type="radio">
```

We can identify the element as follows.

Locate element by xpath

```
//input[@value='1']
```

■Identify radio button (Male)

HTML of element

```
<input name="sex" value="2" id="u_0_7" type="radio">
```

We can identify the element as follows.

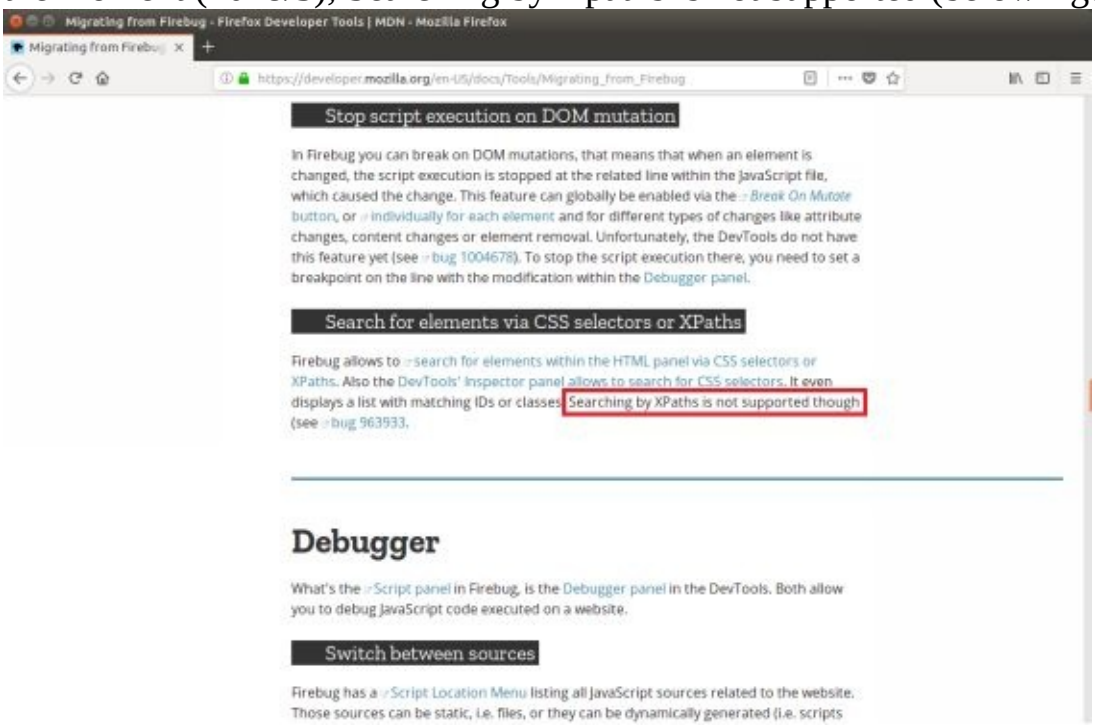
Locate element by xpath

```
//input[@value='2']
```

1-4-4 How to check xpath

In earlier versions of Firefox, installing Firebug and FirePath add-ons made it easier to check xpath but those add-ons are no longer available in the current version.

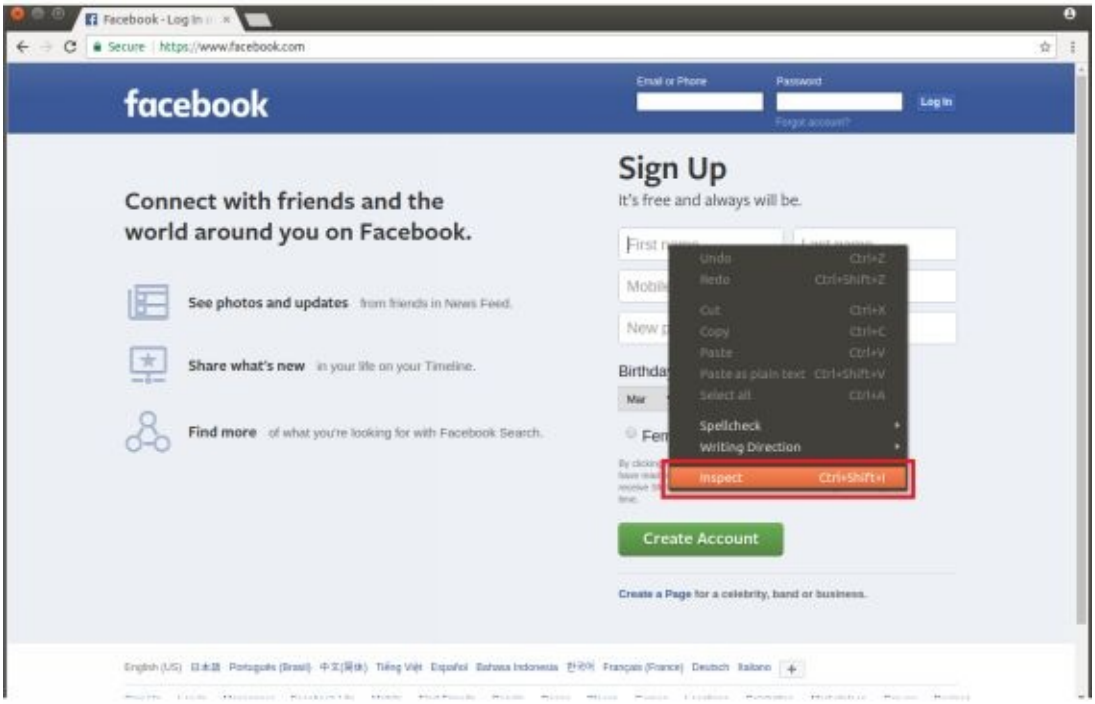
Although Firebug's functionality has been migrated to Firefox, unfortunately, at the moment (2018/3), Searching by Xpaths is not supported (below figure).



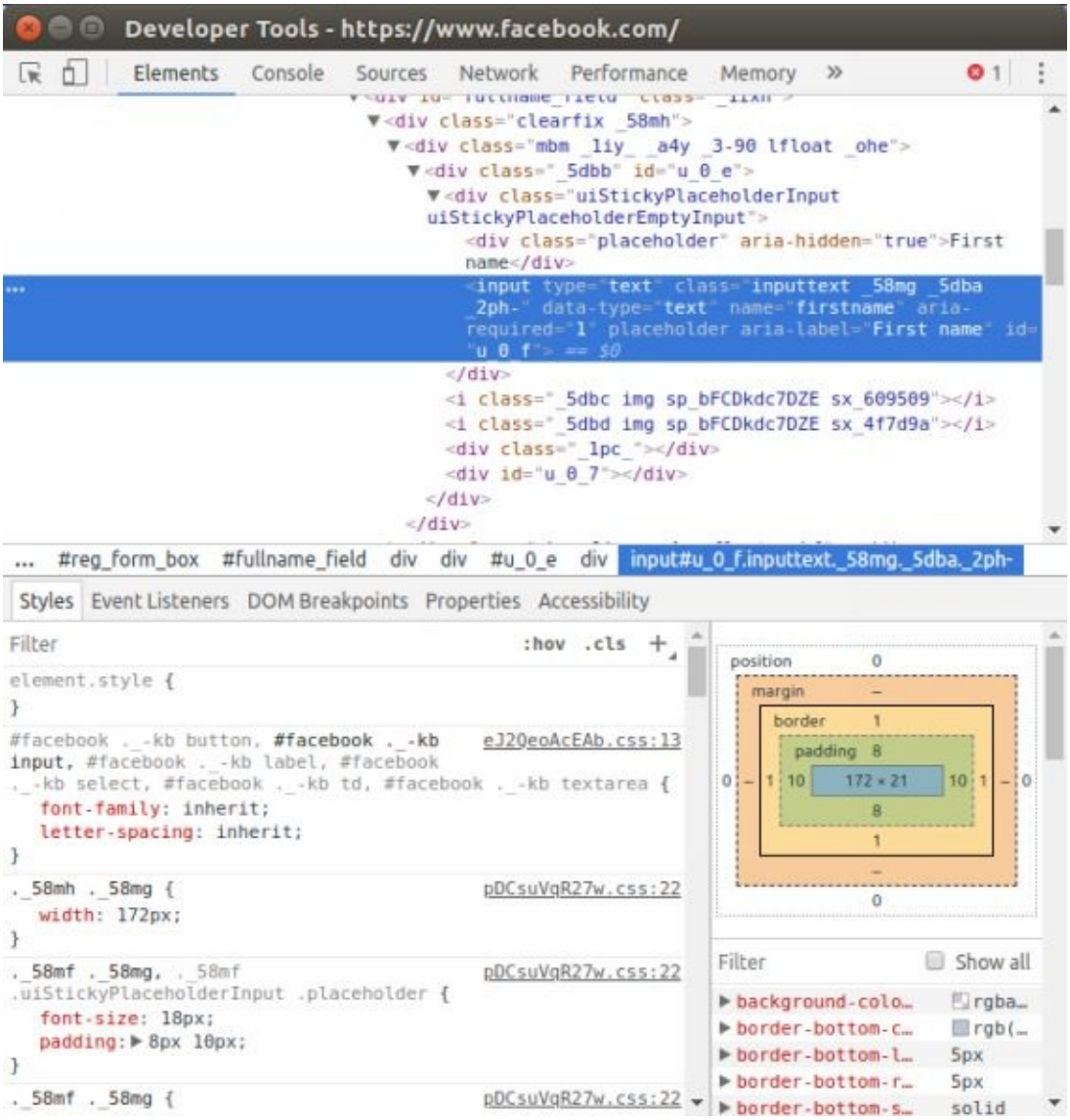
https://developer.mozilla.org/en-US/docs/Tools/Migrating_from_Firebug

So, here we check xpath using the feature of Chrome's developer tool.

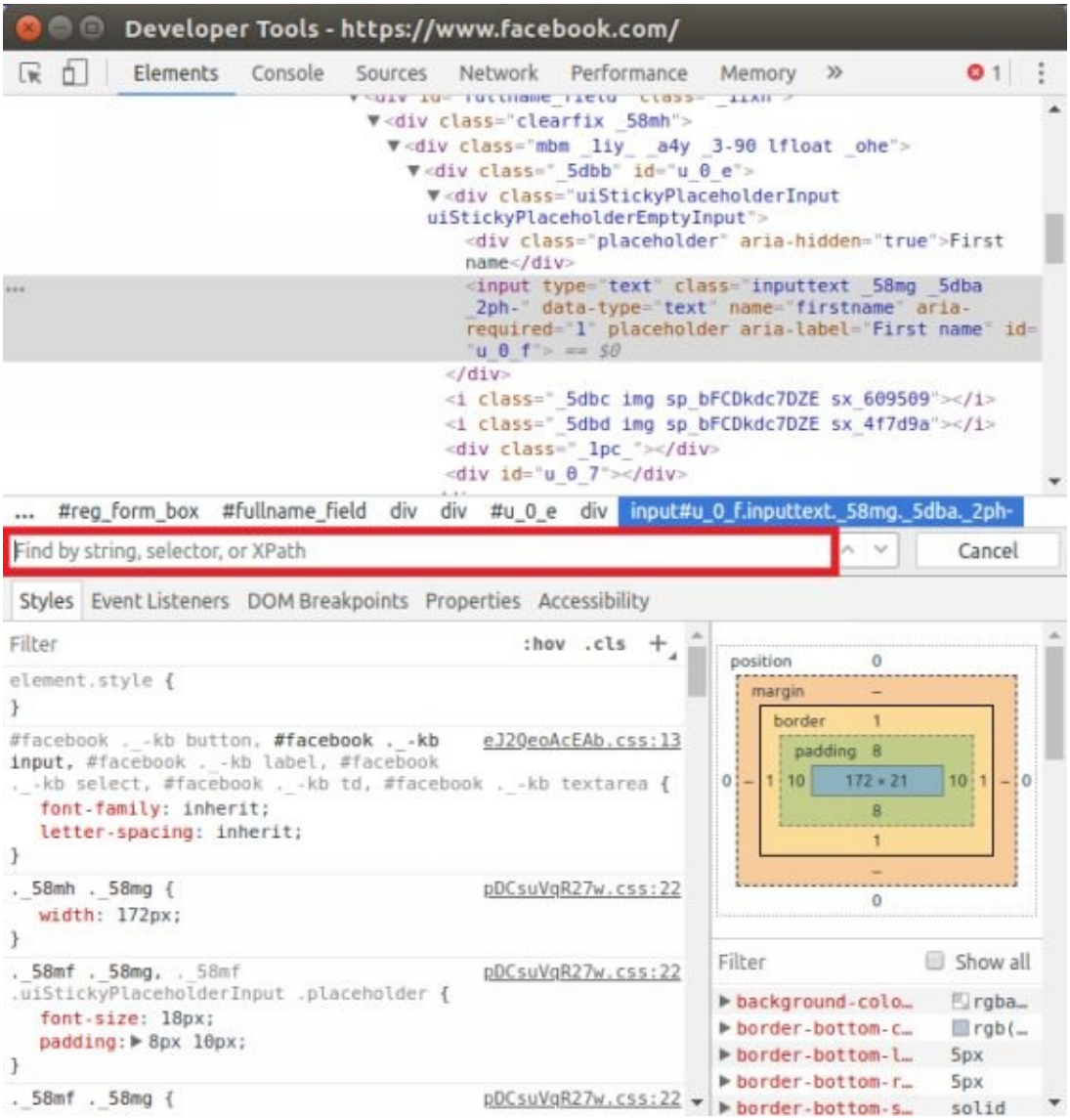
To display the developer tool, right-click on an element and select "Inspect" from the displayed menu.



The developer tool is displayed.



If we press "Ctrl + F" while the developer tool is displayed, search box is displayed. We can check xpath by entering xpath in this search box.



Suppose we want to identify text box (First name).

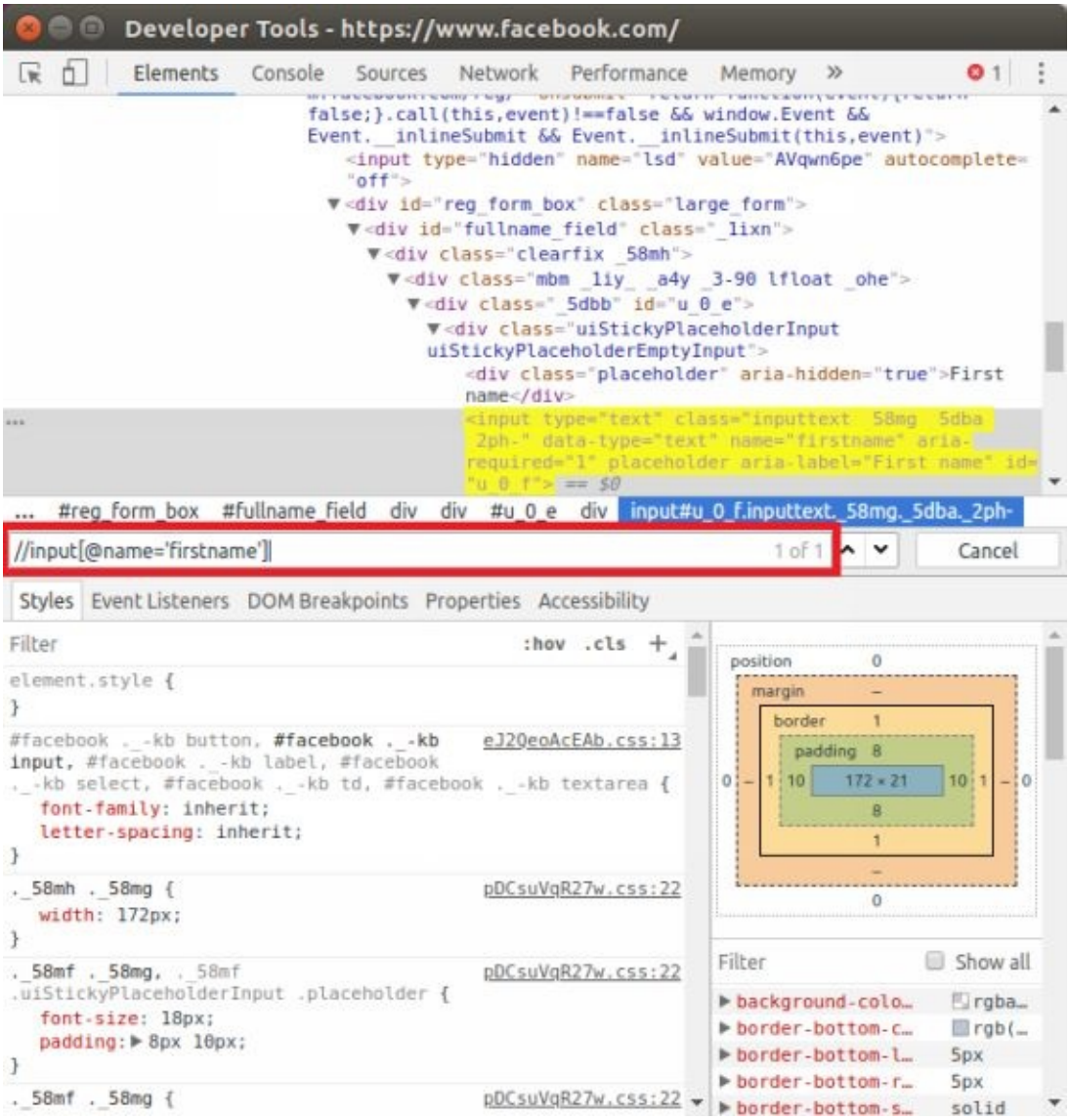
HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="firstname" aria-required="1" placeholder="" aria-label="First name" id="u_0_e" type="text">
```

Enter the following xpath in the search box.

```
//input[@name='firstname']
```

The search result is shown below.

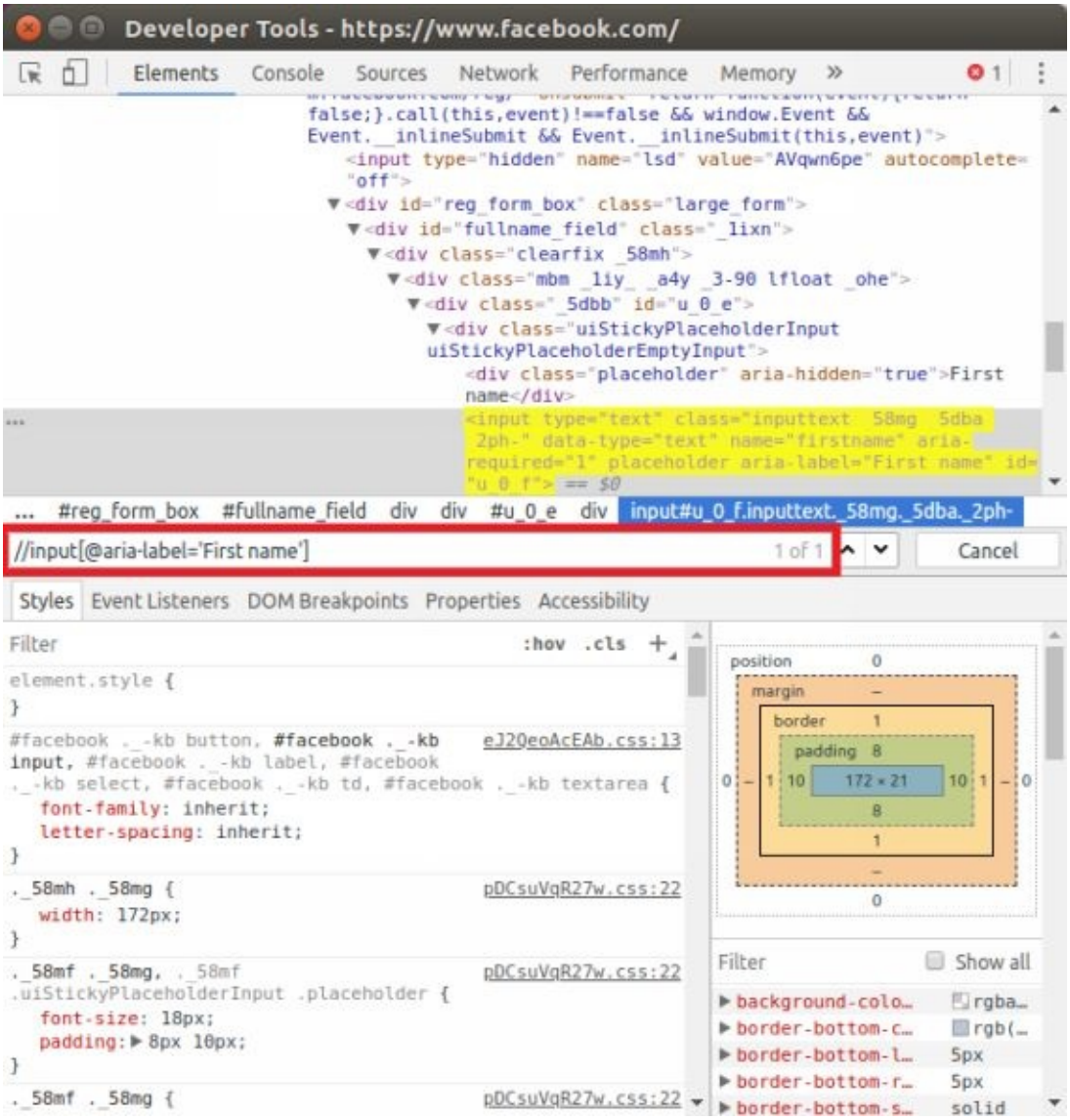


Since the result says "1 of 1", we can see that the element can be identified as one.

Enter the following xpath in the search box.

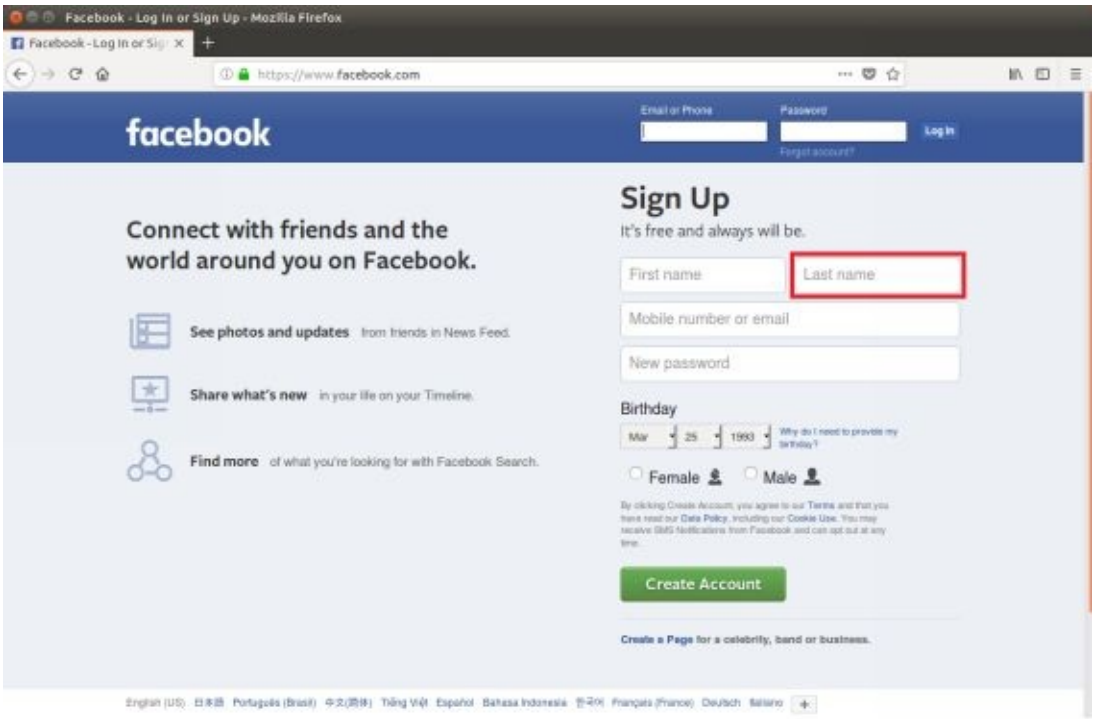
```
//input[@aria-label='First name']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify text box (Last name).



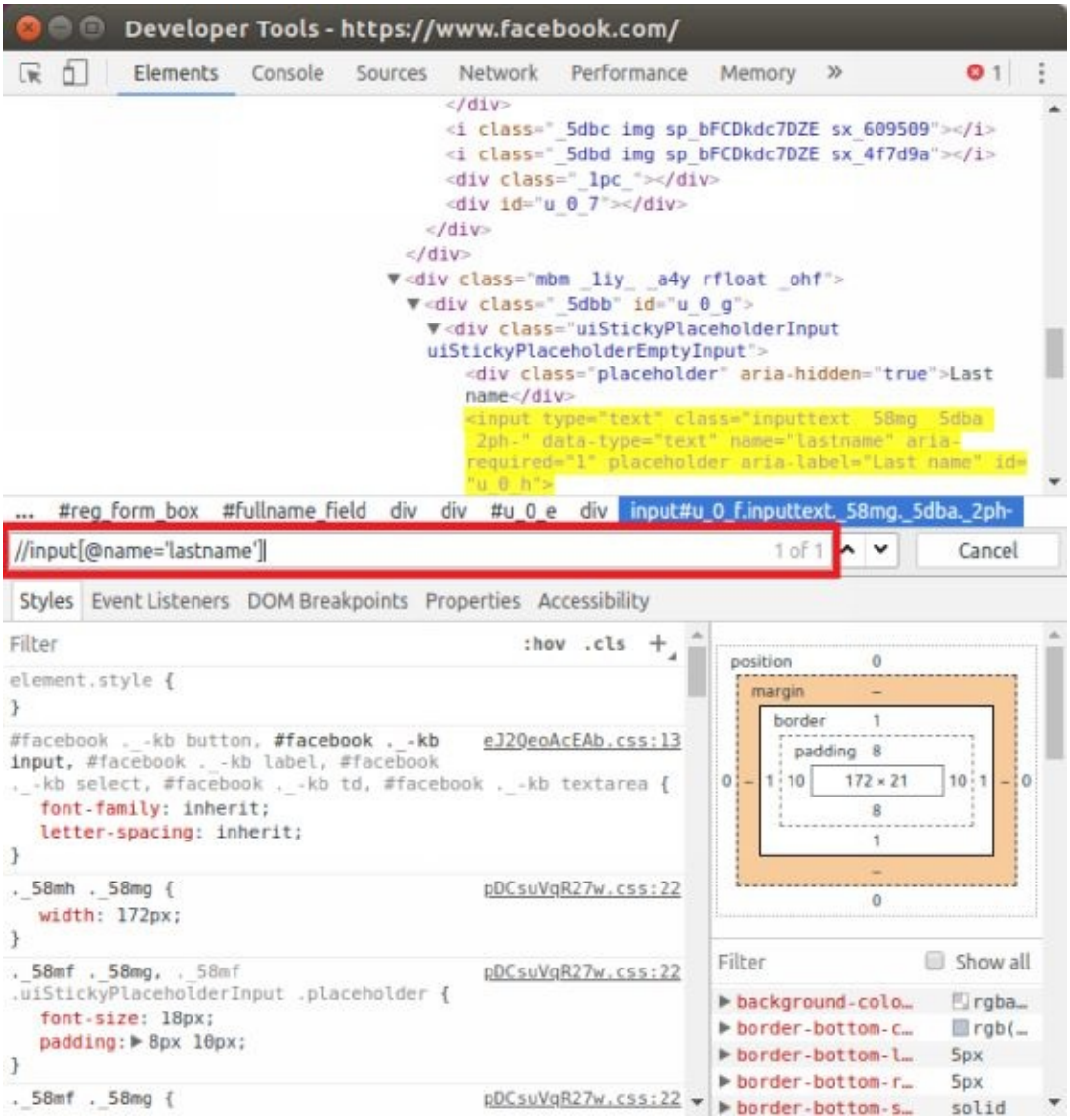
HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="lastname" aria-required="1" placeholder="" aria-label="Last name" id="u_0_g" type="text">
```

Enter the following xpath in the search box.

```
//input[@name='lastname']
```

The search result is shown below.

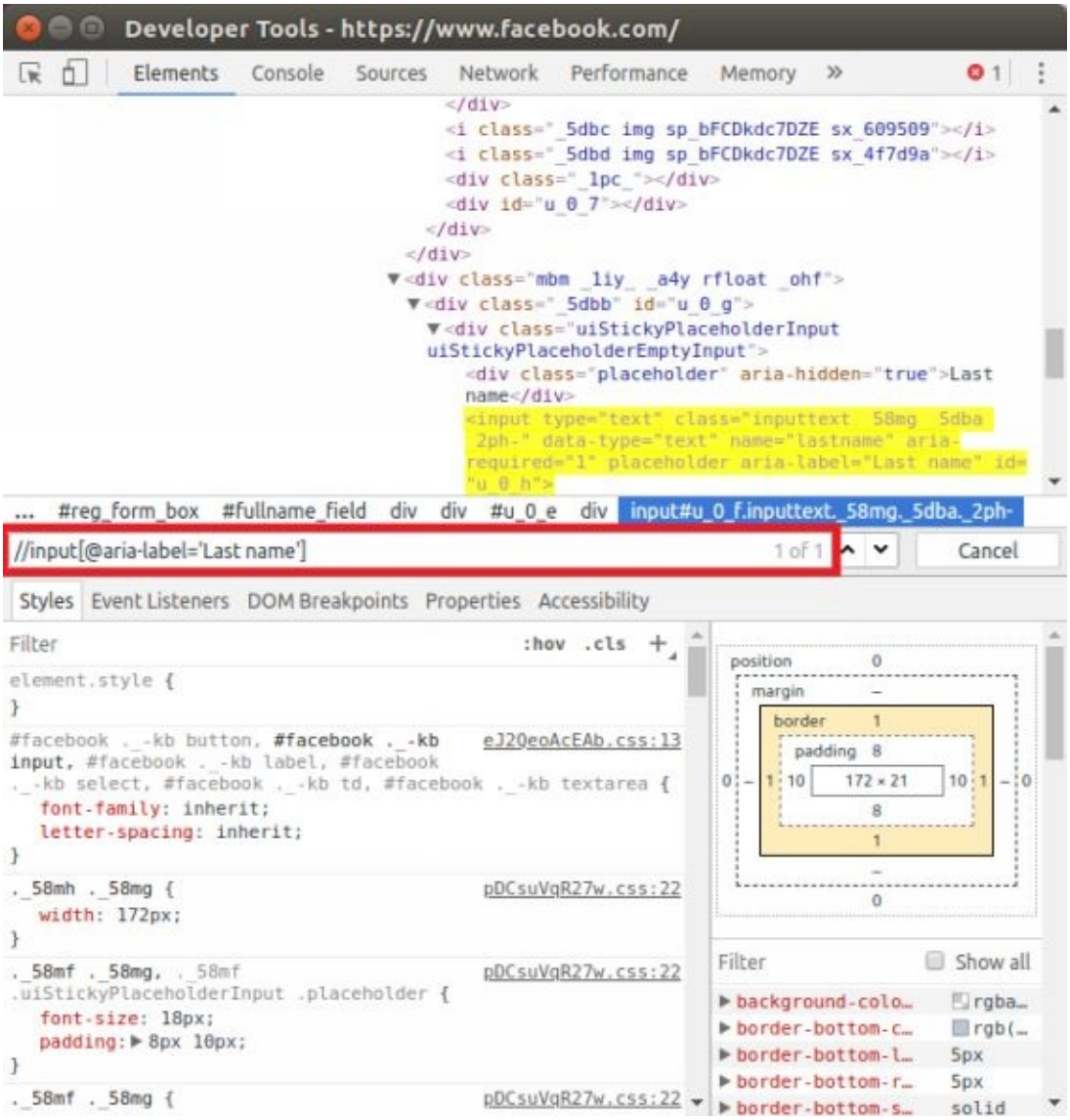


Since the result says "1 of 1", we can see that the element can be identified as one.

Enter the following xpath in the search box.

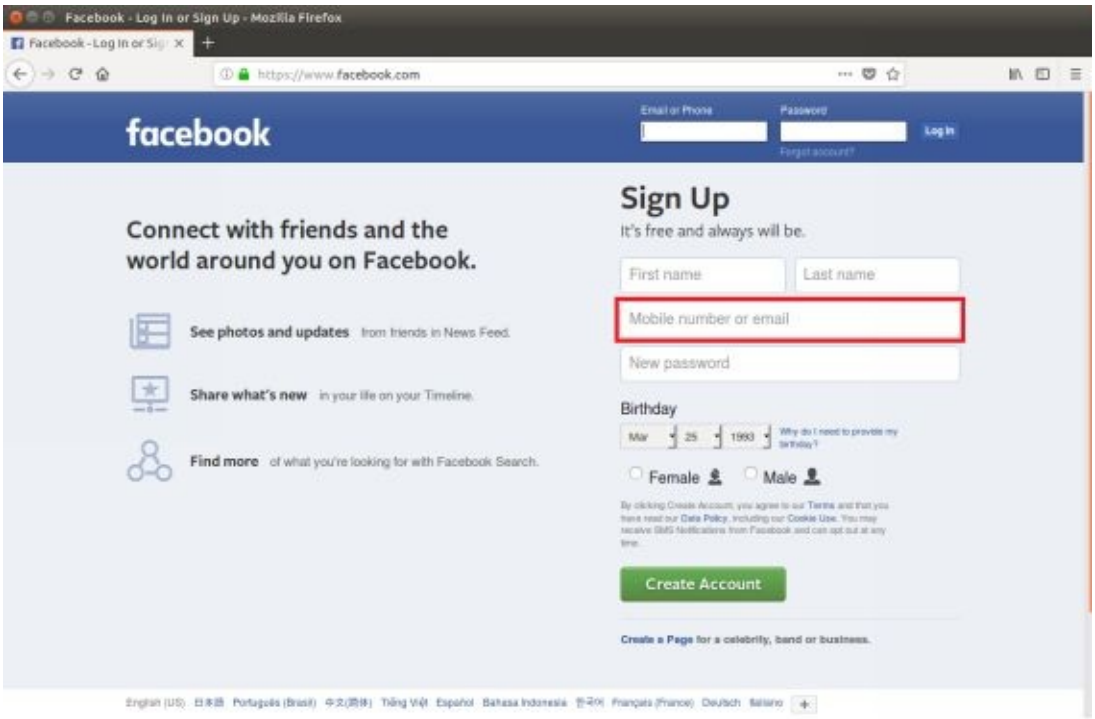
```
//input[@aria-label='Last name']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify text box (Mobile number or email).



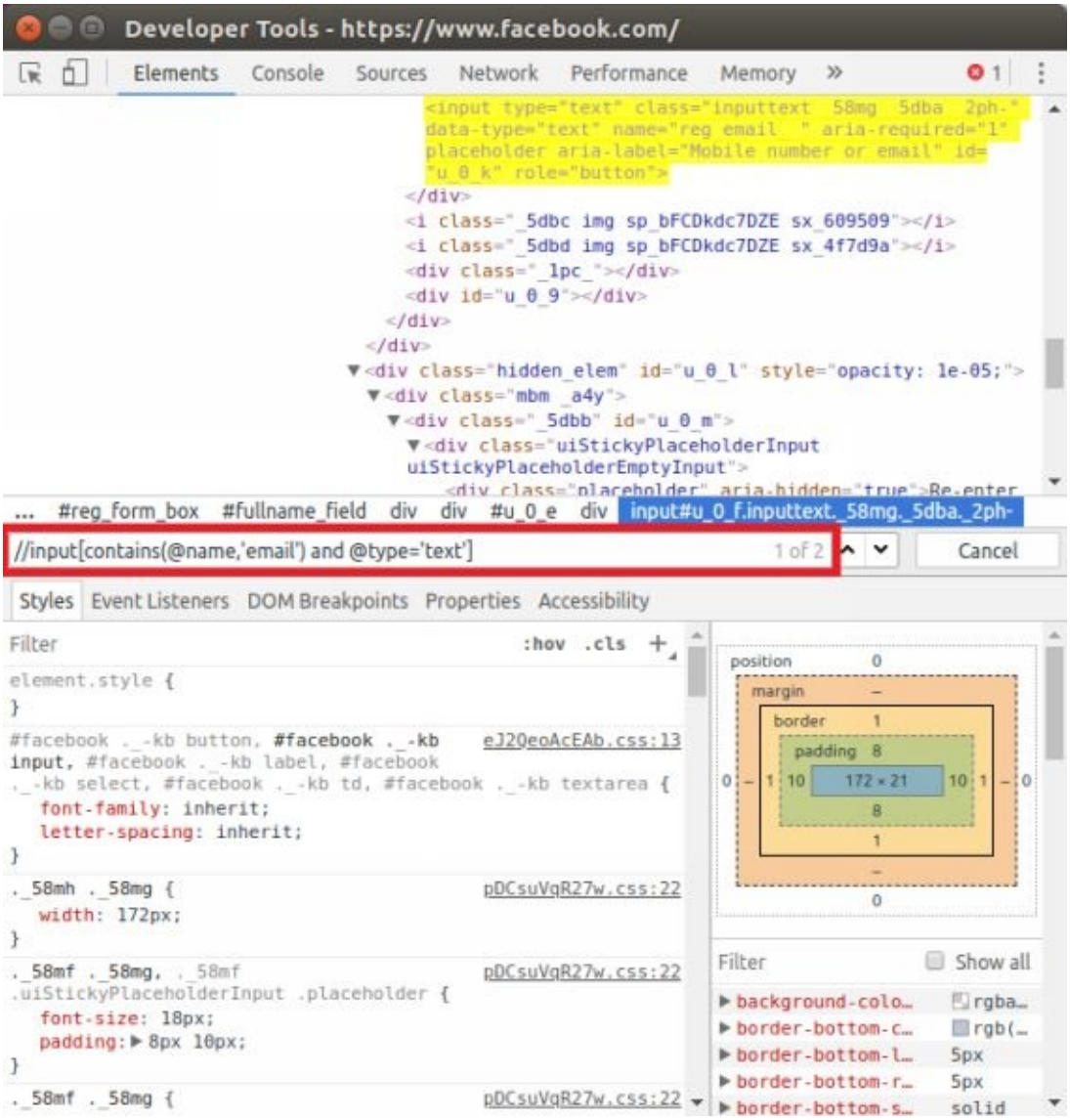
HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="text" name="reg_email__" aria-required="1" placeholder="" aria-label="Mobile number or email" id="u_0_j" type="text">
```

Enter the following xpath in the search box.

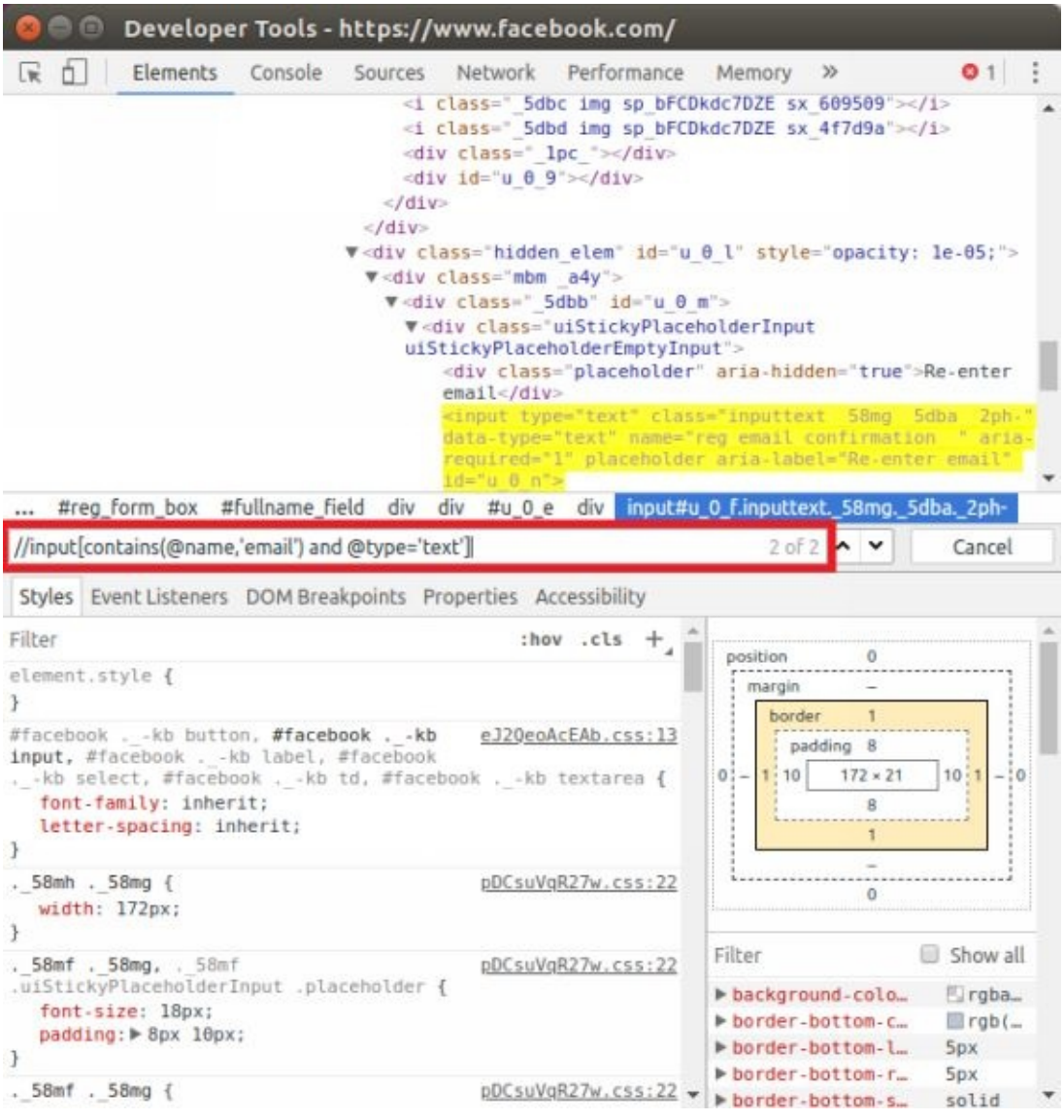
```
//input[contains(@name,'email') and @type='text']
```

The search result is shown below.



Since the result says "1 of 2", we can see that there are two elements which are identified by the same xpath in this page.

Another element identified by the same xpath is text box (Re-enter mail). (This element is displayed on the screen after the mail address is entered in the first

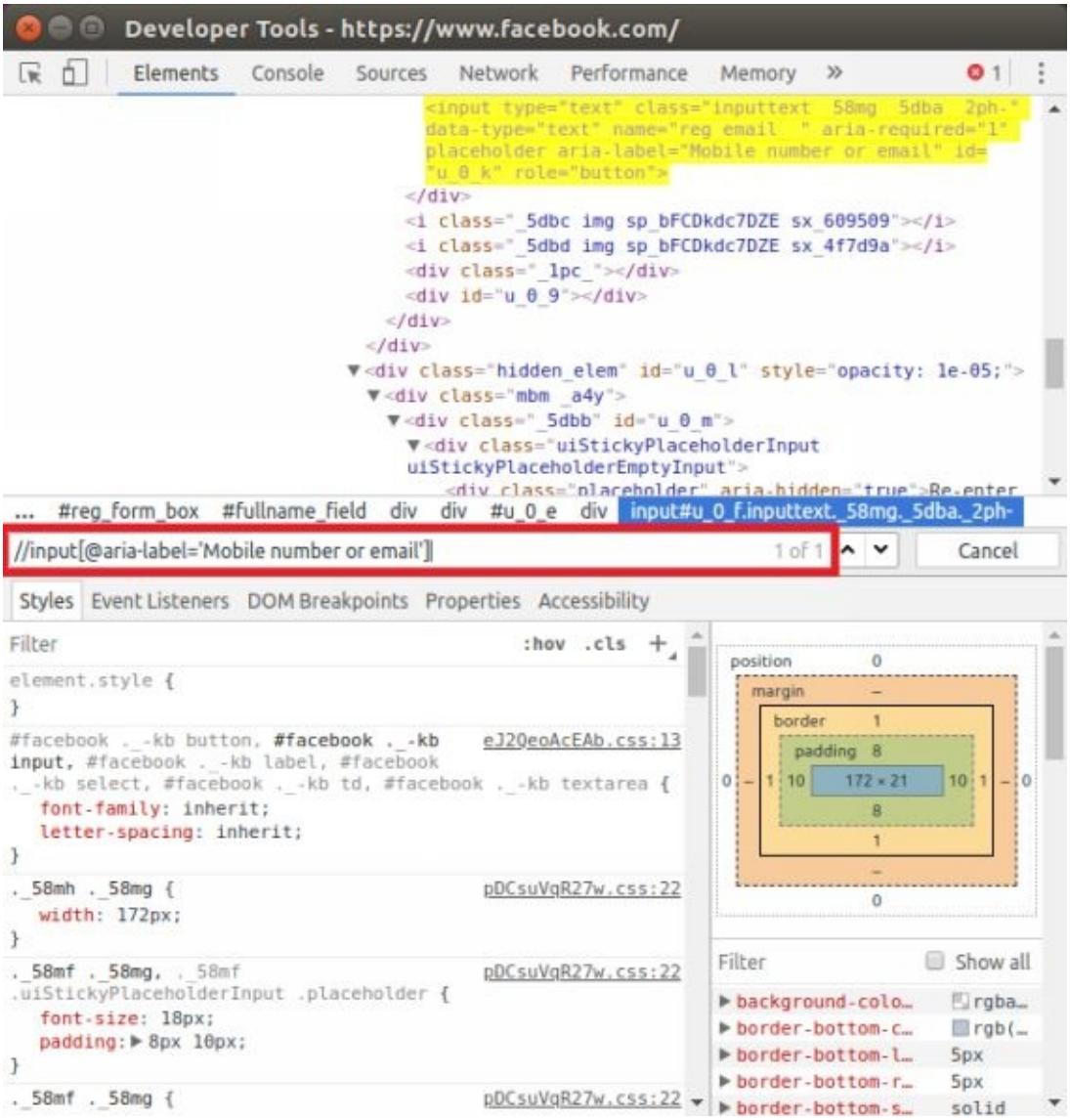


text box)

Enter the following xpath in the search box.

```
//input[@aria-label='Mobile number or email']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify text box (New password).

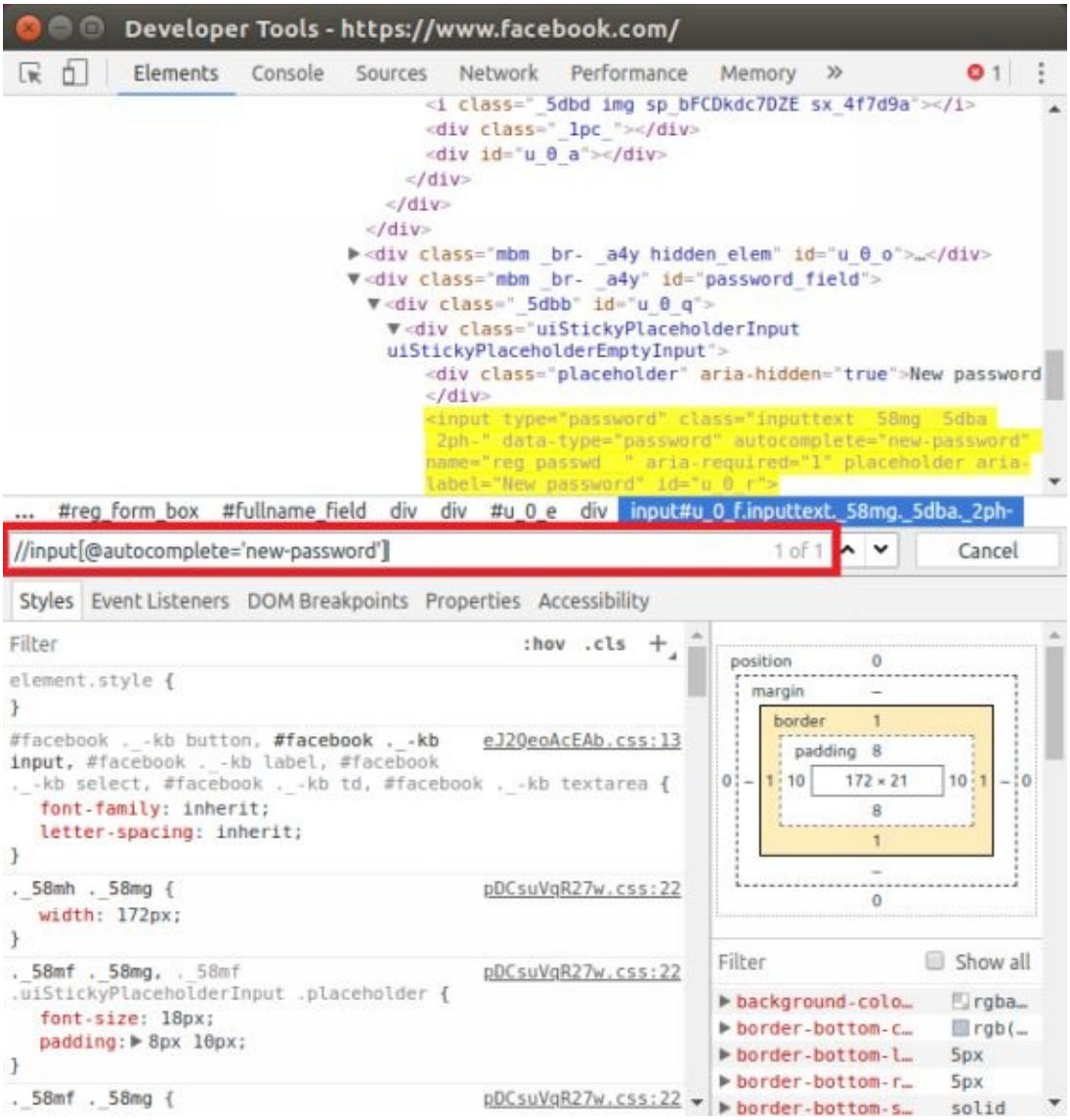
HTML of element

```
<input class="inputtext _58mg _5dba _2ph-" data-type="password" autocomplete="new-password" name="reg_passwd__" aria-required="1" placeholder="" aria-label="New password" id="u_0_q" type="password">
```

Enter the following xpath in the search box.

```
//input[@autocomplete='new-password']
```

The search result is shown below.

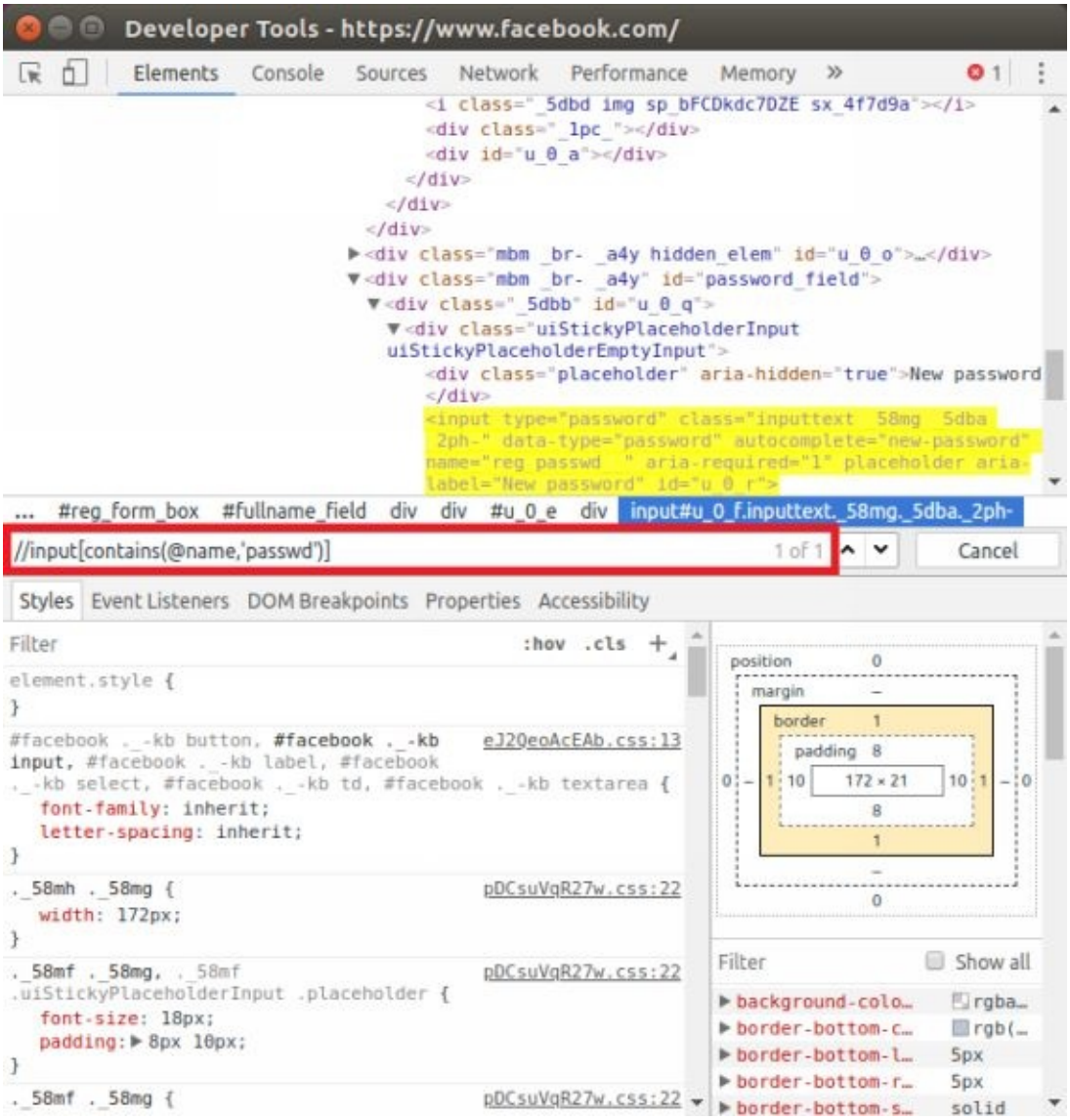


Since the result says "1 of 1", we can see that the element can be identified as one.

Enter the following xpath in the search box.

```
//input[contains(@name,'passwd')]
```

The search result is shown below.

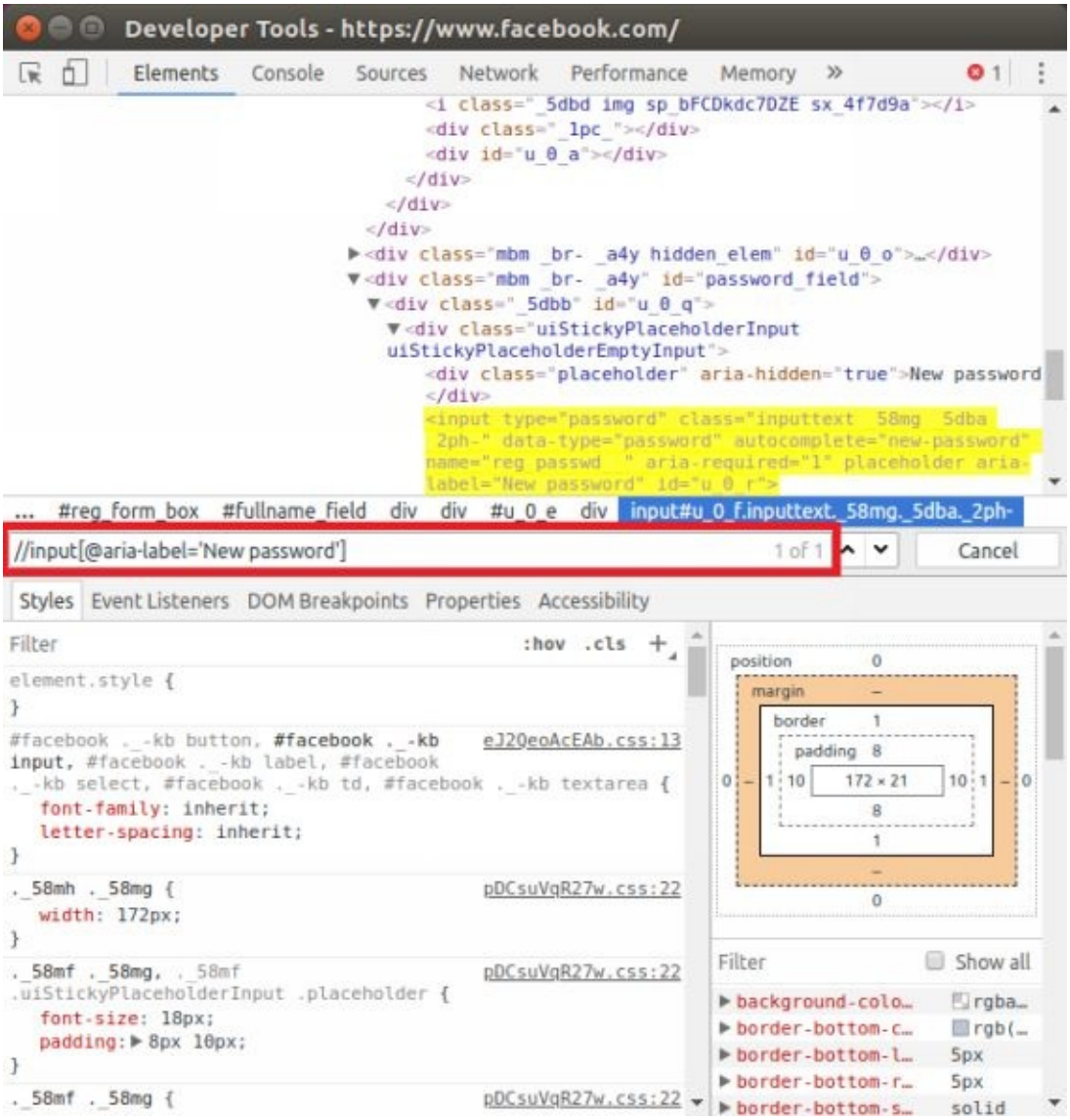


Since the result says "1 of 1", we can see that the element can be identified as one.

Enter the following xpath in the search box.

```
//input[@aria-label='New password']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify dropdown menu option (Jul).

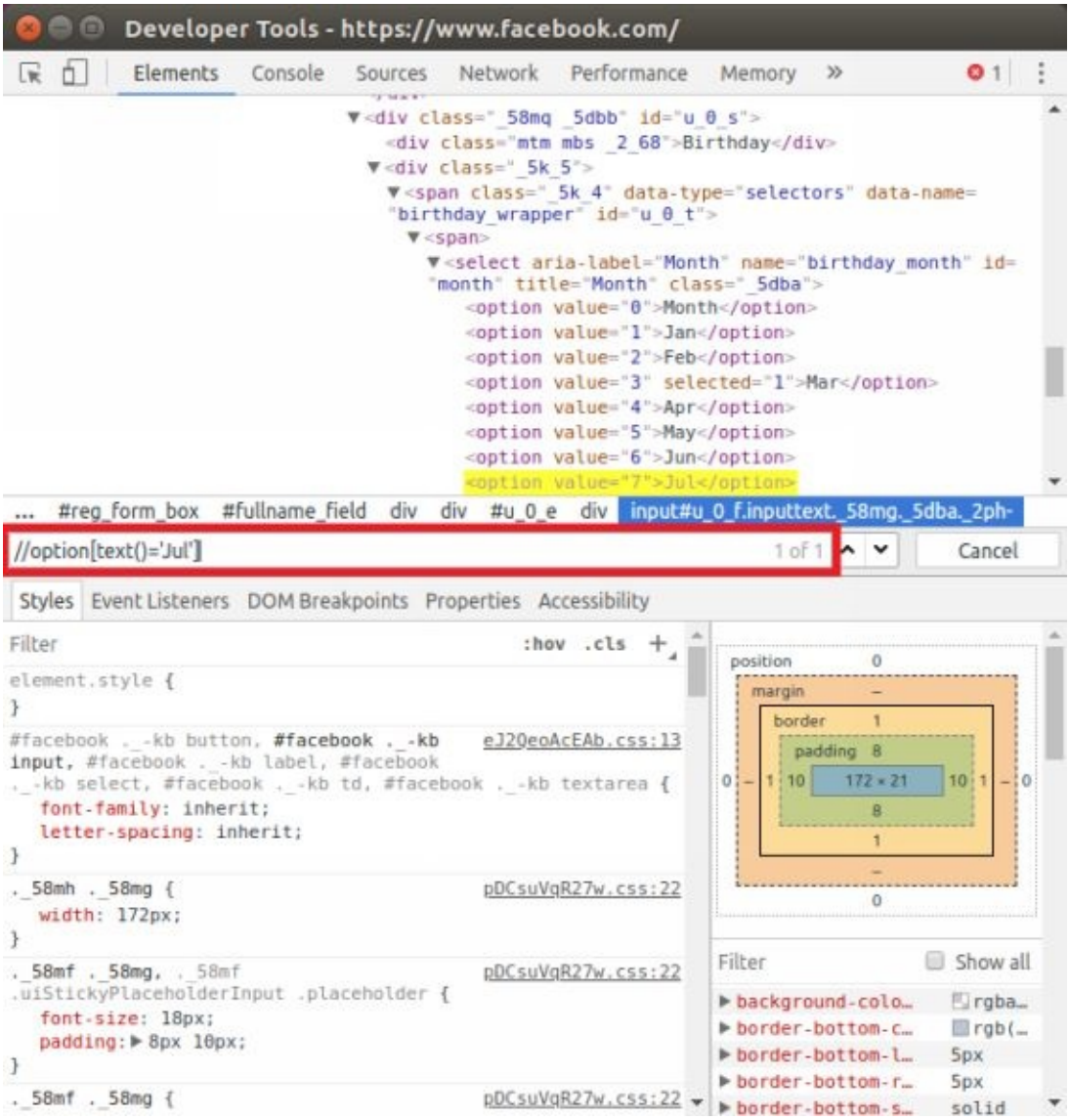
HTML of element

```
<select aria-label="Month" name="birthday_month" id="month" title="Month" class="_5dba"> <option value="0">Month</option>
<option value="1">Jan</option>
<option value="2">Feb</option>
<option value="3" selected="1">Mar</option>
<option value="4">Apr</option>
<option value="5">May</option>
<option value="6">Jun</option>
<option value="7">Jul</option> <option value="8">Aug</option>
<option value="9">Sep</option>
<option value="10">Oct</option>
<option value="11">Nov</option>
<option value="12">Dec</option>
</select>
```

Enter the following xpath in the search box.

```
//option[text()='Jul']
```

The search result is shown below.

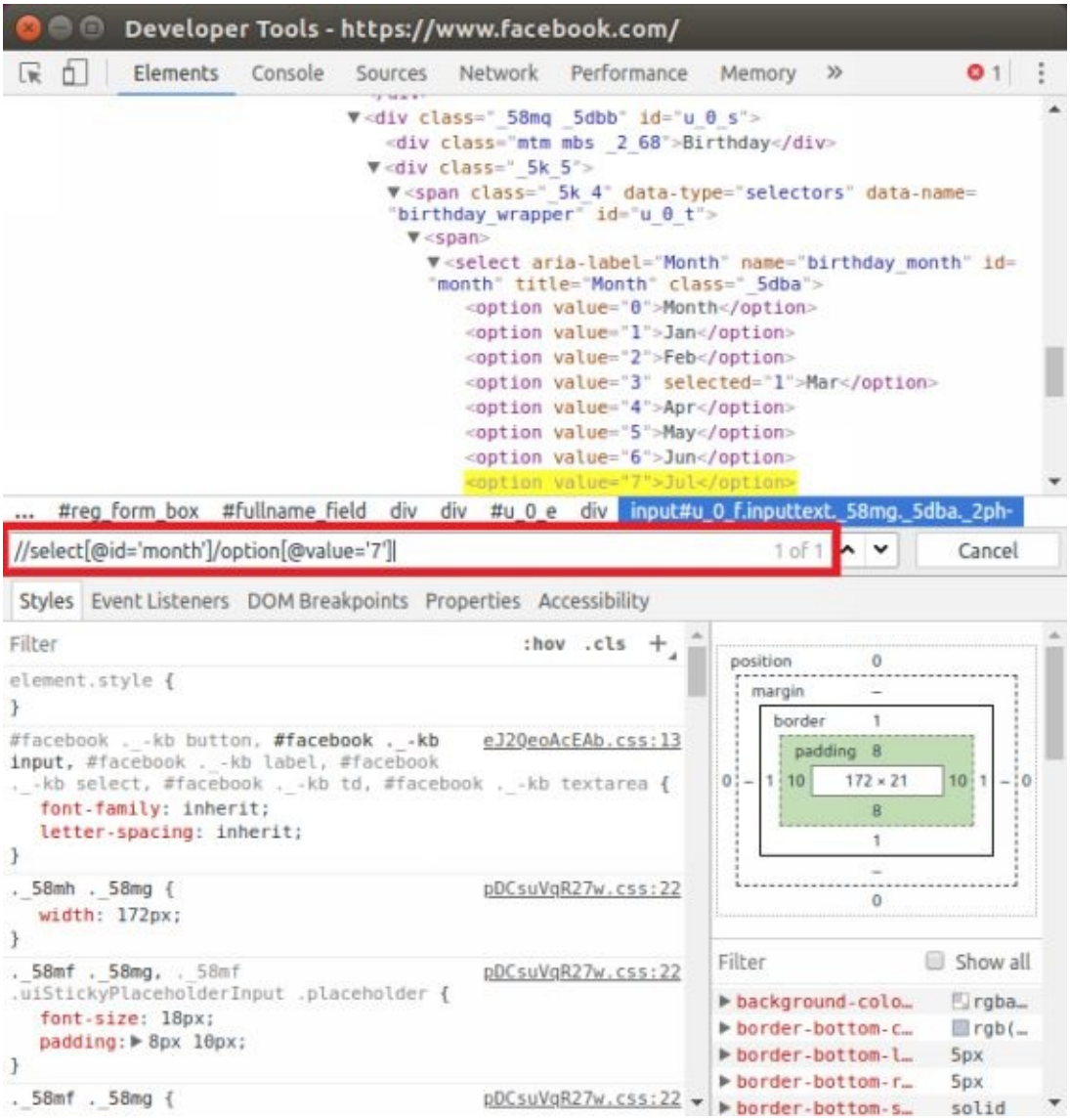


Since the result says "1 of 1", we can see that the element can be identified as one.

Enter the following xpath in the search box.

```
//select[@id='month']/option[@value='7']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify radio button (Female).

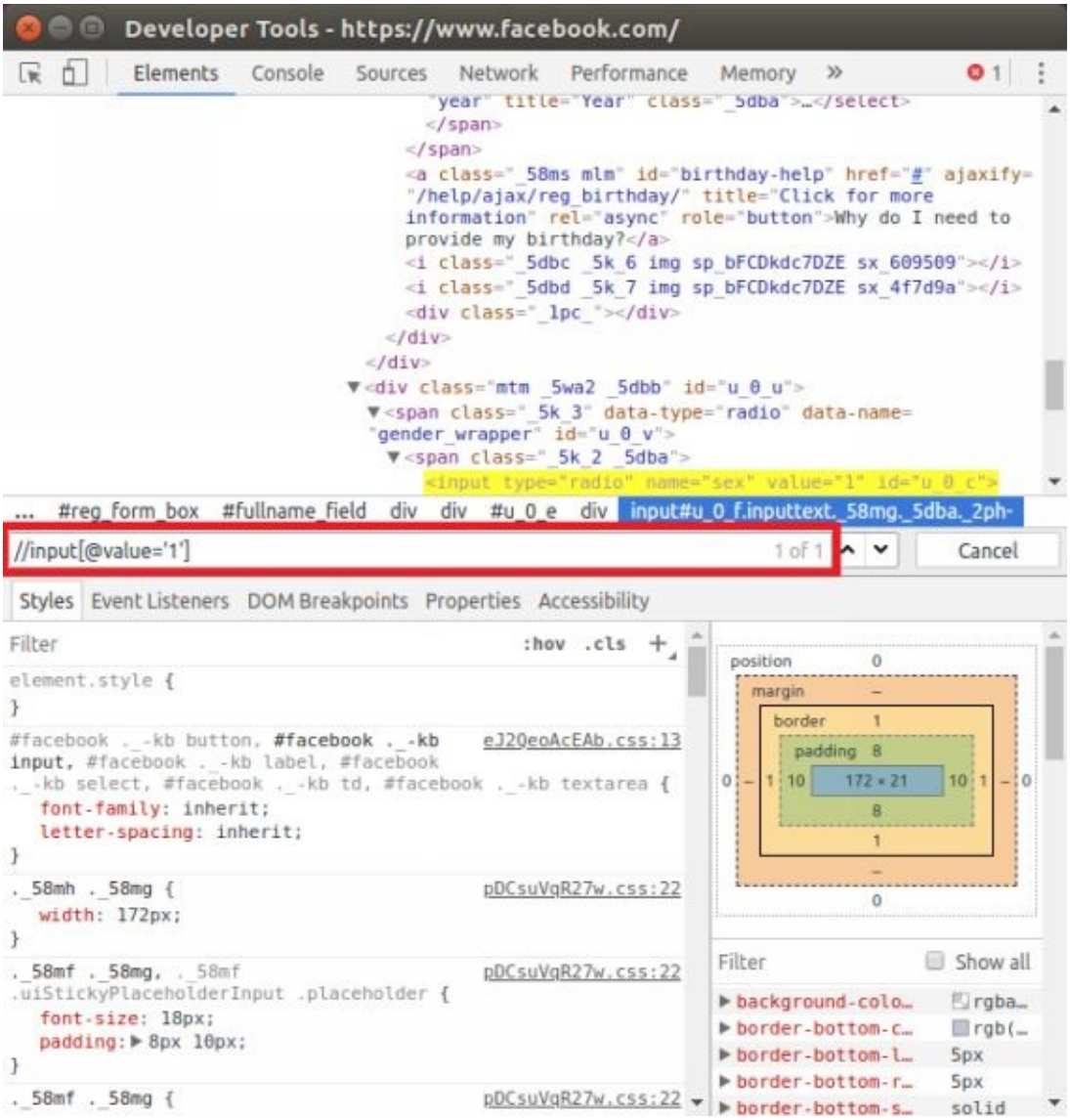
HTML of element

```
<input name="sex" value="1" id="u_0_6" type="radio">
```

Enter the following xpath in the search box.

```
//input[@value='1']
```

The search result is shown below.



Since the result says "1 of 1", we can see that the element can be identified as one.

Suppose we want to identify radio button (Male).

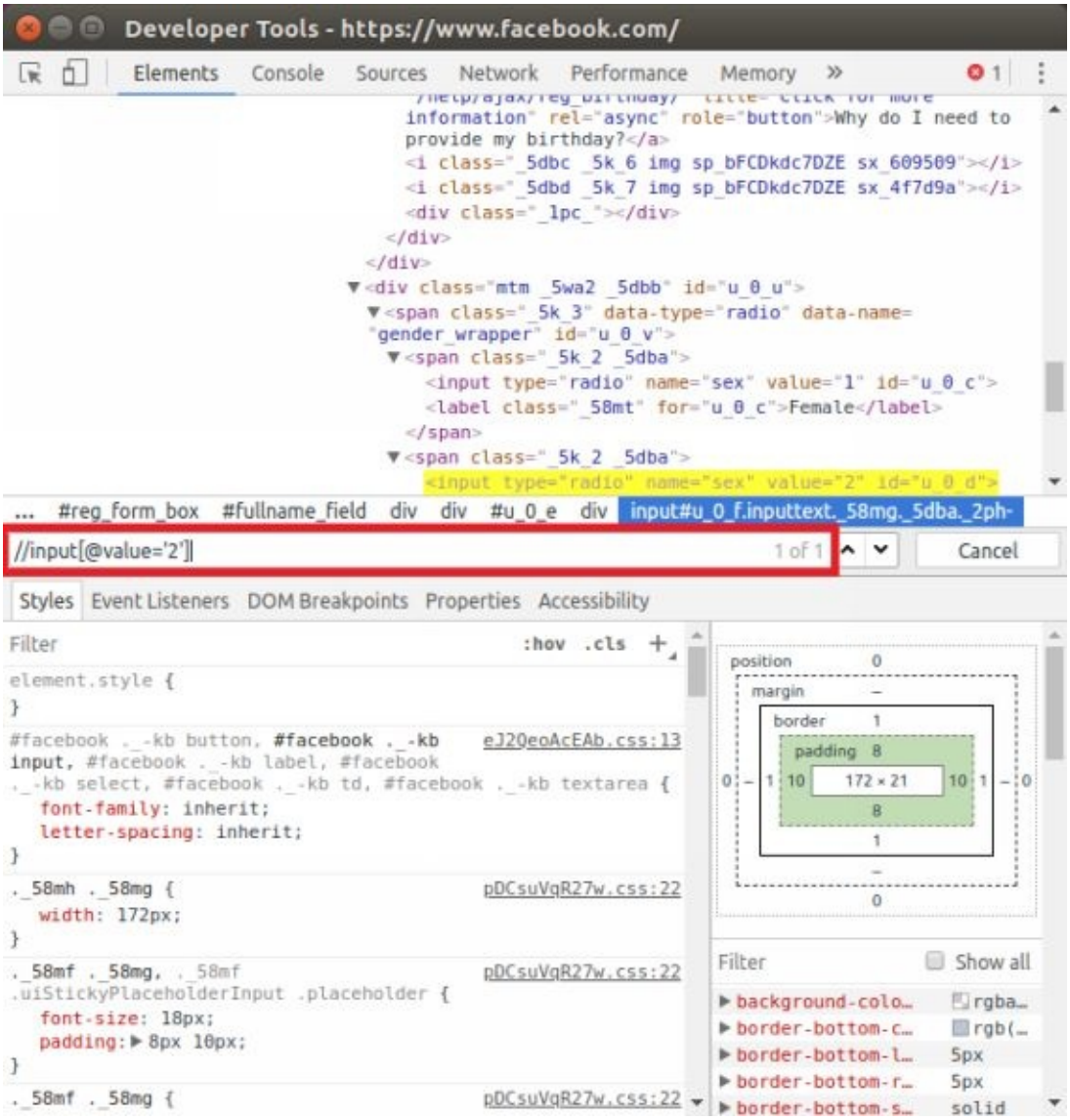
HTML of element

```
<input name="sex" value="2" id="u_0_7" type="radio">
```

Enter the following xpath in the search box.

```
//input[@value='2']
```

The search result is shown below.



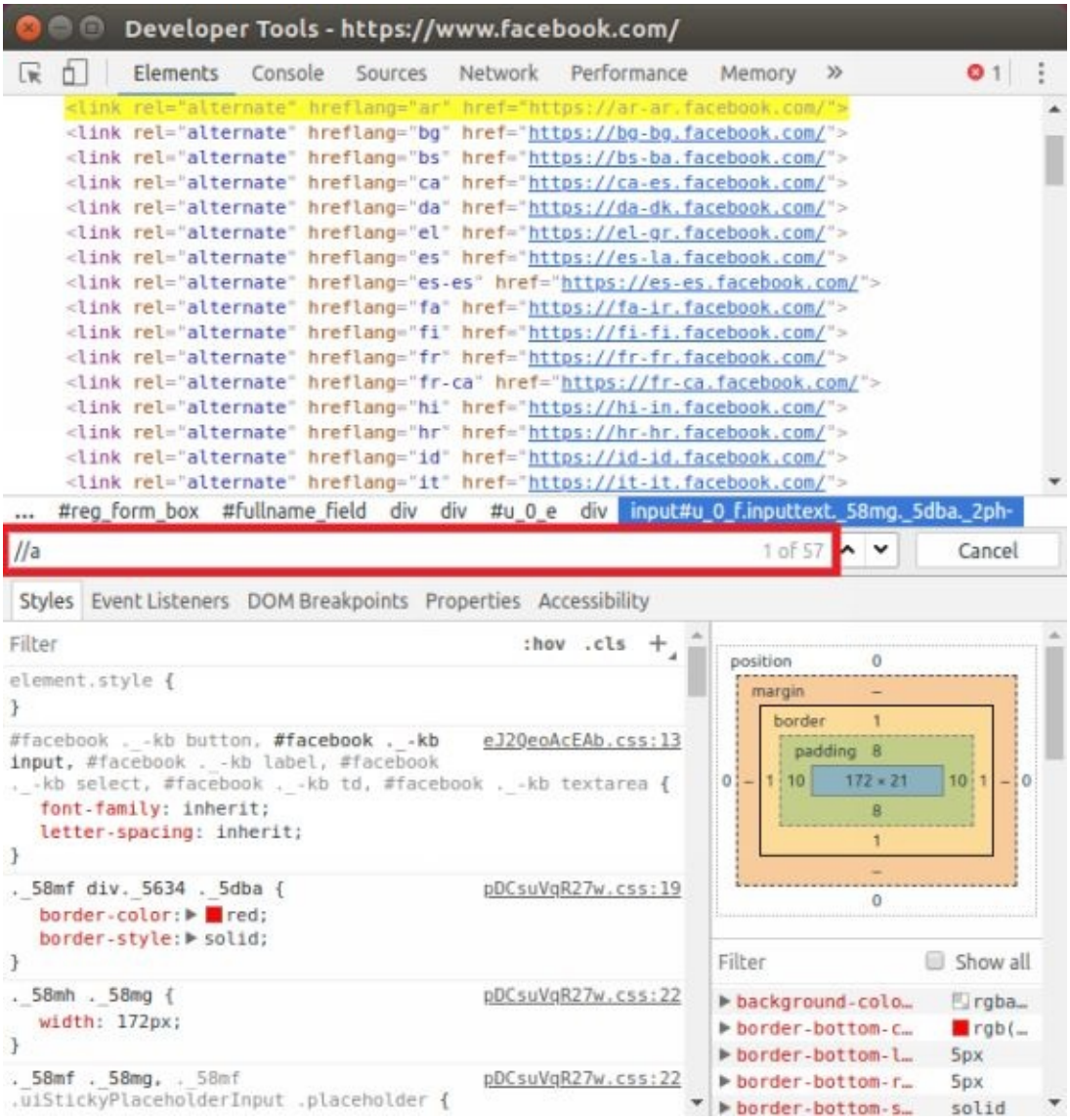
Since the result says "1 of 1", we can see that the element can be identified as one.

Next, we practice how to use "starts-with" and "following" in xpath.

Enter the following xpath in the search box.

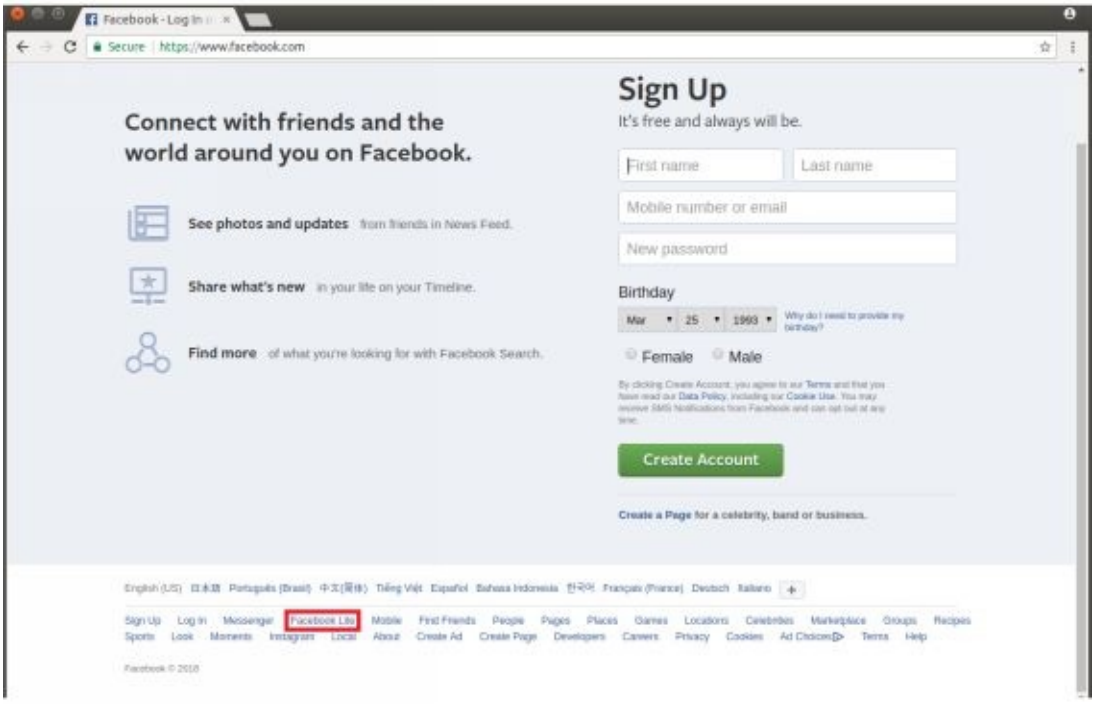
```
//a
```

The search result is shown below.



Since the result says "1 of 57", we can see that there are 57 elements of "a" tag in this page.

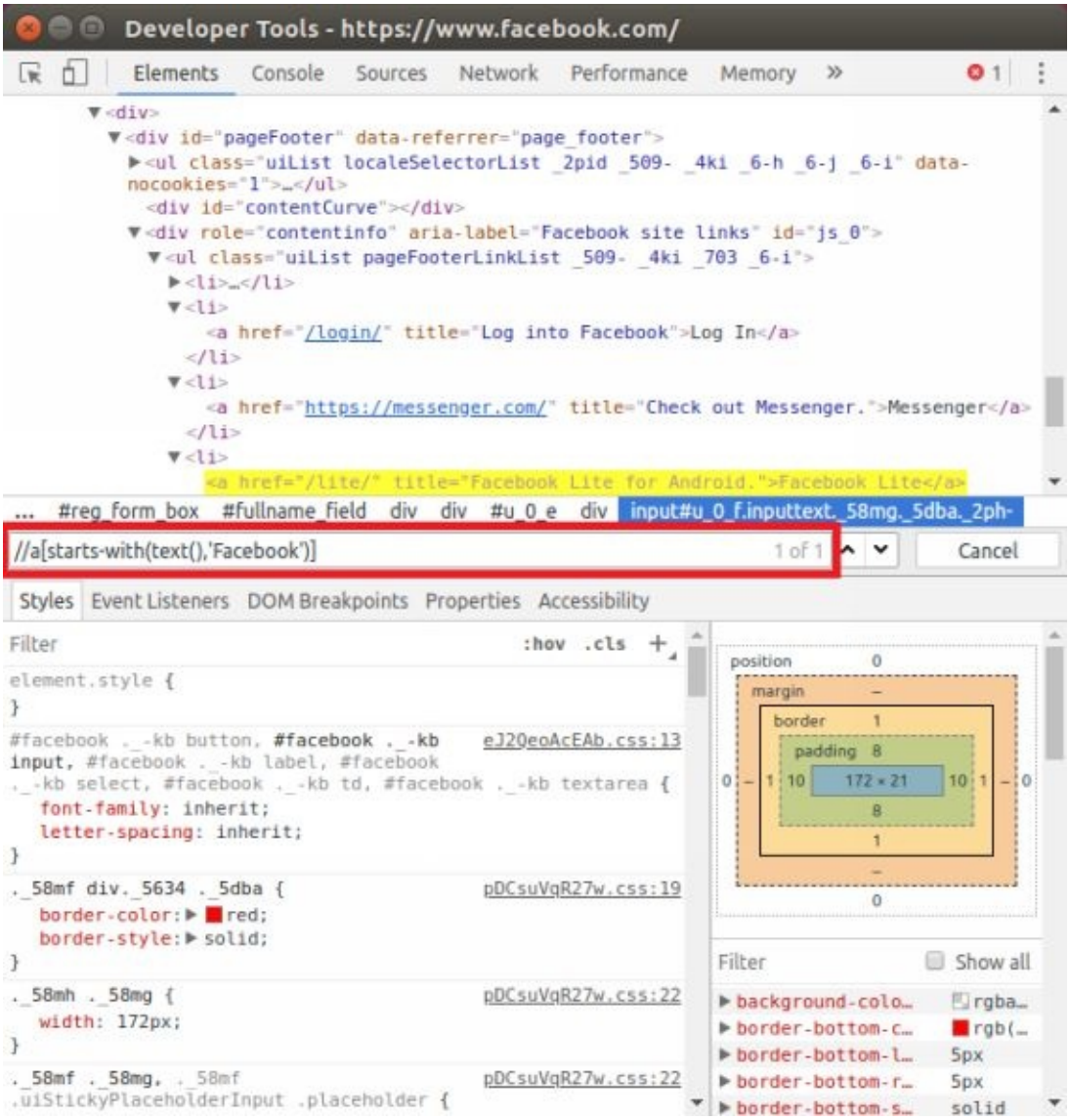
There is one "a" tag element that starts with the word "Facebook" in this page.



Enter the following xpath in the search box.

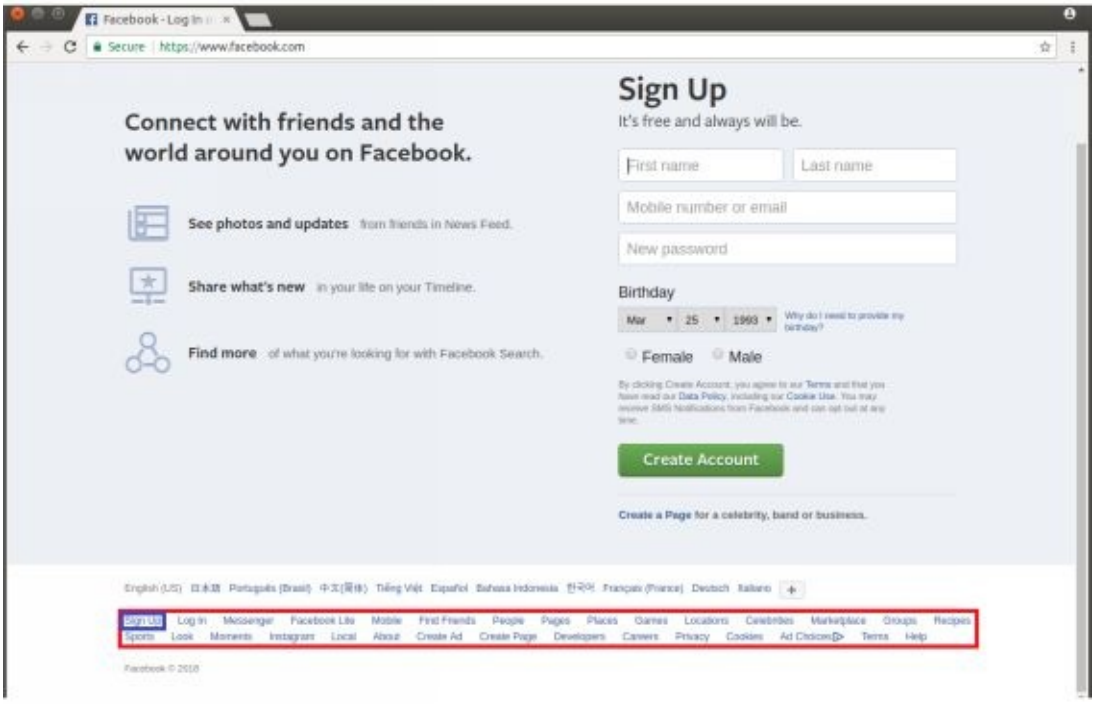
```
//a[starts-with(text(),'Facebook')]
```

The search result is shown below.



Since the result says "1 of 1", there is one "a" tag element that starts with the word "Facebook" in this page.

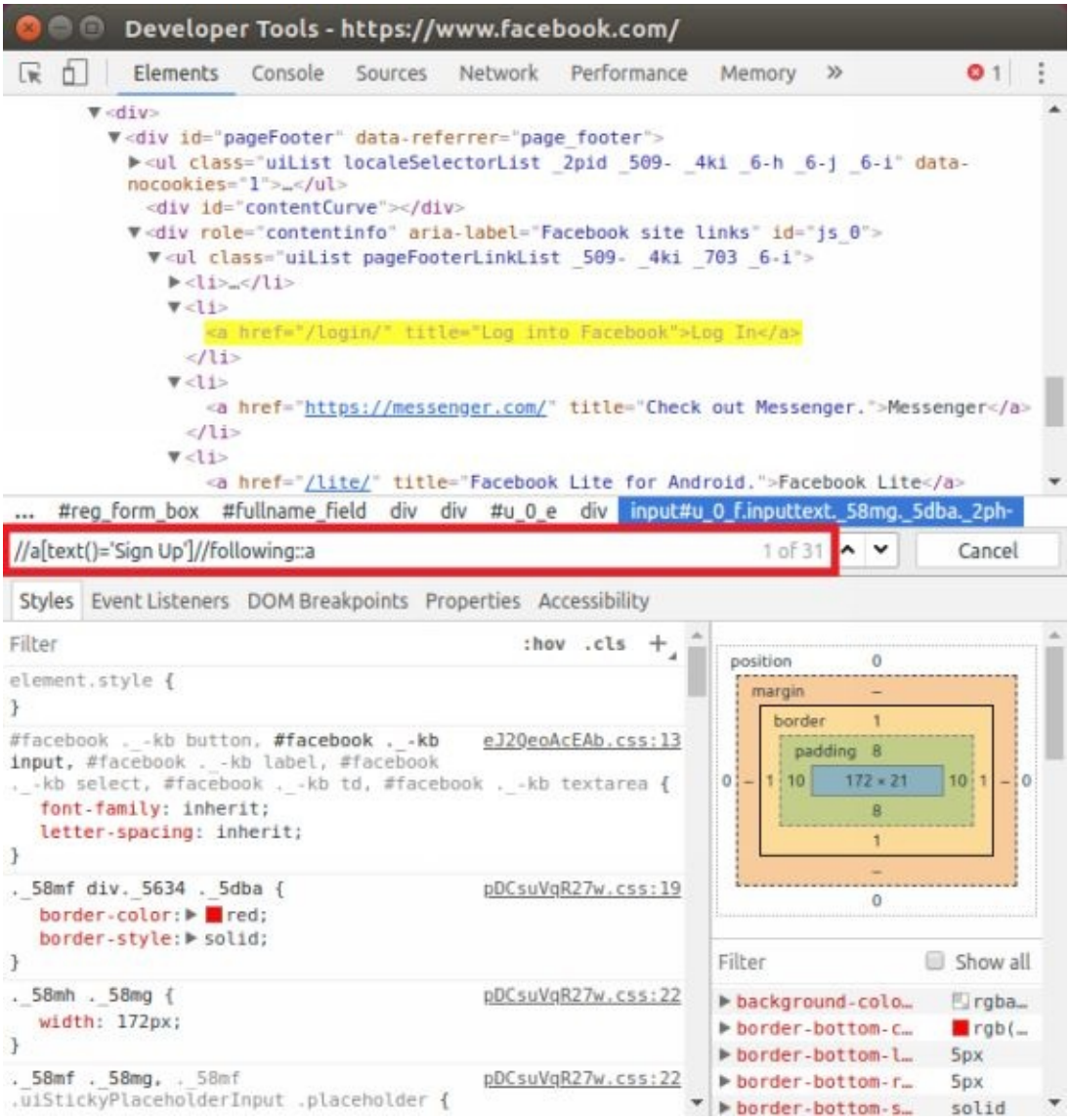
Suppose we want to examine "a" tag elements after "Sign Up" link.



Enter the following xpath in the search box.

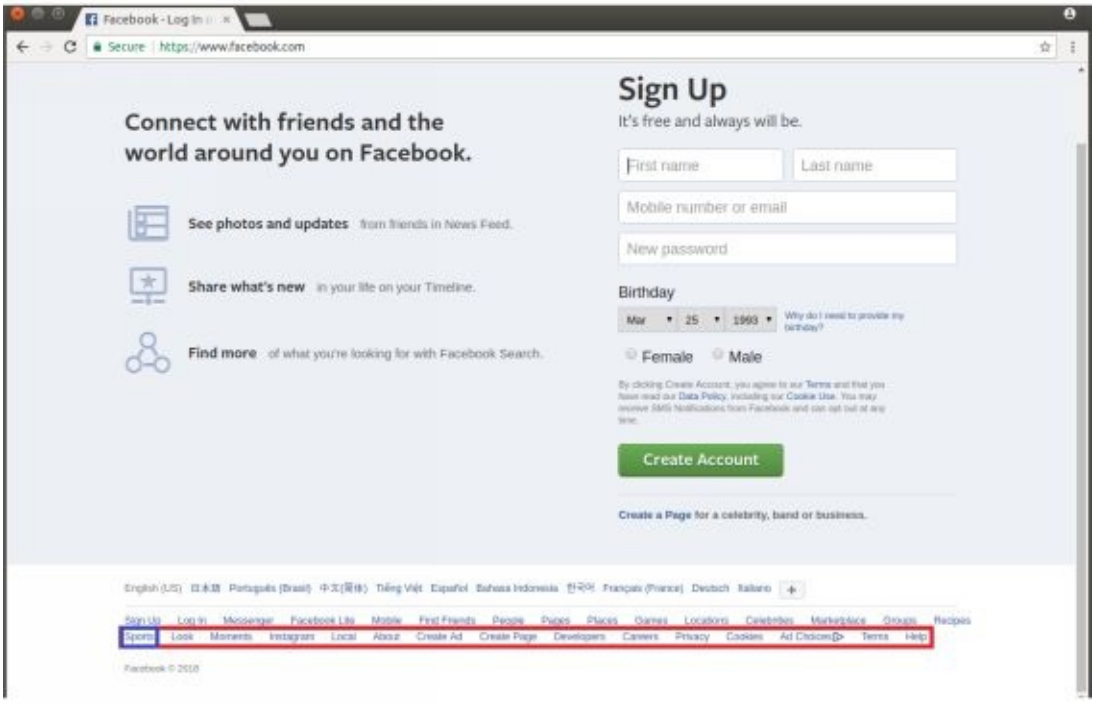
```
//a[text()='Sign Up']//following::a
```

The search result is shown below.



By using "following" it is possible to search for "a" tag elements after the element identified by "//a[text()='Sign Up']".

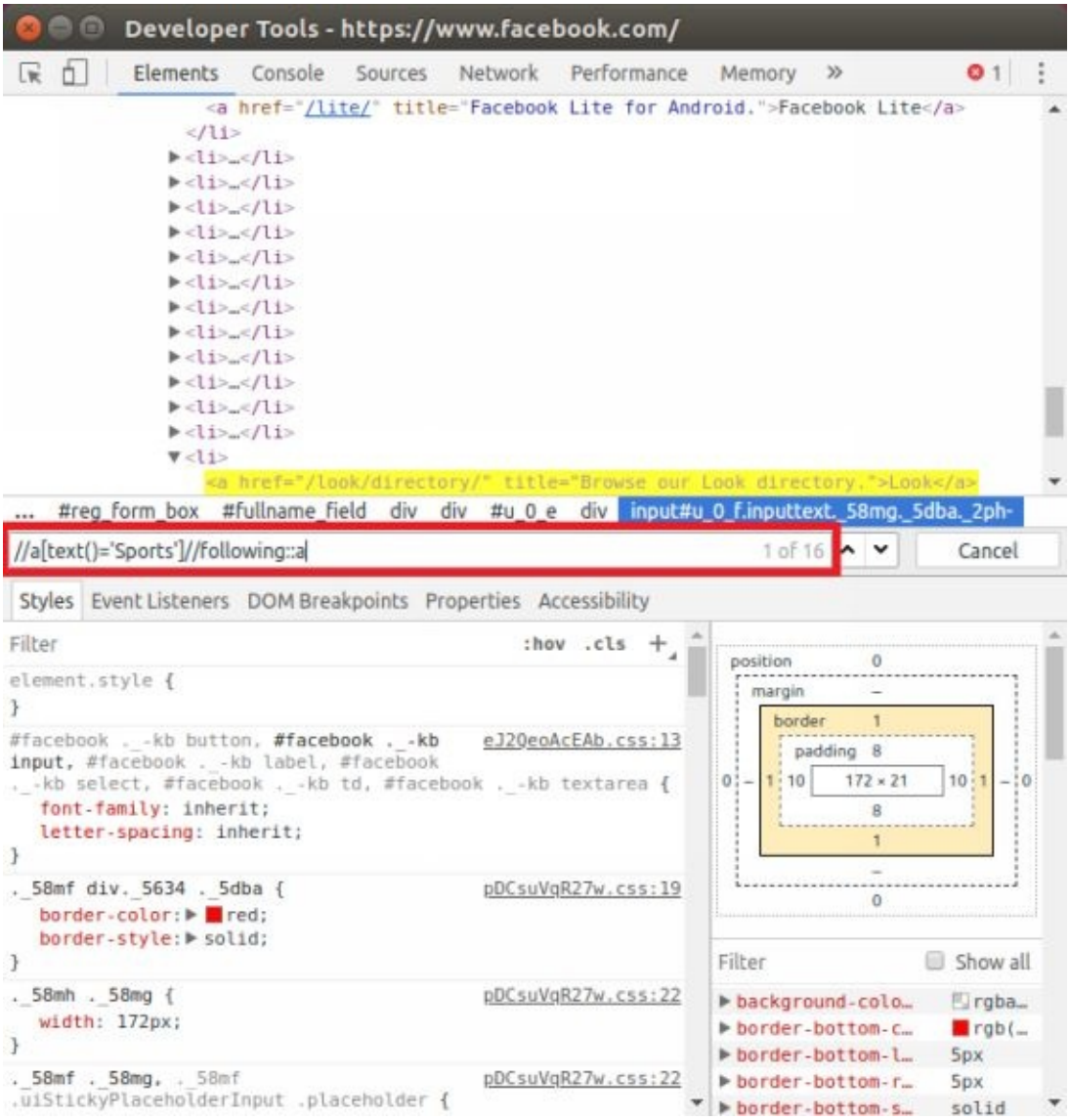
Next, suppose we want to examine "a" tag elements after "Sports" link.



Enter the following xpath in the search box.

```
//a[text()='Sports']//following::a
```

The search result is shown below.



By using "following" it is possible to search for "a" tag elements after the element identified by "//a[text()='Sports']".

In this chapter, we learned the knowledge necessary for automatic operation of browser using Selenium WebDriver.

Chapter 2 Prepare automatic operation environment

In this chapter, we prepare an environment for automatic operation of browser using Selenium WebDriver.

2-1 Prepare browser and driver

Here we prepare the environment surrounded by blue line (browser and driver) of below figure.

2-1-1 Prepare Mozilla Firefox

Firefox is preinstalled on Ubuntu so we can use Firefox after installing Ubuntu.

For example, when installing Ubuntu using "ubuntu-16.04.4-desktop-amd64.iso", the version of Firefox is as follows.

We update Ubuntu to make Firefox the latest one.

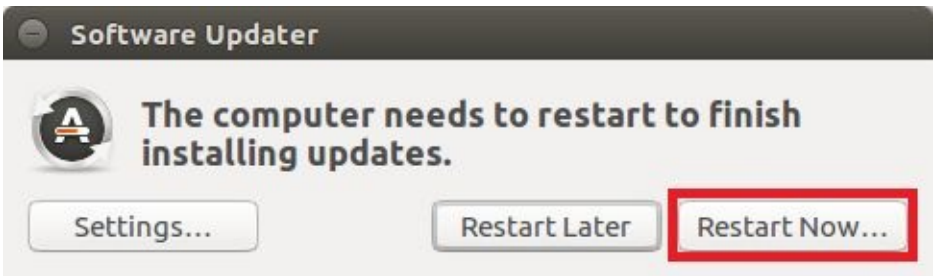
Click "System Setting" icon

Click "Details" icon

Click "Install Updates"

Click "Install Now"

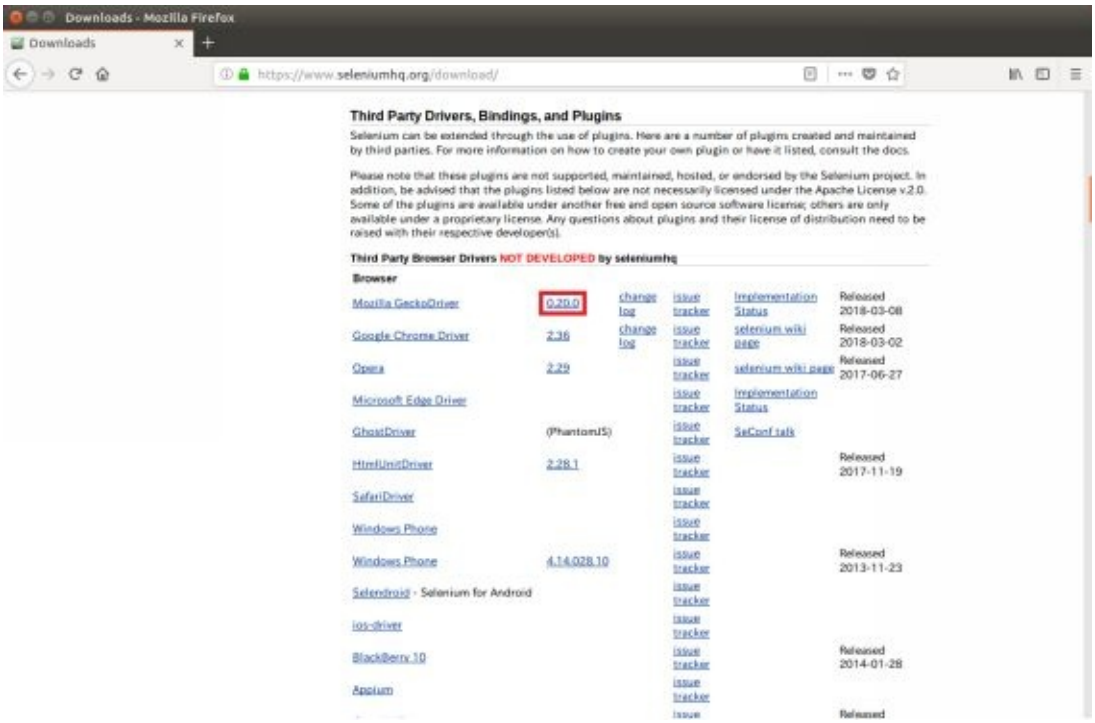
Click "Restart Now..."



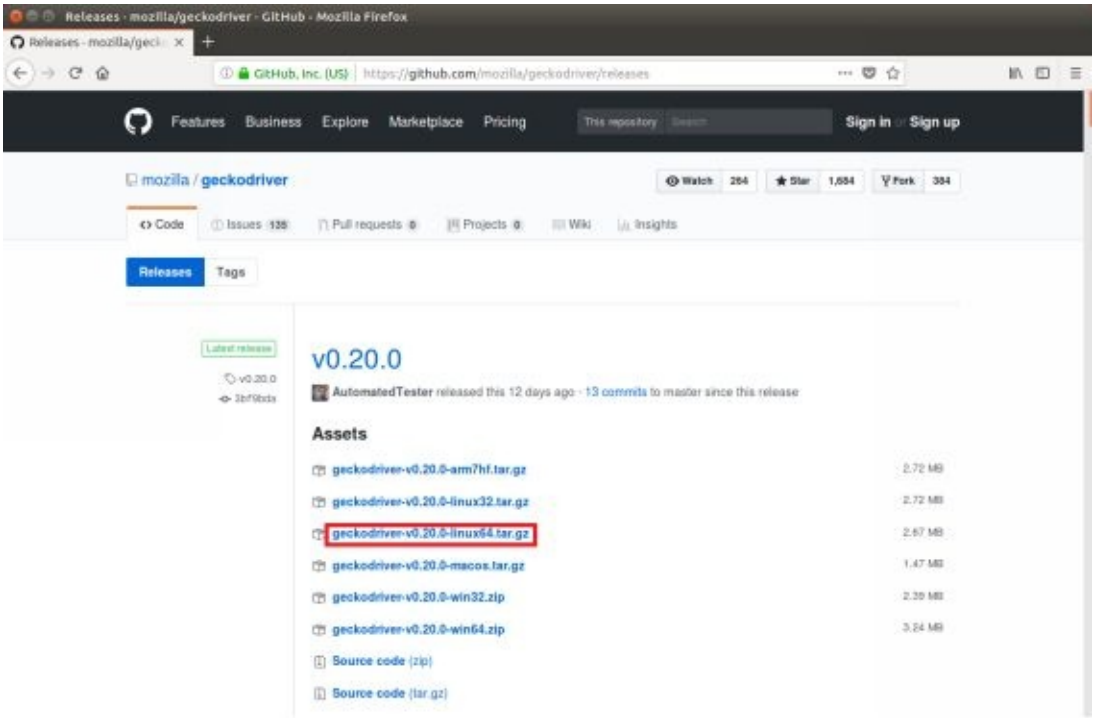
After restarting, the version of Firefox is up to date as shown below.

Next, we prepare Mozilla GeckoDriver.

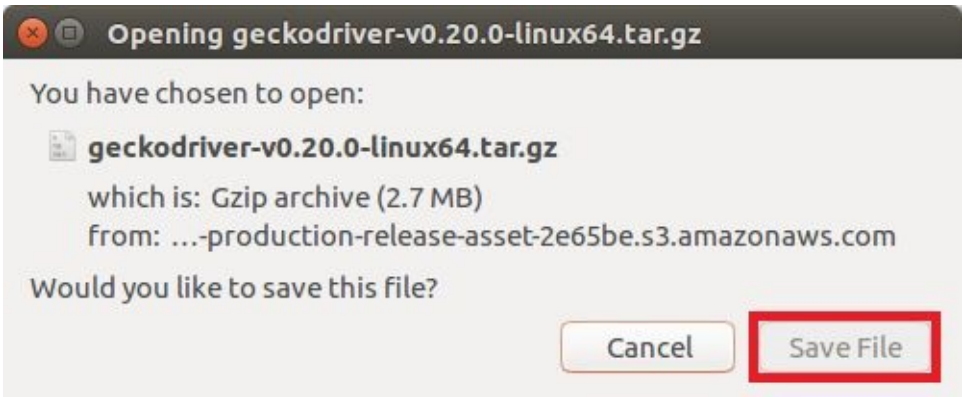
Select Mozilla GeckoDriver on Selenium Downloads site



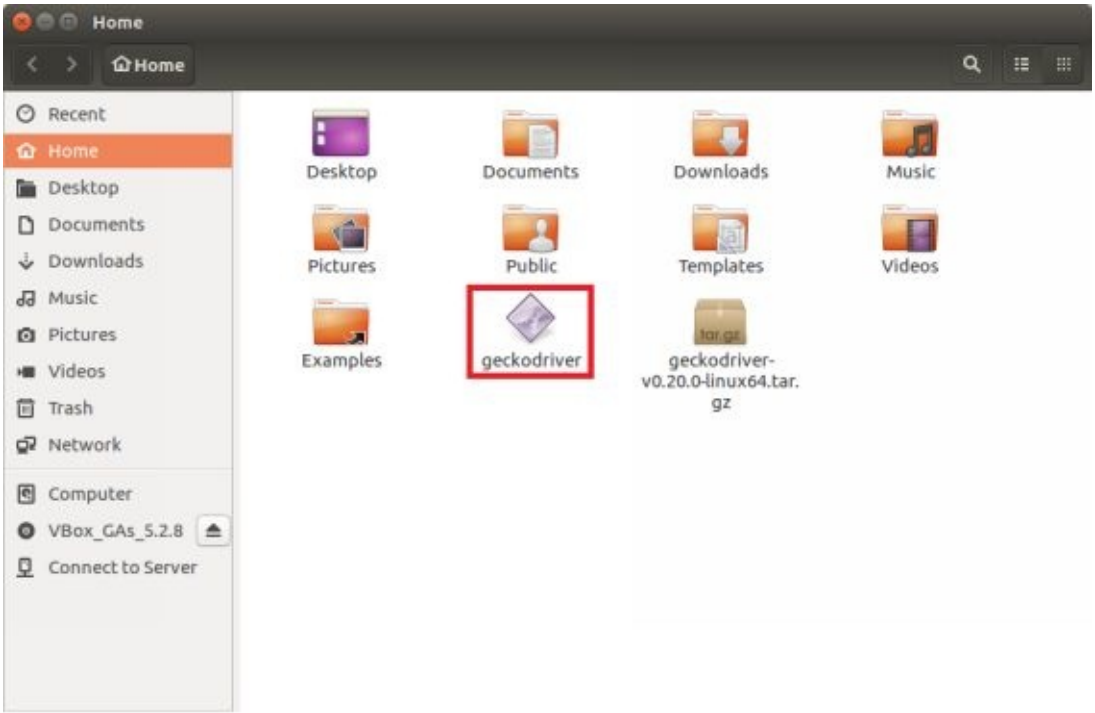
Select driver to download



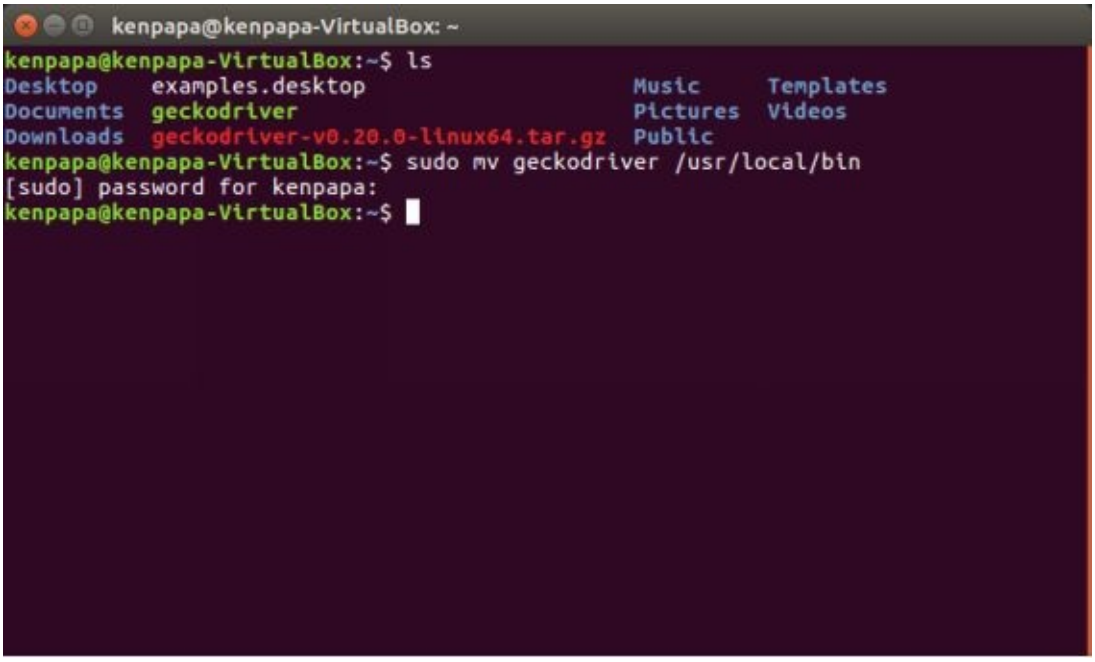
Click "Save File"



Extract the file after moving to Home



Move executable file to "/usr/local/bin"



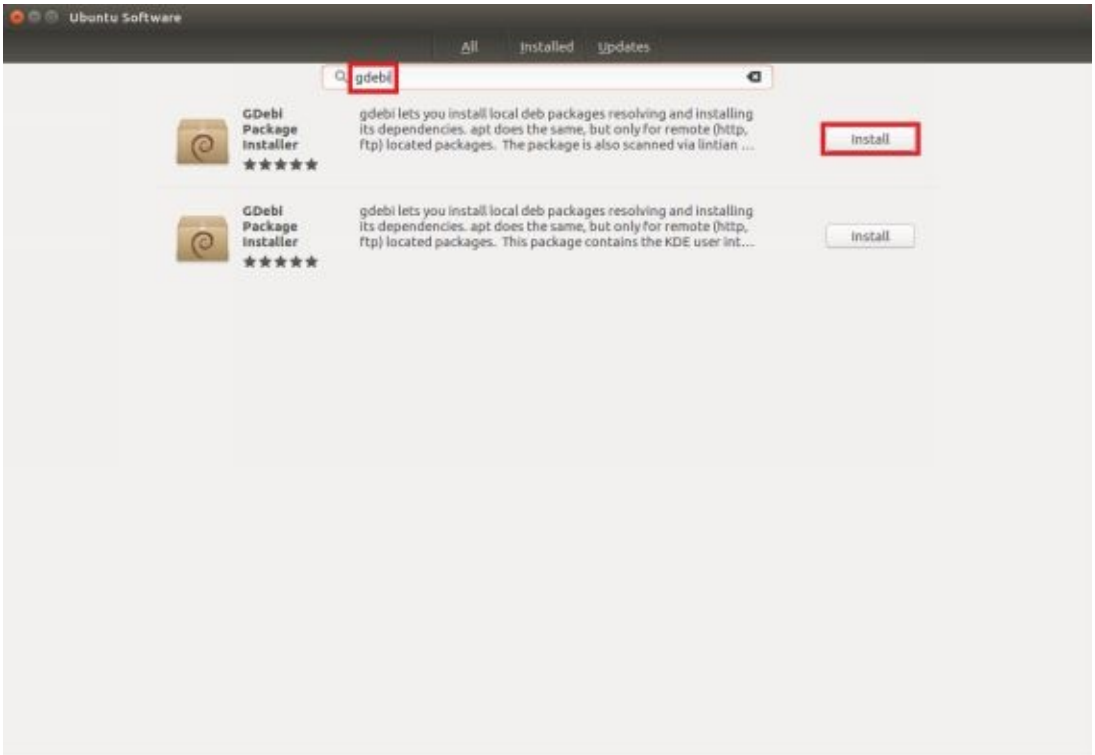
This completes the preparation of Firefox.

2-1-2 Prepare Google Chrome

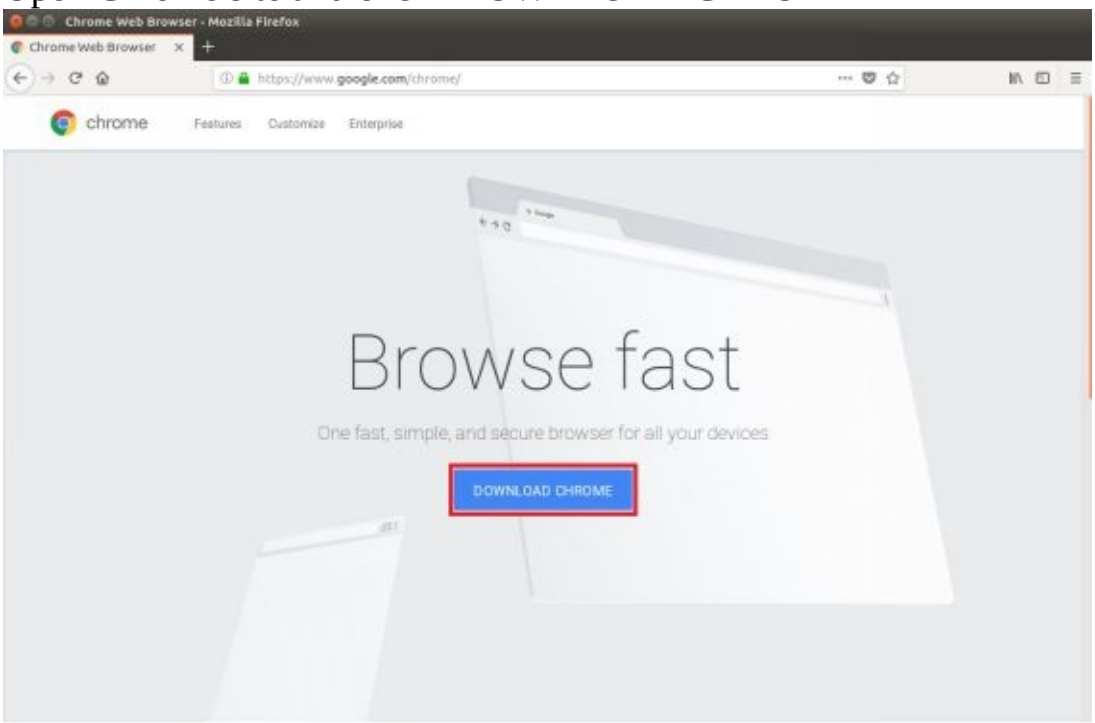
We can install Chrome in the following procedure.
Install GDebi package
Download Chrome
Install Chrome using GDebi

Click "Ubuntu Software" icon

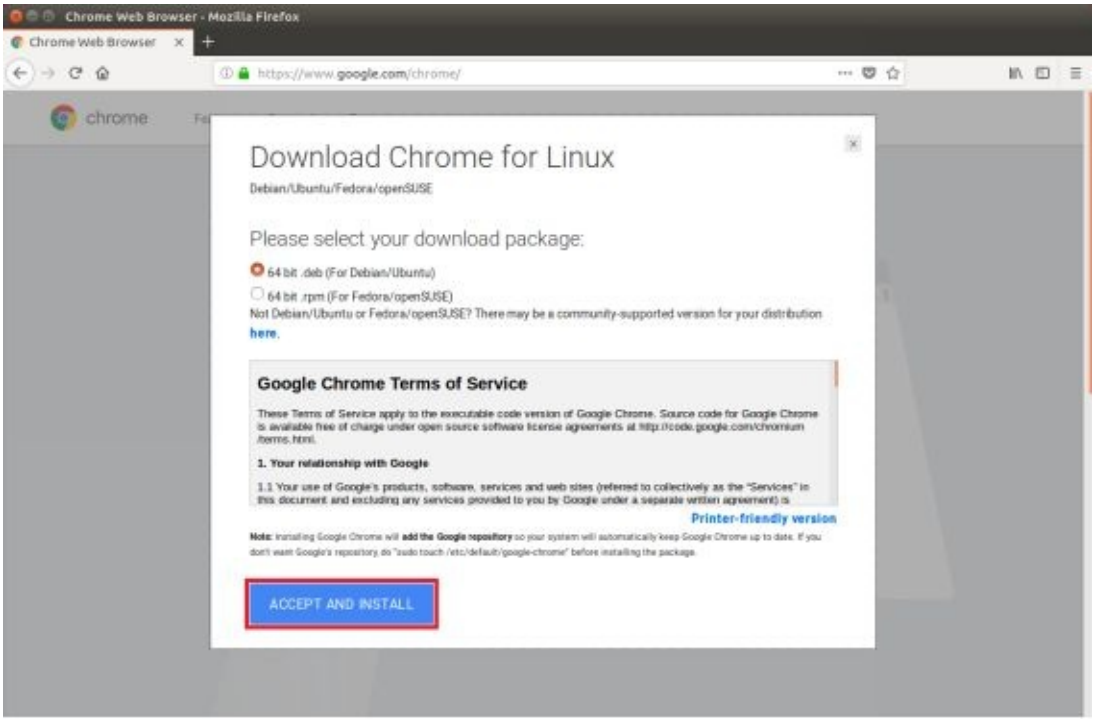
Search and install the GDebi package



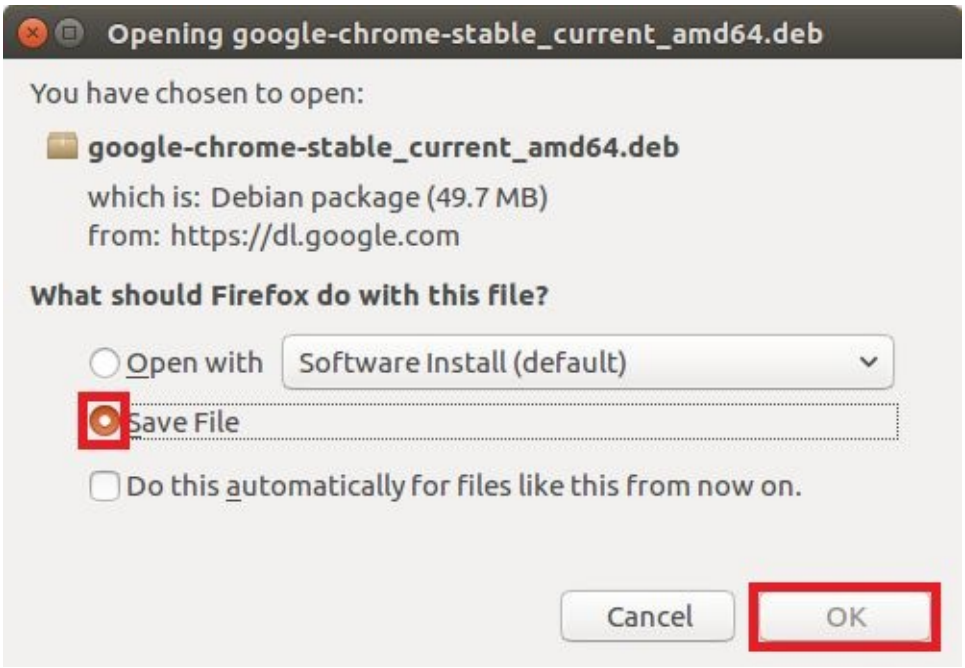
Open Chrome site and click "DOWNLOAD CHROME"



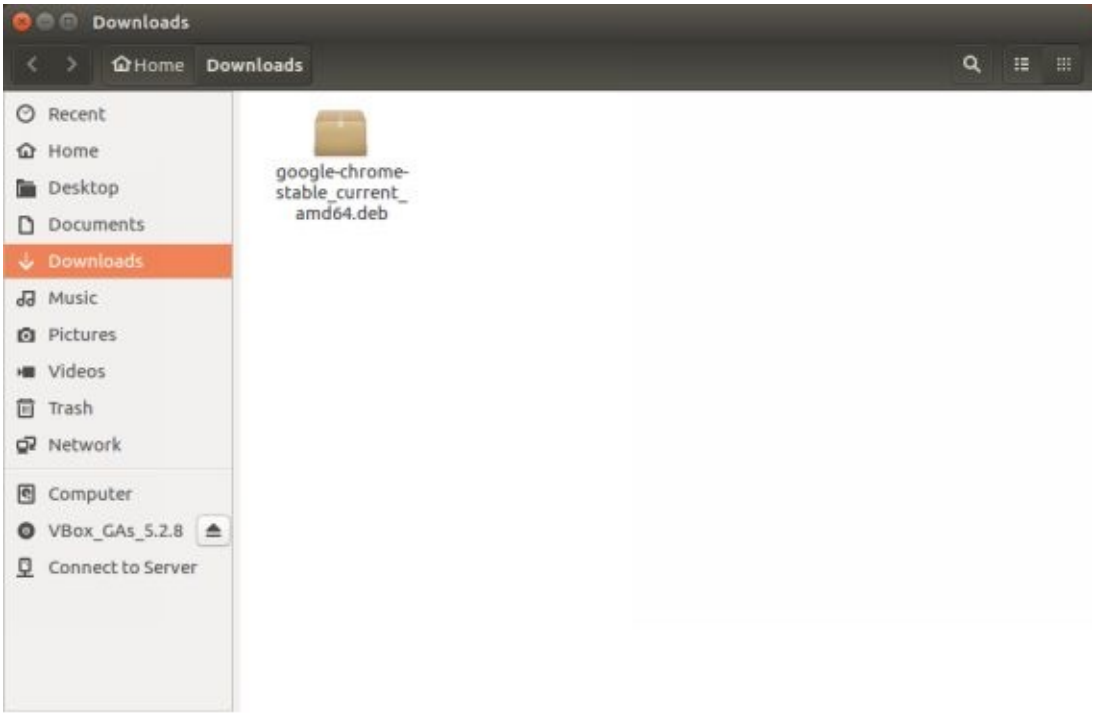
Click "ACCEPT AND INSTALL"



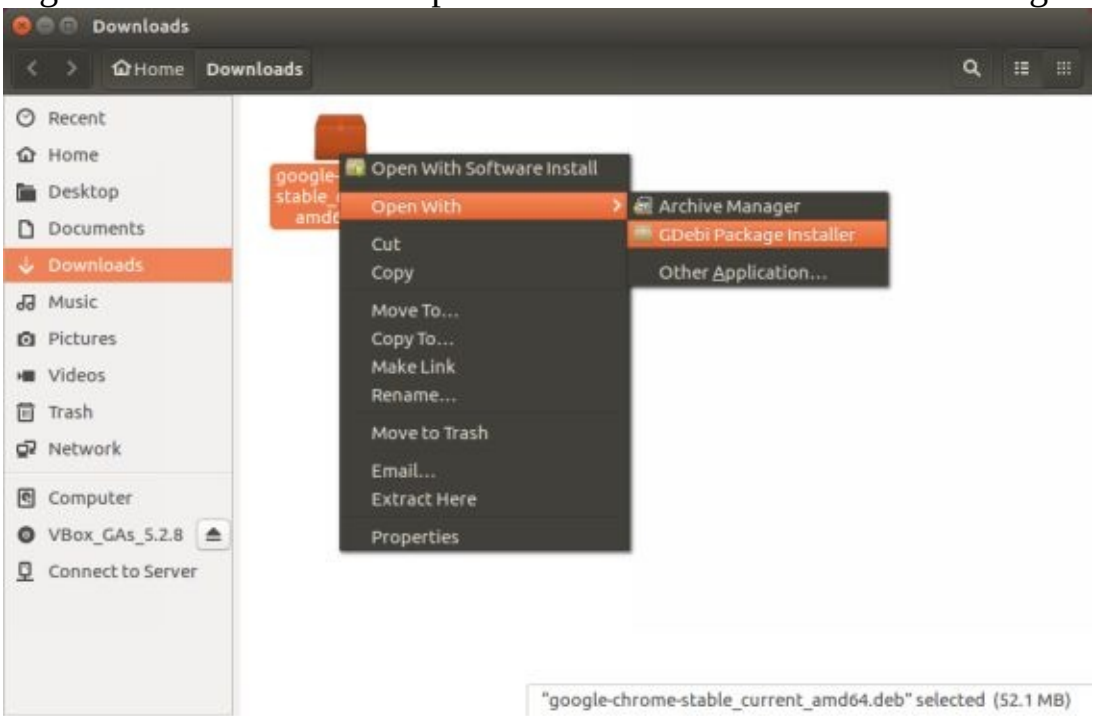
Check "Save File" and click "OK"



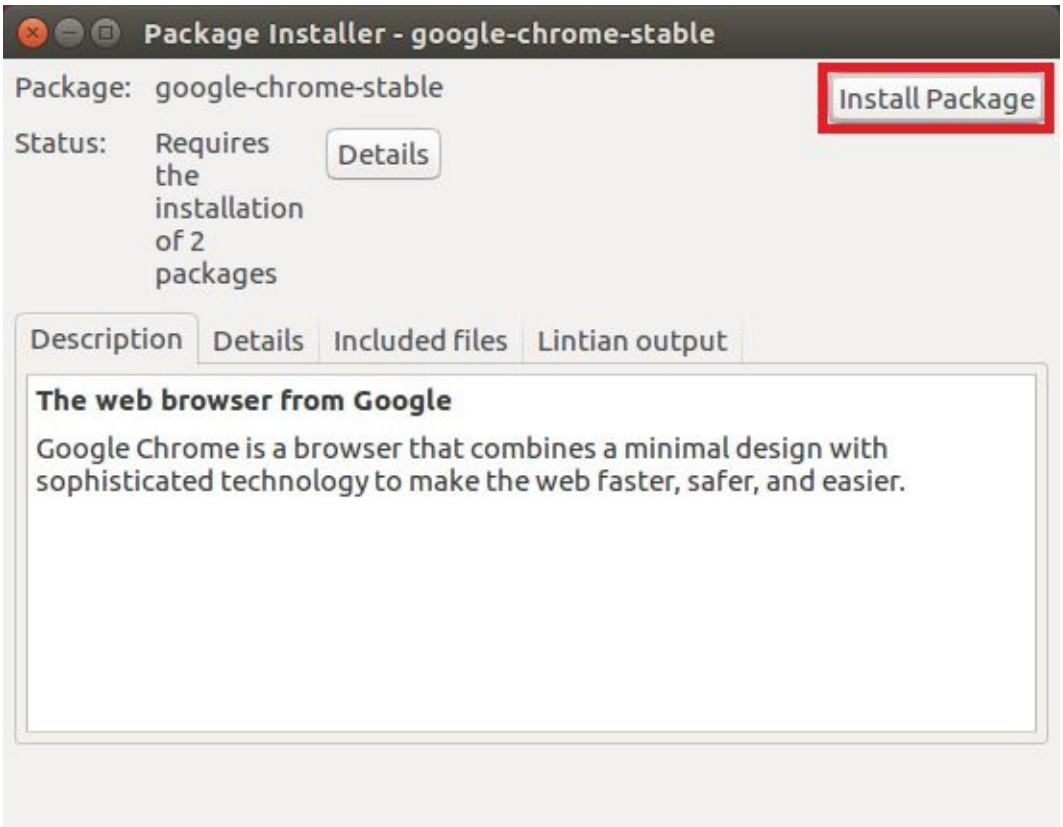
Open "Downloads" folder



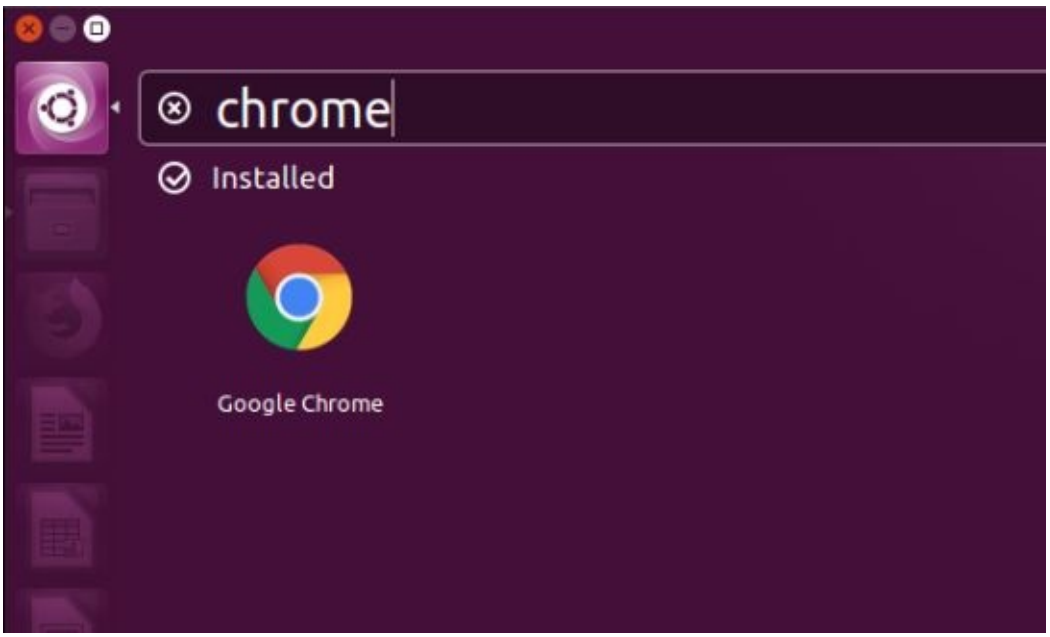
Right click on the file and open the menu and select "GDebi Package Installer"



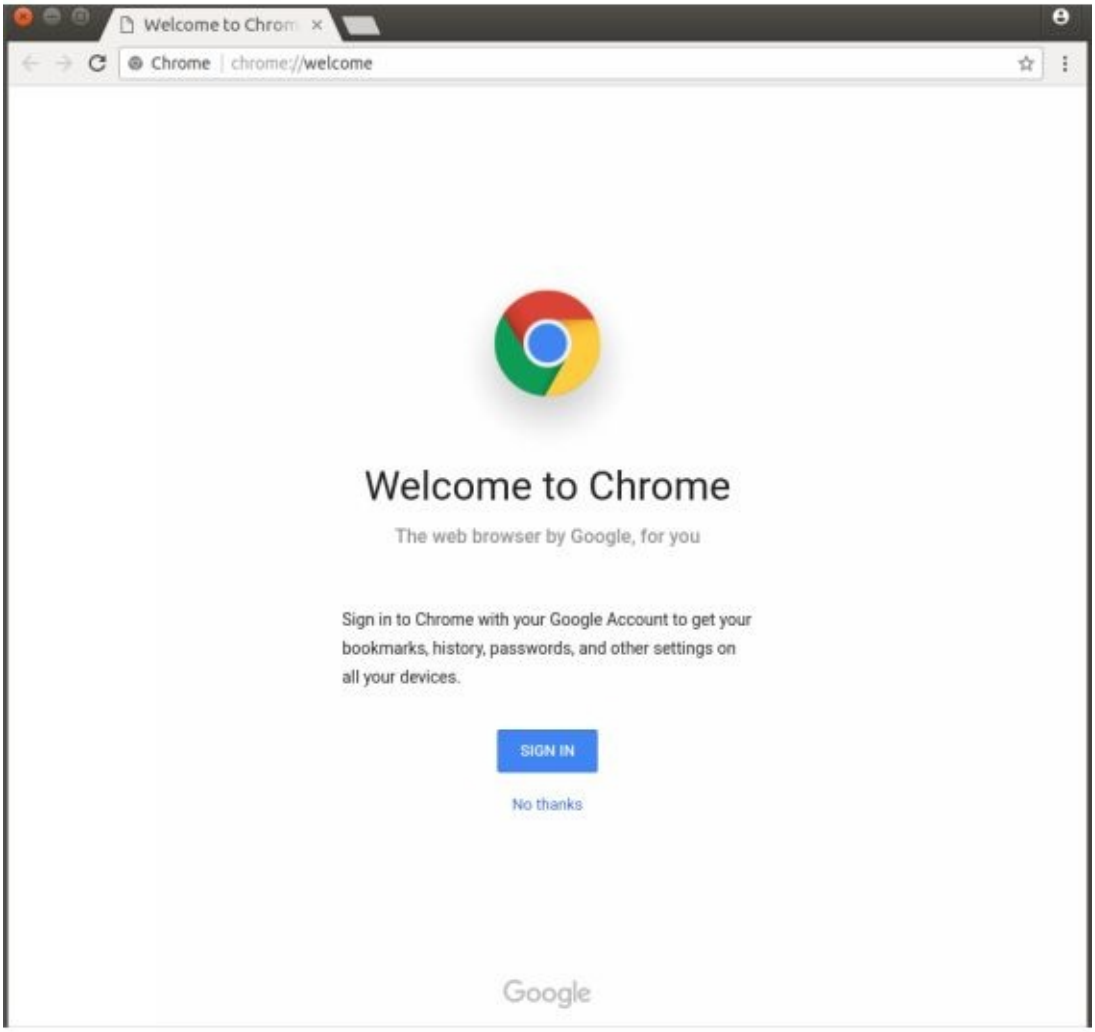
Click "Install Package"



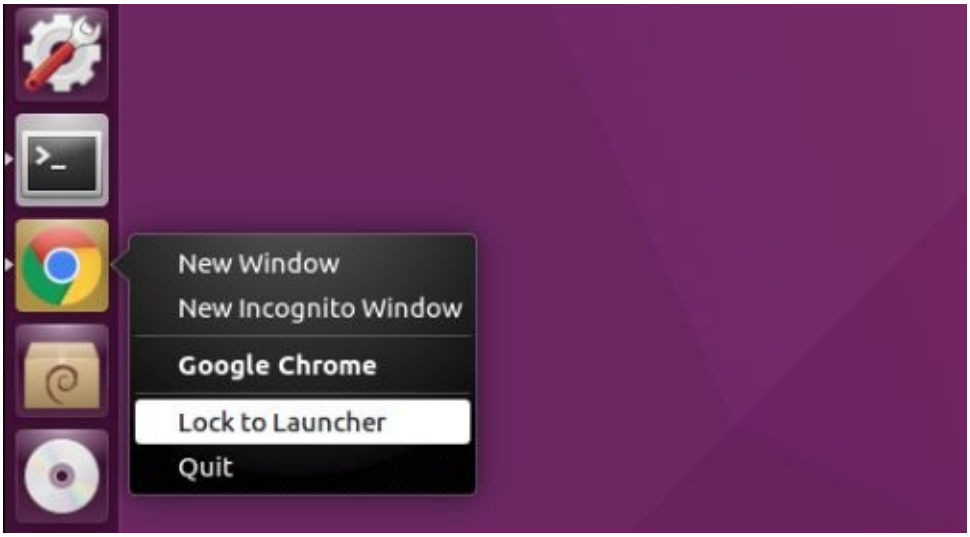
Search for Chrome with "Search your computer" icon and select "Google Chrome" icon



Chrome starts up.

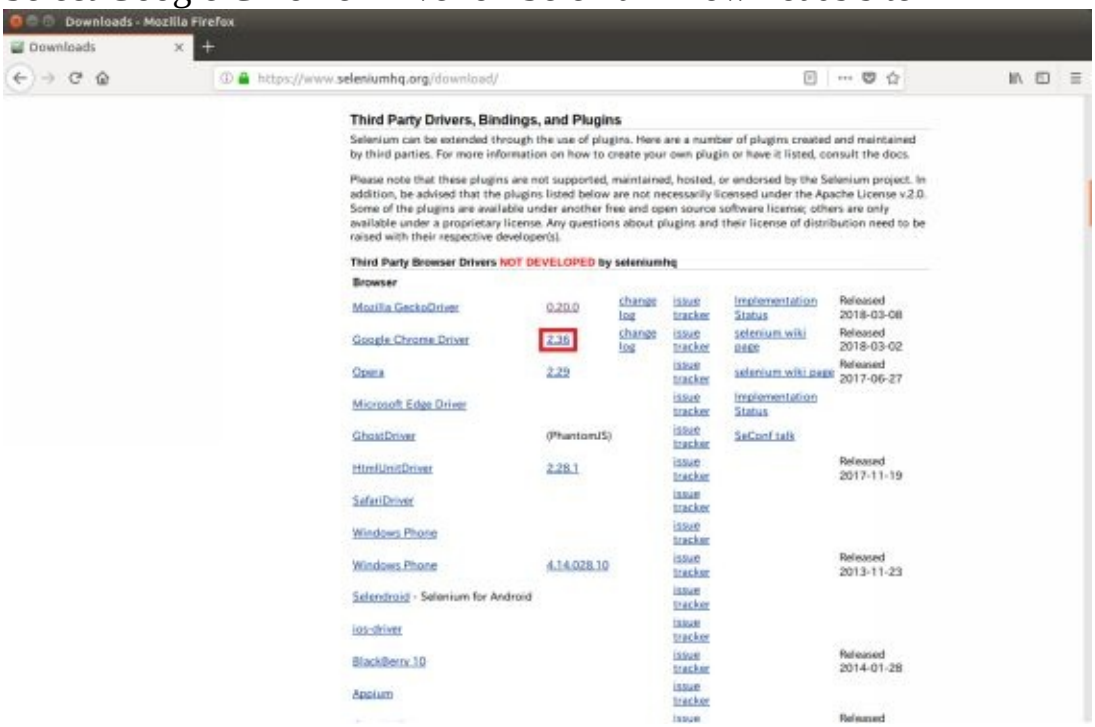


It is convenient to lock to Launcher.

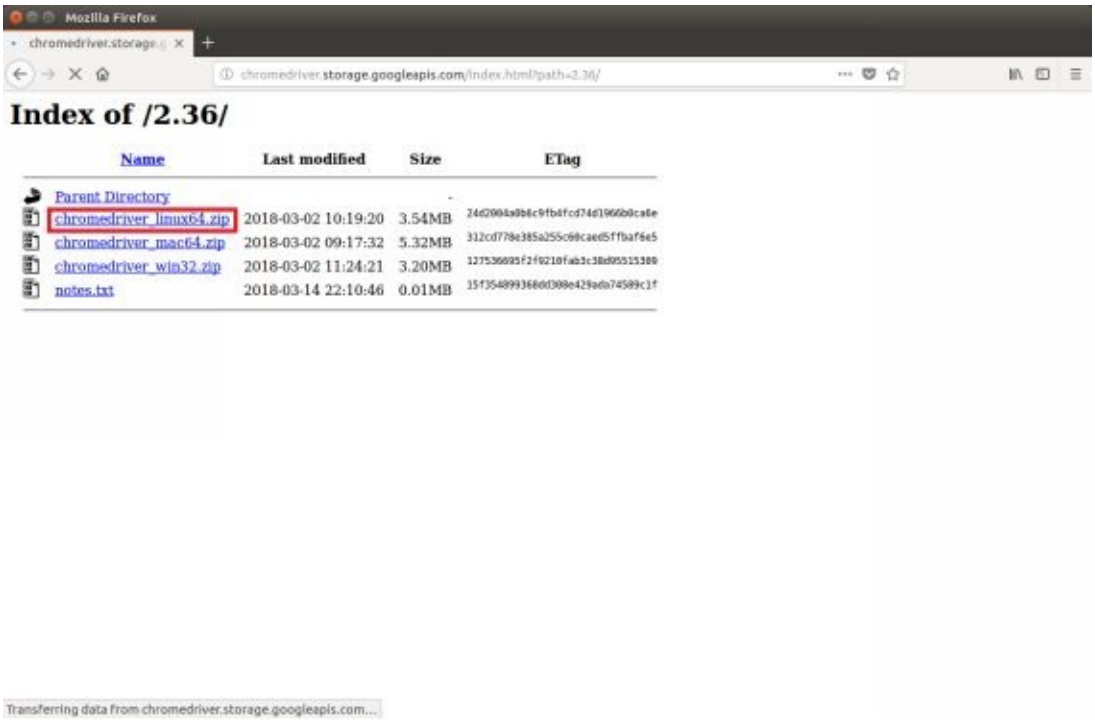


Next, we prepare Google Chrome Driver.

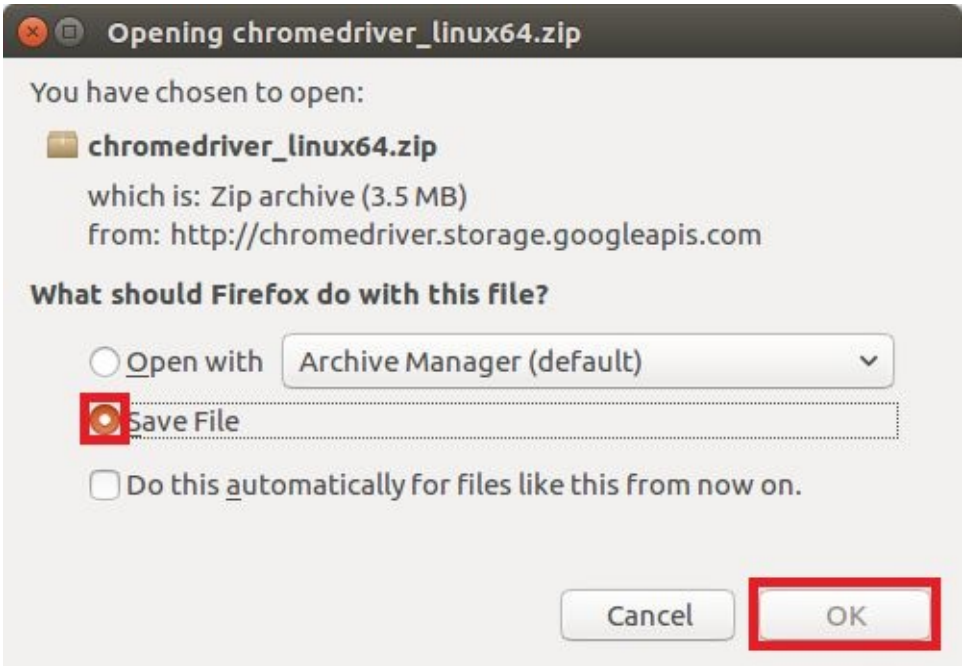
Select Google Chrome Driver on Selenium Downloads site



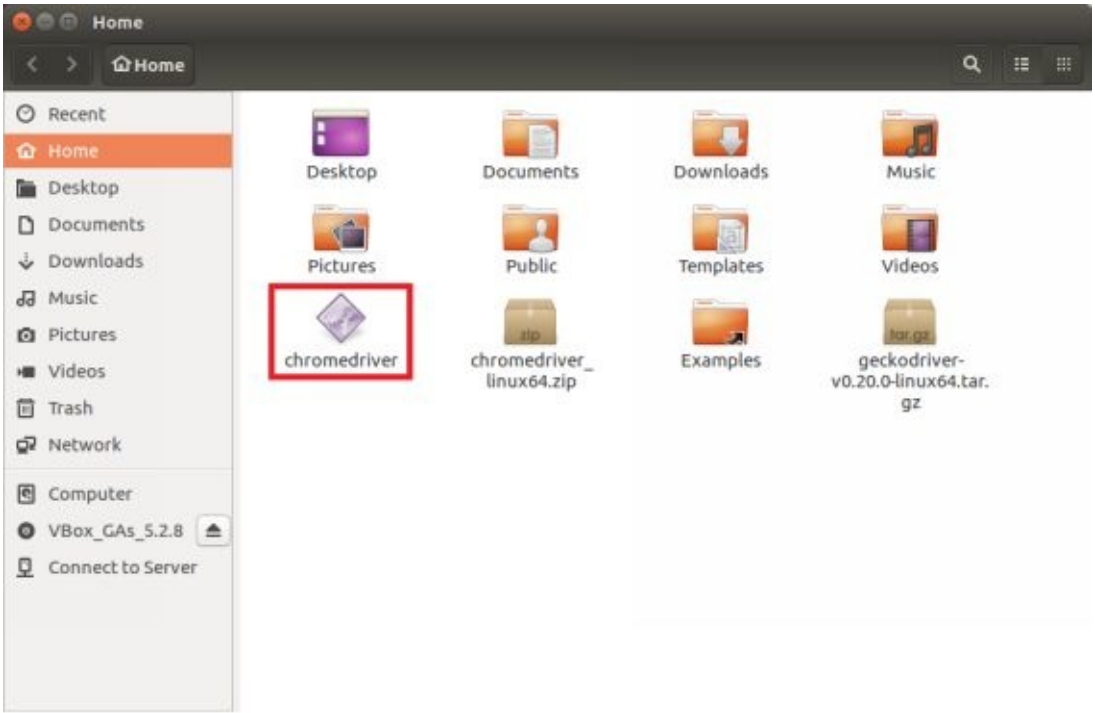
Select driver to download



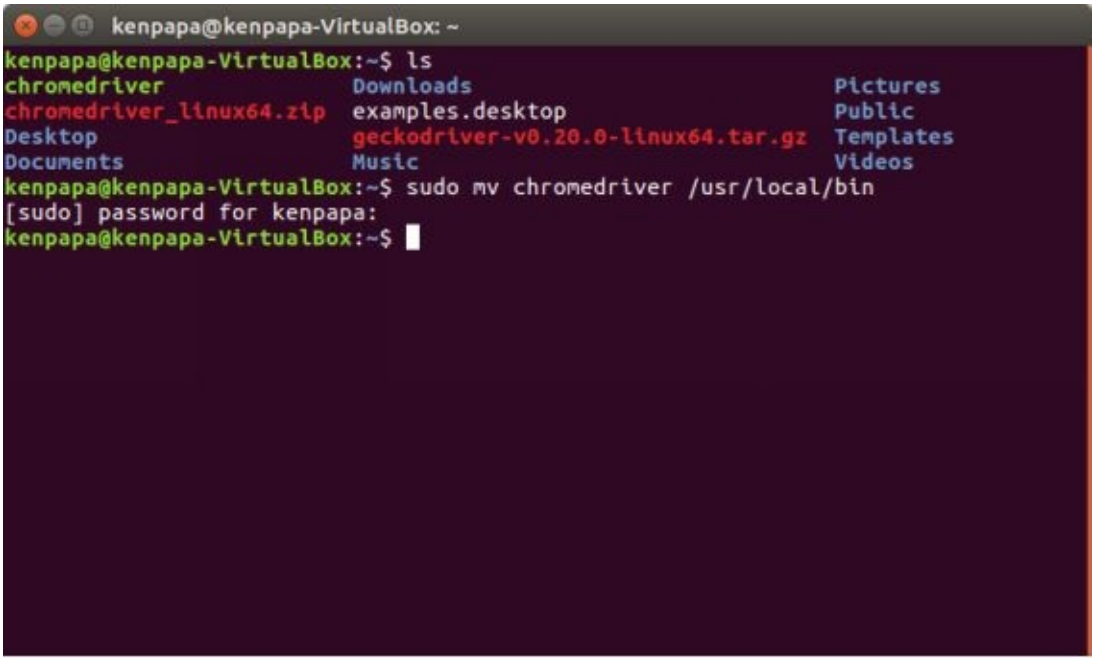
Check "Save File" and click "OK"



Extract the file after moving to Home



Move executable file to "/usr/local/bin"



This completes the preparation of Chrome.

2-2 Browser operation using JsonWireProtocol

When automatic operation is performed with Selenium WebDriver, Selenium Client internally uses JsonWireProtocol to interact with browser driver so that automatic operation is performed.

Although we do not need to be conscious of JsonWireProtocol when using Selenium WebDriver, here we use JsonWireProtocol to interact with the browser driver directly to see what the Selenium Client is doing internally.

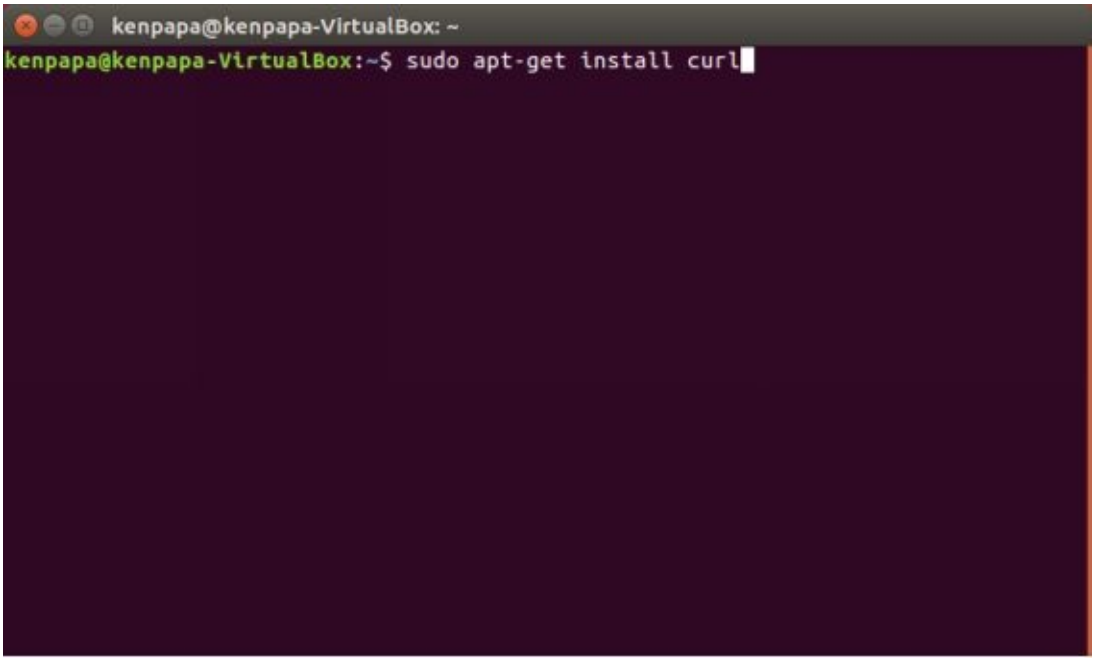
※If you are not interested in JsonWireProtocol, you can skip this section.

Since we use "curl" command here, installation is necessary in advance.

How to install curl

```
sudo apt-get install curl
```

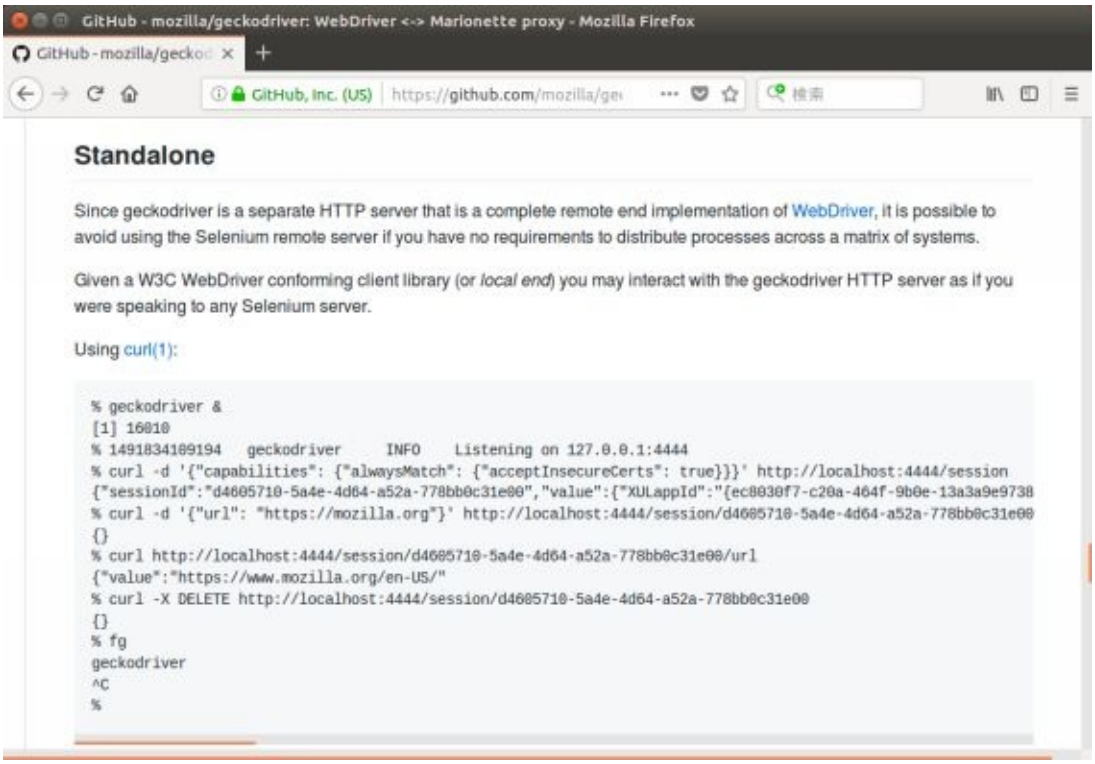
Install curl



2-2-1 Operate Mozilla Firefox using JsonWireProtocol

At first, we interact with Firefox driver (geckodriver) directly to operate Firefox.

The way to operate Firefox with only geckodriver is described on the geckodriver site shown below.



https://github.com/mozilla/geckodriver

Here we perform the following automatic operations.
Launch Firefox
Open http://www.google.com
Get Title

The JsonWireProtocol commands corresponding to each of the above operations are as follows.
POST /session
POST /session/:sessionId/url
GET /session/:sessionId/title

The commands actually executed using curl are as follows.
curl -d '{"capabilities": {"alwaysMatch": {"acceptInsecureCerts": true}}}' http://localhost:4444/session curl -d '{"url": "http://www.google.com"}' http://localhost:4444/session/:sessionId/url
curl http://localhost:4444/session/:sessionId/title

geckodriver &

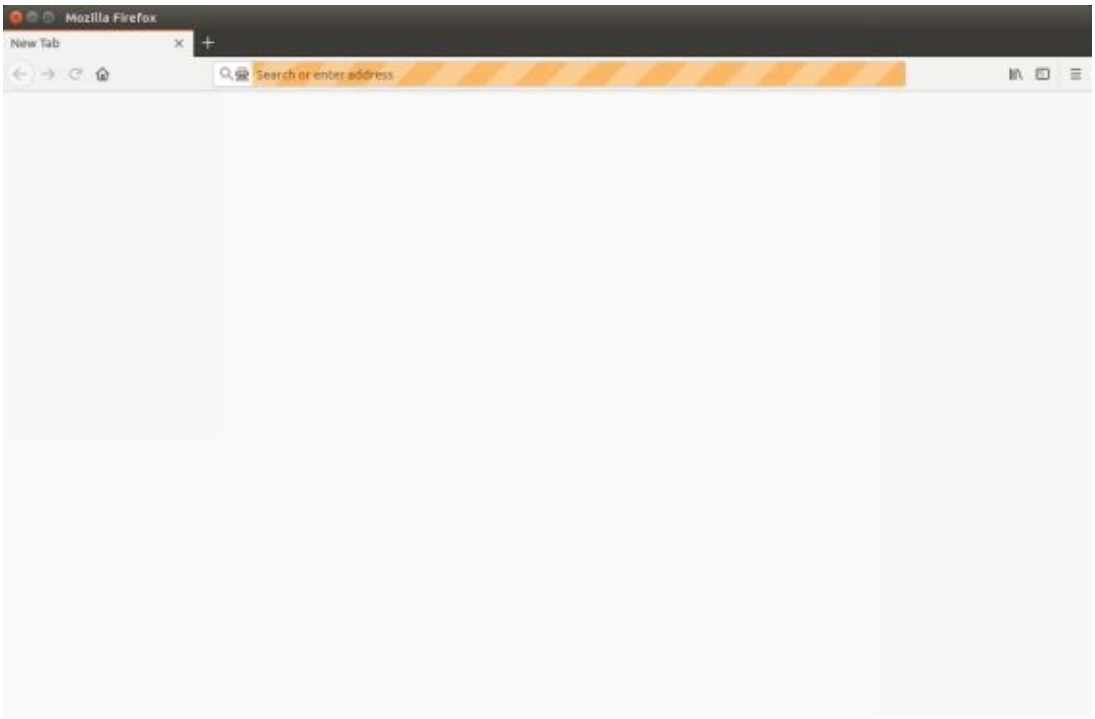
```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ geckodriver &
[1] 7229
kenpapa@kenpapa-VirtualBox:~$ 1521966531971     geckodriver     INFO    geckodri
ver 0.20.0
1521966531988   geckodriver     INFO    Listening on 127.0.0.1:4444
```

curl -d '{"capabilities": {"alwaysMatch": {"acceptInsecureCerts": true}}}' http://localhost:4444/session

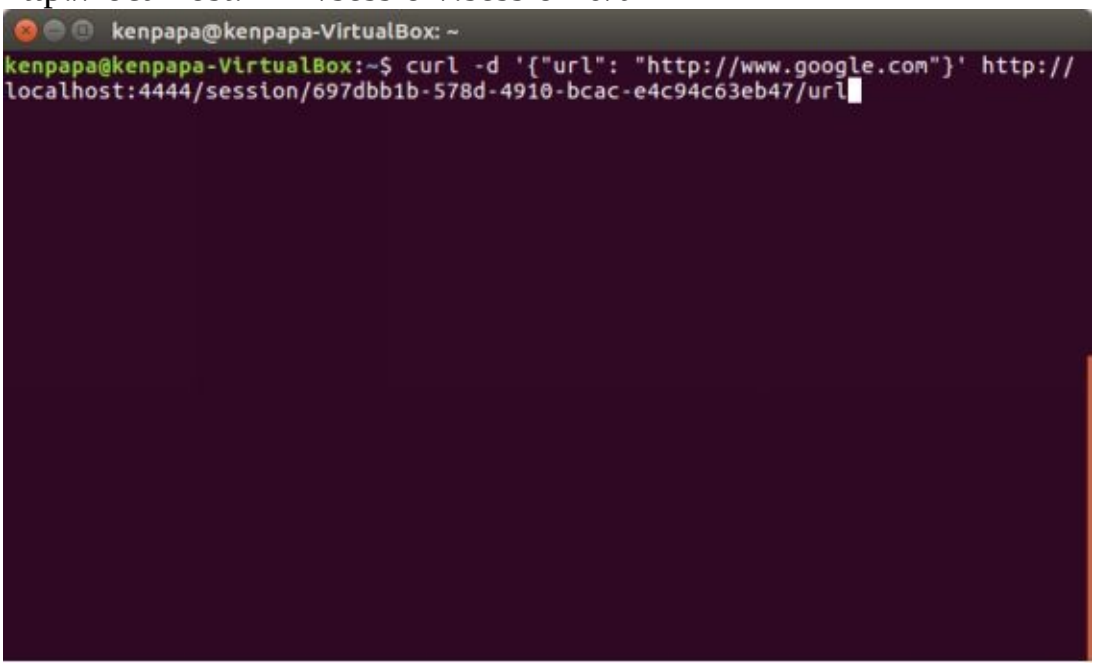
```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ geckodriver &
[1] 7229
kenpapa@kenpapa-VirtualBox:~$ 1521966531971     geckodriver     INFO    geckodri
ver 0.20.0
1521966531988   geckodriver     INFO    Listening on 127.0.0.1:4444

kenpapa@kenpapa-VirtualBox:~$ curl -d '{"capabilities": {"alwaysMatch": {"accept
InsecureCerts": true}}}' http://localhost:4444/session
1521966573101   mozrunner::runner       INFO    Running command: "/usr/bin/firef
ox" "-marionette" "-profile" "/tmp/rust_mozprofile.ex1GR0UMjngh"
1521966573890   Marionette      INFO    Enabled via --marionette
1521966576396   Marionette      INFO    Listening on port 45523
1521966576504   Marionette      WARN    TLS certificate errors will be ignored f
or this session
{"value": {"sessionId":"697dbb1b-578d-4910-bcac-e4c94c63eb47","capabilities":{"a
cceptInsecureCerts":true,"browserName":"firefox","browserVersion":"59.0.1","moz:
accessibilityChecks":false,"moz:headless":false,"moz:processID":7259,"moz:profil
e":"/tmp/rust_mozprofile.ex1GR0UMjngh","moz:useNonSpecCompliantPointerOrigin":fa
lse,"moz:webdriverClick":true,"pageLoadStrategy":"normal","platformName":"linux"
,"platformVersion":"4.13.0-37-generic","rotatable":false,"timeouts":{"implicit":
0,"pageLoad":300000,"script":30000}}}}kenpapa@kenpapa-VirtualBox:~$
```

We can get sessionId and it is used for subsequent commands.

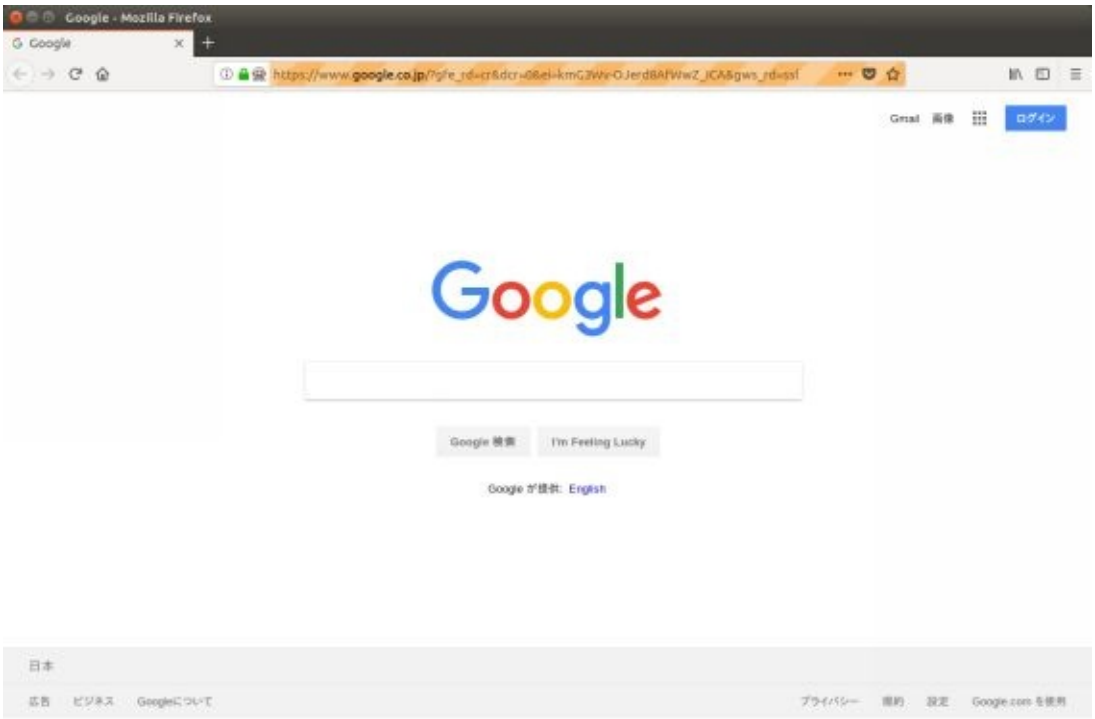
The browser starts up.



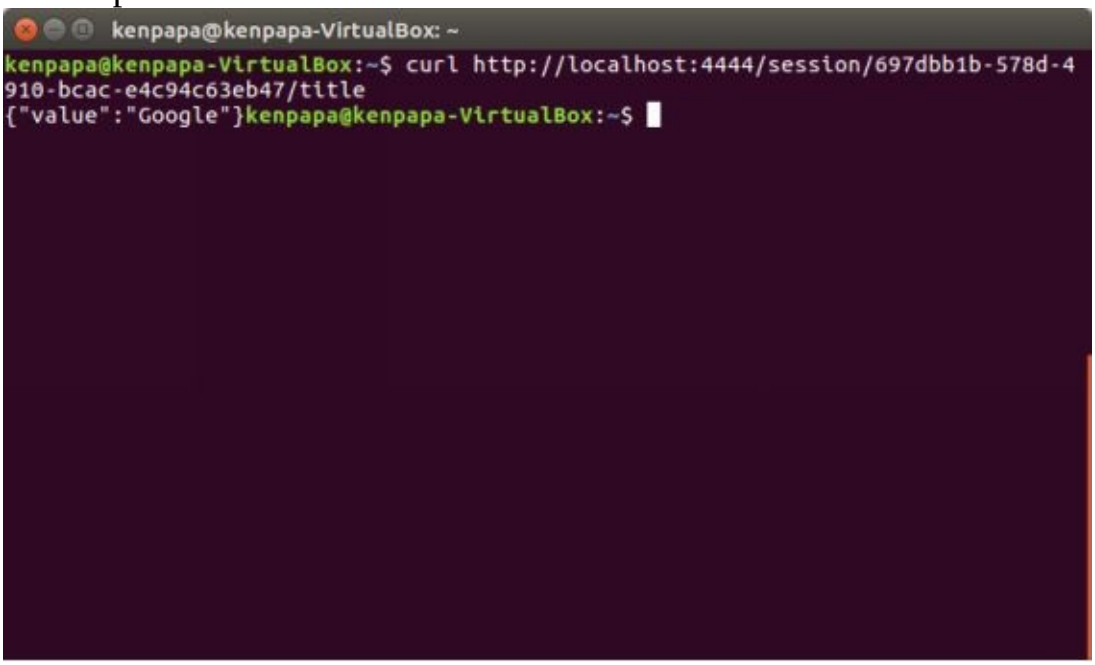
curl -d '{"url": "http://www.google.com"}' http://localhost:4444/session/:sessionId/url



"http://www.google.com" opens.



curl http://localhost:4444/session/:sessionId/title



We can get "Google" which is the title of "http://www.google.com".

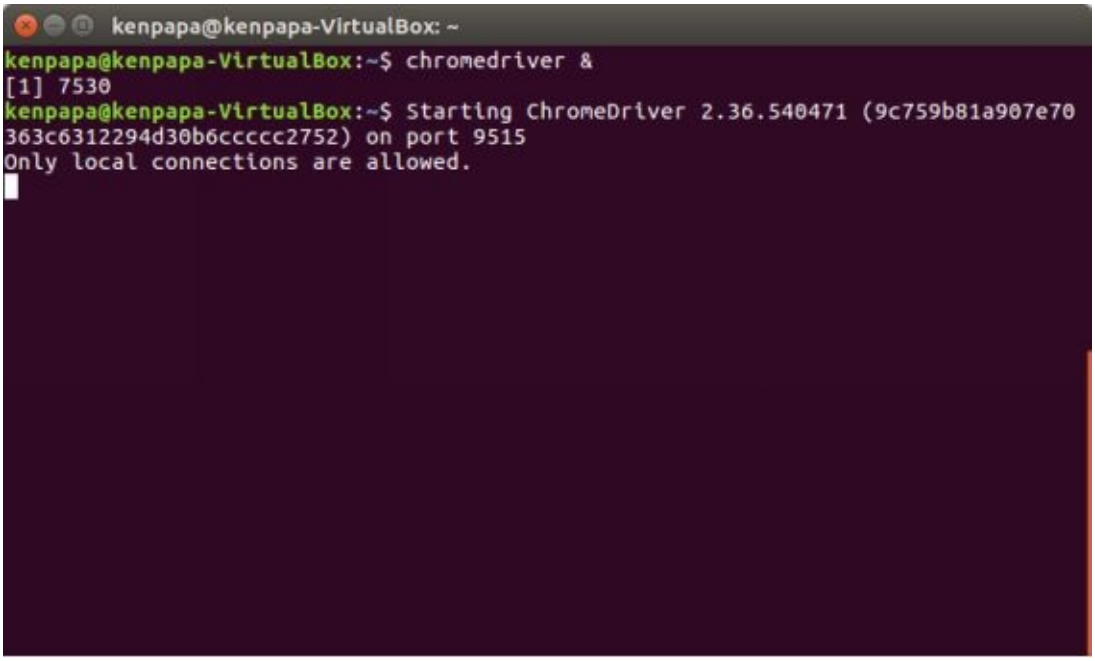
Quit geckodriver

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ curl http://localhost:4444/session/697dbb1b-578d-4
910-bcac-e4c94c63eb47/title
{"value":"Google"}kenpapa@kenpapa-VirtualBox:~$ fg
geckodriver
^C
kenpapa@kenpapa-VirtualBox:~$ [GFX1-]: Receive IPC close with reason=AbnormalShu
tdown
[Child 7352, Chrome_ChildThread] WARNING: pipe error (3): Connection reset by pe
er: file /build/firefox-CQifnS/firefox-59.0.1+build1/ipc/chromium/src/chrome/com
mon/ipc_channel_posix.cc, line 353
[GFX1-]: Receive IPC close with reason=AbnormalShutdown
[Child 7308, Chrome_ChildThread] WARNING: pipe error (3): Connection reset by pe
er: file /build/firefox-CQifnS/firefox-59.0.1+build1/ipc/chromium/src/chrome/com
mon/ipc_channel_posix.cc, line 353
```

2-2-2 Operate Google Chrome using JsonWireProtocol

Next, we interact with Chrome driver (chromedriver) directly to operate Chrome but it is almost the same procedure as Firefox driver.

chromedriver &

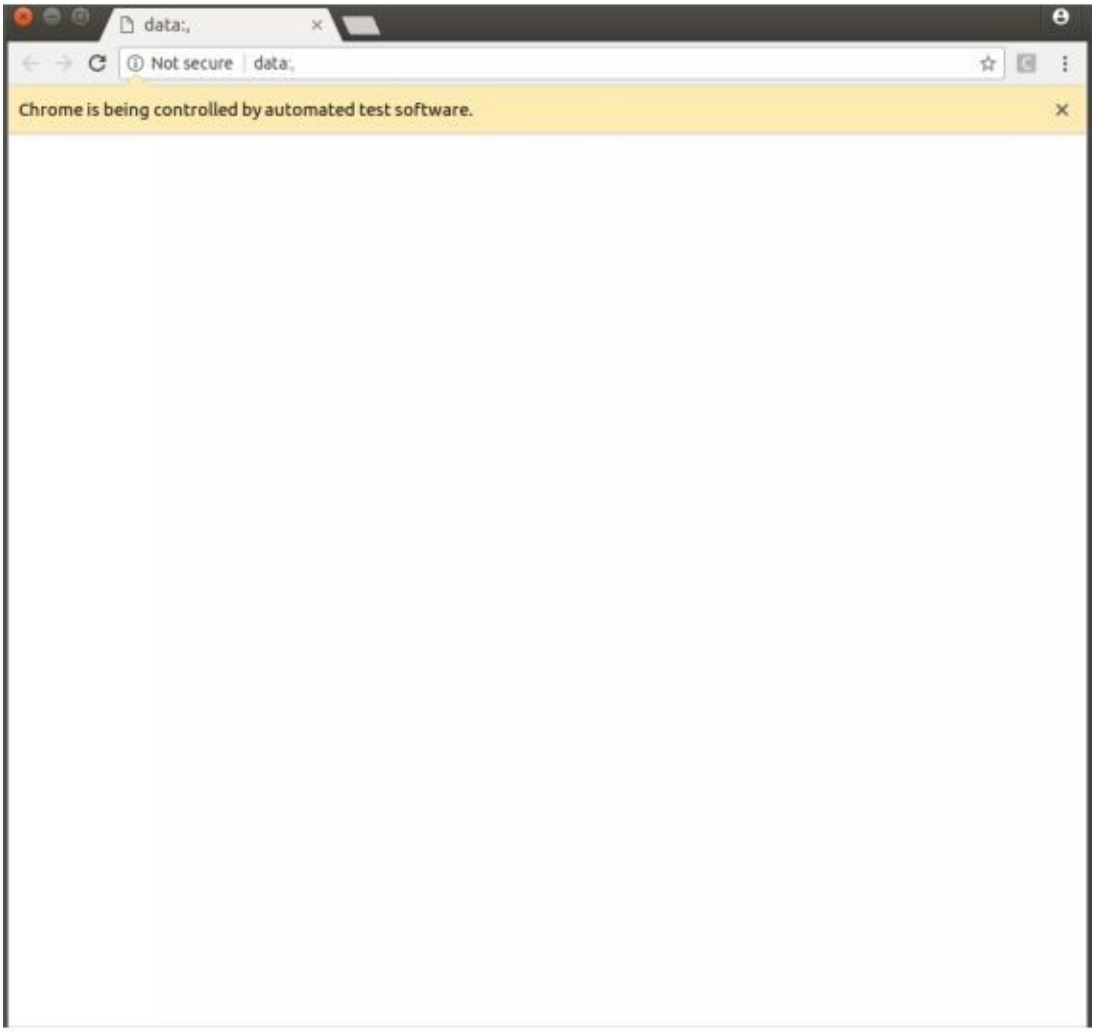


curl -d '{"desiredCapabilities": {}}' http://localhost:9515/session

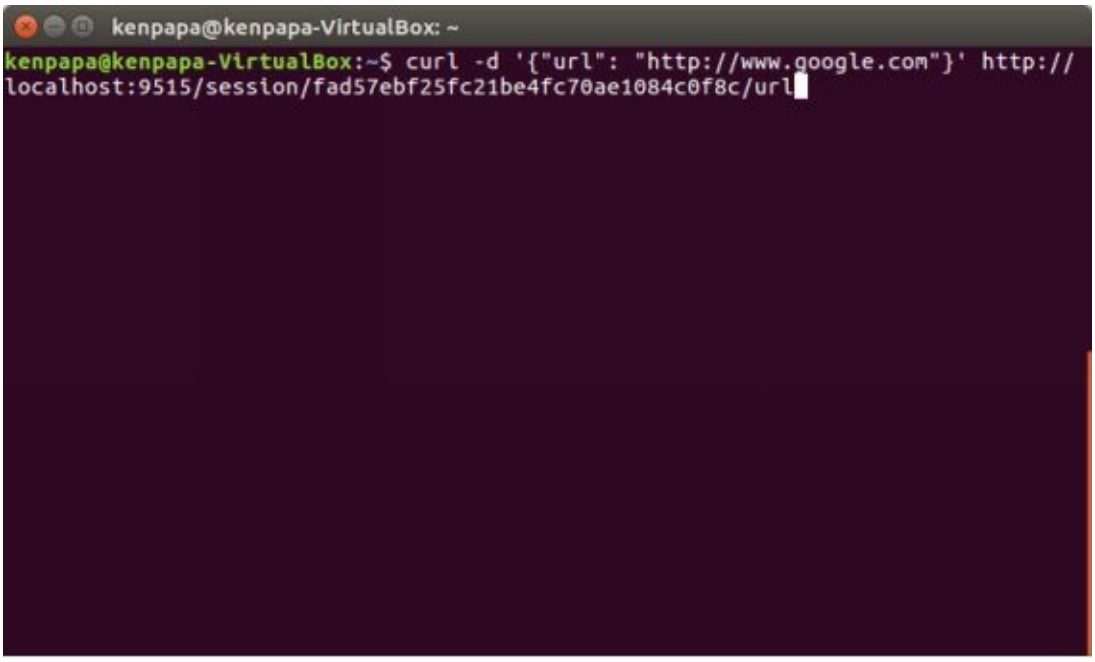
```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ chromedriver &
[1] 7530
kenpapa@kenpapa-VirtualBox:~$ Starting ChromeDriver 2.36.540471 (9c759b81a907e70
363c6312294d30b6ccccc2752) on port 9515
Only local connections are allowed.

kenpapa@kenpapa-VirtualBox:~$ curl -d '{"desiredCapabilities": {}}' http://local
host:9515/session
{"sessionId":"fad57ebf25fc21be4fc70ae1084c0f8c","status":0,"value":{"acceptInsec
ureCerts":false,"acceptSslCerts":false,"applicationCacheEnabled":false,"browserC
onnectionEnabled":false,"browserName":"chrome","chrome":{"chromedriverVersion":"
2.36.540471 (9c759b81a907e70363c6312294d30b6ccccc2752)","userDataDir":"/tmp/.org
.chromium.Chromium.cdCAgd"},"cssSelectorsEnabled":true,"databaseEnabled":false,"
handlesAlerts":true,"hasTouchScreen":false,"javascriptEnabled":true,"locationCon
textEnabled":true,"mobileEmulationEnabled":false,"nativeEvents":true,"networkCon
nectionEnabled":false,"pageLoadStrategy":"normal","platform":"Linux","rotatable"
:false,"setWindowRect":true,"takesHeapSnapshot":true,"takesScreenshot":true,"une
xpectedAlertBehaviour":"","version":"65.0.3325.181","webStorageEnabled":true}}ke
npapa@kenpapa-VirtualBox:~$
```

We can get sessionId and it is used for subsequent commands.

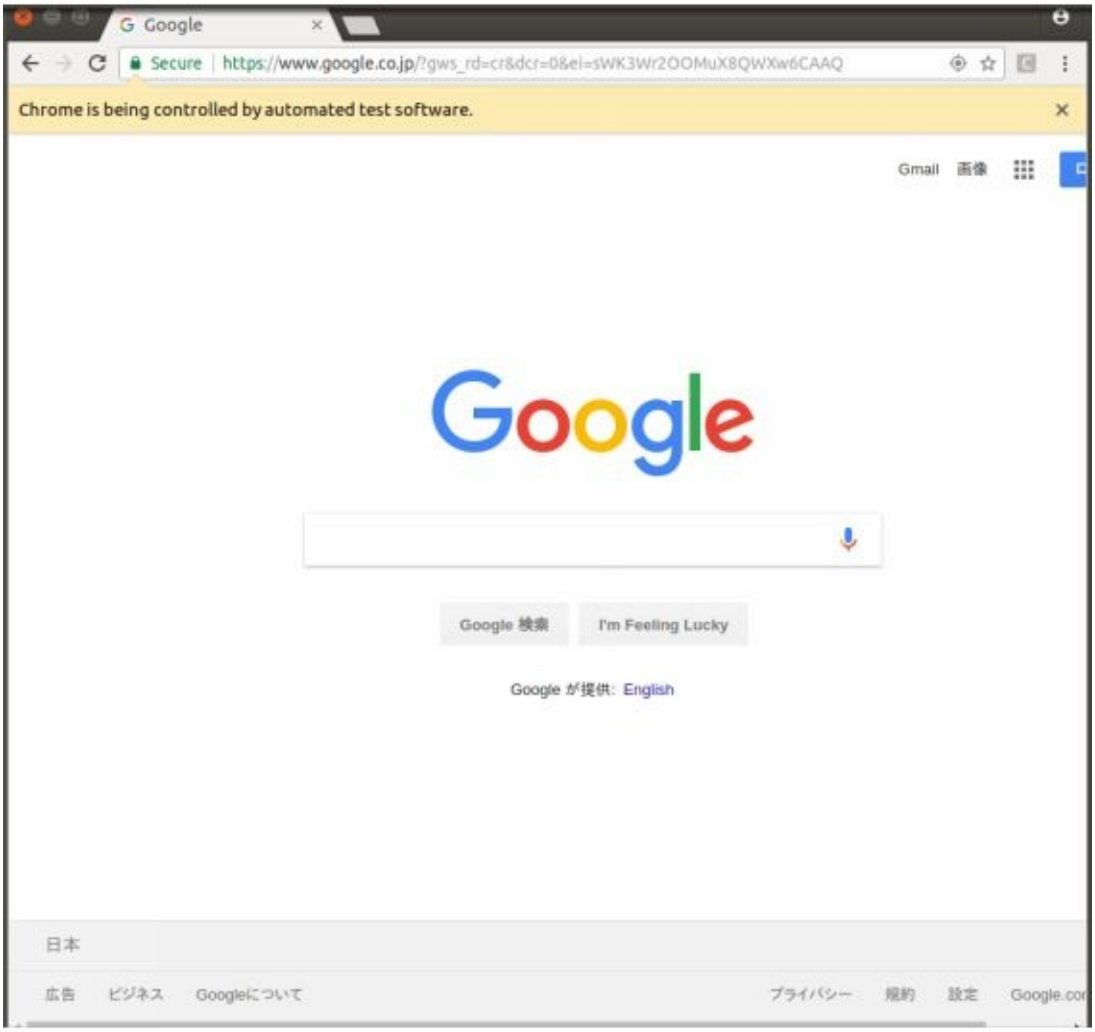
The browser starts up.



curl -d '{"url": "http://www.google.com"}' http://localhost:9515/session/:sessionId/url



"http://www.google.com" opens.



curl http://localhost:9515/session/:sessionId/title

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ curl http://localhost:9515/session/fad57ebf25fc21b
e4fc70ae1084c0f8c/title
{"sessionId":"fad57ebf25fc21be4fc70ae1084c0f8c","status":0,"value":"Google"}kenp
apa@kenpapa-VirtualBox:~$
```

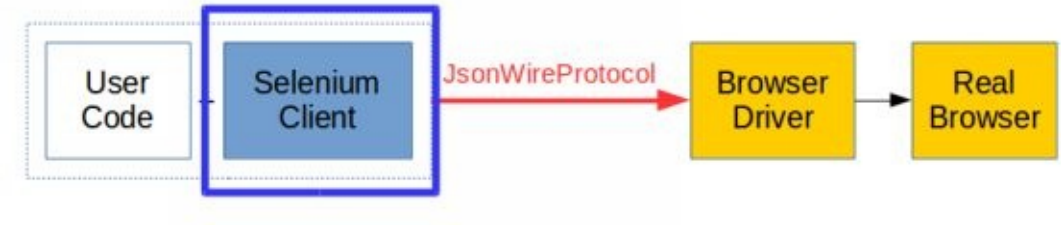
We can get "Google" which is the title of "http://www.google.com".

Quit chromedriver

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ curl http://localhost:9515/session/fad57ebf25fc21b
e4fc70ae1084c0f8c/title
{"sessionId":"fad57ebf25fc21be4fc70ae1084c0f8c","status":0,"value":"Google"}kenp
apa@kenpapa-VirtualBox:~$ fg
chromedriver
^C
kenpapa@kenpapa-VirtualBox:~$
```

2-3 Prepare Selenium Client

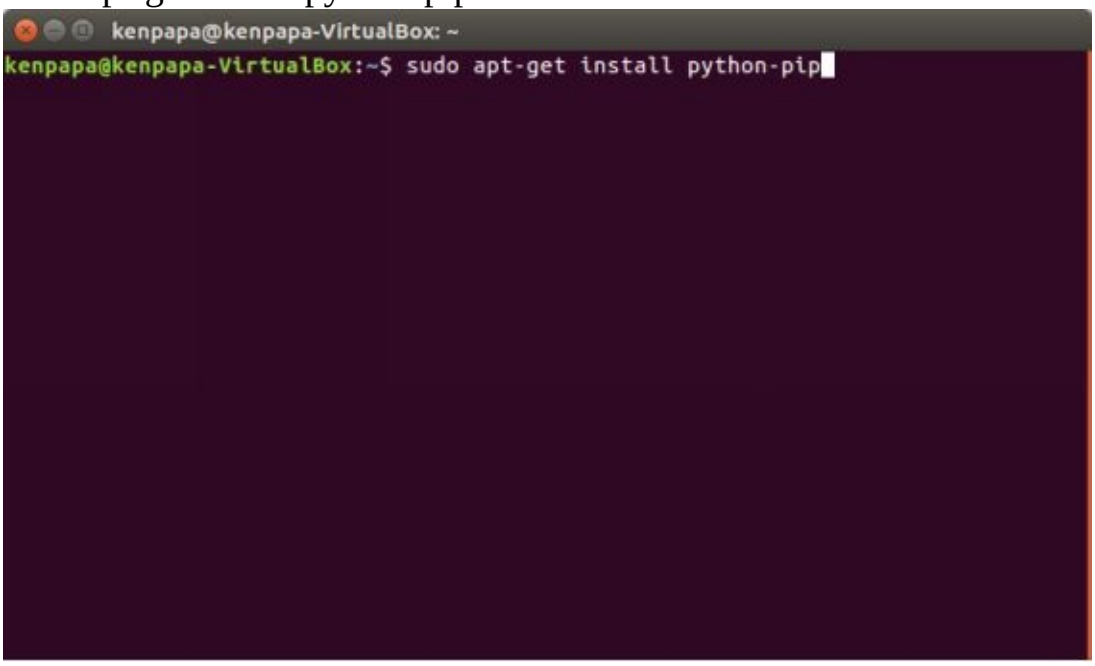
Here we prepare the environment surrounded by blue line of below figure.



Installation procedure of Selenium Client

```
sudo apt-get install python-pip
sudo pip install selenium
```

sudo apt-get install python-pip



sudo pip install selenium

```
kenpapa@kenpapa-VirtualBox: ~
Unpacking python-pkg-resources (20.7.0-1) ...
Selecting previously unselected package python-setuptools.
Preparing to unpack .../python-setuptools_20.7.0-1_all.deb ...
Unpacking python-setuptools (20.7.0-1) ...
Selecting previously unselected package python-wheel.
Preparing to unpack .../python-wheel_0.29.0-1_all.deb ...
Unpacking python-wheel (0.29.0-1) ...
Processing triggers for doc-base (0.10.7) ...
Processing 1 added doc-base file...
Processing triggers for man-db (2.7.5-1) ...
Setting up libexpat1-dev:amd64 (2.1.0-7ubuntu0.16.04.3) ...
Setting up libpython2.7-dev:amd64 (2.7.12-1ubuntu0~16.04.3) ...
Setting up libpython-dev:amd64 (2.7.12-1~16.04) ...
Setting up libpython-all-dev:amd64 (2.7.12-1~16.04) ...
Setting up python-all (2.7.12-1~16.04) ...
Setting up python2.7-dev (2.7.12-1ubuntu0~16.04.3) ...
Setting up python-dev (2.7.12-1~16.04) ...
Setting up python-all-dev (2.7.12-1~16.04) ...
Setting up python-pip-whl (8.1.1-2ubuntu0.4) ...
Setting up python-pip (8.1.1-2ubuntu0.4) ...
Setting up python-pkg-resources (20.7.0-1) ...
Setting up python-setuptools (20.7.0-1) ...
Setting up python-wheel (0.29.0-1) ...
kenpapa@kenpapa-VirtualBox:~$ sudo pip install selenium
```

selenium-3.11.0 has been installed.

```
kenpapa@kenpapa-VirtualBox: ~
Setting up python-dev (2.7.12-1~16.04) ...
Setting up python-all-dev (2.7.12-1~16.04) ...
Setting up python-pip-whl (8.1.1-2ubuntu0.4) ...
Setting up python-pip (8.1.1-2ubuntu0.4) ...
Setting up python-pkg-resources (20.7.0-1) ...
Setting up python-setuptools (20.7.0-1) ...
Setting up python-wheel (0.29.0-1) ...
kenpapa@kenpapa-VirtualBox:~$ sudo pip install selenium
The directory '/home/kenpapa/.cache/pip/http' or its parent directory is not own
ed by the current user and the cache has been disabled. Please check the permiss
ions and owner of that directory. If executing pip with sudo, you may want sudo'
s -H flag.
The directory '/home/kenpapa/.cache/pip' or its parent directory is not owned by
 the current user and caching wheels has been disabled. check the permissions an
d owner of that directory. If executing pip with sudo, you may want sudo's -H fl
ag.
Collecting selenium
  Downloading selenium-3.11.0-py2.py3-none-any.whl (943kB)
    100% |[                                    ]| 952kB 747kB/s
Installing collected packages: selenium
Successfully installed selenium-3.11.0
You are using pip version 8.1.1, however version 9.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
kenpapa@kenpapa-VirtualBox:~$
```

2-4 Automatic operation with simple code

Now that we are ready to use the Selenium WebDriver for automatic browser operation. We perform automatic operation of Firefox and Chrome with simple code.

Create the following two files.

google_search_firefox.py

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
```

google_search_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()
```

First of all, execute "google_search_firefox.py" with python command.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ ls
google_search_chrome.py  google_search_firefox.py
kenpapa@kenpapa-VirtualBox:~/python$ python google_search_firefox.py
```

As a result, Firefox starts up and Google Search is executed. The below is the search result screen.

Next, execute "google_search_chrome.py" with python command.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ ls
google_search_chrome.py  google_search_firefox.py
kenpapa@kenpapa-VirtualBox:~/python$ python google_search_firefox.py
kenpapa@kenpapa-VirtualBox:~/python$ python google_search_chrome.py
```

As a result, Chrome starts up and Google Search is executed. The below is the search result screen.

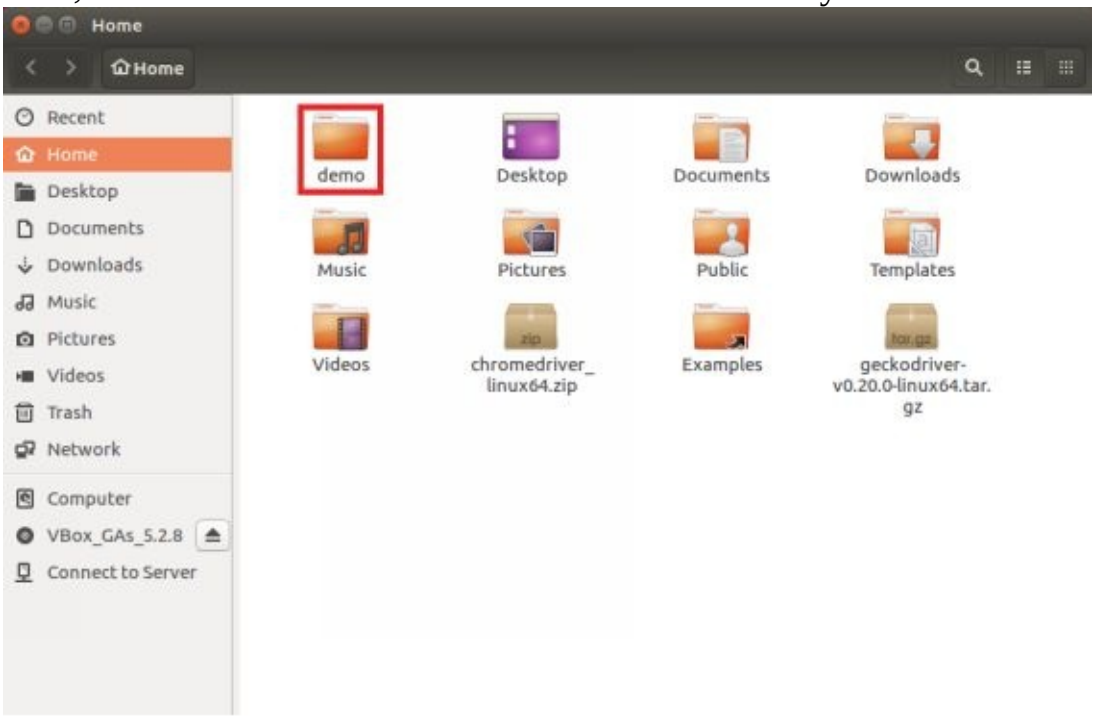
In this chapter, we prepared an environment for automatic operation of browser using Selenium WebDriver.

Chapter 3 Basic practice of automatic operation

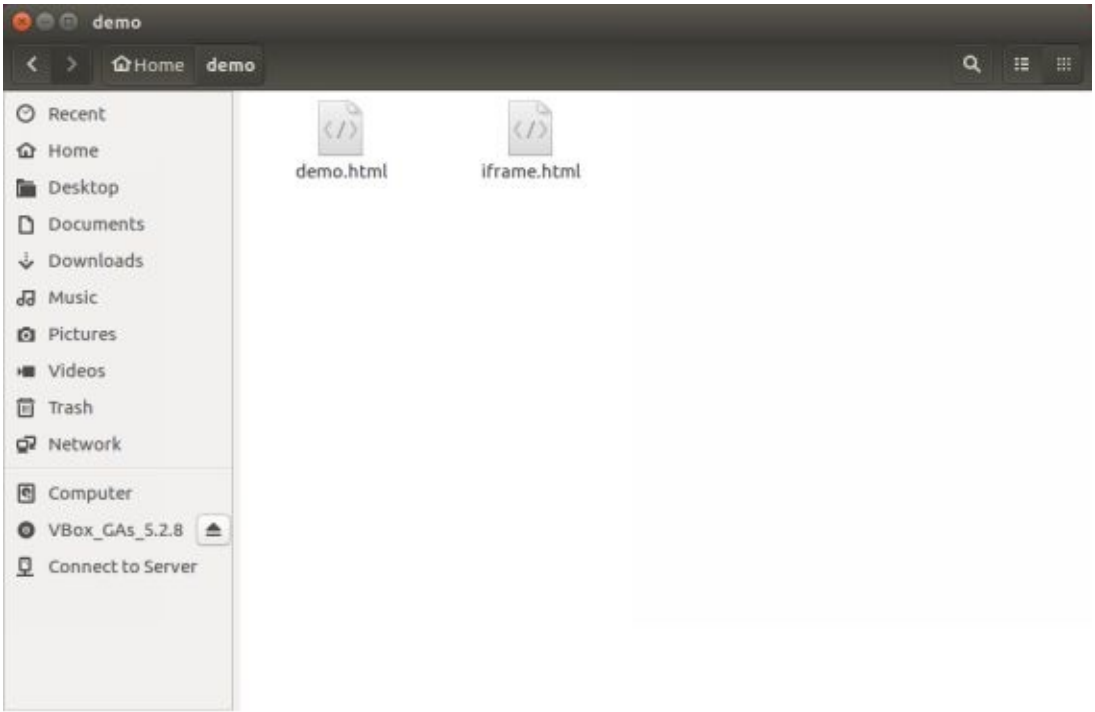
In this chapter, we perform automatic operation of demo site as a basic practice of automatic operation using Selenium WebDriver.

3-1 Prepare demo site

First, create a folder named "demo" at Home directory.



Create "demo.html" and "iframe.html" in demo folder.



The contents of each file are as follows.

demo.html

```
<!DOCTYPE html>
<html>
<head>
   <title>Demo Site</title>
</head>
<body>
   <button id="button" name="Button" onclick="clicked()">Button</button>
<script>
   function clicked() {
   document.getElementById("button").style.color = "red";
   }
   </script>

   <div>
   <a class="Google_Japan_URL" href="http://www.google.co.jp">Google</a>
   </div>
   <br>

   <form>
   <div>
   <input type="text" id="firstname" name="FirstName" placeholder="First Name"> <input type="text" id="lastname" name="LastName"placeholder="Last Name"> </div>
   <br>

   <div>
   <label><input type="radio" name="genderoptions" value="Male">Male</label> <label><input type="radio" name="genderoptions" value="FeMale">FeMale</label> </div>
   <br>

   <div>
   <div><input type="checkbox" value="Dog"><label>Dog</label></div> <div><input type="checkbox" value="Cat"><label>Cat</label></div> <div> <input type="checkbox" value="Ferret"><label>Ferret</label></div> </div>
```

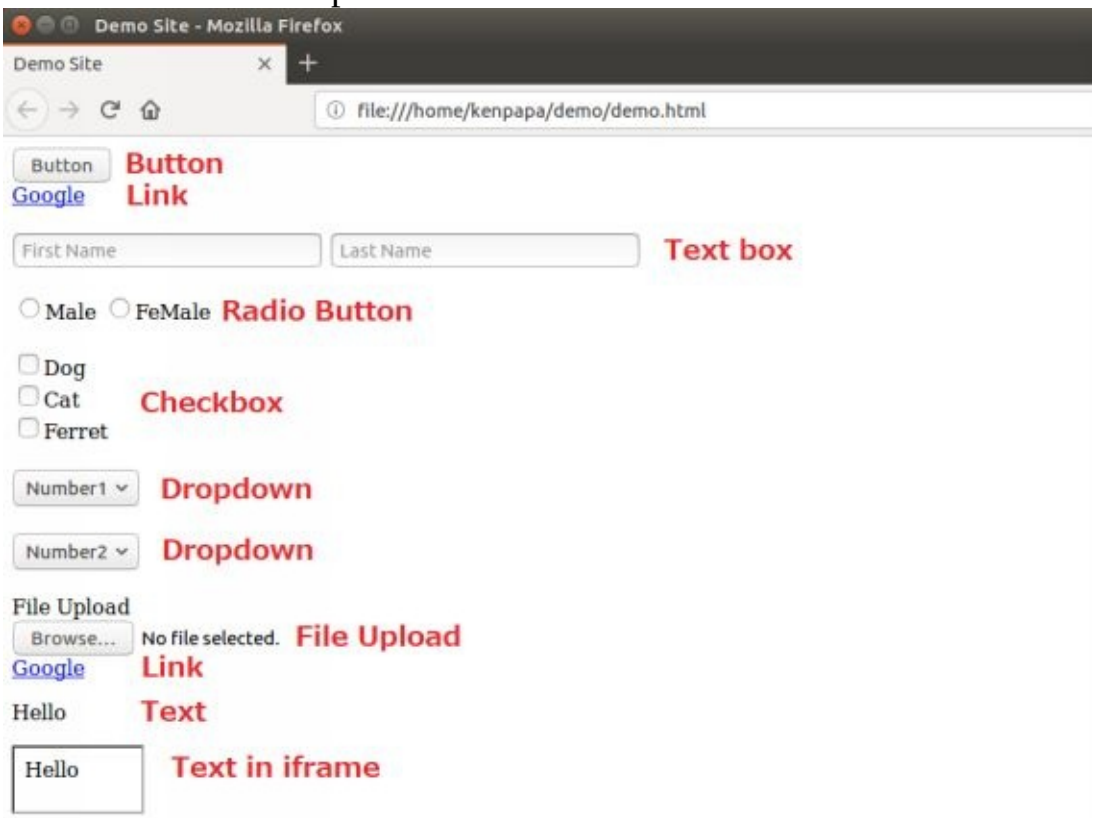
```
<br>

<div>
<select type="text" placeholder="Number1">
<option value="">Number1</option>
<option>0</option>
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
</select>
</div>
<br>

<div>
<select type="text" placeholder="Number2">
<option value="">Number2</option>
<option>0</option>
<option>1</option>
<option>2</option>
<option>3</option>
<option>4</option>
<option>5</option>
</select>
</div>
<br>
</form>

<label>File Upload</label>
<div>
<input id="uploadfile" type="file">
</div>

<div>
<a class="Google_USA_URL" href="http://www.google.com">Google</a>
</div>

<div>
```

```
<p id="inner">Hello</p>
</div>

<div>
<iframe src="iframe.html" width=100 height=50 id="content_ifr">
</iframe> </div>

</body>
</html>
```

iframe.html

```
<p id="inner">Hello</p>
```

The demo site is composed of various elements as shown below.



As for above demo site, the notes on operating with Selenium are as follows.

There are two "Google" links
There are two Dropdown menus

→To specify the second element, use the advanced pattern of xpath

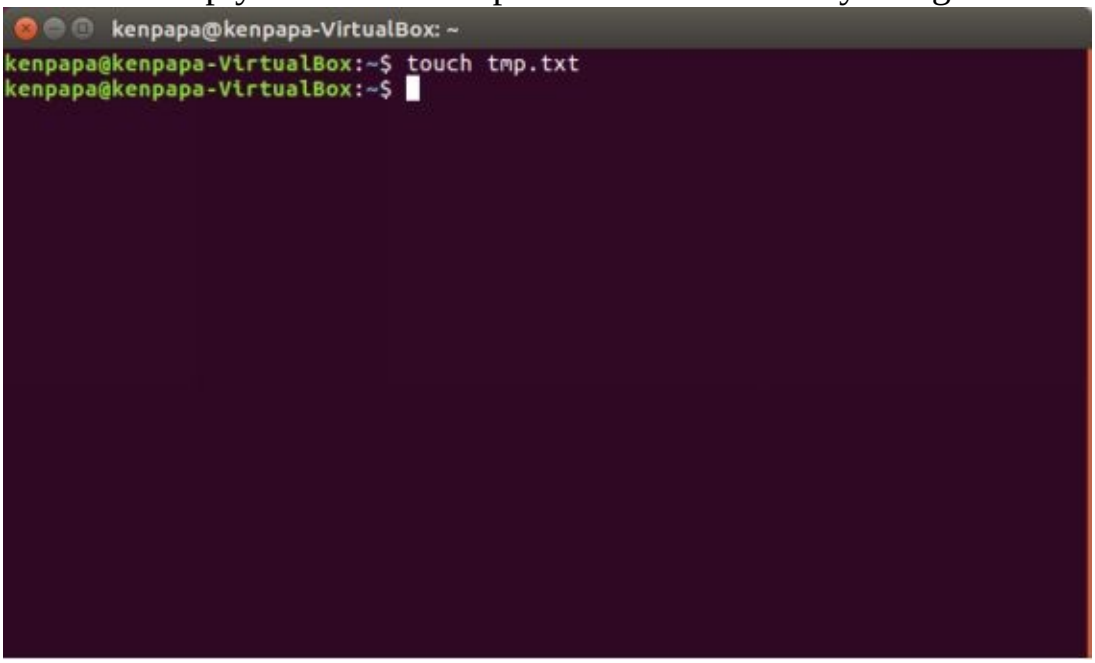
"Browse" button for file upload exists
→Use "send_keys" instead of "click"

Text exists
→Use Javascript to change text

Text in iframe exists
→Change the text using Javascript after switching frame

Next prepare the file to be used for the file upload operation.

Create an empty file named "tmp.txt" at home directory using touch command.

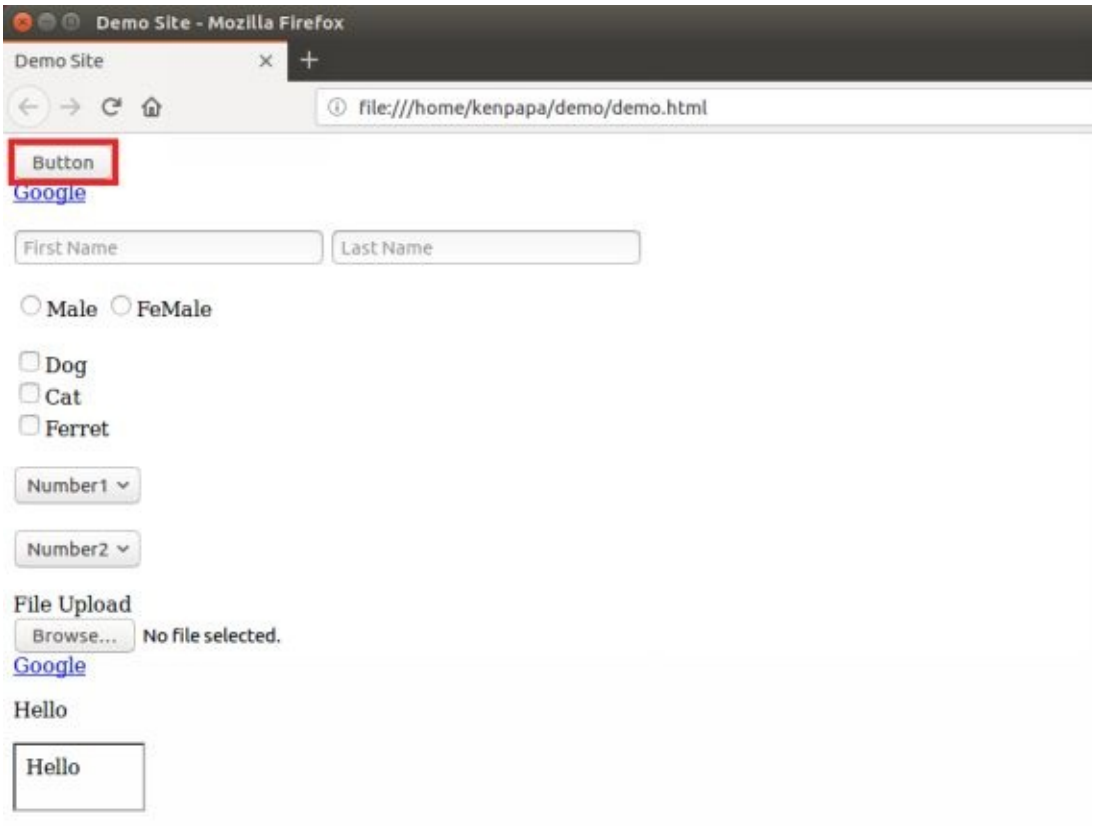


This is the end of the preparations.

3-2 Create automatic operation code for operation target element

Here we create an automatic operation code from the HTML of the operation target element. The procedure is as follows.
Identify element (id attribute/name attribute/link text/xpath)
Determine the operation method of the element

■Click button



HTML of operation target element

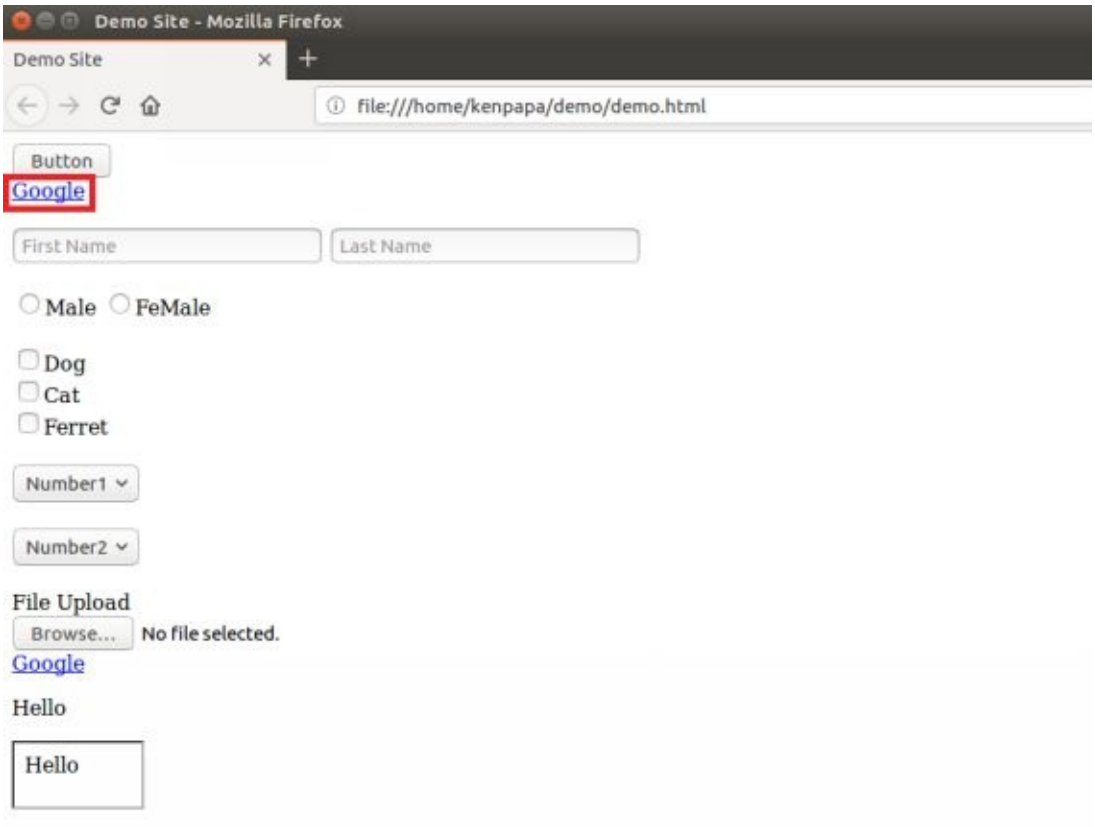
```
<button id="button" name="Button" onclick="clicked()">Button</button>
```

There is an id attribute and a name attribute but here we identify the element by the id attribute.

Automatic operation code

```
driver.find_element_by_id("button").click()
```

■Click link (Google Japan URL)



HTML of operation target element

```
<a class="Google_Japan_URL" href="http://www.google.co.jp">Google</a>
```

Since there is a link text, we identify the element by the link text.

Automatic operation code (link text)

```
driver.find_element_by_link_text("Google").click()
```

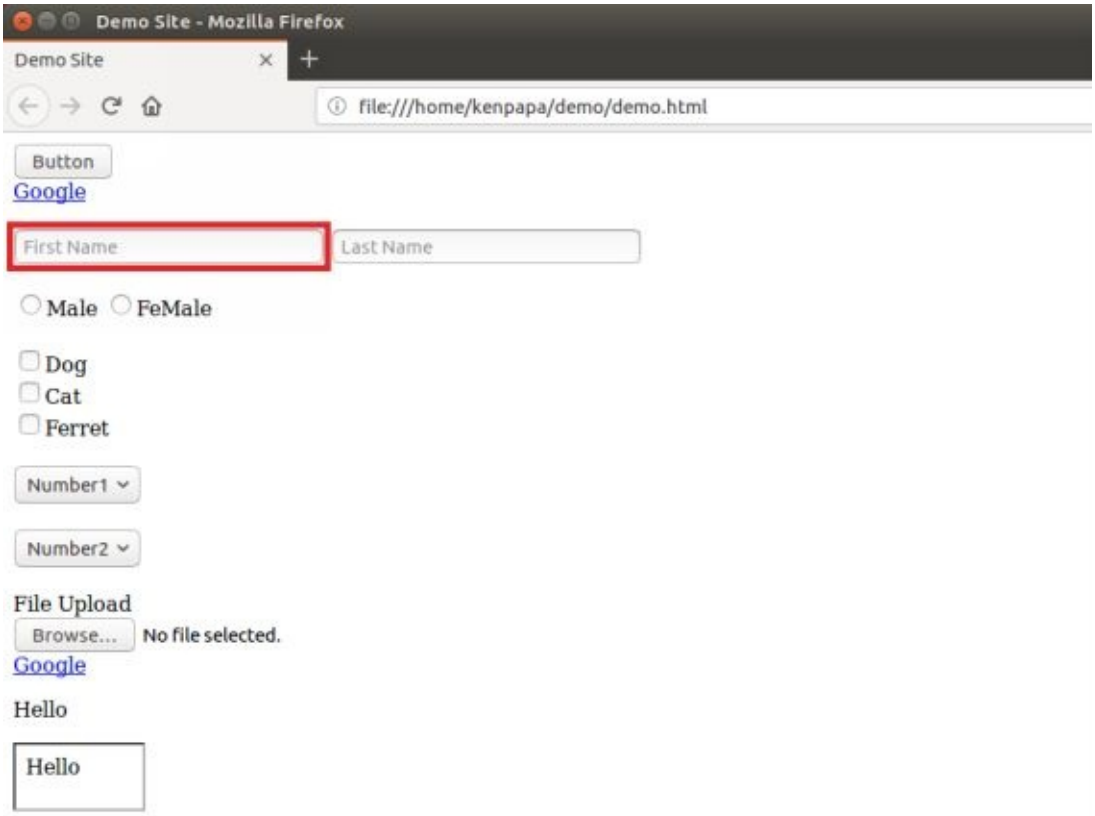
Also, using xpath we can identify element as follows.

Automatic operation code (xpath)

```
driver.find_element_by_xpath("//a[text()='Google']").click()
```

find_element returns the first found element, so this element is identified.

■Enter text box (First Name)



HTML of operation target element

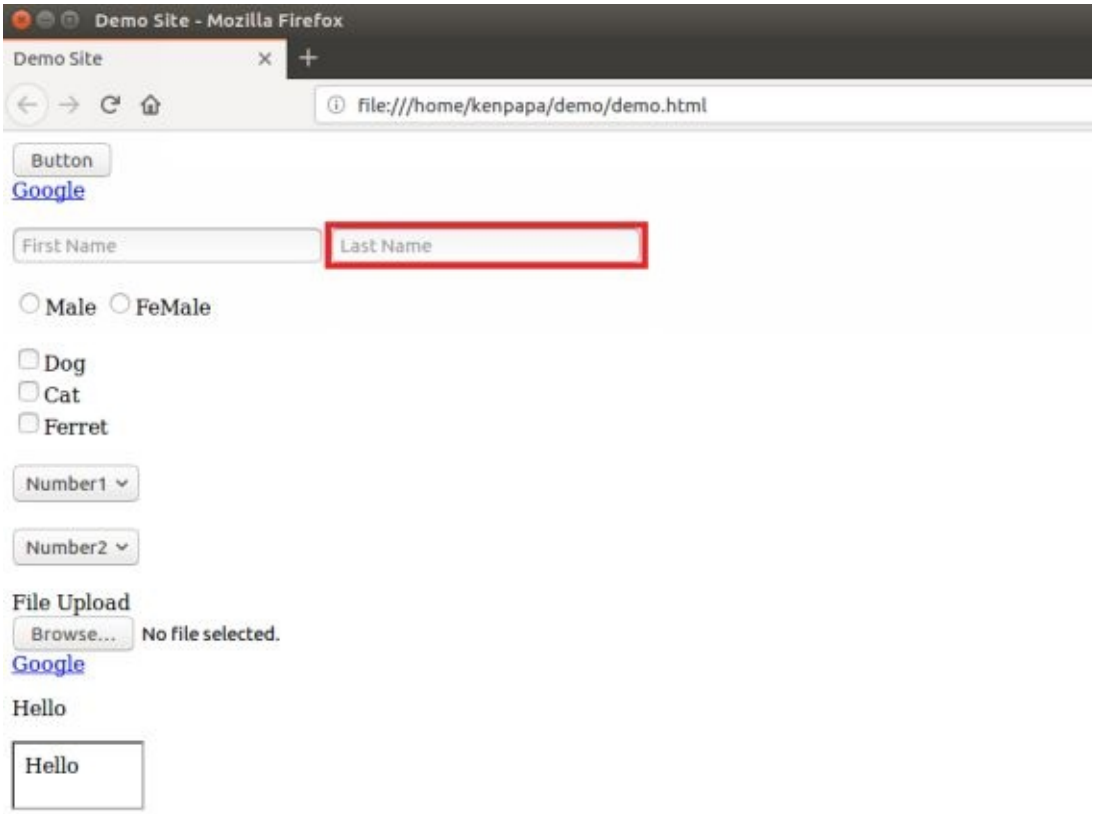
```
<input type="text" id="firstname" name="FirstName" placeholder="First Name">
```

There is an id attribute and a name attribute but here we identify the element by the id attribute.

Automatic operation code

```
driver.find_element_by_id("firstname").send_keys("Taro")
```

■Enter text box (Last Name)



HTML of operation target element

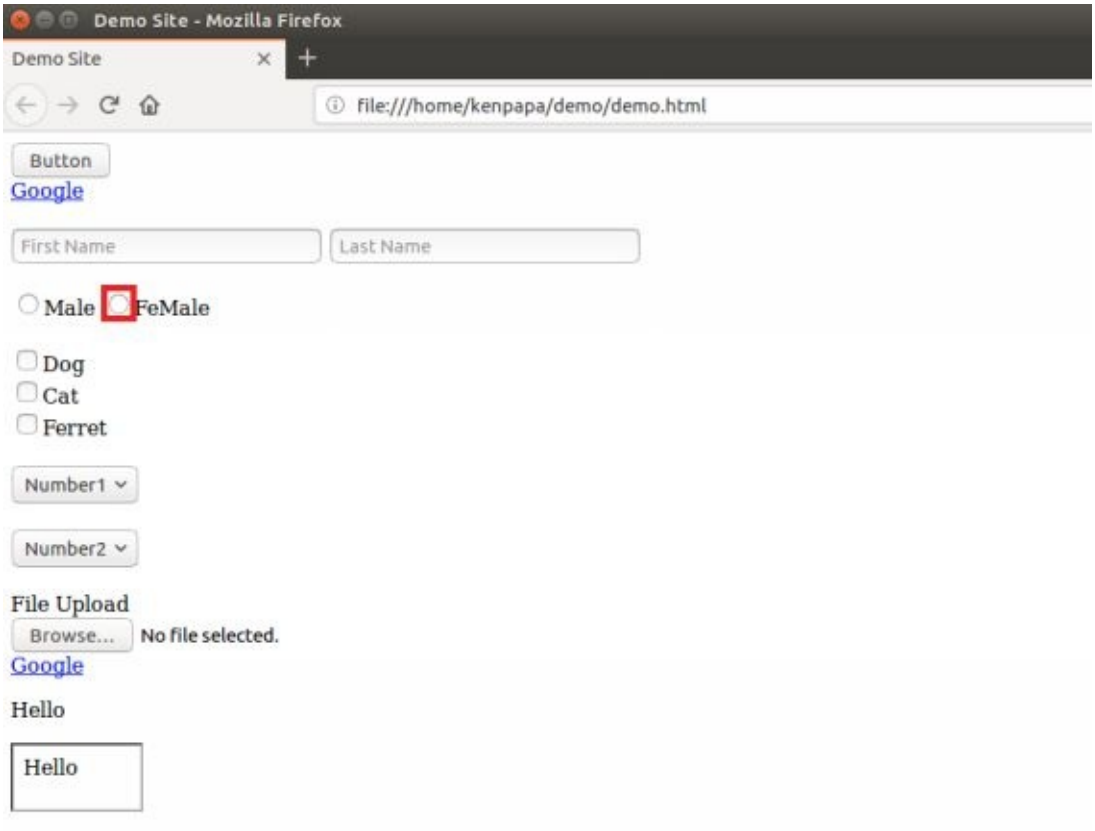
```
<input type="text" id="lastname" name="LastName" placeholder="Last Name">
```

There is an id attribute and a name attribute but here we identify the element by the name attribute.

Automatic operation code

```
driver.find_element_by_name("LastName").send_keys("Yamada")
```

■Click radio button (FeMale)



HTML of operation target element

```
<input type="radio" name="genderoptions" value="FeMale">
```

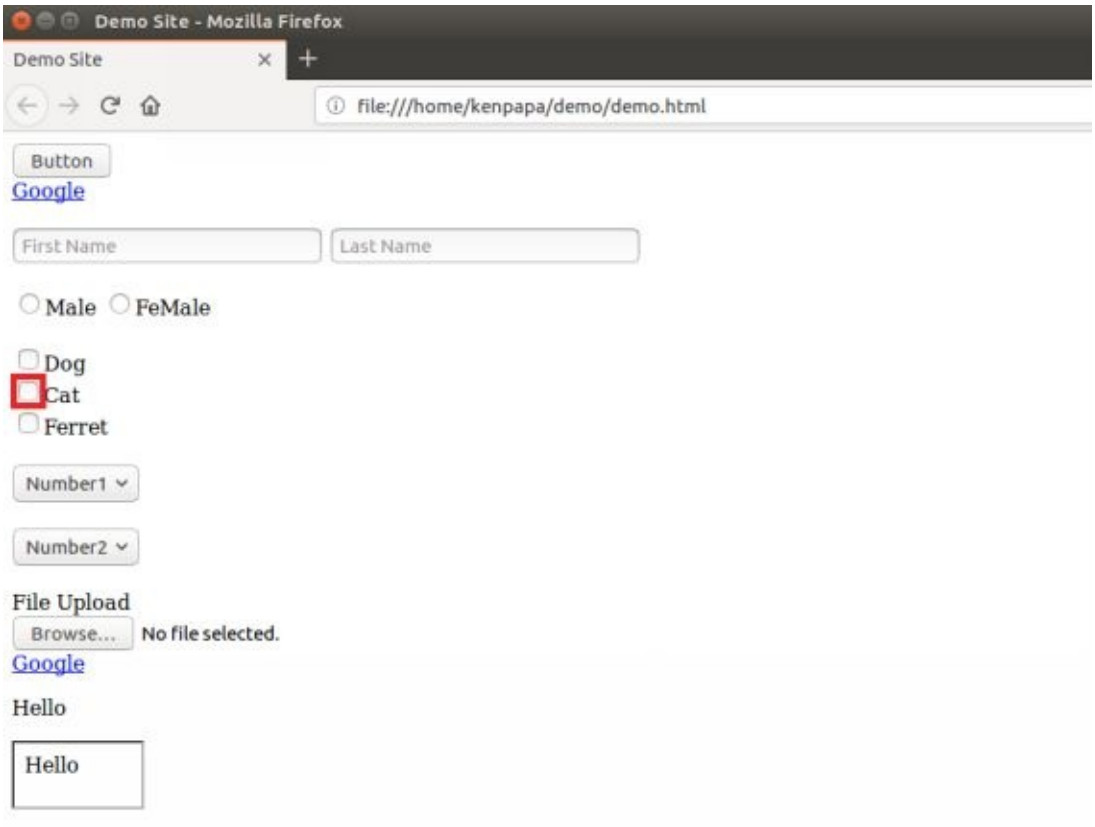
There is a name attribute but we can't use the name attribute to identify the element as one. Because in the case of a radio button, the other element ("Male" in this case) also has the same name attribute value.

So, we use xpath to identify the element.

Automatic operation code

```
driver.find_element_by_xpath("//input[@value='FeMale']").click()
```

■Click checkbox (Cat)



HTML of operation target element

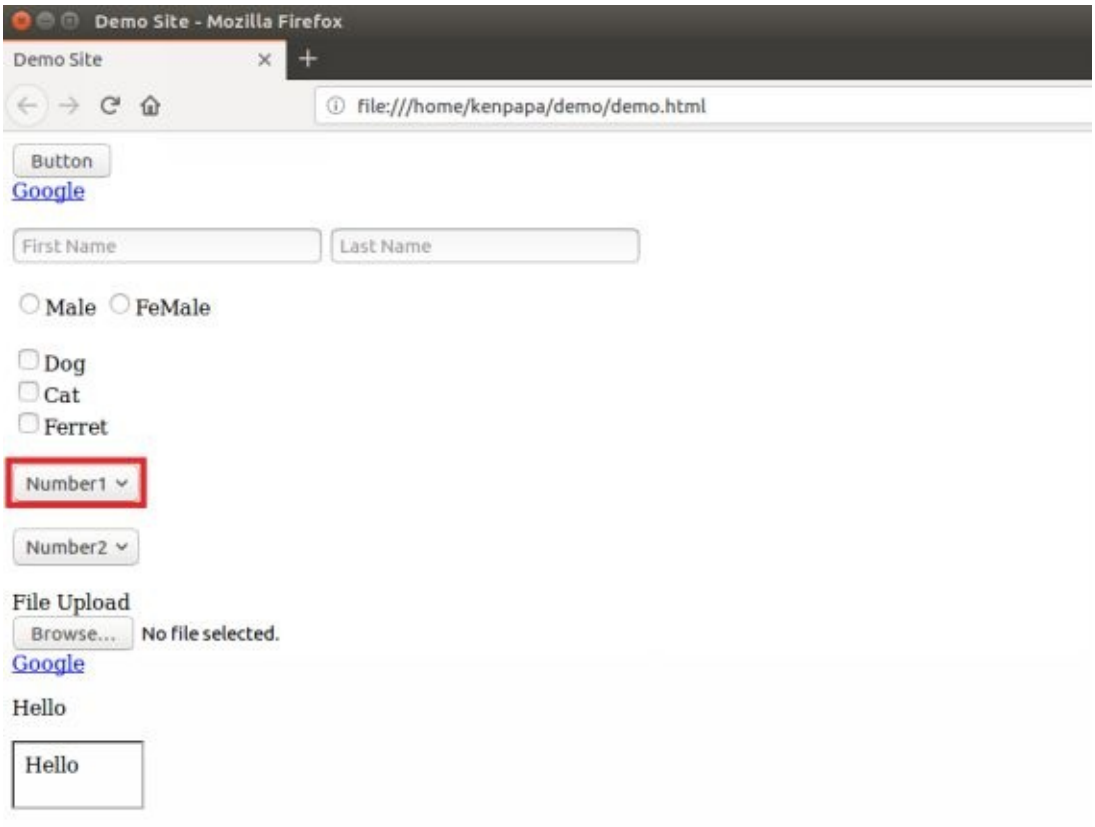
```
<input type="checkbox" value="Cat">
```

Since there is neither an id attribute nor a name attribute, we use xpath to identify the element.

Automatic operation code

```
driver.find_element_by_xpath("//input[@value='Cat']").click()
```

■Click dropdown menu option (3)



HTML of operation target element

```
<select type="text" placeholder="Number1">
   <option value="">Number1</option>
   <option>0</option>
   <option>1</option>
   <option>2</option>
   <option>3</option> <option>4</option>
   <option>5</option>
</select>
```

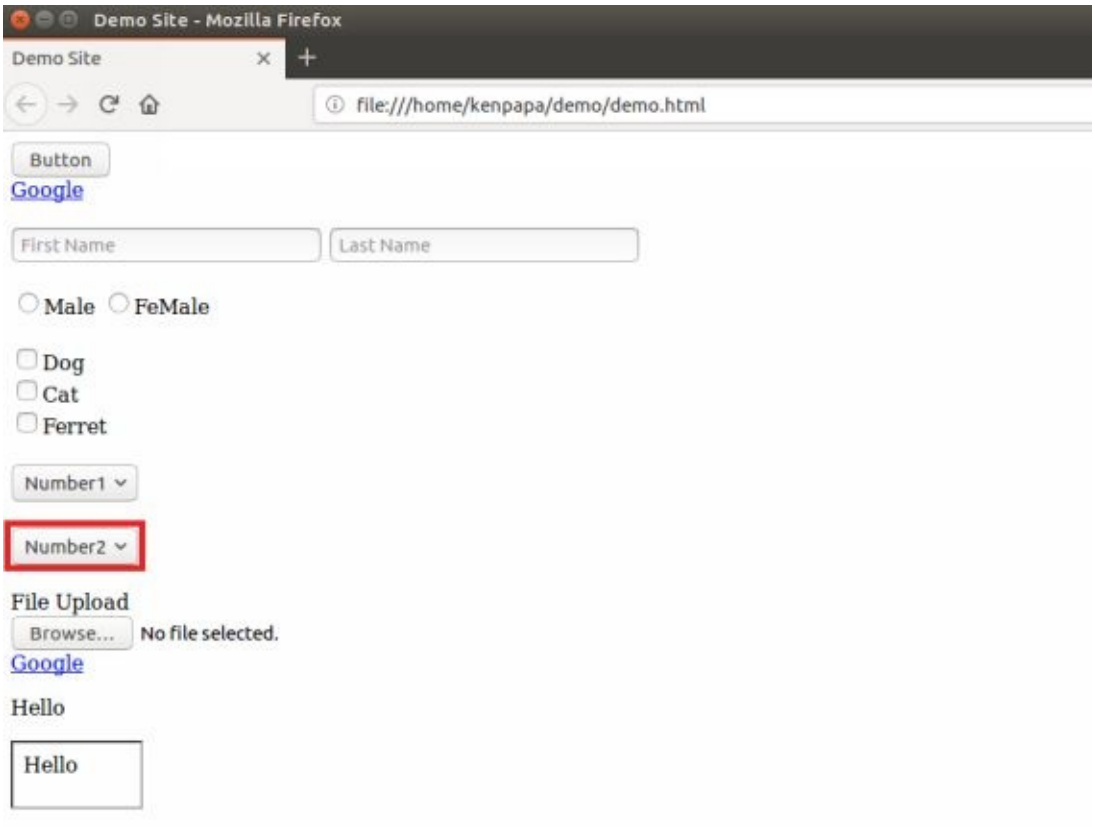
Since there is neither an id attribute nor a name attribute, we use xpath to identify the element.

Automatic operation code

```
driver.find_element_by_xpath("//option[text()='3']").click()
```

find_element returns the first found element, so this element is identified.

■Click dropdown menu option (2)



HTML of operation target element

```
<select type="text" placeholder="Number2"> <option value="">Number2</option>
  <option>0</option>
  <option>1</option>
  <option>2</option> <option>3</option>
  <option>4</option>
  <option>5</option>
</select>
```

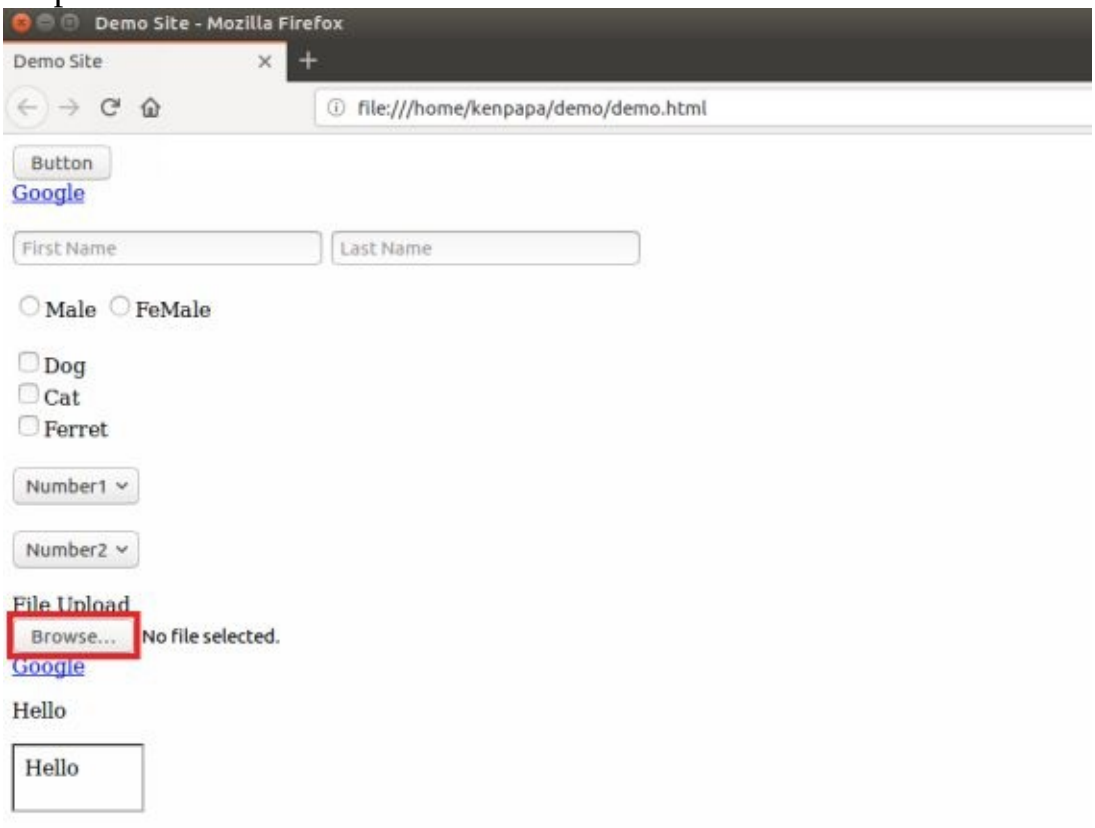
Since there is neither an id attribute nor a name attribute, we use xpath to identify the element. However, since this element is the second Dropdown menu, we use the advanced pattern of xpath to identify it.

We use "hierarchy" of this dropdown menu to identify the element.

Automatic operation code (xpath: "hierarchy")

```
driver.find_element_by_xpath("//select[@placeholder='Number2']/option[text()=
```

■Upload File

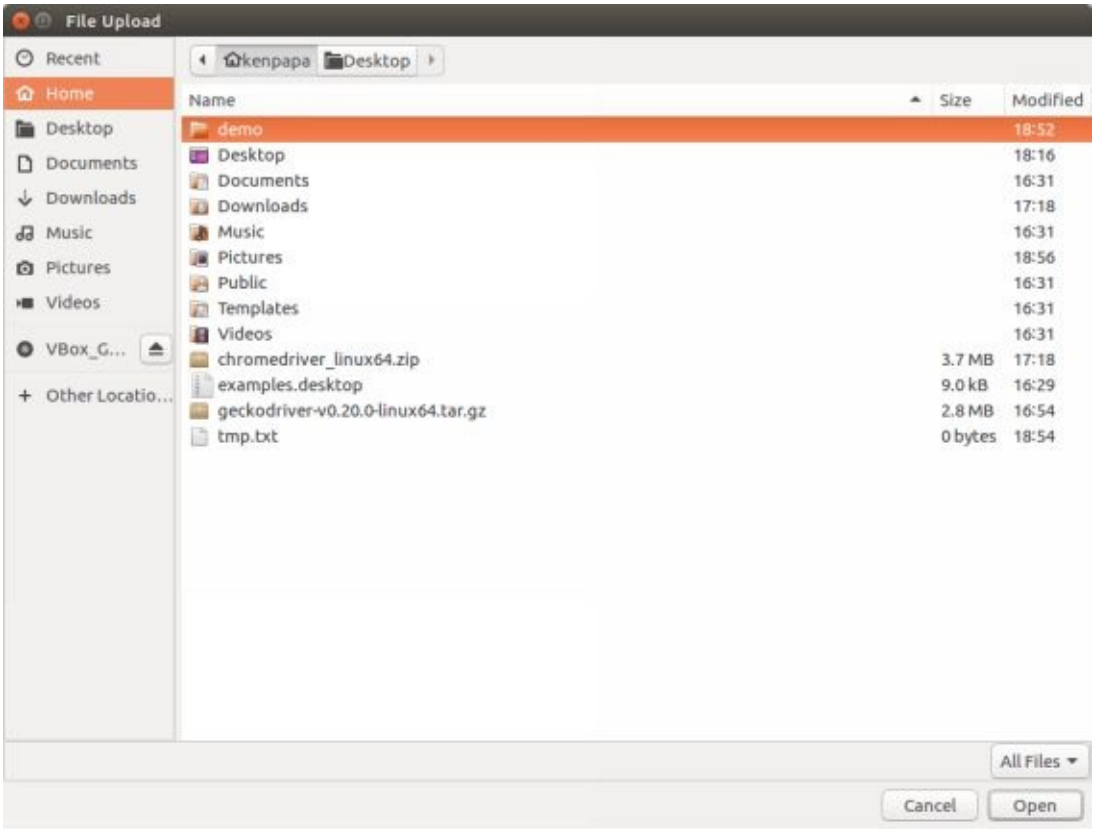


HTML of operation target element

```
<input id="uploadfile" type="file">
```

Since there is an id attribute, we identify the element by the id attribute.

If we click the "Browse" button here, file selection window opens.



Since the window is not a browser, it can't be operated with Selenium WebDriver, so when we do file uploading automatically, we need to use "sendKeys" instead of "click" method.

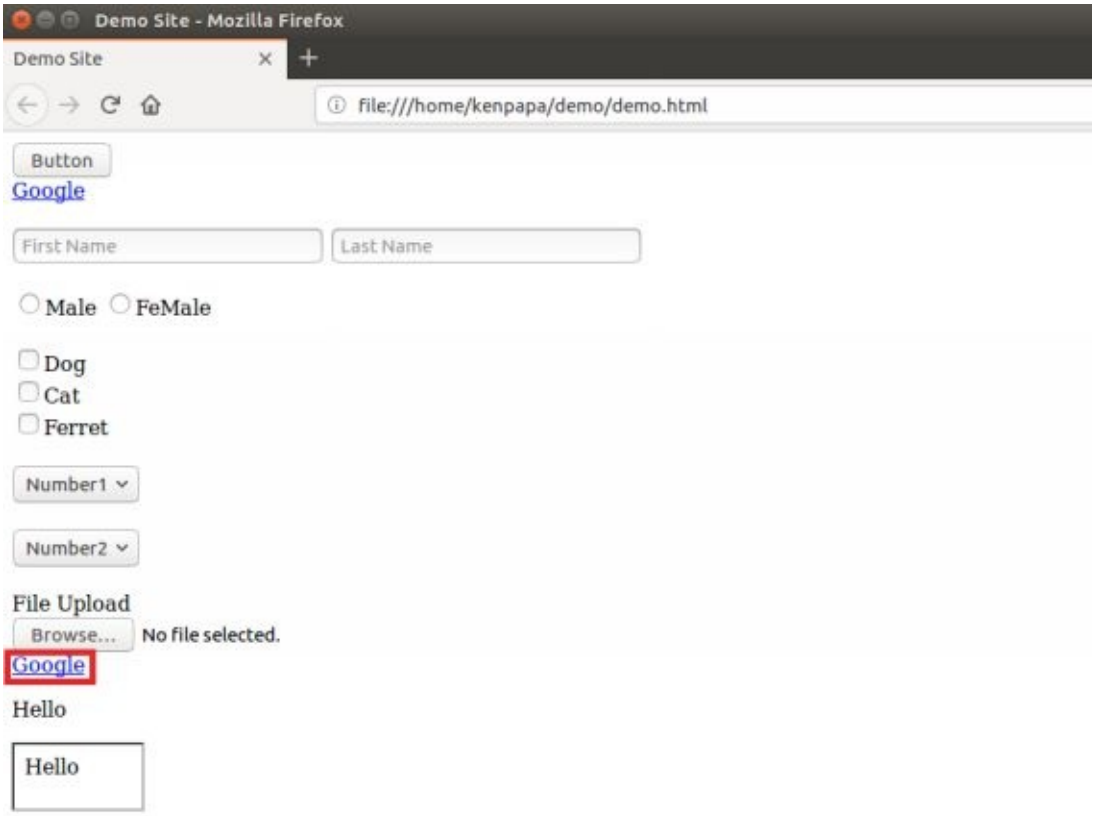
For details please check the following site.

How to upload file using Selenium WebDriver in Java
https://stackoverflow.com/questions/16896685/how-to-upload-file-using-selenium-webdriver-in-java

Automatic operation code

```
driver.find_element_by_id("uploadfile").send_keys("/home/kenpapa/tmp.txt")
```

■Click link (Google USA URL)



HTML of operation target element

```
<a class="Google_USA_URL" href="http://www.google.com">Google</a>
```

There is a link text but since this is the second "Google" link text, we use the advanced pattern of xpath to identify the element.

We use "and" to identify the element by using the class attribute and the link text.

Automatic operation code (xpath: "and")

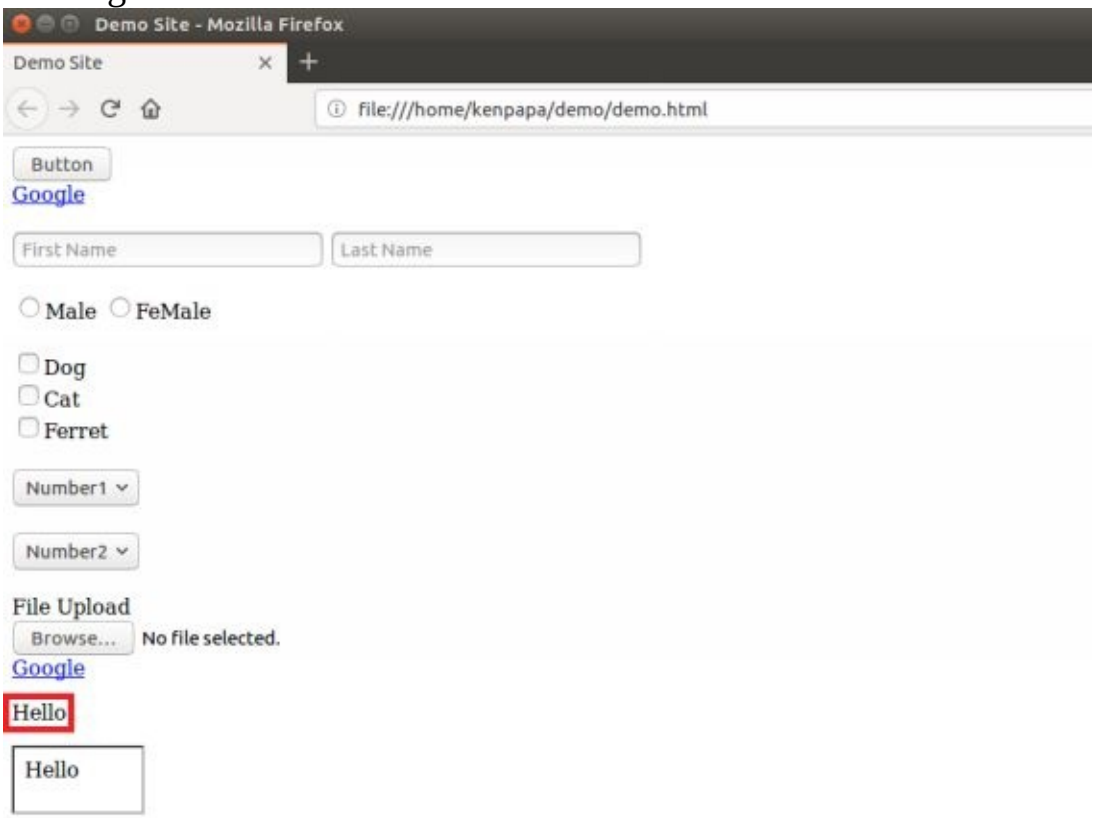
```
driver.find_element_by_xpath("//a[text()='Google' and @class='Google_USA_URL']").click()
```

Also, we use "following" to identify the element by using the relation that the target element is after "File Upload" label element.

Automatic operation code (xpath: "following")

```
driver.find_element_by_xpath("//label[text()='File Upload']//following::a[text()='Google']").click()
```

■Change text



HTML of operation target element

```
<p id="inner">Hello</p>
```

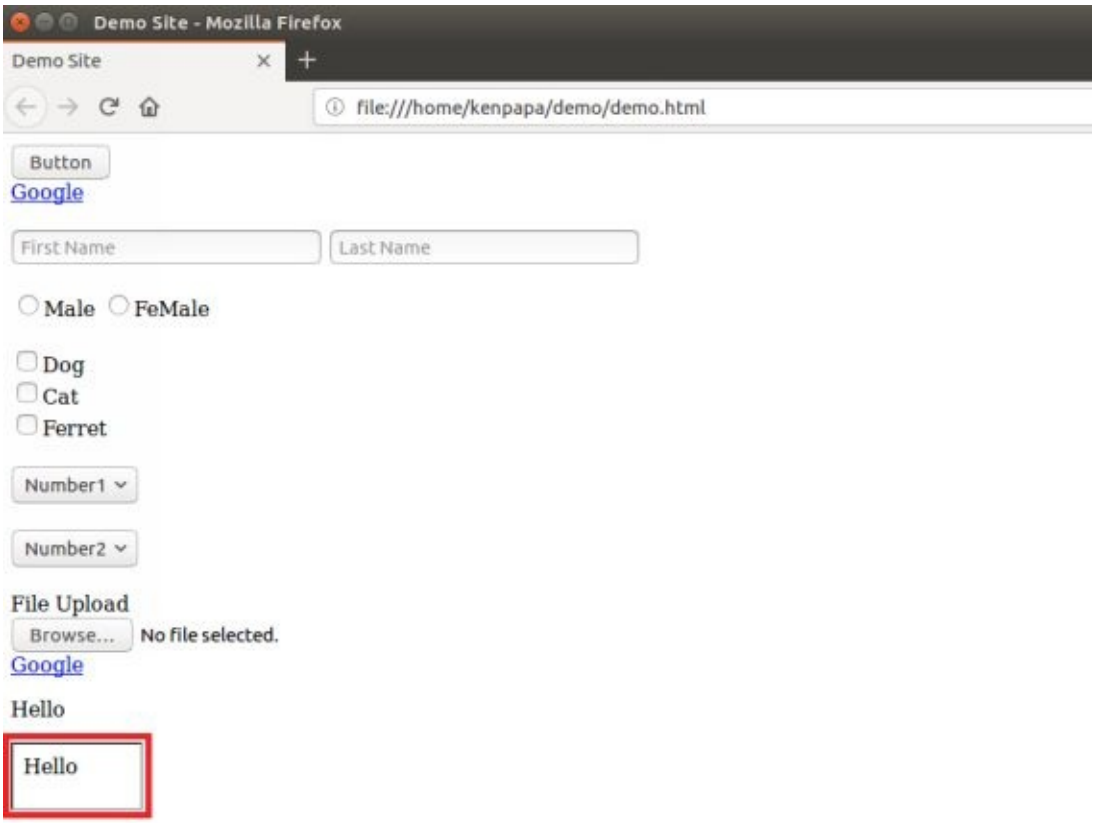
Since there is an id attribute, we identify the element by the id attribute. However, instead of using Selenium's find_element, we identify the element in Javascript.

Automatic operation code

```
driver.execute_script('document.getElementById("inner").innerHTML = "Bye";')
```

We change the text using Javascript.

■Change text in iframe



HTML of operation target element

```
<iframe src="iframe.html" id="content_ifr" width="100" height="50"> <html>
  <body>
  <p id="inner">Hello</p> </body>
</html>
</iframe>
```

Since there is an id attribute, we identify the element by the id attribute.

Automatic operation code

```
driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("inner").innerHTML = "Bye";')
driver.switch_to.default_content()
```

After changing frame, we use Javascript to change the text.

3-3 Complete and execute automatic operation code

Since all the codes for automatic operation of each element in the demo site have already been created, the following code completes the code of automatic operation of demo site in one file.

demo_firefox.py

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.get("file:///home/kenpapa/demo/demo.html")

driver.find_element_by_id("button").click()

driver.find_element_by_link_text("Google").click()
driver.back()

driver.find_element_by_xpath("//a[text()='Google']").click()
driver.back()

driver.find_element_by_id("firstname").send_keys("Taro")
driver.find_element_by_name("LastName").send_keys("Yamada")

driver.find_element_by_xpath("//input[@value='FeMale']").click()

driver.find_element_by_xpath("//input[@value='Cat']").click()

driver.find_element_by_xpath("//option[text()='3']").click()

driver.find_element_by_xpath("//select[@placeholder='Number2']/option[text()=

driver.find_element_by_id("uploadfile").send_keys("/home/kenpapa/tmp.txt")

driver.find_element_by_xpath("//a[text()='Google'
@class='Google_USA_URL']").click()
driver.back()

driver.find_element_by_xpath("//label[text()='File
Upload']//following::a[text()='Google']").click()
```

```
driver.back()

driver.execute_script('document.getElementById("inner").innerHTML = "Bye";'

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("inner").innerHTML = "Bye";'
driver.switch_to.default_content()
```

demo_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.get("file:///home/kenpapa/demo/demo.html")

driver.find_element_by_id("button").click()

driver.find_element_by_link_text("Google").click()
driver.back()

driver.find_element_by_xpath("//a[text()='Google']").click()
driver.back()

driver.find_element_by_id("firstname").send_keys("Taro")
driver.find_element_by_name("LastName").send_keys("Yamada")

driver.find_element_by_xpath("//input[@value='FeMale']").click()

driver.find_element_by_xpath("//input[@value='Cat']").click()

driver.find_element_by_xpath("//option[text()='3']").click()

driver.find_element_by_xpath("//select[@placeholder='Number2']/option[text()=

driver.find_element_by_id("uploadfile").send_keys("/home/kenpapa/tmp.txt")

driver.find_element_by_xpath("//a[text()='Google'
@class='Google_USA_URL']").click()
driver.back()
```

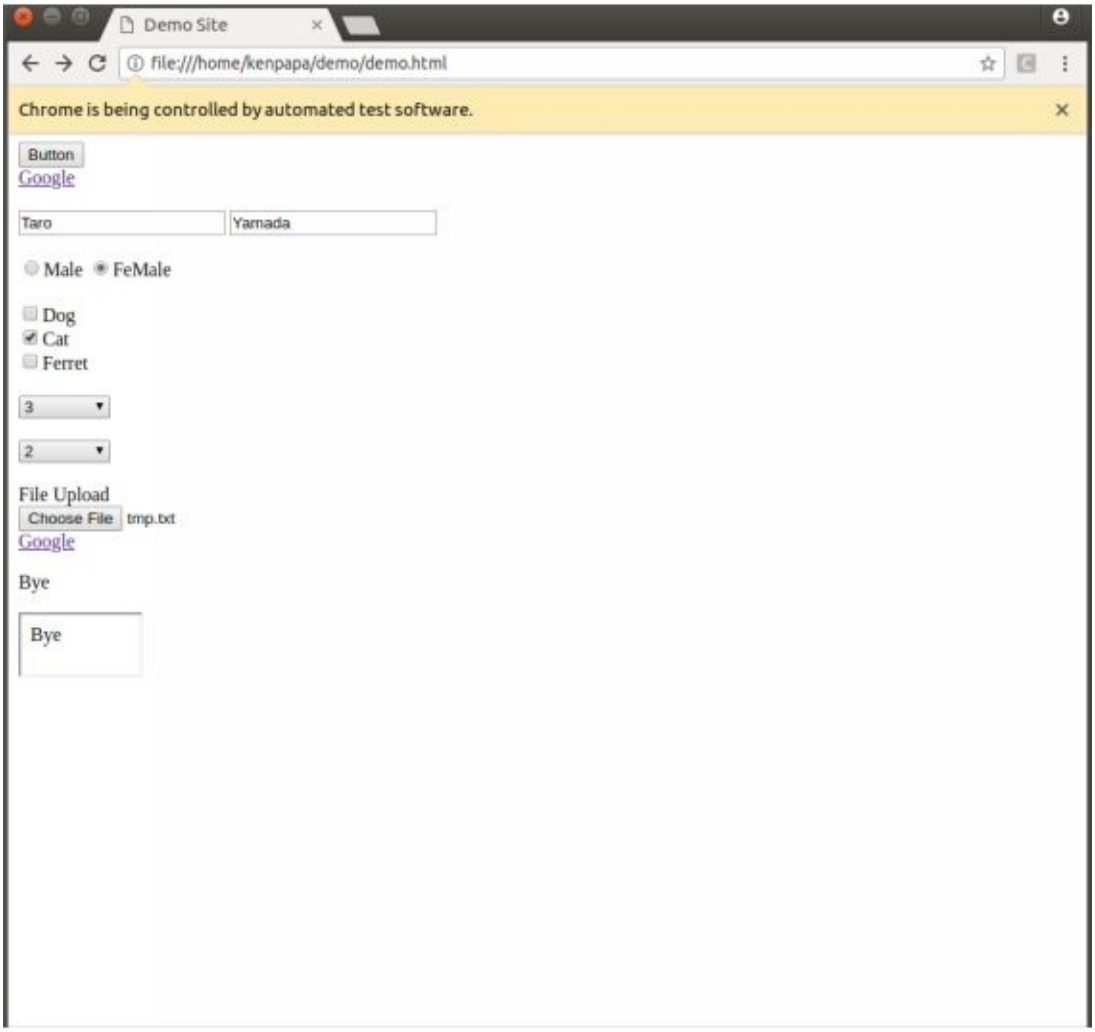
```
driver.find_element_by_xpath("//label[text()='File
Upload']//following::a[text()='Google']").click()
driver.back()

driver.execute_script('document.getElementById("inner").innerHTML = "Bye";'

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("inner").innerHTML = "Bye";'
driver.switch_to.default_content()
```

Two Google links are clicked twice in different ways.
(It returns to the original page after transitioning to Google page)

The above code works fine for both Firefox and Chrome browsers.

Result in Firefox

Result in Chrome



In this chapter, we performed automatic operation of demo site as a basic practice of Selenium WebDriver.

Chapter 4 Advanced practice of automatic operation

In this chapter, we perform automatic operation of WordPress site as an advanced practice of automatic operation using Selenium WebDriver.

4-1 Prepare WordPress site

Here we install the following softwares.
WordPress
WooCommerce plugin

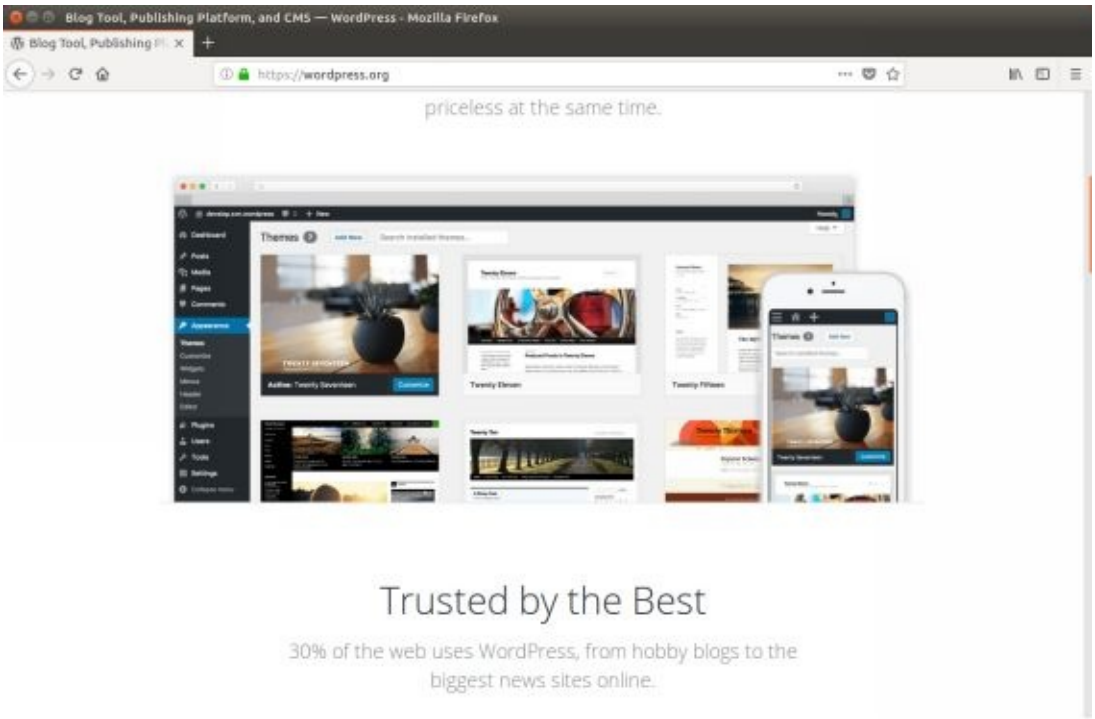
4-1-1 What is WordPress ?

WordPress is an open source content management system (CMS) written in PHP. With WordPress we can build and operate a multifunctional and excellent website without programming.

WordPress official website

https://wordpress.org/

The official website says "30% of the web uses WordPress".


Blog Tool, Publishing Platform, and CMS — WordPress - Mozilla Firefox
https://wordpress.org
priceless at the same time.
Trusted by the Best
30% of the web uses WordPress, from hobby blogs to the biggest news sites online.

4-1-2 Install WordPress

In this book we install WordPress as follows.

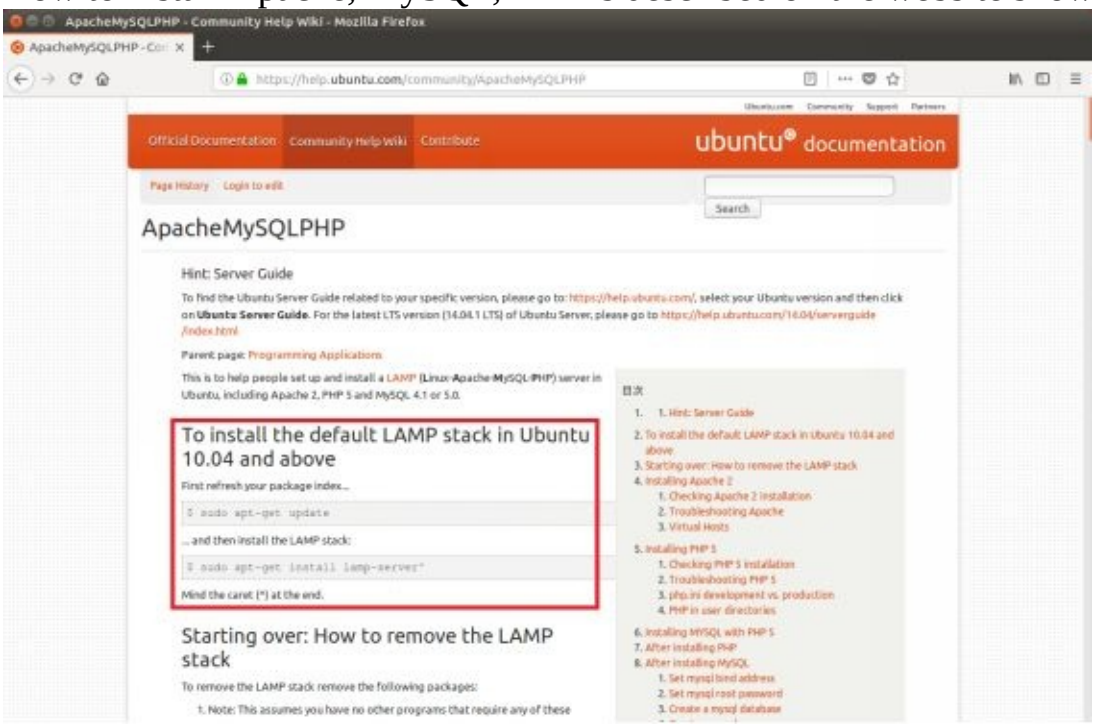
WordPress installation procedure

```
■Install Apache, MySQL, PHP
sudo apt-get update
sudo apt-get install lamp-server^

■Create database
mysql -u root -p
CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;
GRANT ALL ON wordpressdb.* TO 'wordpressuser'@'localhost' IDENTIFIED BY 'password';
FLUSH PRIVILEGES;
EXIT;

■Download and deploy WordPress
wget http://wordpress.org/latest.tar.gz
tar -xzvf latest.tar.gz
sudo mv wordpress /var/www/html

■Change Wordpress directory setting
sudo chown -R www-data:www-data /var/www/html/wordpress

■Change Apache configuration
sudo gedit /etc/apache2/apache2.conf
-----
<Directory /var/www/html/wordpress>
   AllowOverride All
</Directory>
-----

■Activate mod_rewrite and restart Apache
sudo a2enmod rewrite
sudo service apache2 restart
```

■Open "http://localhost/wordpress" in the browser

※After that, continue installing on WordPress screen.

How to install Apache, MySQL, PHP is described on the website shown below.



https://help.ubuntu.com/community/ApacheMySQLPHP

Let's start the installation.

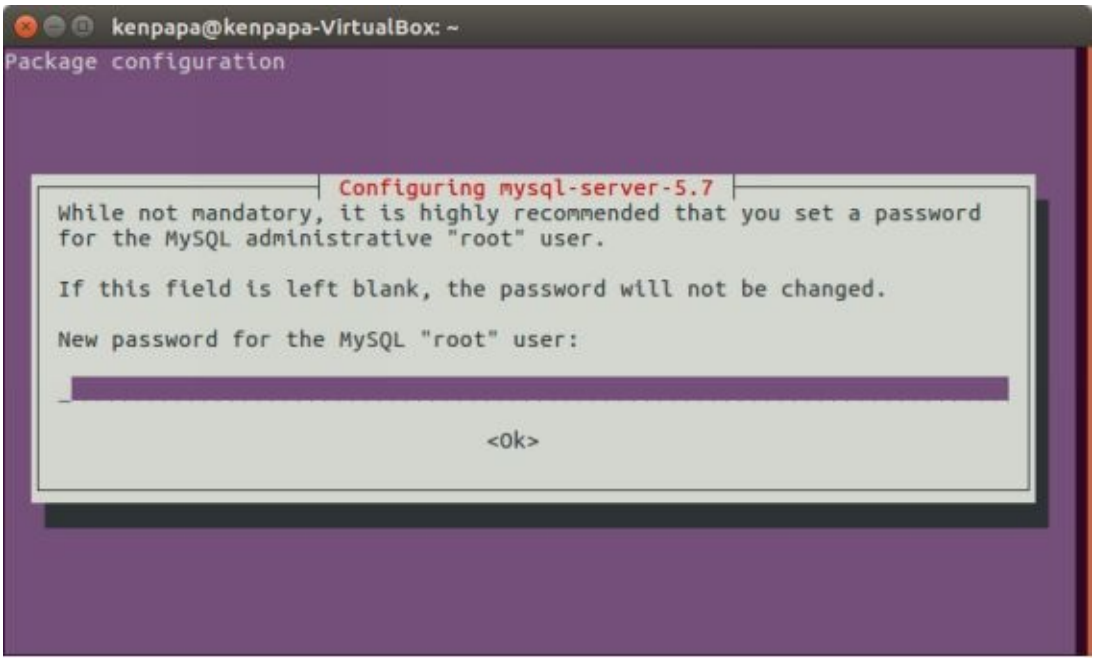
sudo apt-get update

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ sudo apt-get update
```

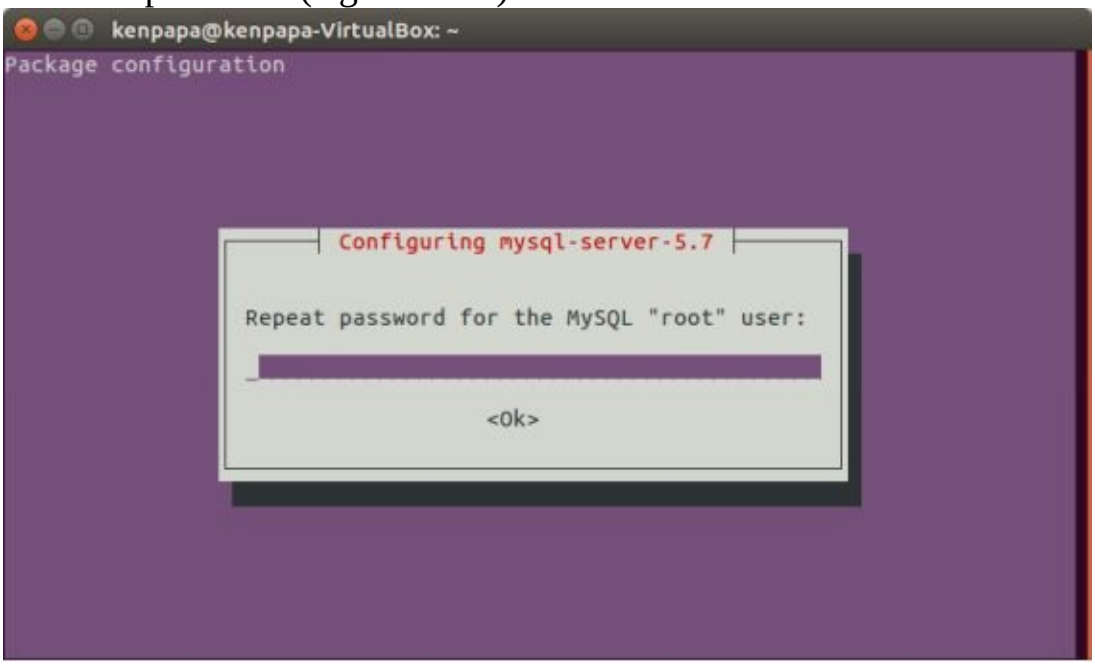
sudo apt-get install lamp-server^

```
kenpapa@kenpapa-VirtualBox: ~
Get:9 http://jp.archive.ubuntu.com/ubuntu xenial-updates/universe amd64 DEP-11 M
etadata [193 kB]
Get:10 http://jp.archive.ubuntu.com/ubuntu xenial-updates/universe DEP-11 64x64
Icons [275 kB]
Get:11 http://dl.google.com/linux/chrome/deb stable Release.gpg [819 B]
Get:12 http://jp.archive.ubuntu.com/ubuntu xenial-updates/multiverse amd64 DEP-1
1 Metadata [5,888 B]
Get:13 http://jp.archive.ubuntu.com/ubuntu xenial-backports/main amd64 DEP-11 Me
tadata [3,328 B]
Get:14 http://jp.archive.ubuntu.com/ubuntu xenial-backports/universe amd64 DEP-1
1 Metadata [5,088 B]
Get:15 http://dl.google.com/linux/chrome/deb stable/main amd64 Packages [1,346 B
]
Get:16 http://security.ubuntu.com/ubuntu xenial-security/main amd64 DEP-11 Metad
ata [67.5 kB]
Get:17 http://security.ubuntu.com/ubuntu xenial-security/main DEP-11 64x64 Icons
 [72.2 kB]
Get:18 http://security.ubuntu.com/ubuntu xenial-security/universe amd64 DEP-11 M
etadata [56.0 kB]
Get:19 http://security.ubuntu.com/ubuntu xenial-security/universe DEP-11 64x64 I
cons [93.6 kB]
Fetched 1,625 kB in 8s (198 kB/s)
Reading package lists... Done
kenpapa@kenpapa-VirtualBox:~$ sudo apt-get install lamp-server^
```

Enter the password (e.g. root123)



Re-enter password (e.g. root123)



mysql -u root -p

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ mysql -u root -p
```

For password, enter the one set in 3 and 4.

CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.21-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;
```

wordpressdb: Database Name

GRANT ALL ON wordpressdb.* TO 'wordpressuser'@'localhost' IDENTIFIED

BY 'password';

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.21-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL ON wordpressdb.* TO 'wordpressuser'@'localhost' IDENTIFIED BY '
password';
```

wordpressuser: User Name
password: Password

FLUSH PRIVILEGES;

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ mysql -u root -p
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.21-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL ON wordpressdb.* TO 'wordpressuser'@'localhost' IDENTIFIED BY '
password';
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> FLUSH PRIVILEGES;
```

EXIT;

```
kenpapa@kenpapa-VirtualBox: ~
Enter password:
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 5
Server version: 5.7.21-0ubuntu0.16.04.1 (Ubuntu)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> CREATE DATABASE wordpressdb CHARACTER SET utf8 COLLATE utf8_general_ci;
Query OK, 1 row affected (0.00 sec)

mysql> GRANT ALL ON wordpressdb.* TO 'wordpressuser'@'localhost' IDENTIFIED BY '
password';
Query OK, 0 rows affected, 1 warning (0.01 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.00 sec)

mysql> EXIT;
```

wget http://wordpress.org/latest.tar.gz

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ wget http://wordpress.org/latest.tar.gz
```

tar -xzvf latest.tar.gz

```
kenpapa@kenpapa-VirtualBox: ~
kenpapa@kenpapa-VirtualBox:~$ wget http://wordpress.org/latest.tar.gz
--2018-03-25 19:17:47--  http://wordpress.org/latest.tar.gz
Resolving wordpress.org (wordpress.org)... 198.143.164.252
Connecting to wordpress.org (wordpress.org)|198.143.164.252|:80... connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://wordpress.org/latest.tar.gz [following]
--2018-03-25 19:17:47--  https://wordpress.org/latest.tar.gz
Connecting to wordpress.org (wordpress.org)|198.143.164.252|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 8565525 (8.2M) [application/octet-stream]
Saving to: 'latest.tar.gz'

latest.tar.gz       100%[===================>]   8.17M  2.37MB/s    in 4.3s

2018-03-25 19:17:52 (1.92 MB/s) - 'latest.tar.gz' saved [8565525/8565525]

kenpapa@kenpapa-VirtualBox:~$ tar -xzvf latest.tar.gz
```

sudo mv wordpress /var/www/html

```
kenpapa@kenpapa-VirtualBox: ~
wordpress/wp-admin/js/svg-painter.min.js
wordpress/wp-admin/js/custom-background.js
wordpress/wp-admin/js/color-picker.min.js
wordpress/wp-admin/js/code-editor.js
wordpress/wp-admin/js/common.js
wordpress/wp-admin/js/set-post-thumbnail.min.js
wordpress/wp-admin/js/postbox.min.js
wordpress/wp-admin/js/color-picker.js
wordpress/wp-admin/js/password-strength-meter.js
wordpress/wp-admin/js/customize-nav-menus.js
wordpress/wp-admin/js/editor-expand.js
wordpress/wp-admin/js/code-editor.min.js
wordpress/wp-admin/js/set-post-thumbnail.js
wordpress/wp-admin/options-permalink.php
wordpress/wp-admin/widgets.php
wordpress/wp-admin/setup-config.php
wordpress/wp-admin/install.php
wordpress/wp-admin/admin-header.php
wordpress/wp-admin/post-new.php
wordpress/wp-admin/themes.php
wordpress/wp-admin/options-reading.php
wordpress/wp-trackback.php
wordpress/wp-comments-post.php
kenpapa@kenpapa-VirtualBox:~$ sudo mv wordpress /var/www/html
```

sudo chown -R www-data:www-data /var/www/html/wordpress

```
kenpapa@kenpapa-VirtualBox: ~
wordpress/wp-admin/js/color-picker.min.js
wordpress/wp-admin/js/code-editor.js
wordpress/wp-admin/js/common.js
wordpress/wp-admin/js/set-post-thumbnail.min.js
wordpress/wp-admin/js/postbox.min.js
wordpress/wp-admin/js/color-picker.js
wordpress/wp-admin/js/password-strength-meter.js
wordpress/wp-admin/js/customize-nav-menus.js
wordpress/wp-admin/js/editor-expand.js
wordpress/wp-admin/js/code-editor.min.js
wordpress/wp-admin/js/set-post-thumbnail.js
wordpress/wp-admin/options-permalink.php
wordpress/wp-admin/widgets.php
wordpress/wp-admin/setup-config.php
wordpress/wp-admin/install.php
wordpress/wp-admin/admin-header.php
wordpress/wp-admin/post-new.php
wordpress/wp-admin/themes.php
wordpress/wp-admin/options-reading.php
wordpress/wp-trackback.php
wordpress/wp-comments-post.php
kenpapa@kenpapa-VirtualBox:~$ sudo mv wordpress /var/www/html
kenpapa@kenpapa-VirtualBox:~$ sudo chown -R www-data:www-data /var/www/html/word
press
```

sudo gedit /etc/apache2/apache2.conf

```
kenpapa@kenpapa-VirtualBox: ~
wordpress/wp-admin/js/code-editor.js
wordpress/wp-admin/js/common.js
wordpress/wp-admin/js/set-post-thumbnail.min.js
wordpress/wp-admin/js/postbox.min.js
wordpress/wp-admin/js/color-picker.js
wordpress/wp-admin/js/password-strength-meter.js
wordpress/wp-admin/js/customize-nav-menus.js
wordpress/wp-admin/js/editor-expand.js
wordpress/wp-admin/js/code-editor.min.js
wordpress/wp-admin/js/set-post-thumbnail.js
wordpress/wp-admin/options-permalink.php
wordpress/wp-admin/widgets.php
wordpress/wp-admin/setup-config.php
wordpress/wp-admin/install.php
wordpress/wp-admin/admin-header.php
wordpress/wp-admin/post-new.php
wordpress/wp-admin/themes.php
wordpress/wp-admin/options-reading.php
wordpress/wp-trackback.php
wordpress/wp-comments-post.php
kenpapa@kenpapa-VirtualBox:~$ sudo mv wordpress /var/www/html
kenpapa@kenpapa-VirtualBox:~$ sudo chown -R www-data:www-data /var/www/html/word
press
kenpapa@kenpapa-VirtualBox:~$ sudo gedit /etc/apache2/apache2.conf
```

Add the following contents and save

```
<Directory /var/www/html/wordpress>
    AllowOverride All
```

</Directory>

```
Open ▾   *apache2.conf   /etc/apache2   Save
# not allow access to the root filesystem outside of /usr/share and /var/www.
# The former is used by web applications packaged in Debian,
# the latter may be used for local directories served by the web server. If
# your system is serving content from a sub-directory in /srv you must allow
# access here, or in any related virtual host.
<Directory />
    Options FollowSymLinks
    AllowOverride None
    Require all denied
</Directory>

<Directory /usr/share>
    AllowOverride None
    Require all granted
</Directory>

<Directory /var/www/>
    Options Indexes FollowSymLinks
    AllowOverride None
    Require all granted
</Directory>

<Directory /var/www/html/wordpress>
    AllowOverride All
</Directory>

#<Directory /srv/>
#   Options Indexes FollowSymLinks
#   AllowOverride None
#   Require all granted
#</Directory>



# AccessFileName: The name of the file to look for in each directory
# for additional configuration directives.  See also the AllowOverride
# directive
Plain Text ▾   Tab Width: 4 ▾   Ln 170, Col 1   INS
```

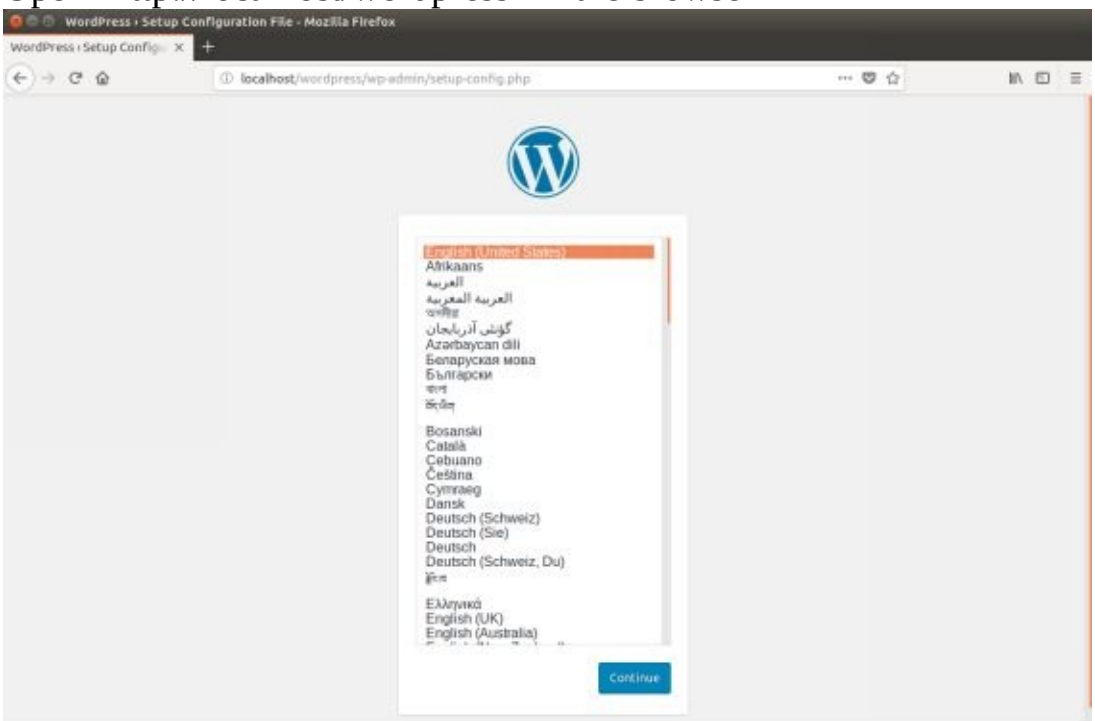
sudo a2enmod rewrite

```
kenpapa@kenpapa-VirtualBox: ~
wordpress/wp-admin/admin-header.php
wordpress/wp-admin/post-new.php
wordpress/wp-admin/themes.php
wordpress/wp-admin/options-reading.php
wordpress/wp-trackback.php
wordpress/wp-comments-post.php
kenpapa@kenpapa-VirtualBox:~$ sudo mv wordpress /var/www/html
kenpapa@kenpapa-VirtualBox:~$ sudo chown -R www-data:www-data /var/www/html/word
press
kenpapa@kenpapa-VirtualBox:~$ sudo gedit /etc/apache2/apache2.conf

(gedit:22928): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedeskt
op.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided
 by any .service files

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-spell-enabled not supported

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-encoding not supported

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-position not supported
kenpapa@kenpapa-VirtualBox:~$ sudo a2enmod rewrite
```

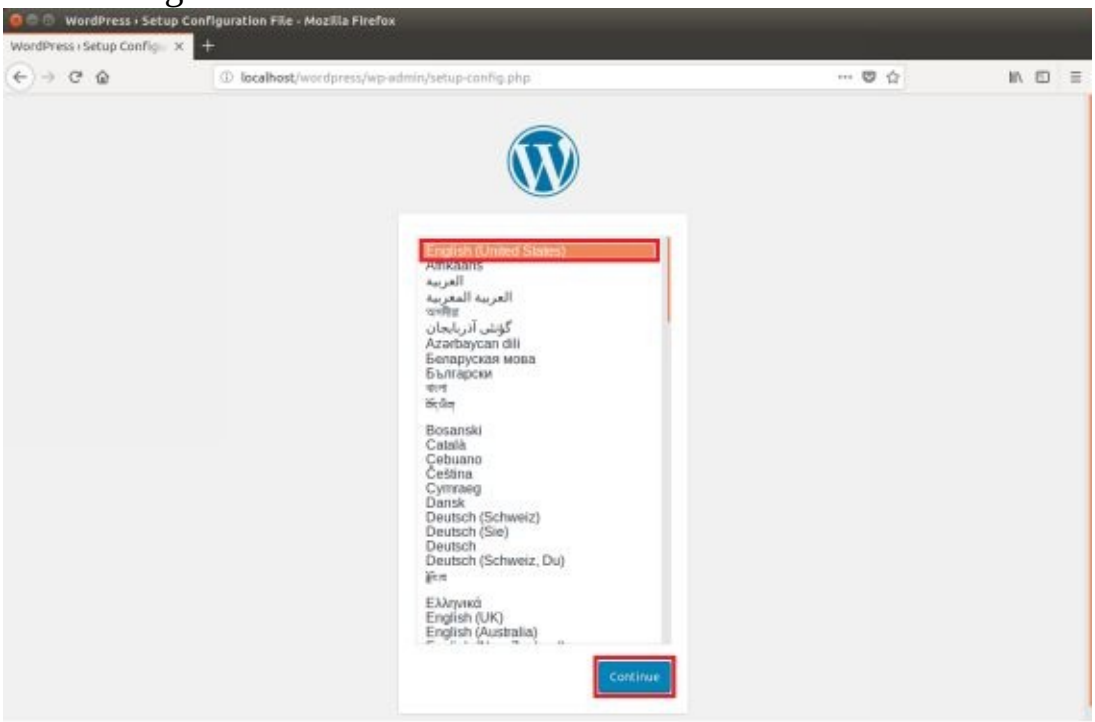
sudo service apache2 restart

```
kenpapa@kenpapa-VirtualBox: ~
wordpress/wp-trackback.php
wordpress/wp-comments-post.php
kenpapa@kenpapa-VirtualBox:~$ sudo mv wordpress /var/www/html
kenpapa@kenpapa-VirtualBox:~$ sudo chown -R www-data:www-data /var/www/html/word
press
kenpapa@kenpapa-VirtualBox:~$ sudo gedit /etc/apache2/apache2.conf

(gedit:22928): Gtk-WARNING **: Calling Inhibit failed: GDBus.Error:org.freedeskt
op.DBus.Error.ServiceUnknown: The name org.gnome.SessionManager was not provided
 by any .service files

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-spell-enabled not supported

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-encoding not supported

** (gedit:22928): WARNING **: Set document metadata failed: Setting attribute me
tadata::gedit-position not supported
kenpapa@kenpapa-VirtualBox:~$ sudo a2enmod rewrite
Enabling module rewrite.
To activate the new configuration, you need to run:
  service apache2 restart
kenpapa@kenpapa-VirtualBox:~$ sudo service apache2 restart
```

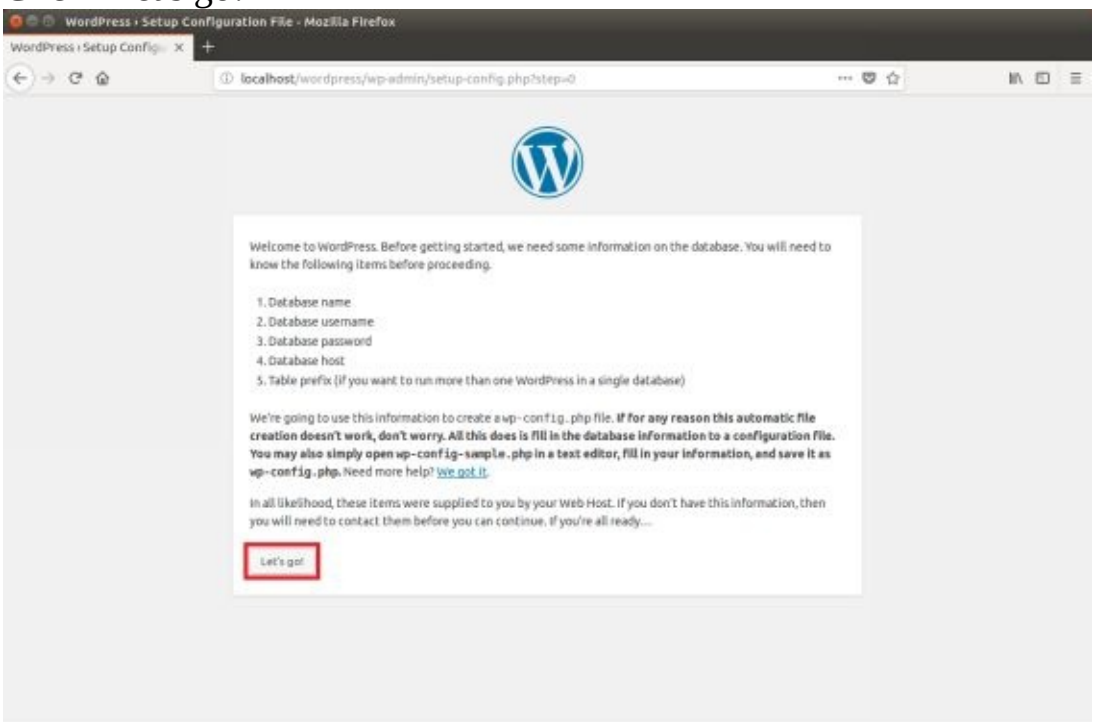
Open "http://localhost/wordpress" in the browser



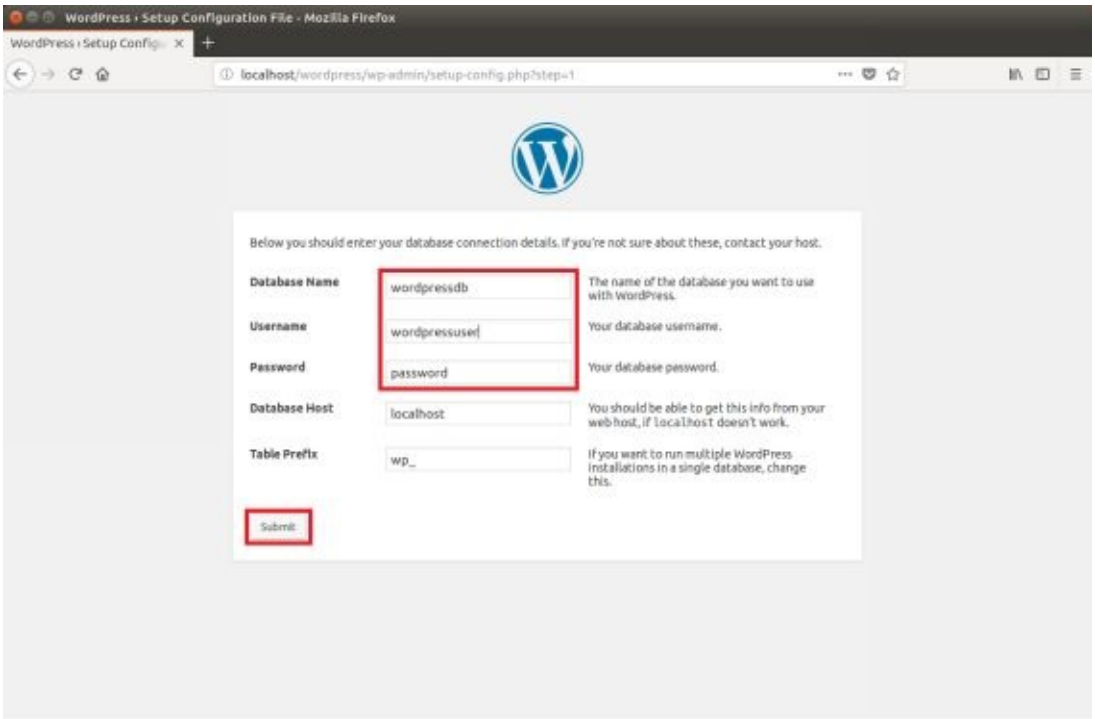
Select "English" and click "Continue"



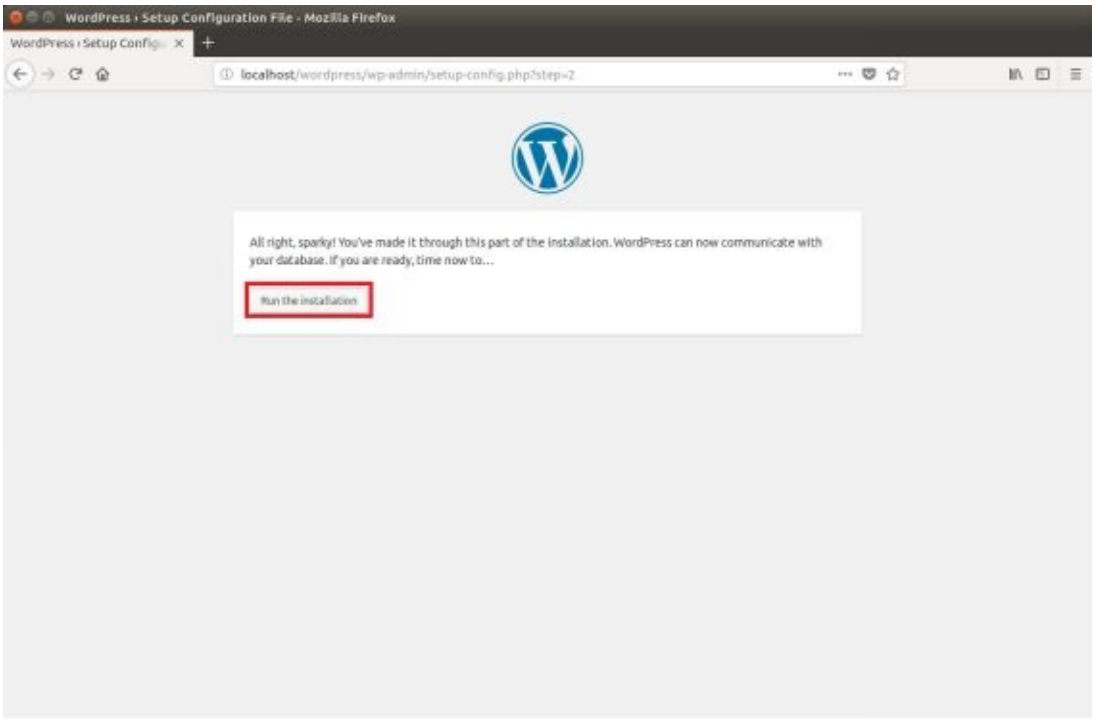
Click "Let's go!"



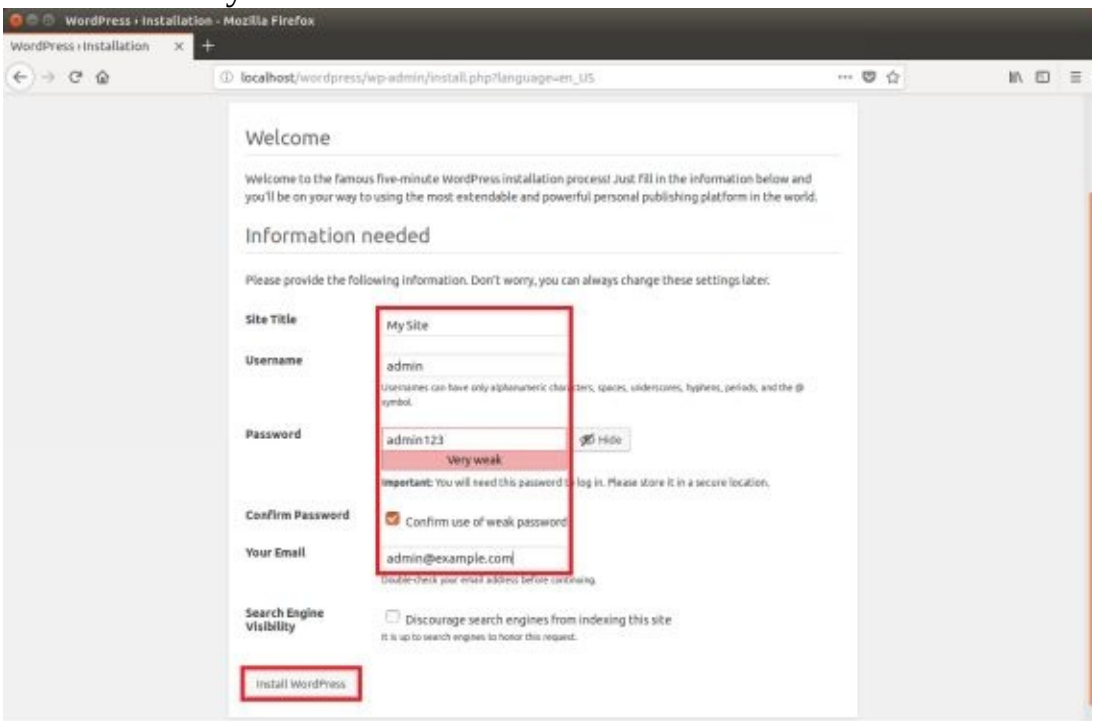
As set in and ,
Enter "wordpressdb" as Database Name
Enter "wordpressuser" as User Name
Enter "password" as Password
then click "Submit"



Click "Run the Installation"



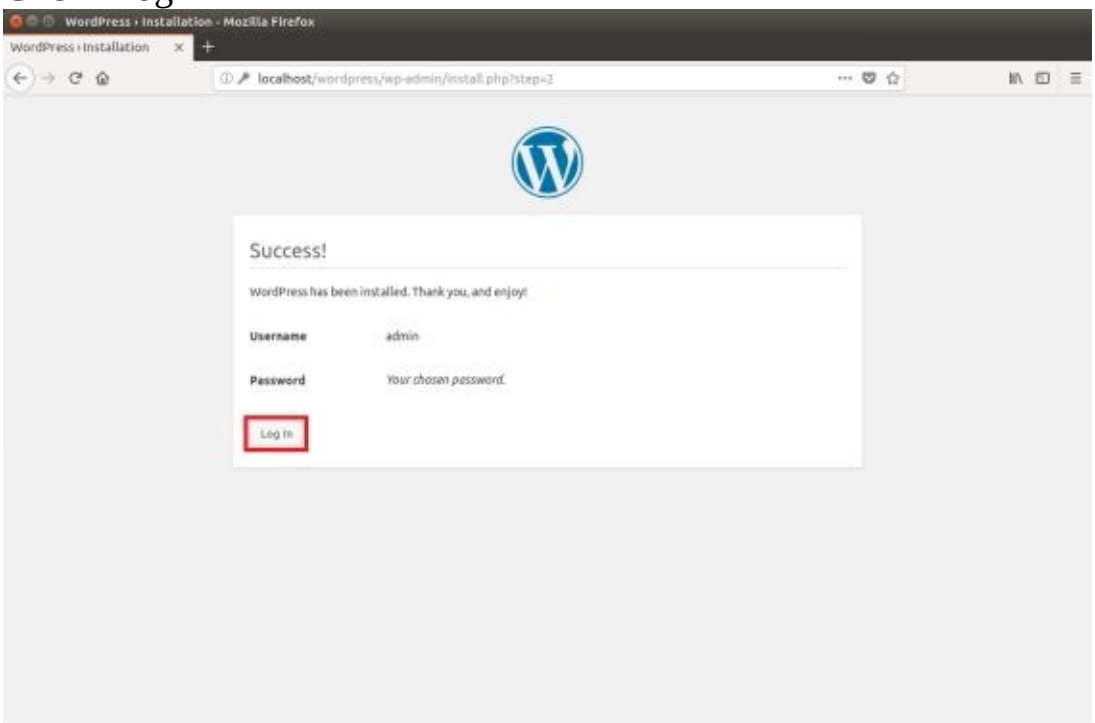
Enter necessary information and click "Install WordPress"



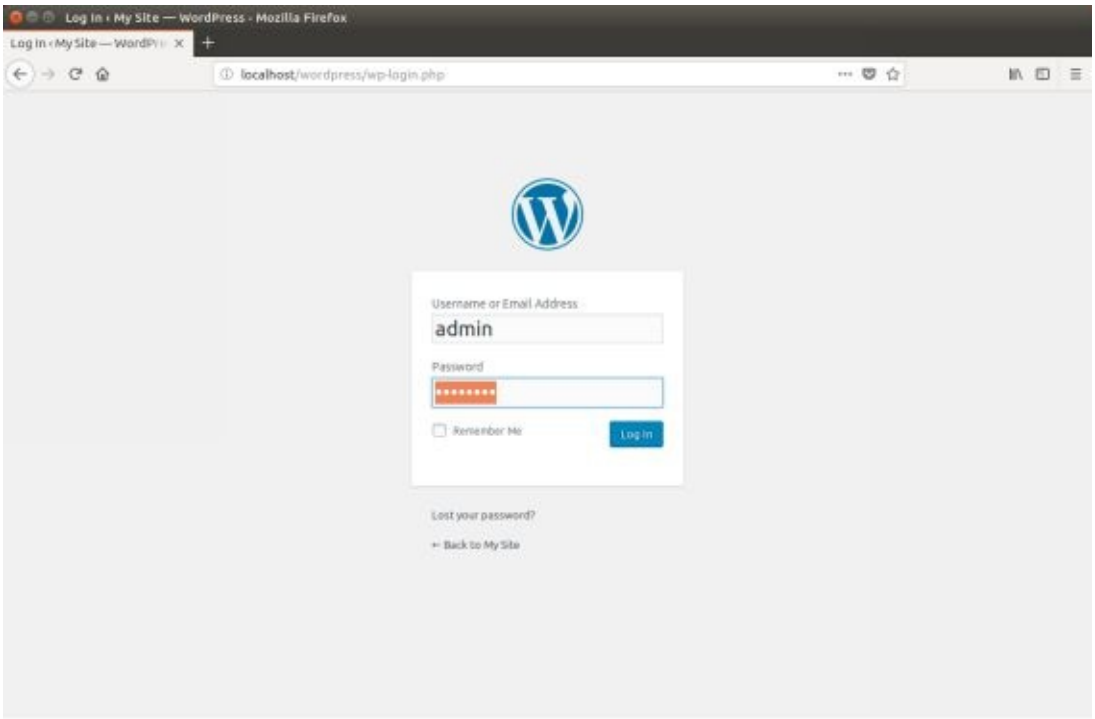
Here enter the data as follows.
Site Title: My Site
Username: admin

Password: admin123
Check to "Confirm use of weak password"
Your Email: admin@example.com

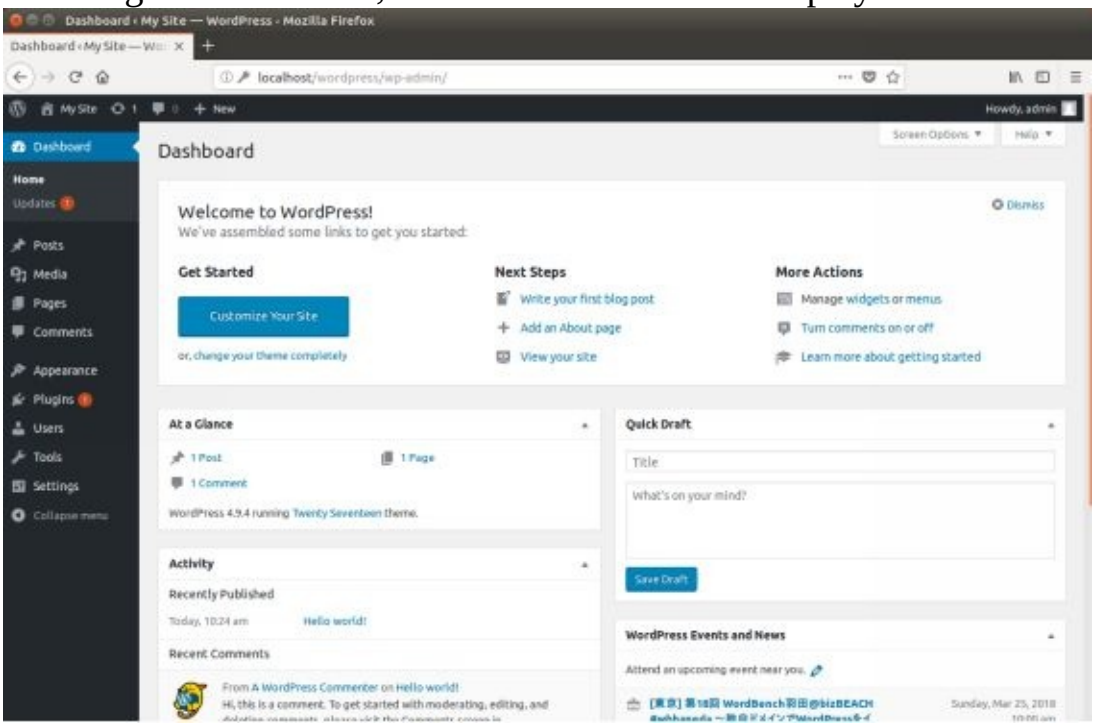
Click "Log In"



Enter Username and Password as set in and click "Log In"



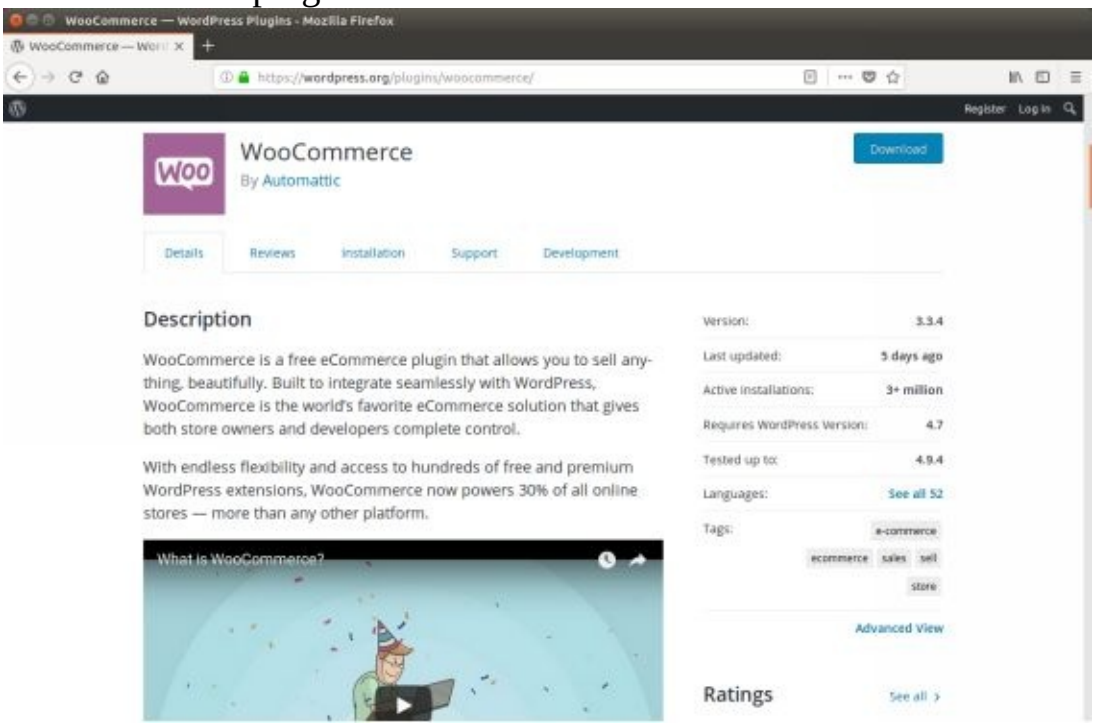
If the login is successful, the dashboard screen is displayed.



4-1-3 Install WooCommerce plugin

WooCommerce is one of WordPress plugins and installing it allows us to build EC site with WordPress.

WooCommerce plugin site

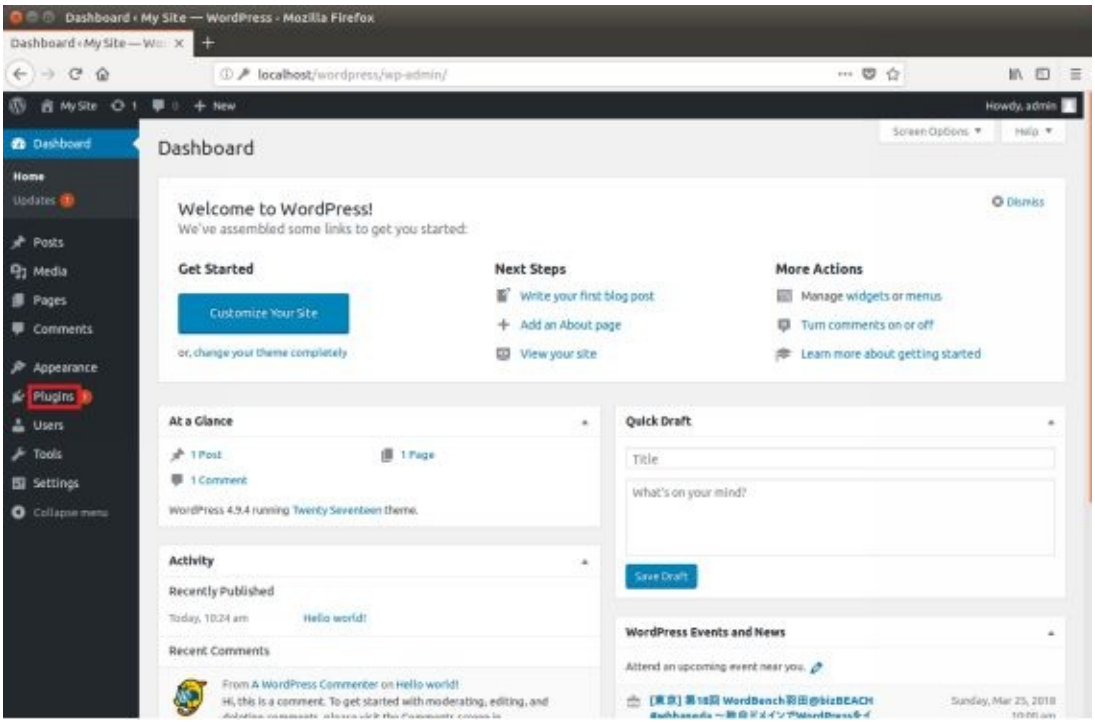


https://wordpress.org/plugins/woocommerce/

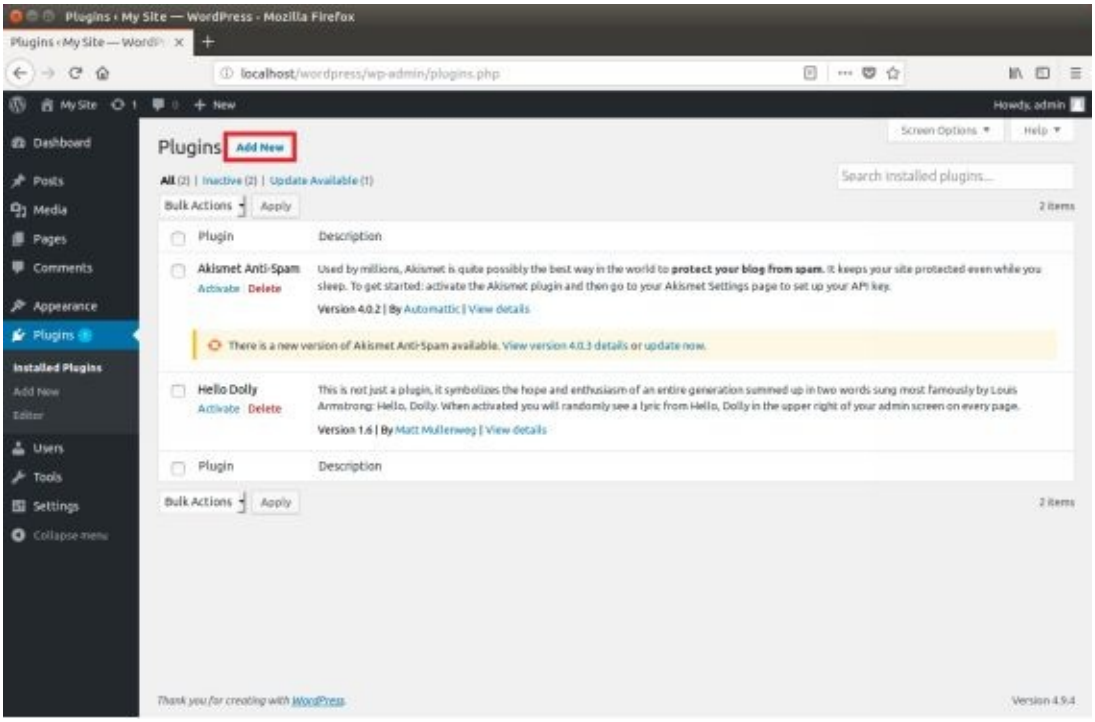
The WooCommerce plugin can be installed from WordPress dashboard screen.

Let's start the installation.

Click "Plugins" at the left of the Dashboard screen

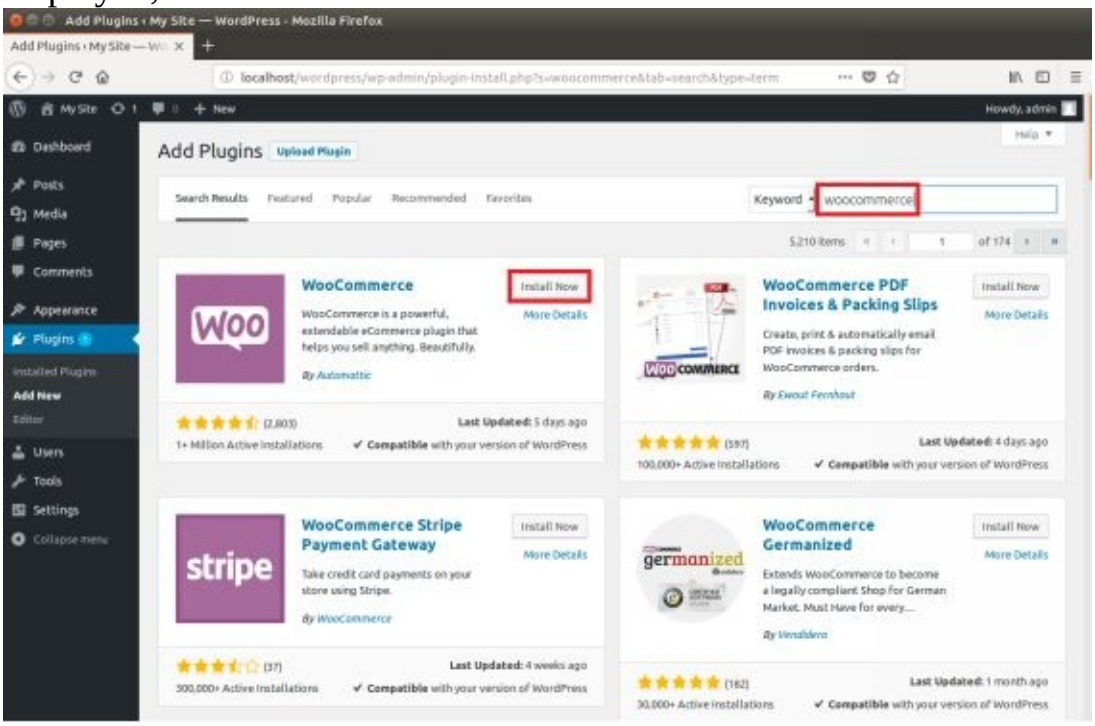


Click "Add New"

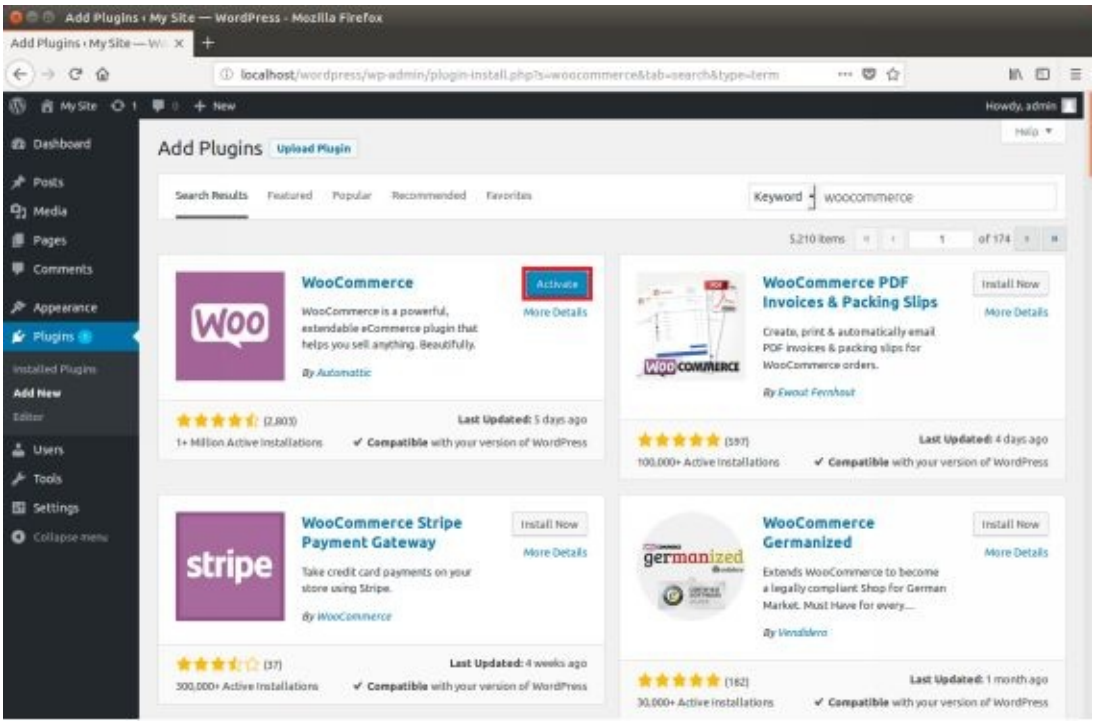


Enter "woocommerce" as the Keyword and the WooCommerce plugin is

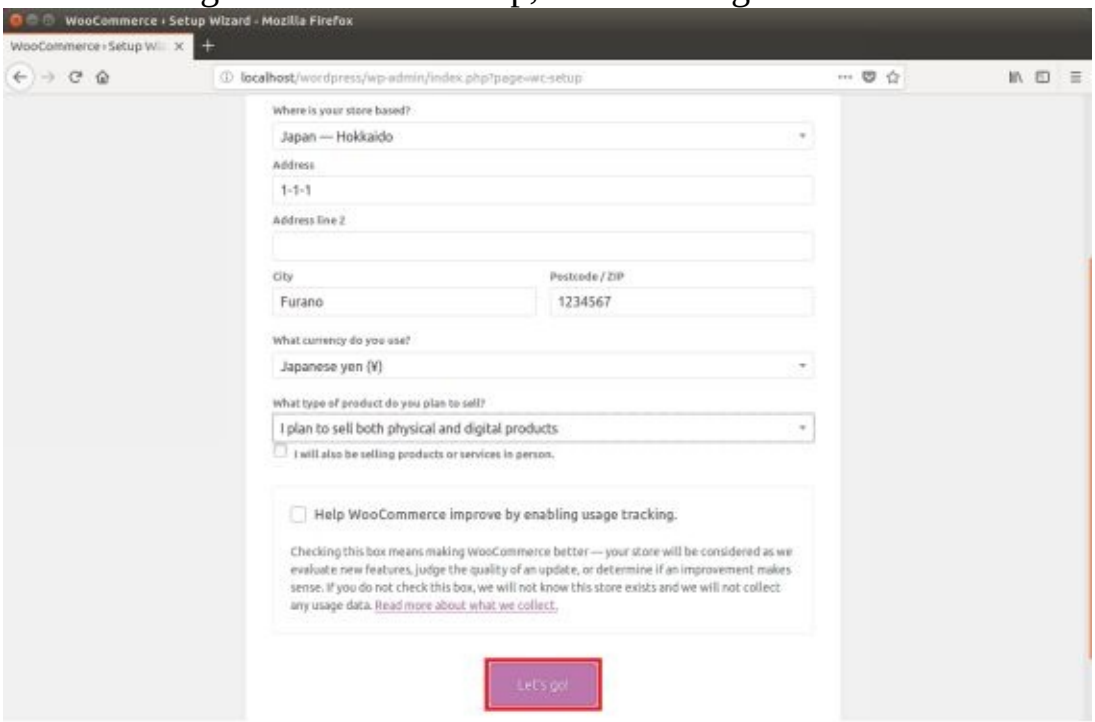
displayed, so click "Install Now"



Click "Activate"

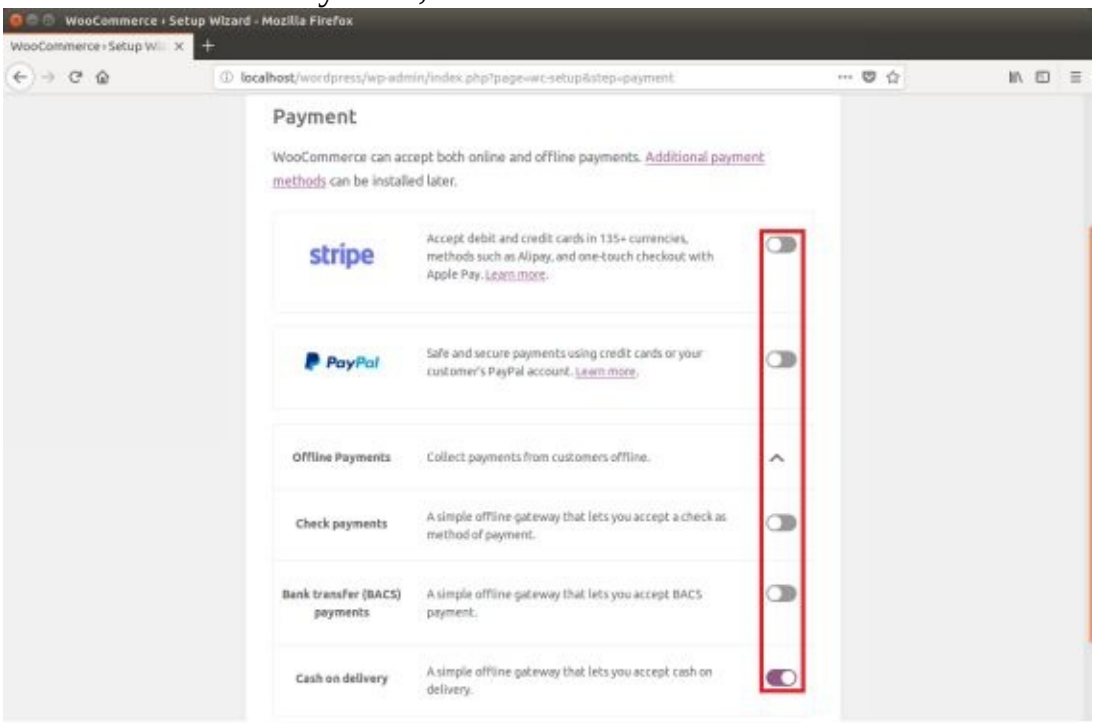


After entering the data of the shop, click "Let's go!"

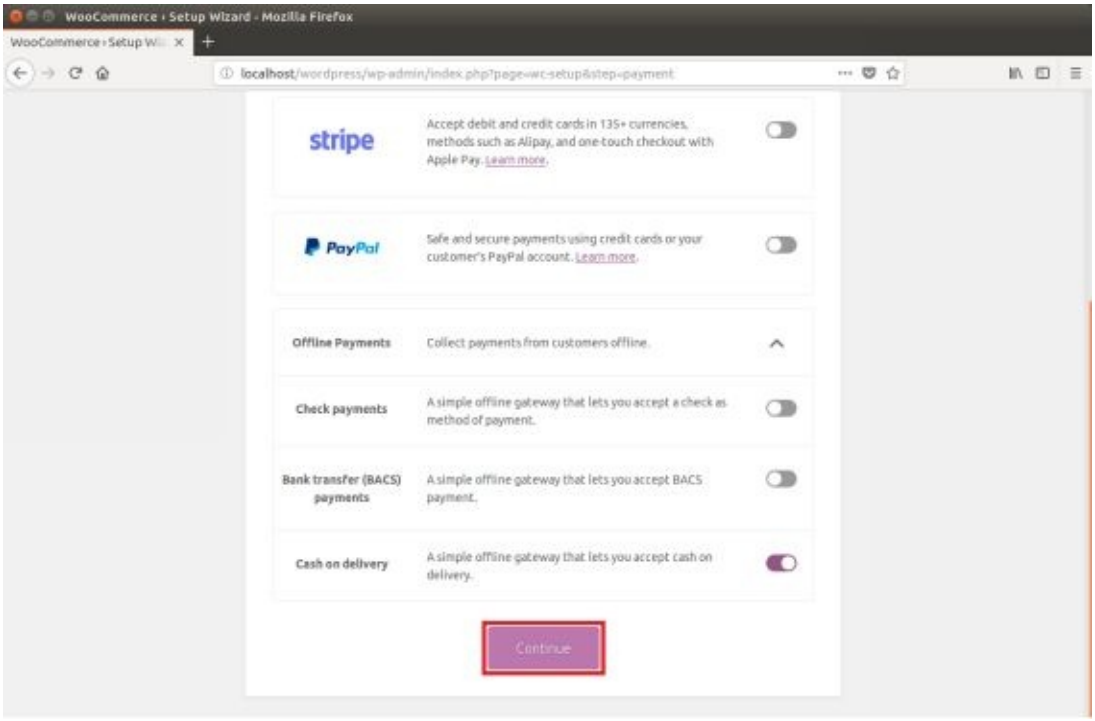


Since we use WooCommerce only for Selenium practice in this book, we enter dummy data.

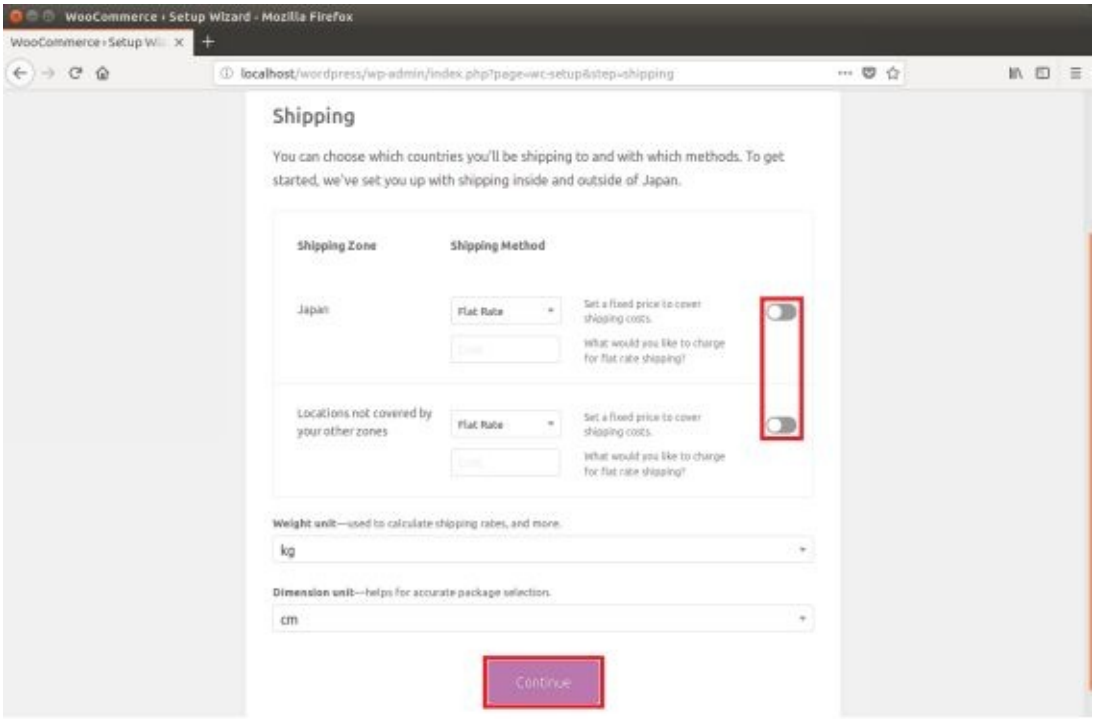
Set "Cash on delivery" ON, all others OFF



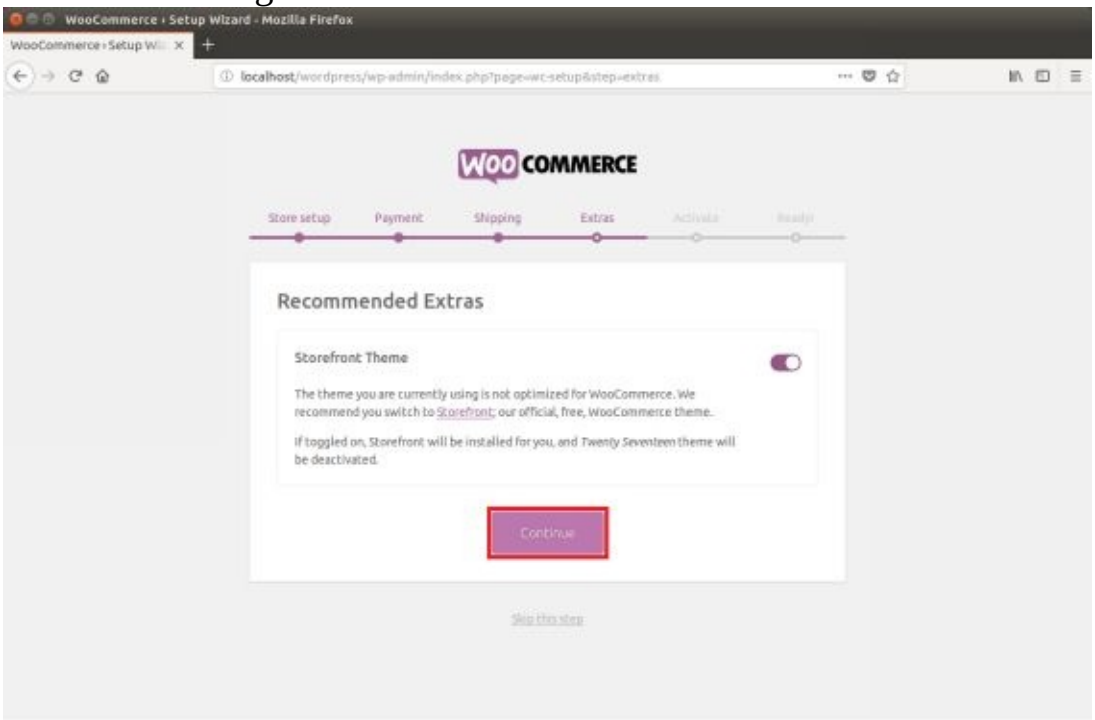
Click "Continue"



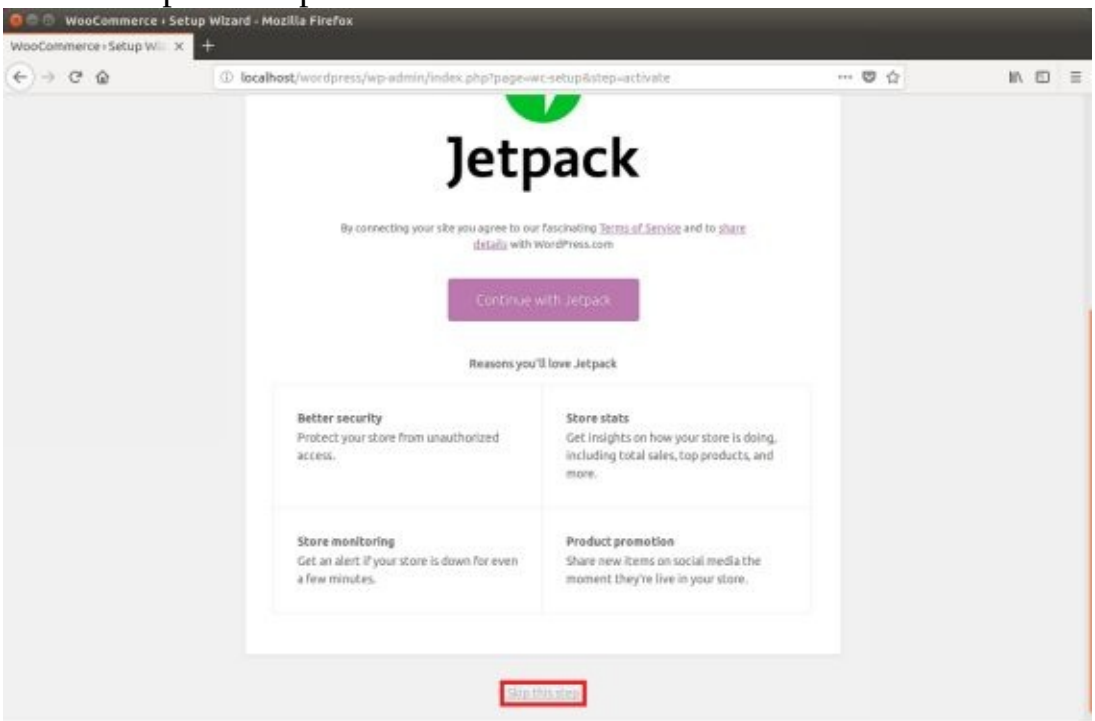
Set all OFF and click "Continue"



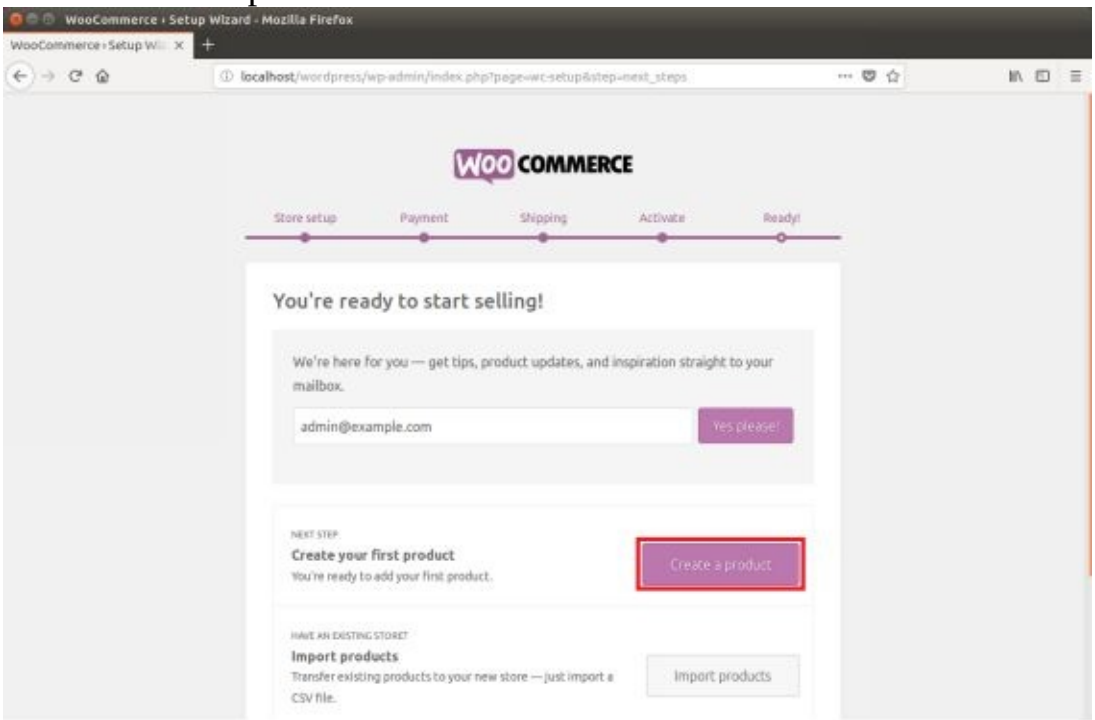
Leave this setting ON here and click "Continue"



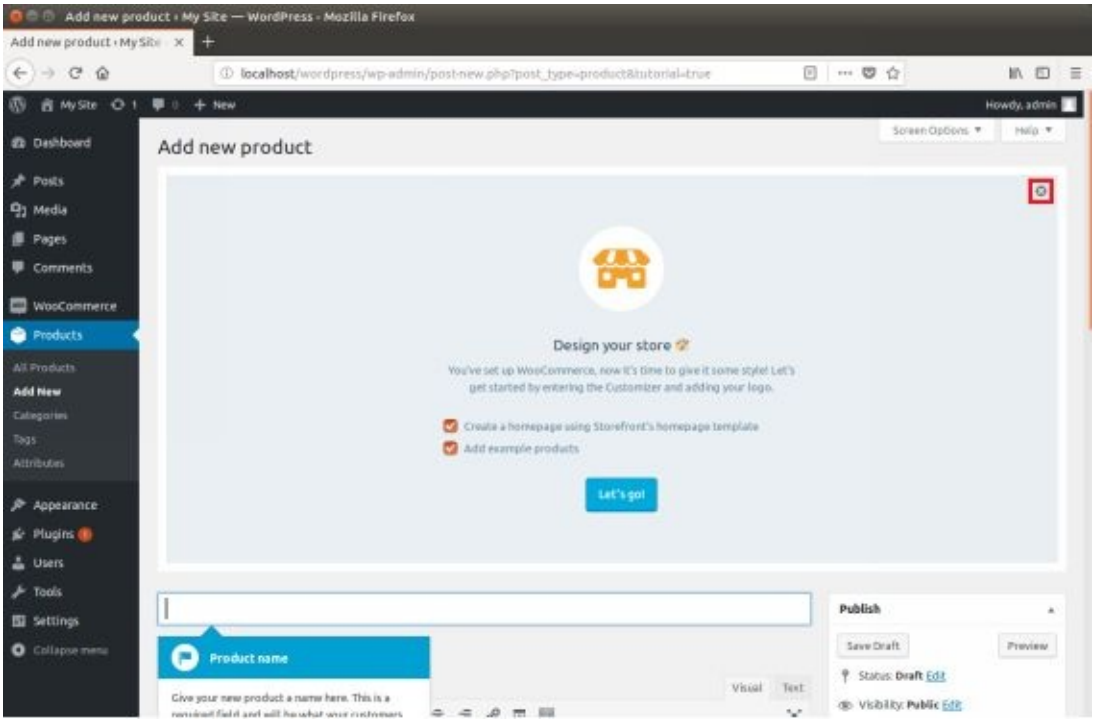
Click "Skip this step"



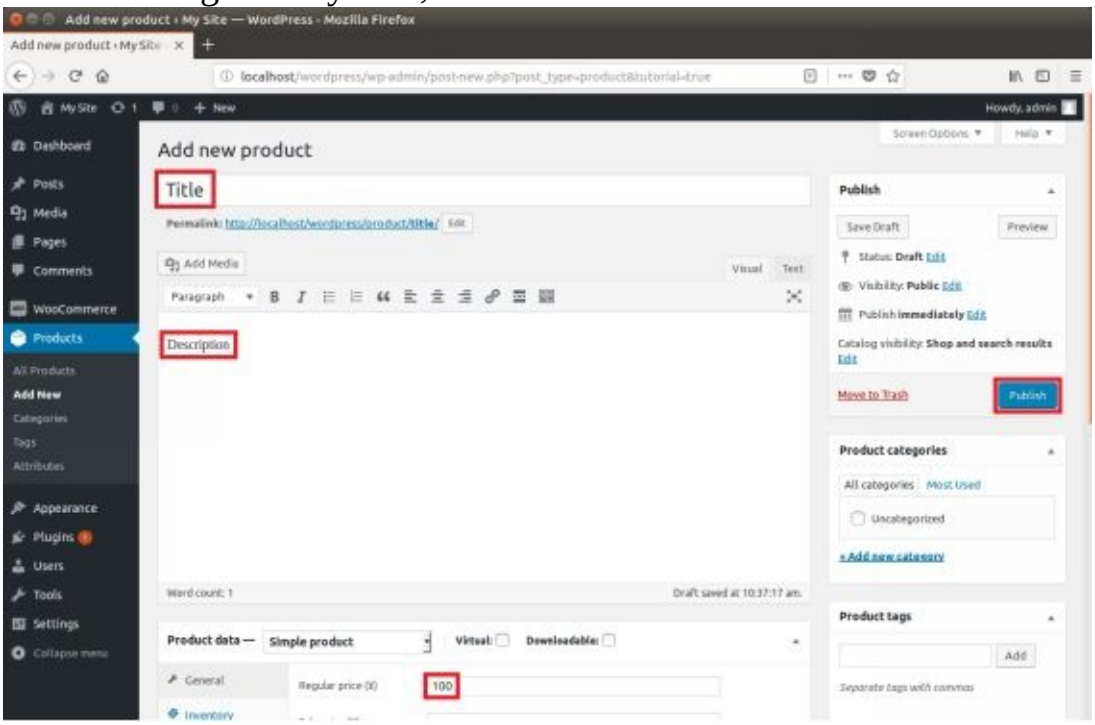
Click "Create a product"



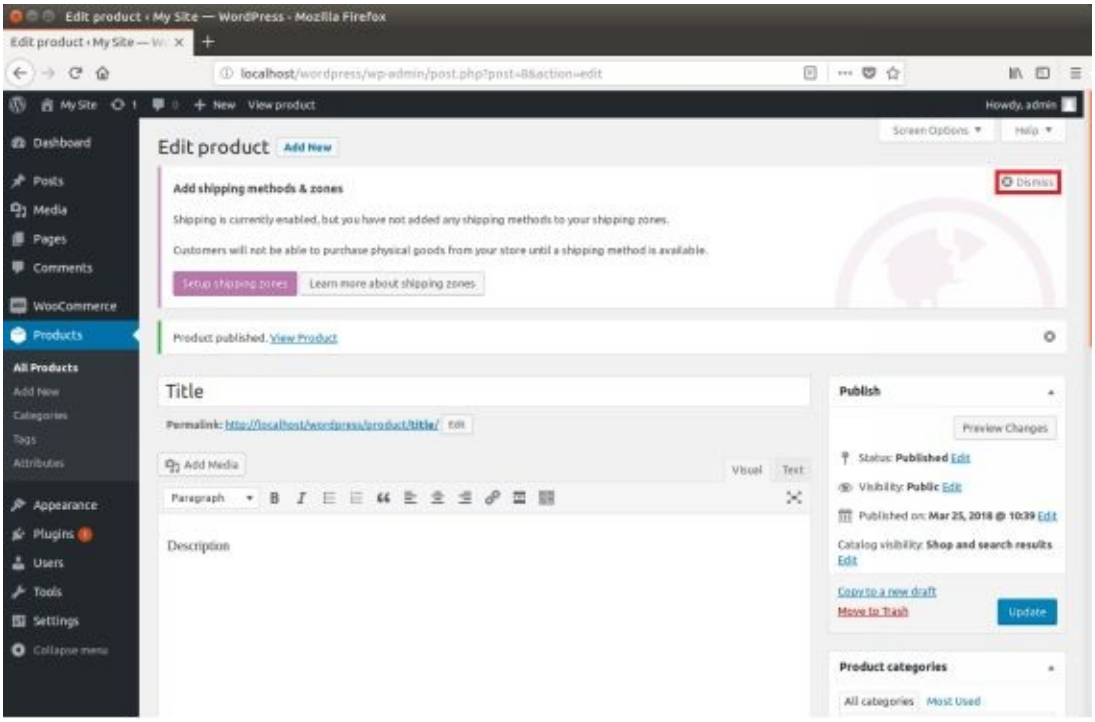
Click "×"



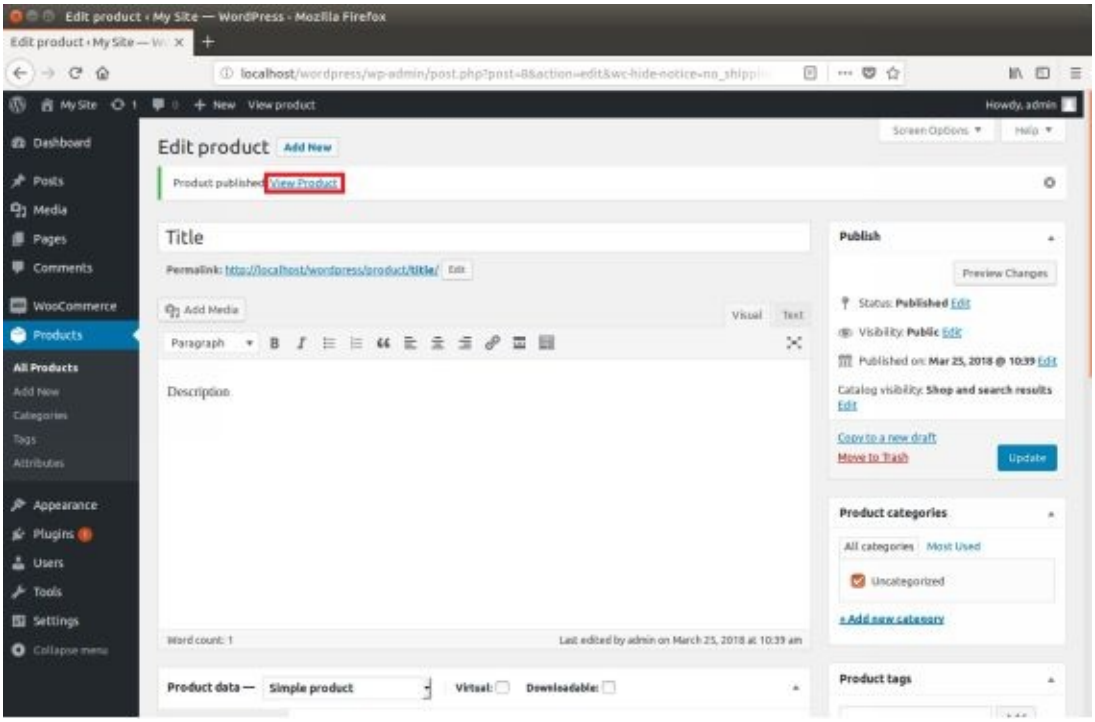
After entering dummy data, click "Publish"



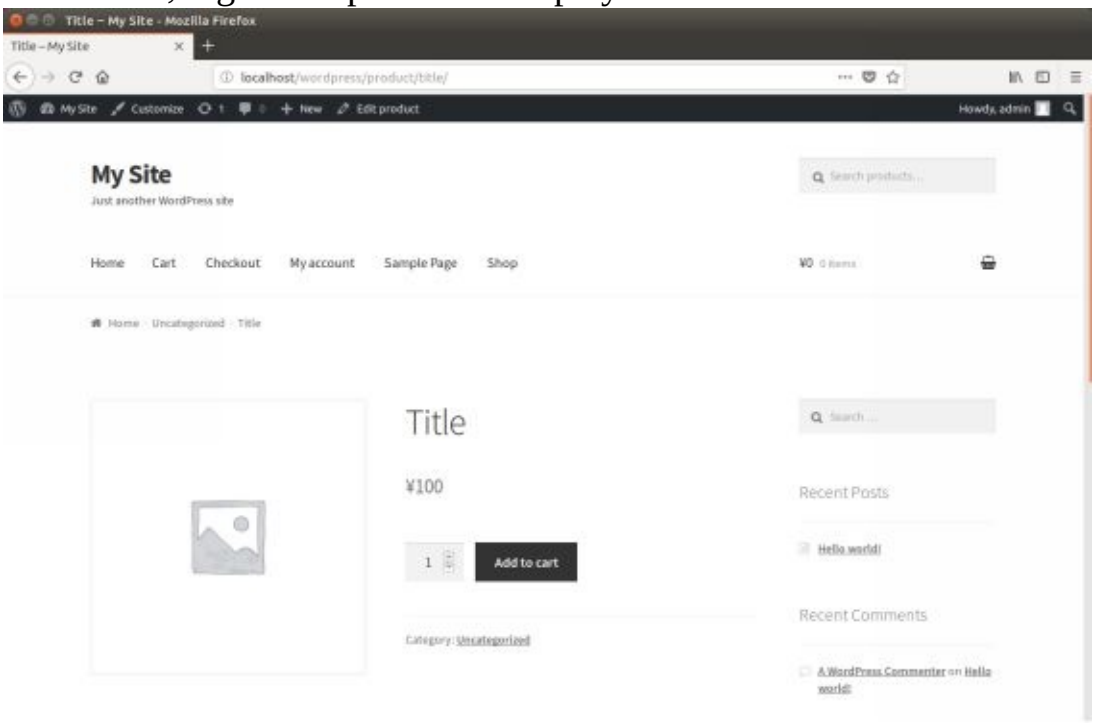
Click "× Dismiss"



Click "View Product"

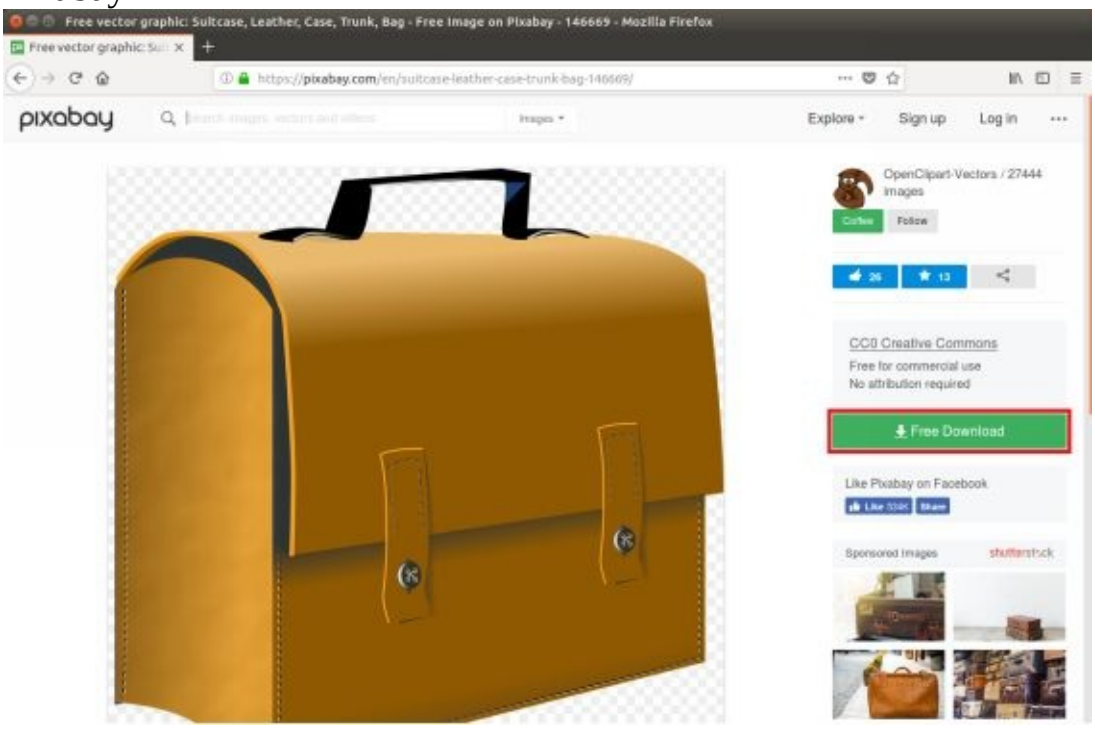


As a result, registered product is displayed.



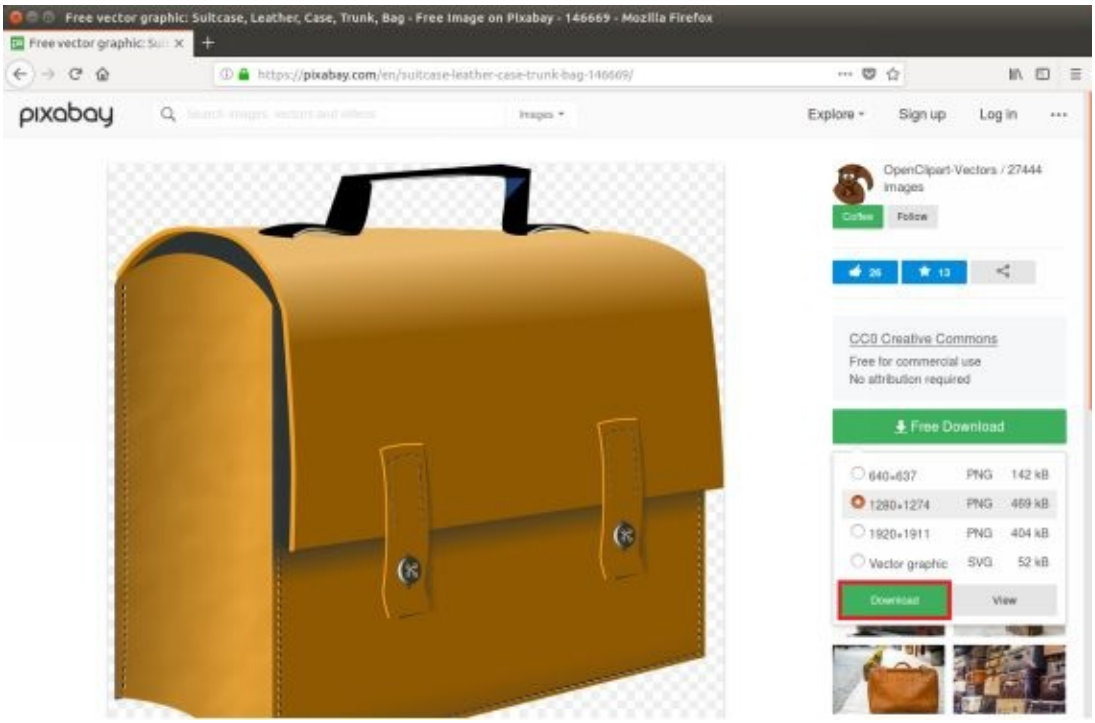
Next, we prepare the image of the product which is used when registering the product on EC site. We download the image from the following site.

Free vector graphic: Suitcase, Leather, Case, Trunk, Bag - Free Image on Pixabay - 146669

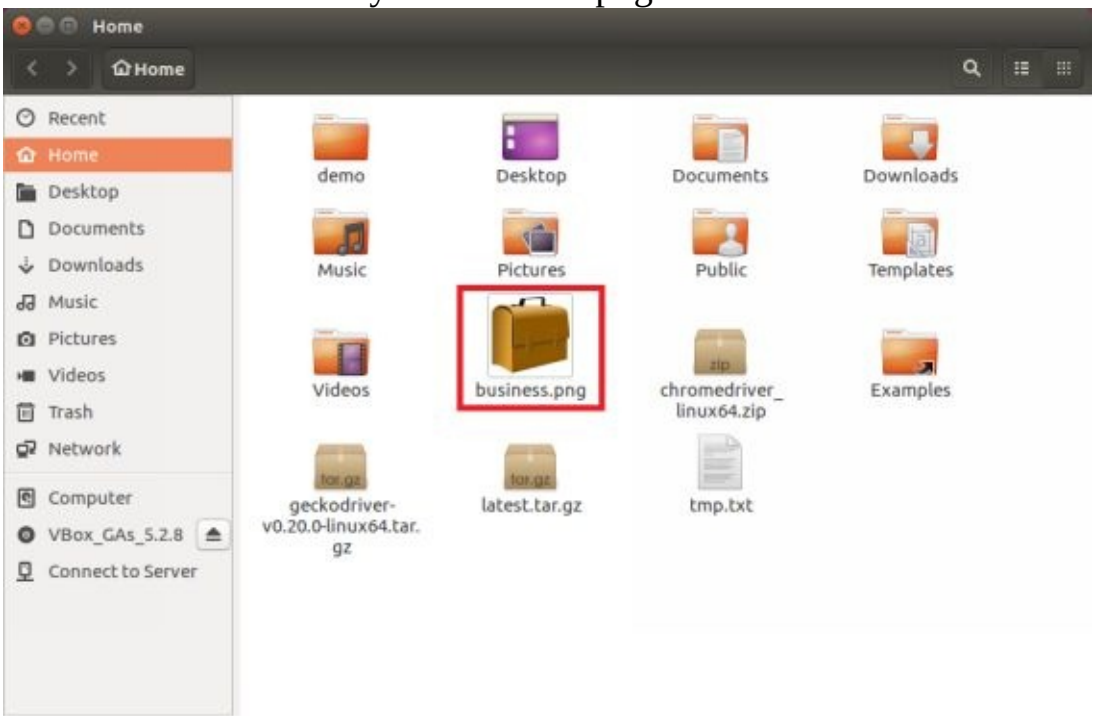


https://pixabay.com/en/suitcase-leather-case-trunk-bag-146669/

Click "Download"



Save it at home directory as "business.png"



4-2 Post blog article

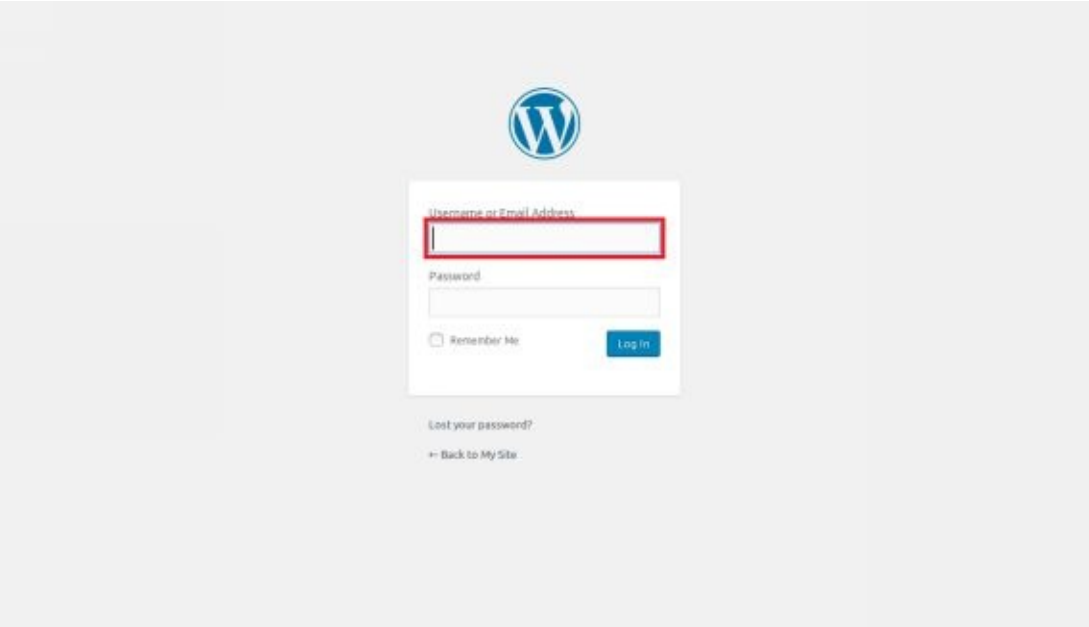
Here we automatically perform the following operations.
Login to WordPress
Enter blog title and blog content
Publish

4-2-1 Flow of posting blog article

The flow of posting blog article is as follows.

Open "http://localhost/wordpress/wp-login.php" in the browser

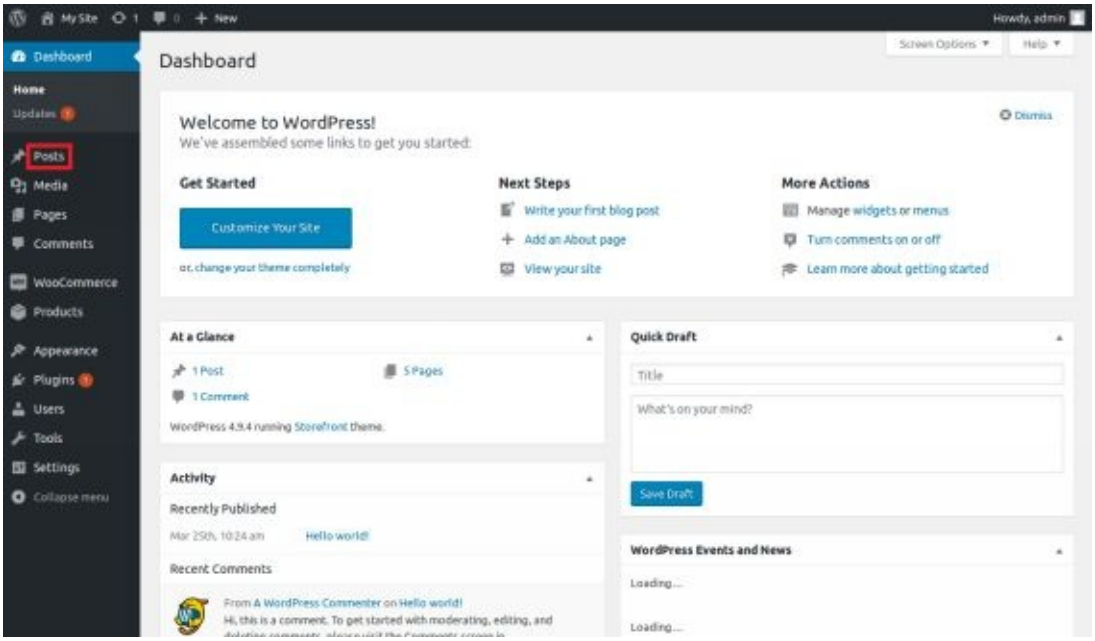
Enter "admin" as Username or Email Address



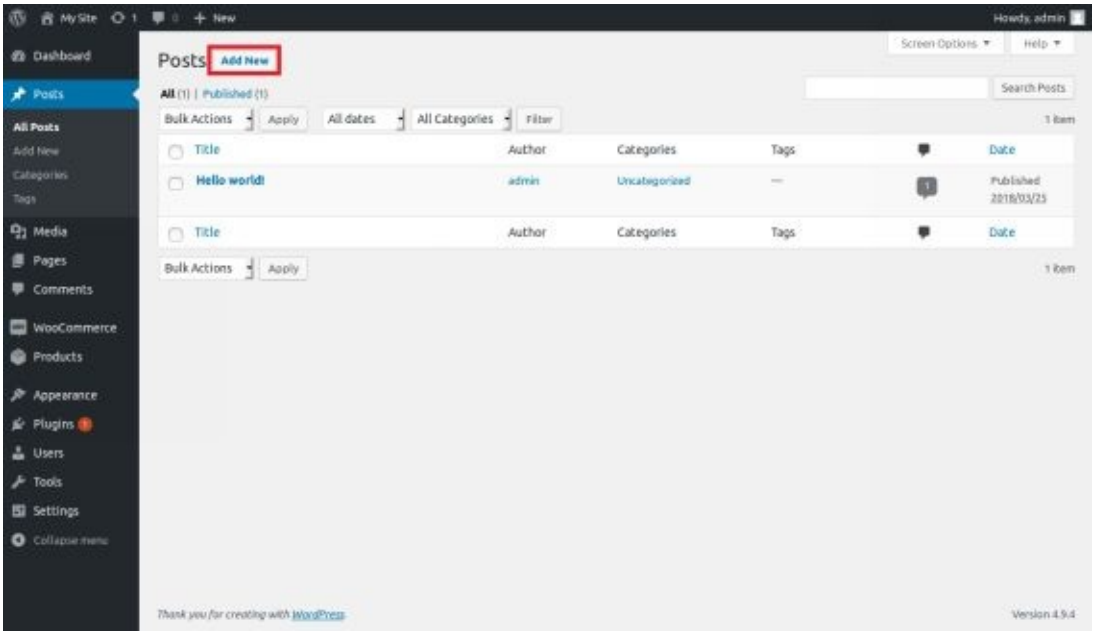
Enter "admin123" as password

Click "Log In"

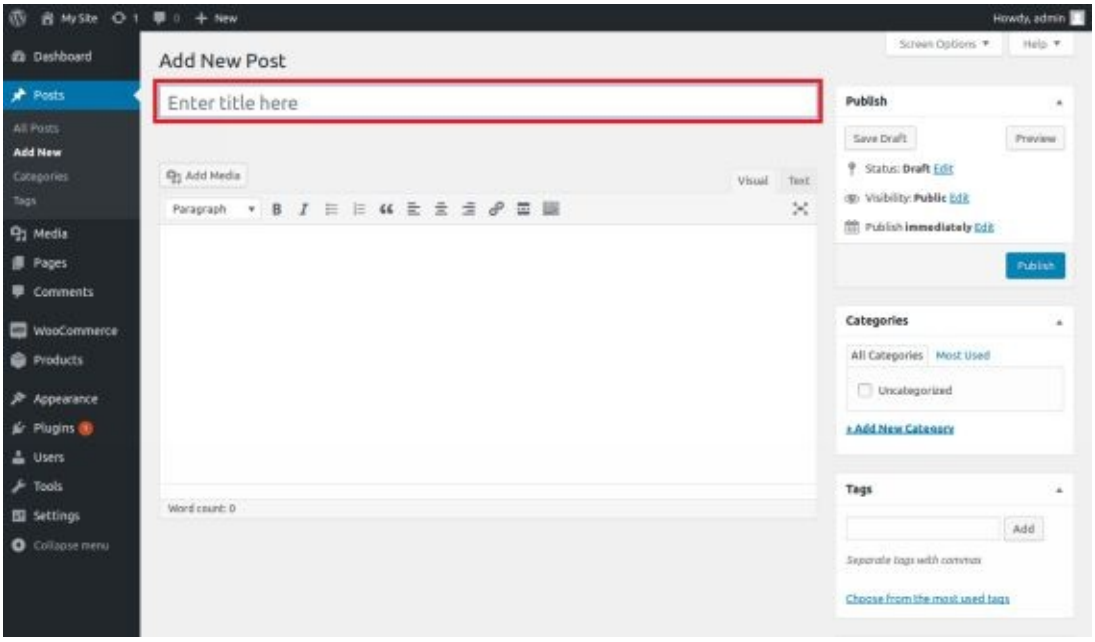
Click "Posts"



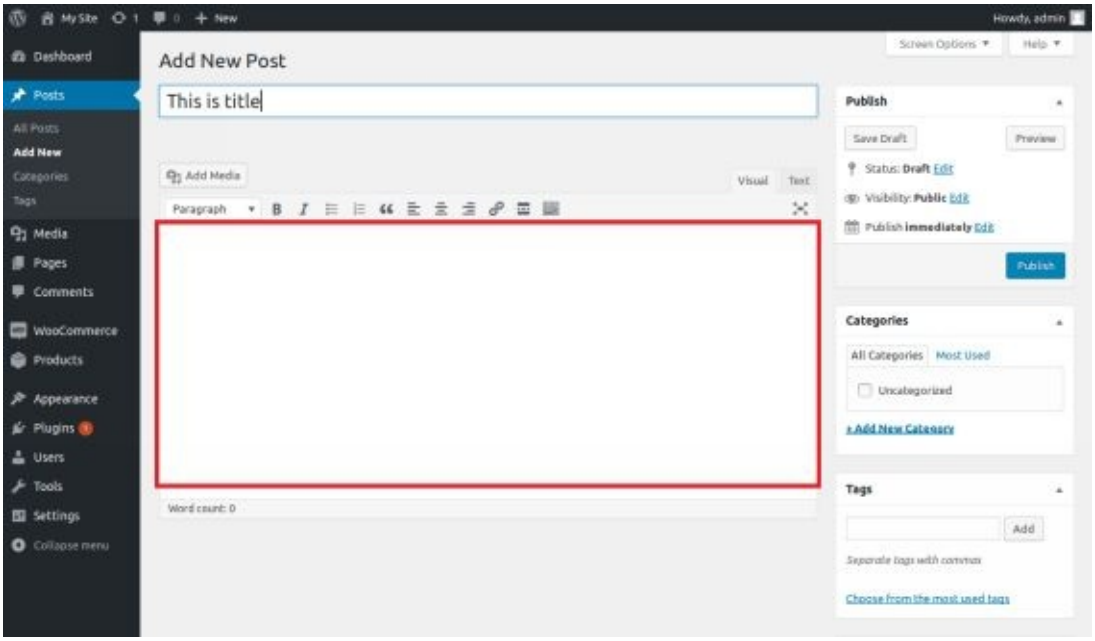
Click "Add New"



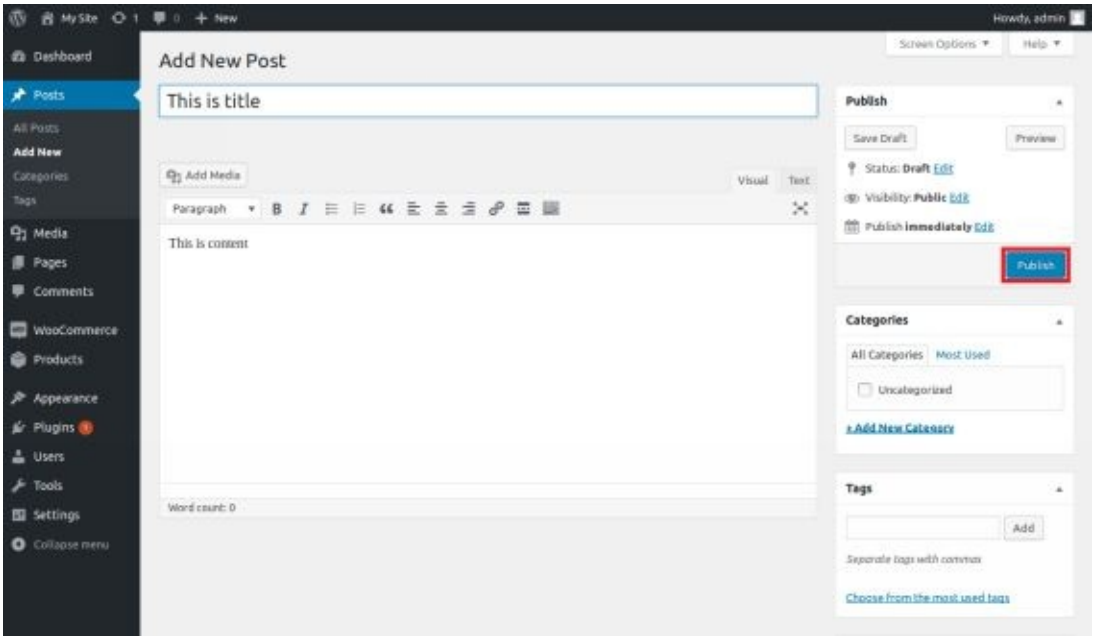
Enter "This is title" in the title



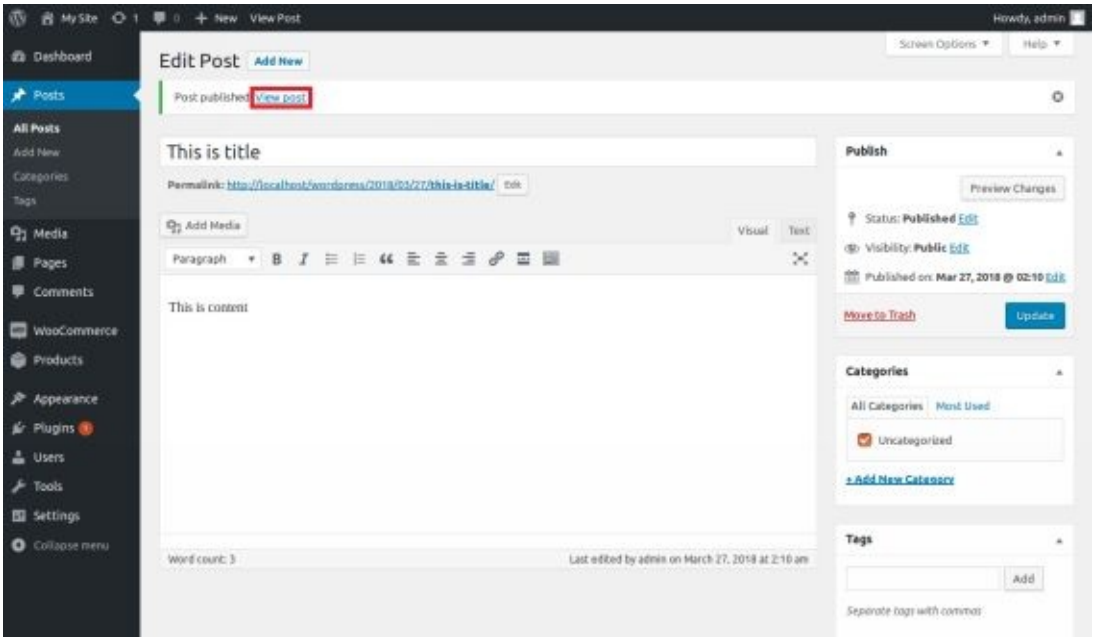
Enter "This is content" in the content



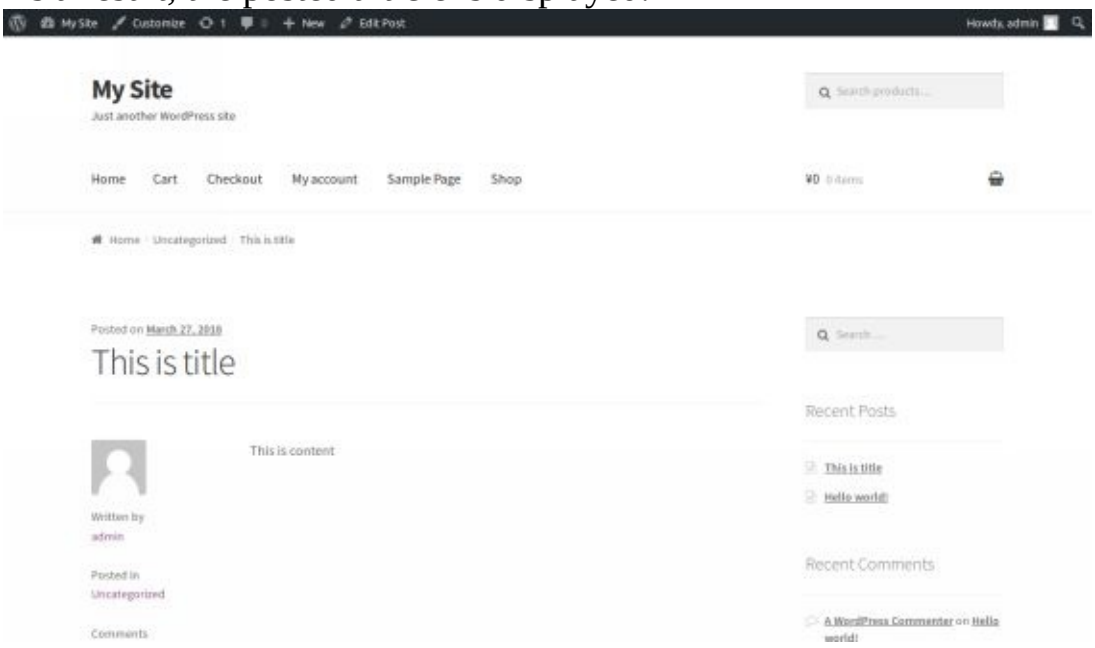
Click "Publish"



Click "View post"



As a result, the posted article is displayed.



4-2-2 Create automatic operation code for operation target element

Here, we create automatic operation code from the HTML of the target element for each operation but please pay attention to the following points.

There are several "Add New" links in the page

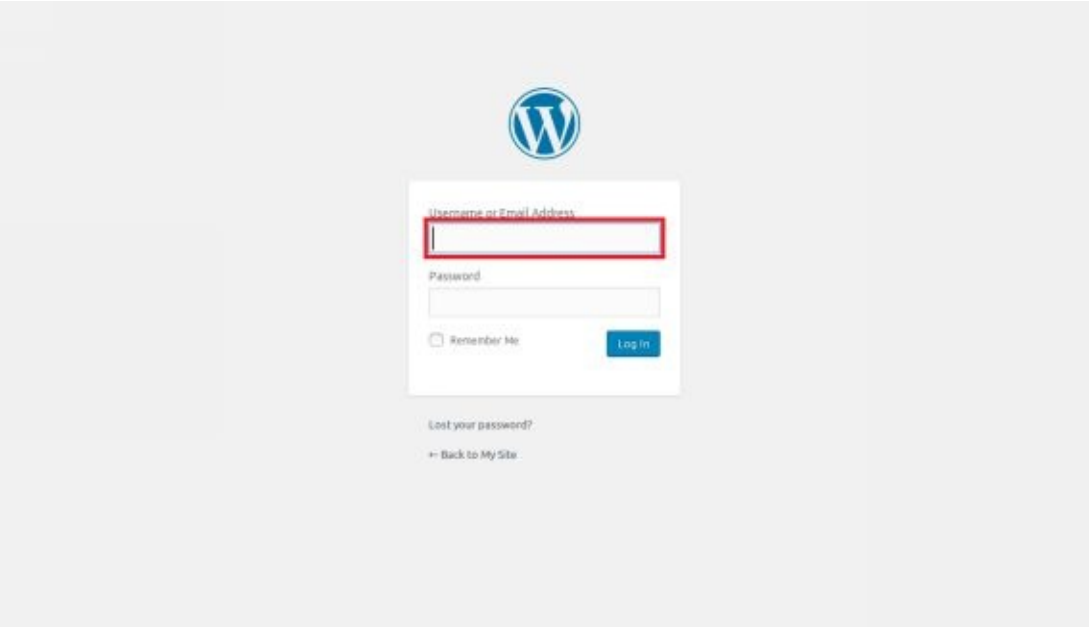
In order to enter text data we need to set the value to innerHTML of body tag element after changing frame

Open "http://localhost/wordpress/wp-login.php" in the browser

Automatic operation code

```
driver.get("http://localhost/wordpress/wp-login.php")
```

Enter "admin" as Username or Email Address



HTML of operation target element

```
<input name="log" id="user_login" class="input" value="" size="20" type="text">
```

Automatic operation code

```
driver.find_element_by_id("user_login").send_keys("admin")
```

Enter "admin123" as password

HTML of operation target element

```
<input name="pwd" id="user_pass" class="input" value="" size="20" type="password">
```

Automatic operation code

```
driver.find_element_by_id("user_pass").send_keys("admin123")
```

Click "Log In"

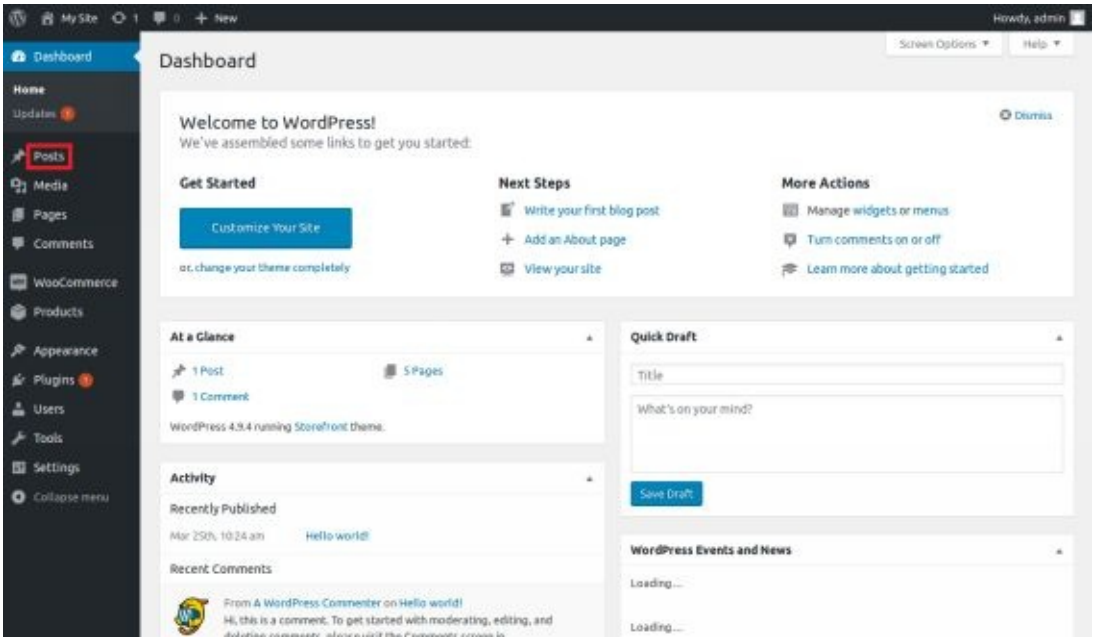
HTML of operation target element

```
<input name="wp-submit" id="wp-submit" class="button button-primary button-large" value="Log In" type="submit">
```

Automatic operation code

```
driver.find_element_by_id("wp-submit").click()
```

Click "Posts"



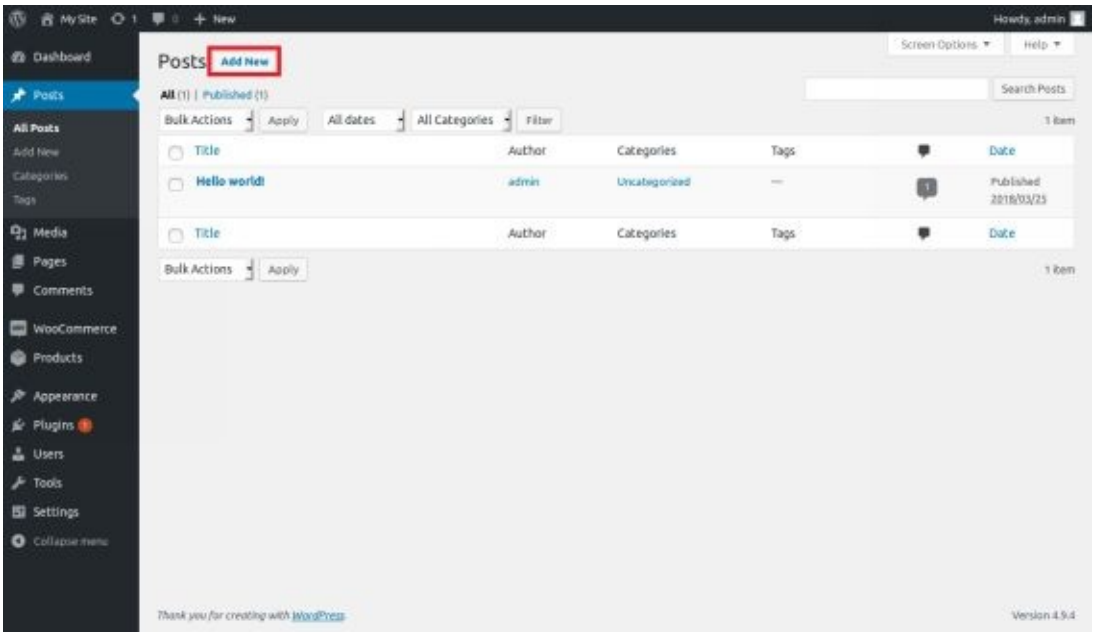
HTML of operation target element

```
<div class="wp-menu-name">Posts</div>
```

Automatic operation code

```
driver.find_element_by_xpath("//div[text()='Posts']").click()
```

Click "Add New"



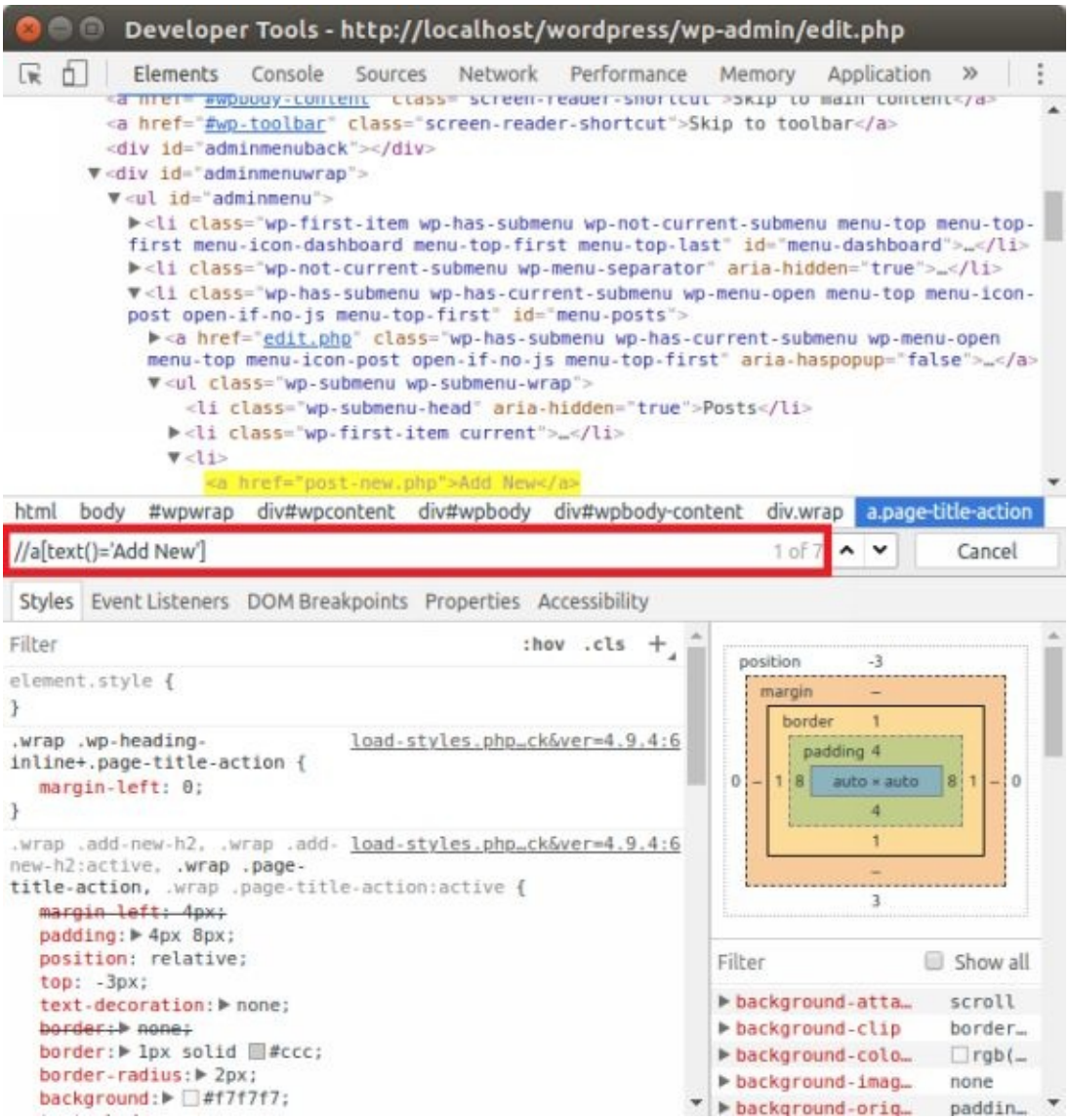
HTML of operation target element

```
<a href="http://localhost/wordpress/wp-admin/post-new.php" class="page-title-action">Add New</a>
```

Let's check the following xpath in the search box.

```
//a[text()='Add New']
```

The search result is shown below.

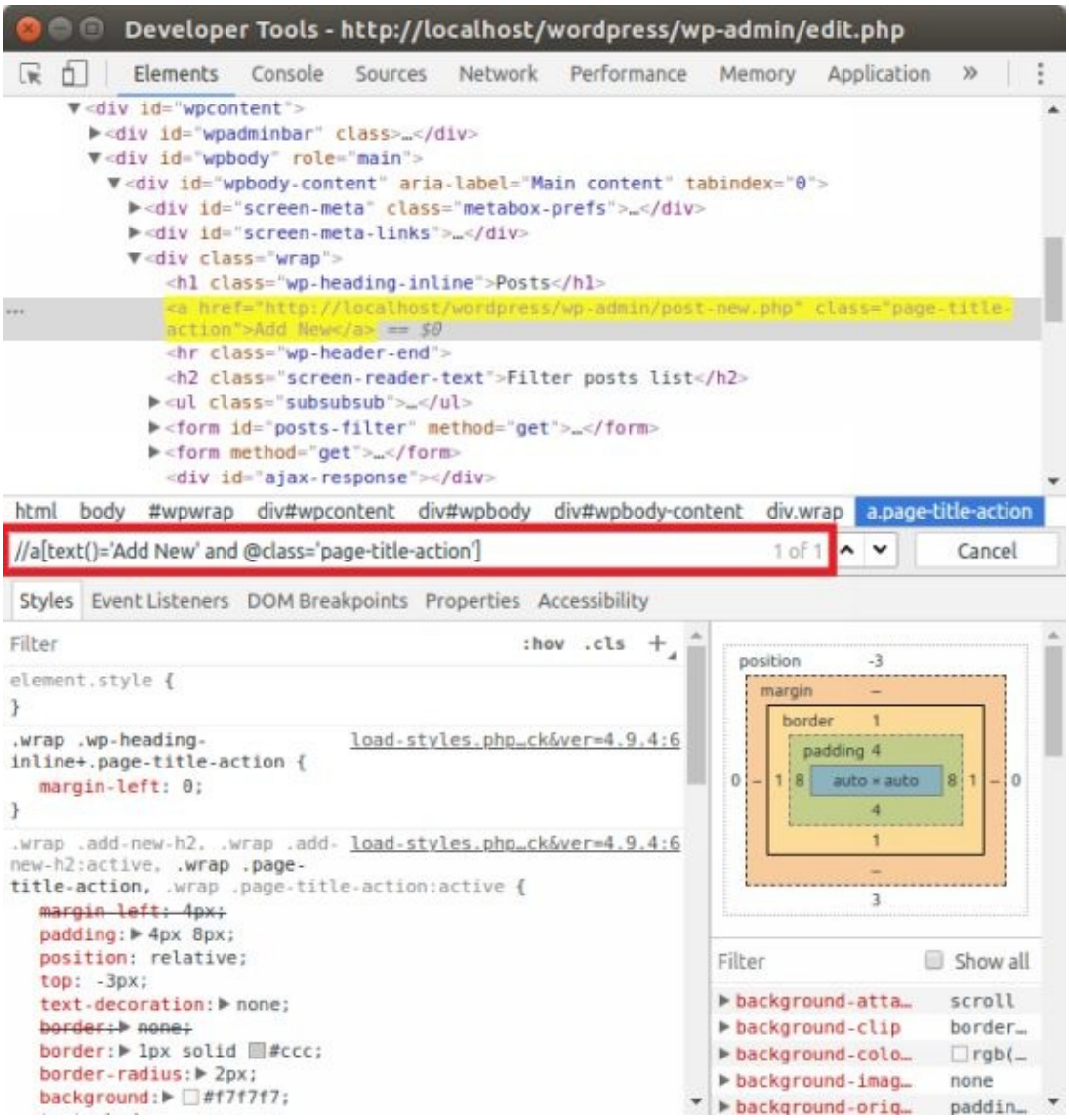


Since the result says "1 of 7", we can see that there are seven elements which are identified by the same xpath in this page.

So, next, let's check the following xpath in the search box.

```
//a[text()='Add New' and @class='page-title-action']
```

The search result is shown below.

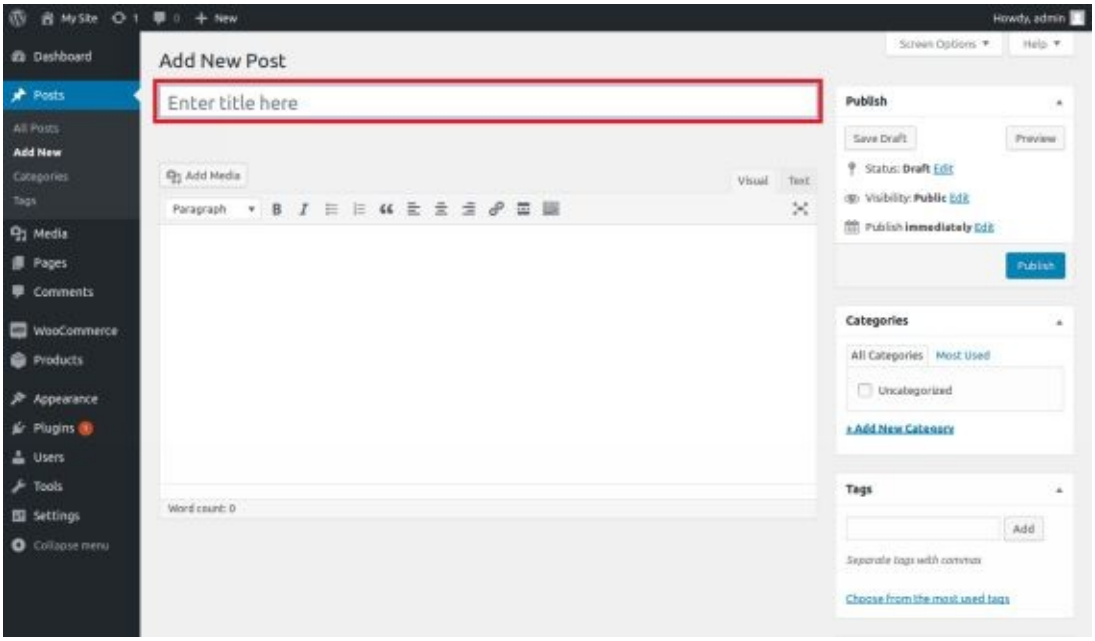


Since the result says "1 of 1", we can see that the element can be identified as one. Therefore, we use this xpath to create code for automatic operation.

Automatic operation code

```
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()
```

Enter "This is title" in the title



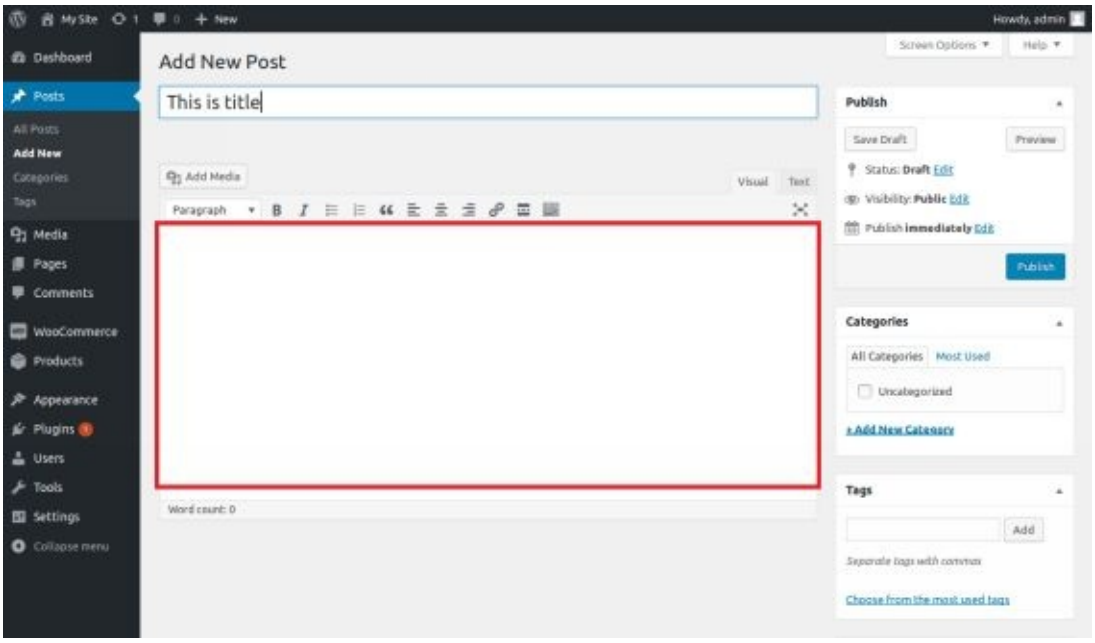
HTML of operation target element

```
<input name="post_title" size="30" value="" id="title" spellcheck="true" autocomplete="off" type="text">
```

Automatic operation code

```
driver.find_element_by_id("title").send_keys("This is title")
```

Enter "This is content" in the content



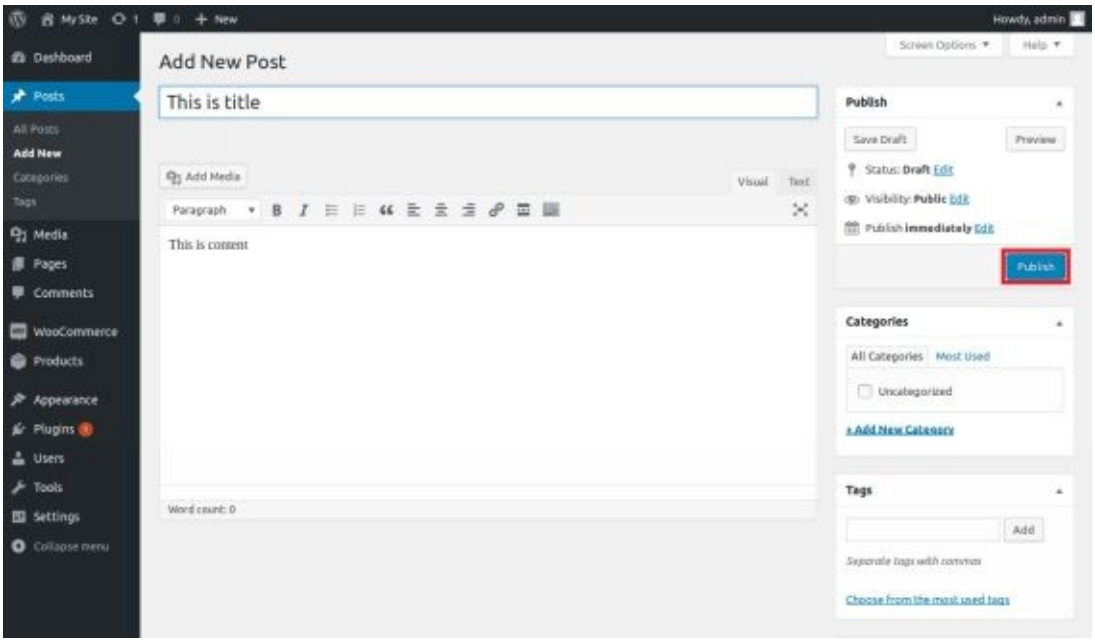
HTML of operation target element

```
<iframe id="content_ifr" ...> ...
<body id="tinymce" class="mce-content-body content post-type-post post-status-auto-draft post-format-standard page-template-default locale-ja mceContentBody wp-editor wp-autoresize html5-captions" data-id="content" style="overflow-y: hidden;" contenteditable="true"><p><br data-mce-bogus="1"></p></body>
...
</iframe>
```

Automatic operation code

```
driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML="This is content";')
driver.switch_to.default_content()
```

Click "Publish"



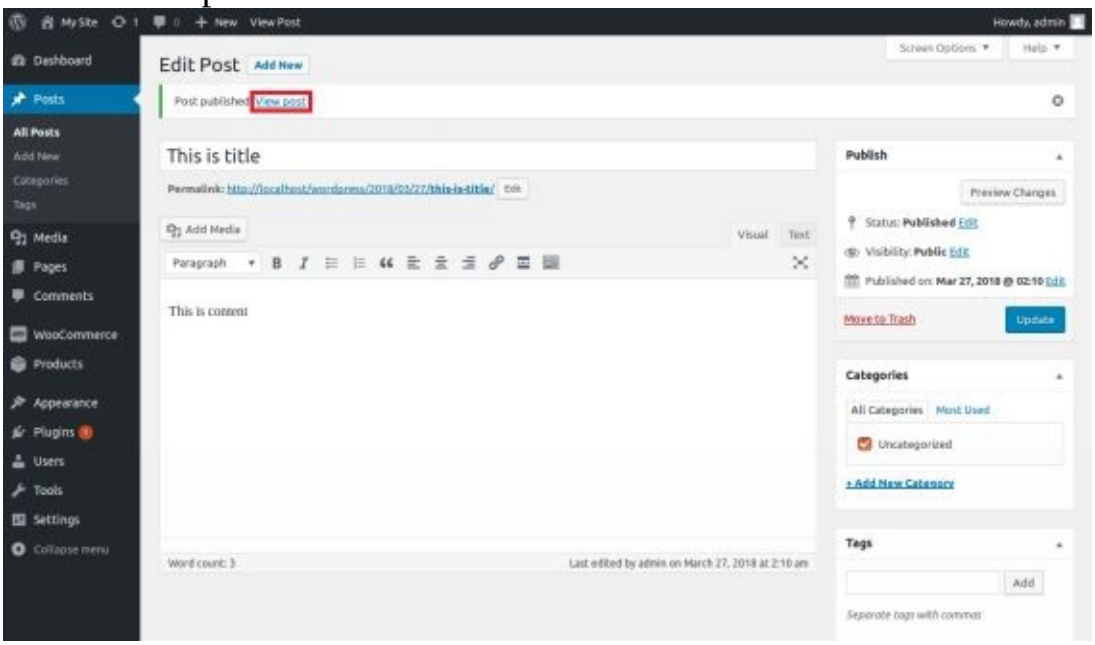
HTML of operation target element

```
<input name="publish" id="publish" class="button button-primary button-large" value="Publish" type="submit">
```

Automatic operation code

```
driver.find_element_by_id("publish").click()
```

Click "View post"



HTML of operation target element

```
<a href="http://localhost/wordpress/2018/03/27/this-is-title-2/">View post</a>
```

Automatic operation code

```
driver.find_element_by_xpath("//a[text()='View post']").click()
```

4-2-3 Complete and execute automatic operation code

Since all the codes of automatic operation of blog article posting have already been created, the following code completes the code of automatic operation of the blog post in one file.

post_article_firefox.py

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Posts']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()

driver.find_element_by_id("title").send_keys("This is title")

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML="This is content";')
driver.switch_to.default_content()

driver.find_element_by_id("publish").click()

driver.find_element_by_xpath("//a[text()='View post']").click()
```

post_article_chrome.py (This code fails in the middle)

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")
```

```
driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Posts']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()

driver.find_element_by_id("title").send_keys("This is title")

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML="This is content";')
driver.switch_to.default_content()

driver.find_element_by_id("publish").click()

driver.find_element_by_xpath("//a[text()='View post']").click()
```

We set 10 seconds for Implicit Wait in the above code.

```
driver.implicitly_wait(10)
```

When using Selenium, the element to be operated may not be displayed immediately and it may take some time to display. Wait is necessary in that case. Selenium waits up to 10 seconds even if the element can’t be found immediately by setting Implicit Wait to 10 seconds.

The above code works fine for Firefox but fails for Chrome.

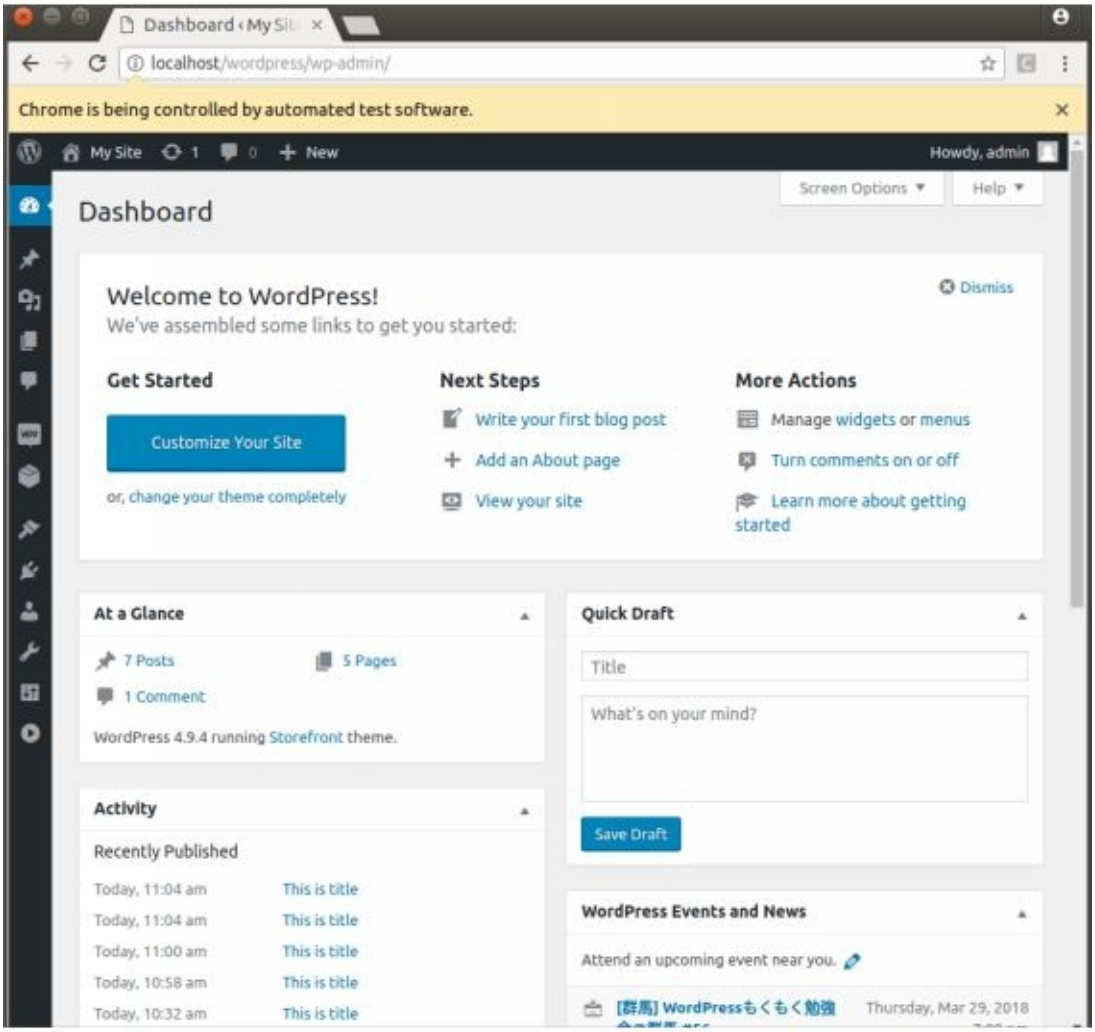
An error occurs in the following part of the code for Chrome.

```
driver.find_element_by_xpath("//div[text()='Posts']").click()
```

When executing the above code, "element not visible" error occurs as shown in the figure below.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ python post_article_chrome.py
Traceback (most recent call last):
  File "post_article_chrome.py", line 11, in <module>
    driver.find_element_by_xpath("//div[text()='Posts']").click()
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webdriv
er.py", line 312, in execute
    self.error_handler.check_response(response)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/errorha
ndler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.ElementNotVisibleException: Message: element not visi
ble
  (Session info: chrome=65.0.3325.181)
  (Driver info: chromedriver=2.36.540471 (9c759b81a907e70363c6312294d30b6ccccc27
52),platform=Linux 4.13.0-37-generic x86_64)

kenpapa@kenpapa-VirtualBox:~/python$
```

The reason is the default window size of Chrome is small and the text of the menu on the left isn’t displayed as shown below.



So, it is necessary to enlarge the window size to select the menu.

The following code is the modified code. Window size change process has been added to prevent error from occurring.

post_article_chrome.py (Modified version)

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")
driver.set_window_size(1280, 800)

driver.find_element_by_id("user_login").send_keys("admin")
```

```
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Posts']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()

driver.find_element_by_id("title").send_keys("This is title")

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML="This is content";')
driver.switch_to.default_content()

driver.find_element_by_id("publish").click()

driver.find_element_by_xpath("//a[text()='View post']").click()
```

4-3 Register product on EC site

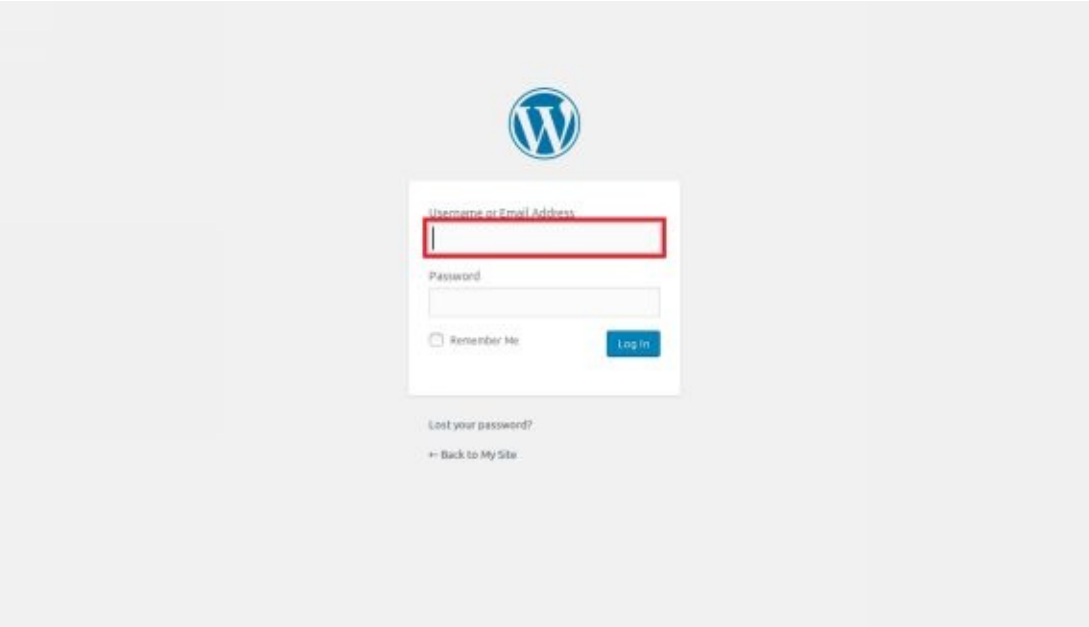
Here we automatically perform the following operations.
Login to WordPress
Enter product name, product description, product price
Set product image
Publish

4-3-1 Flow of product registration

The flow of product registration to the EC site is as follows.

Open "http://localhost/wordpress/wp-login.php" in the browser

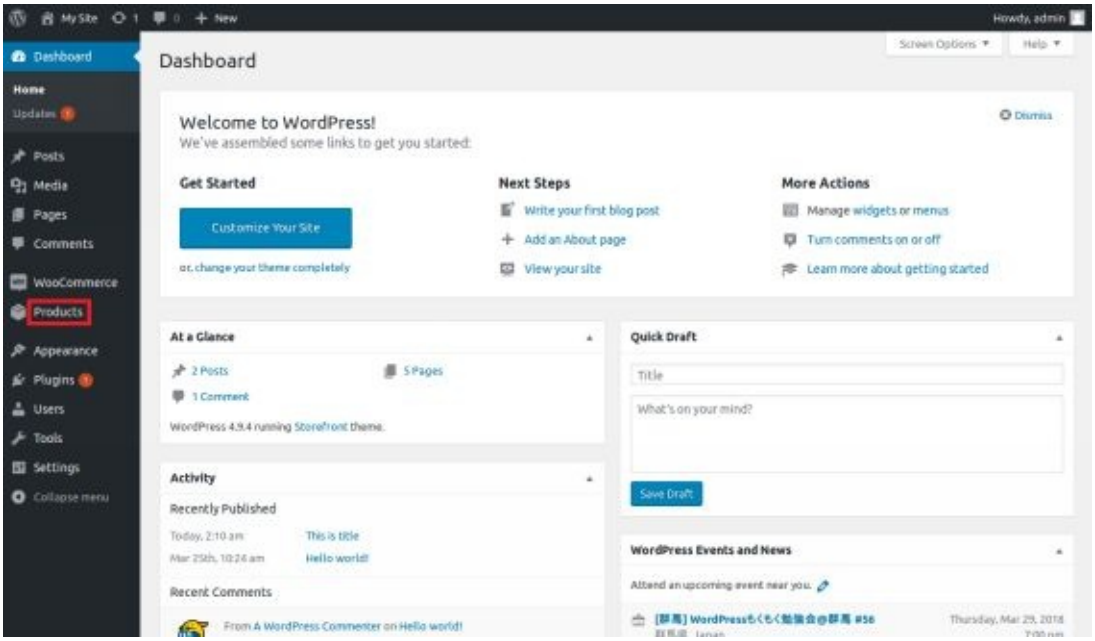
Enter "admin" as Username or Email Address



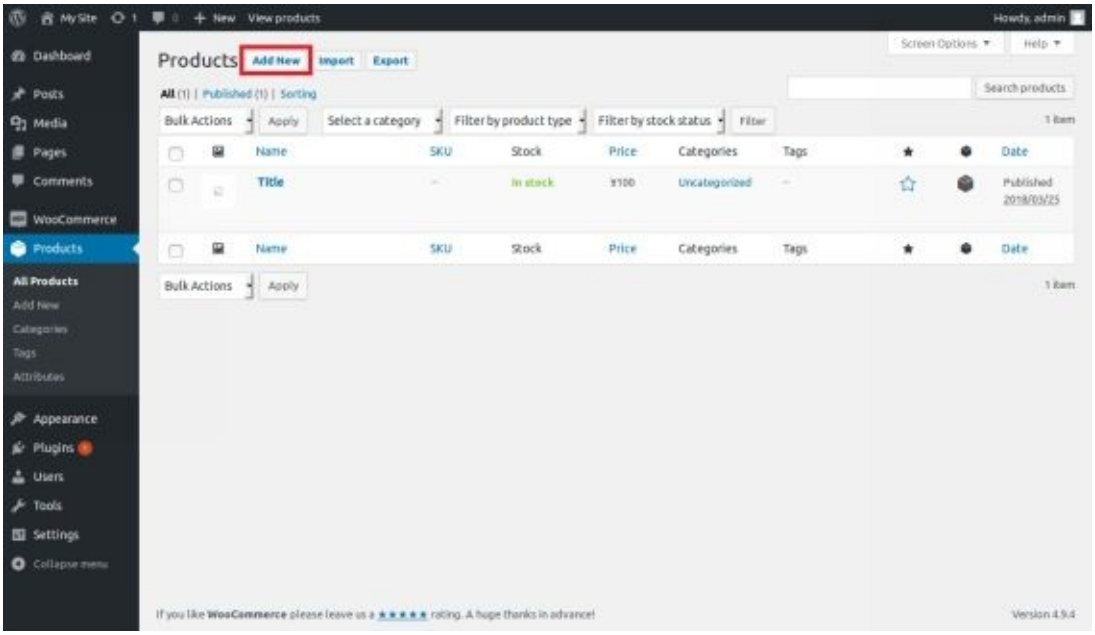
Enter "admin123" as password

Click "Log In"

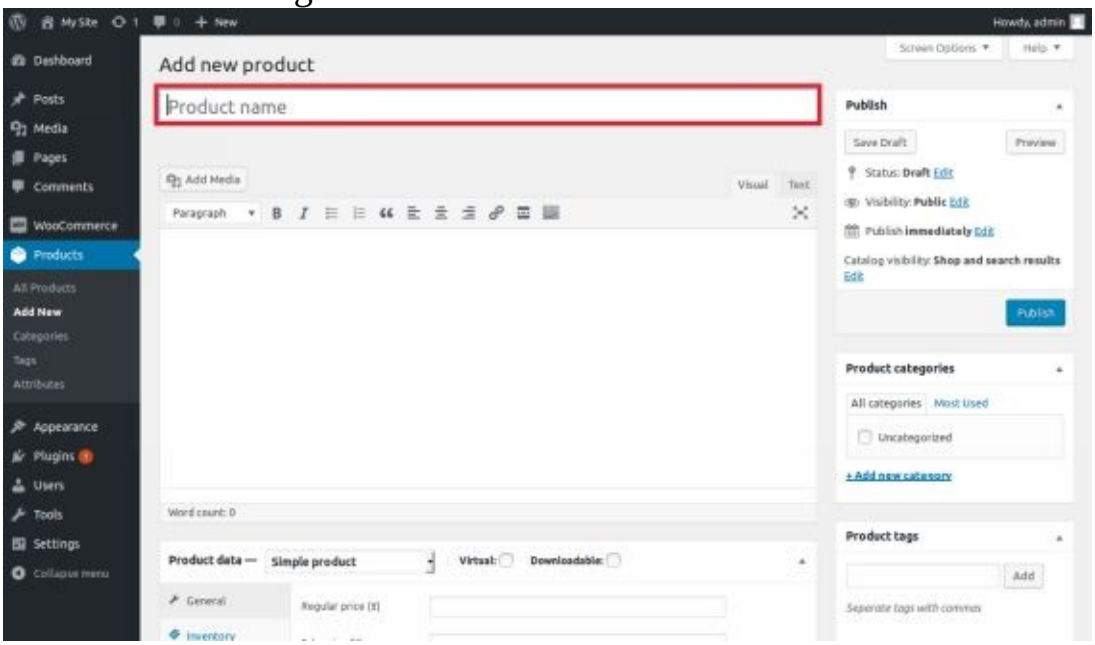
Click "Products"



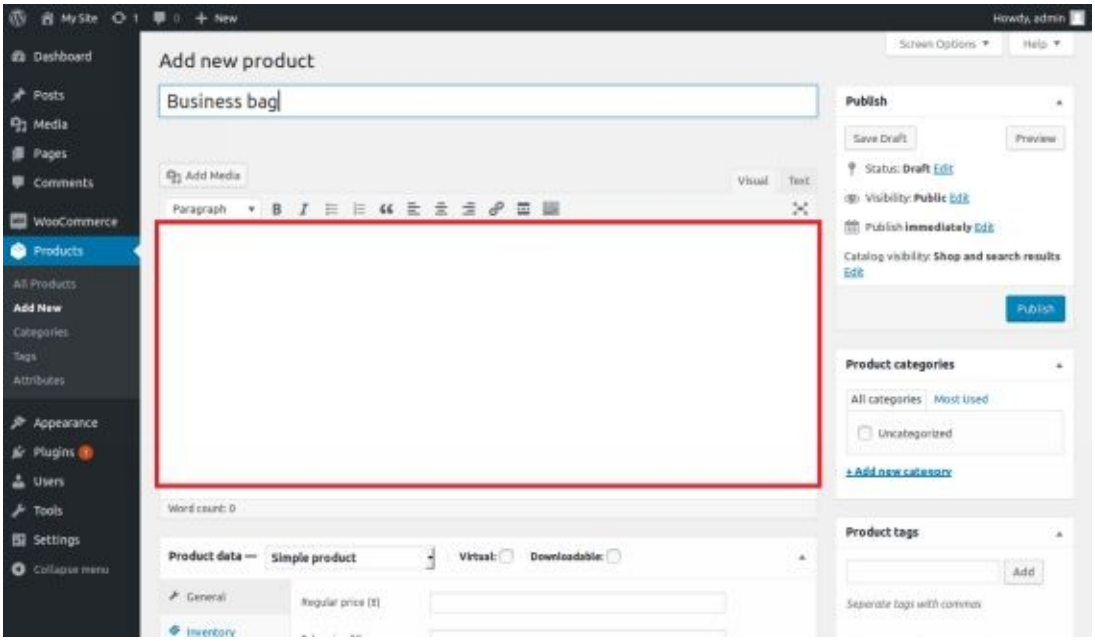
Click "Add New"



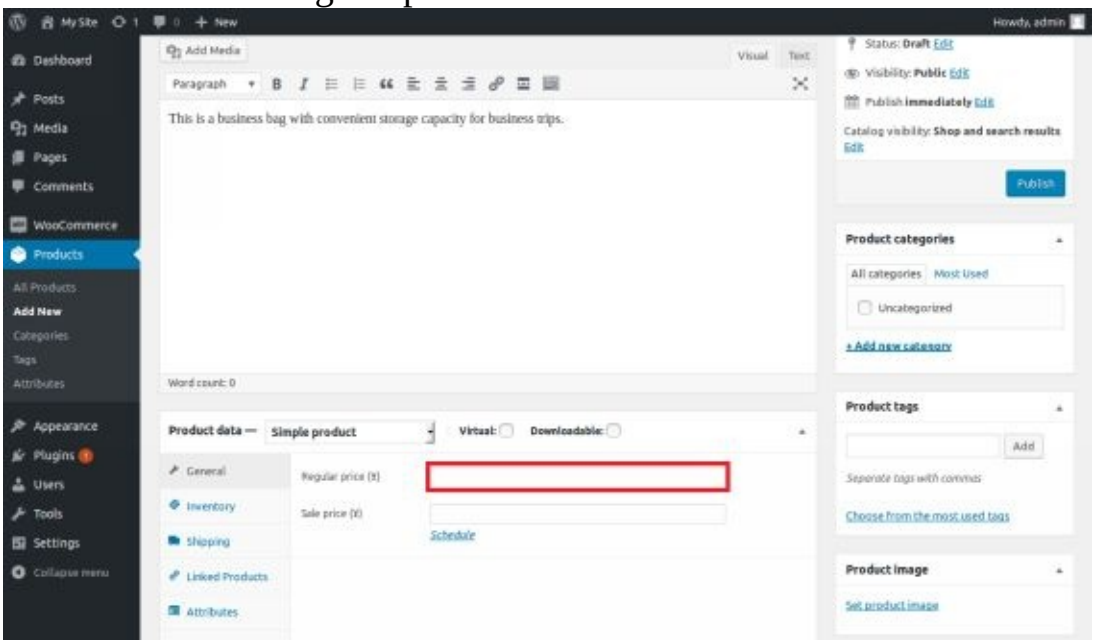
Enter "Business bag" in Product name



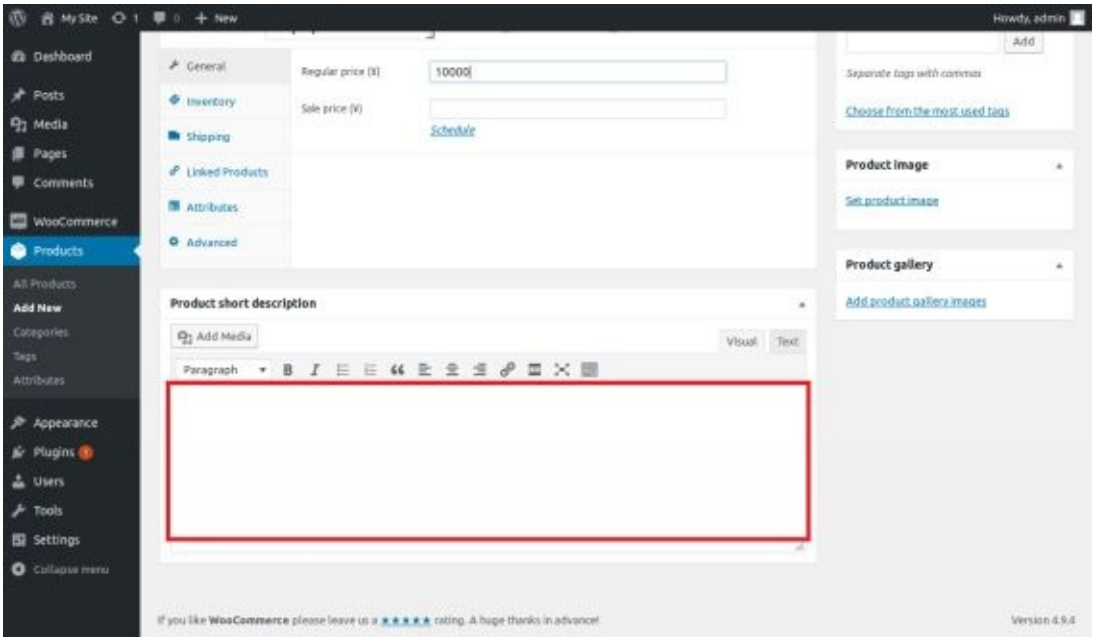
Enter "This is a business bag with convenient storage capacity for business trips." in the description



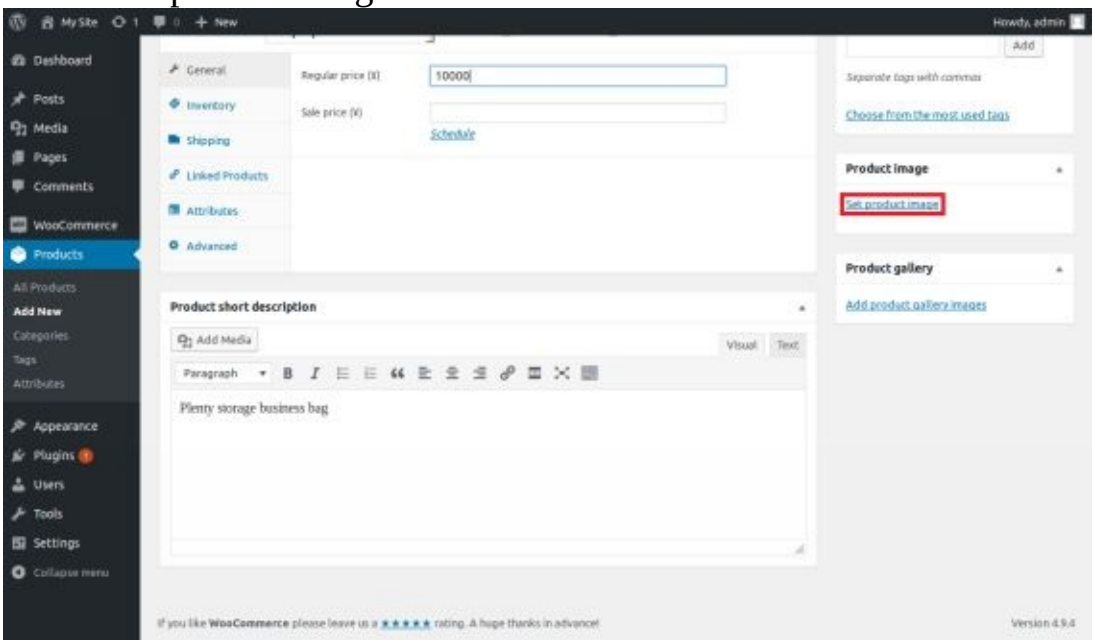
Enter "10000" in Regular price



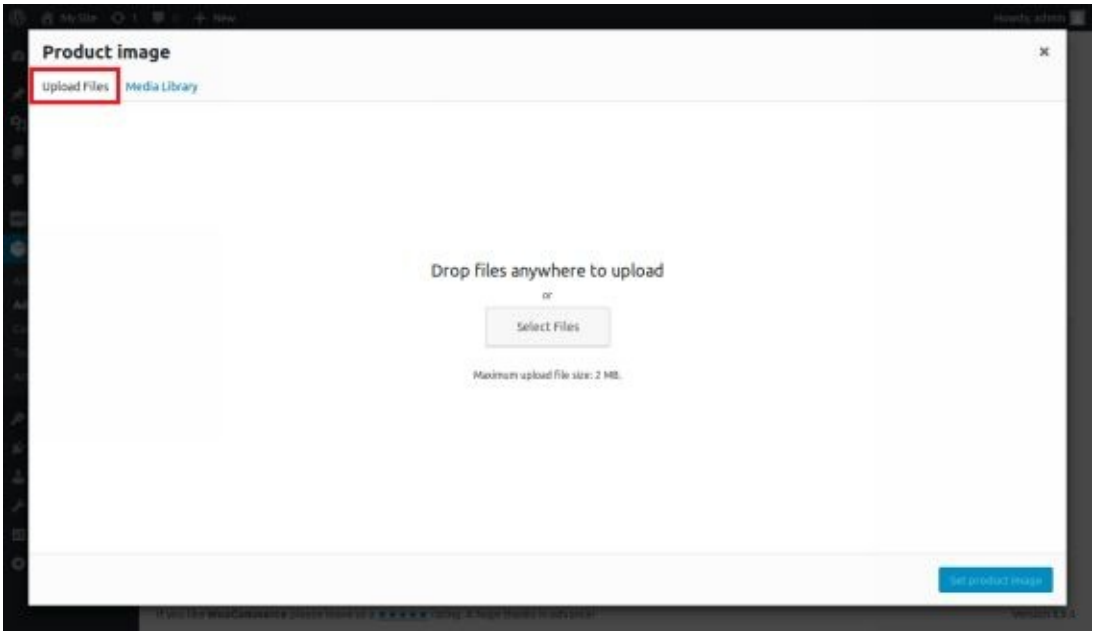
Enter "Plenty storage business bag" in Product short description



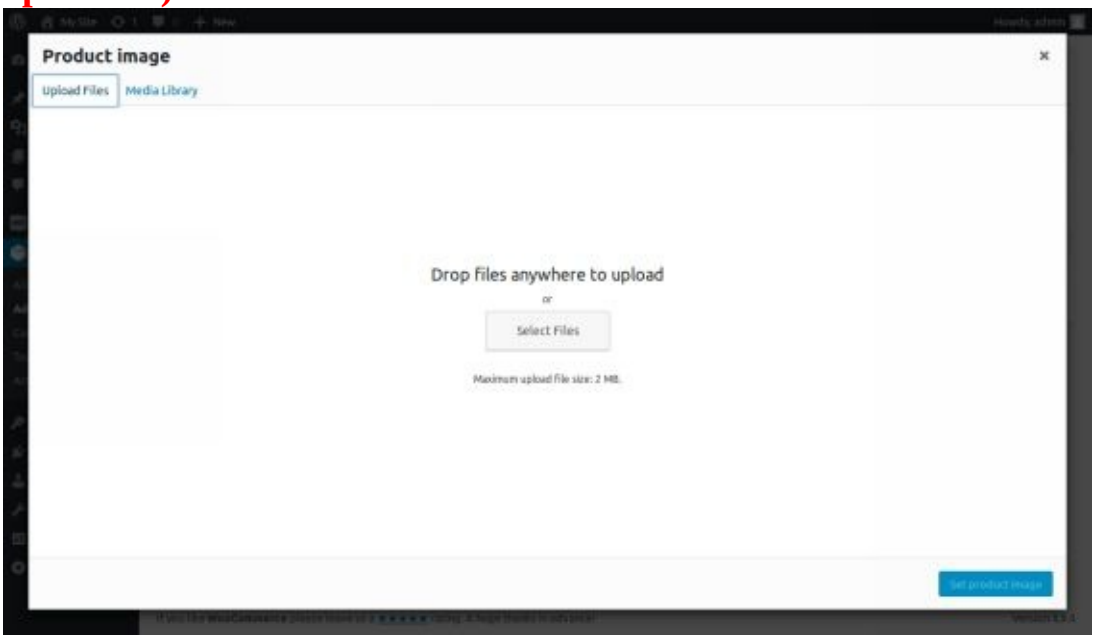
Click "Set product image"



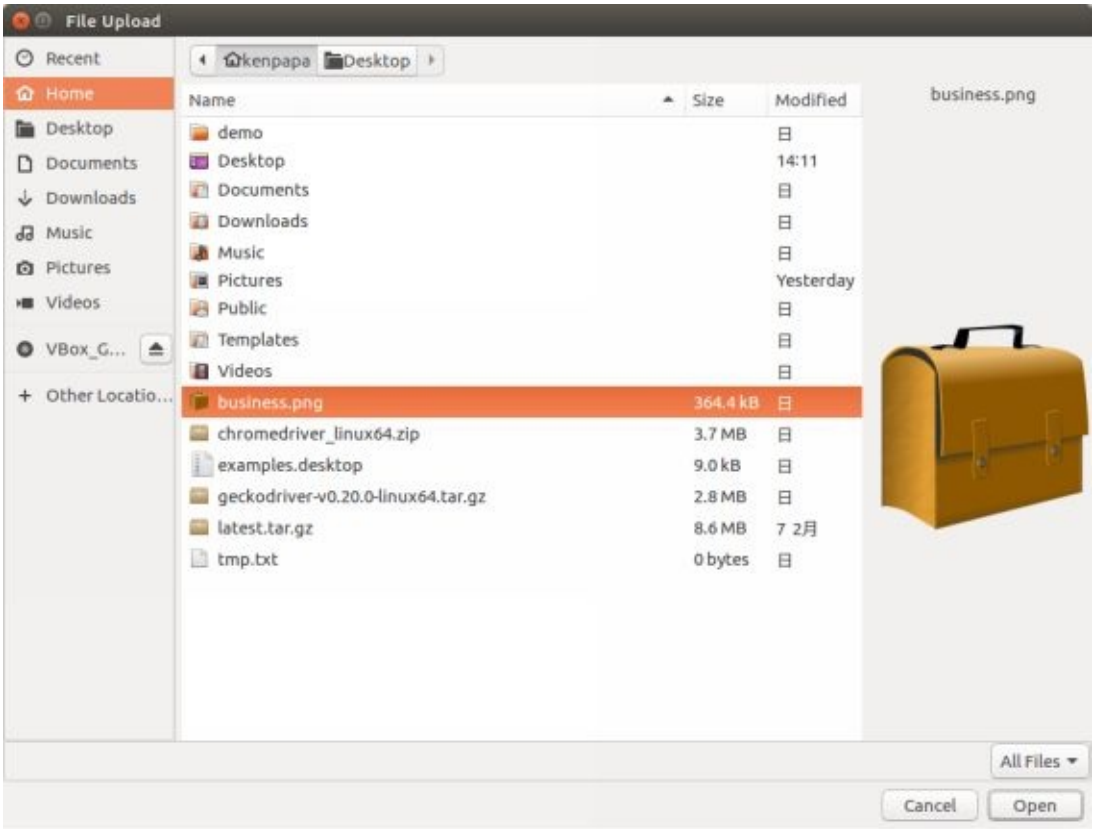
Click "Upload Files"



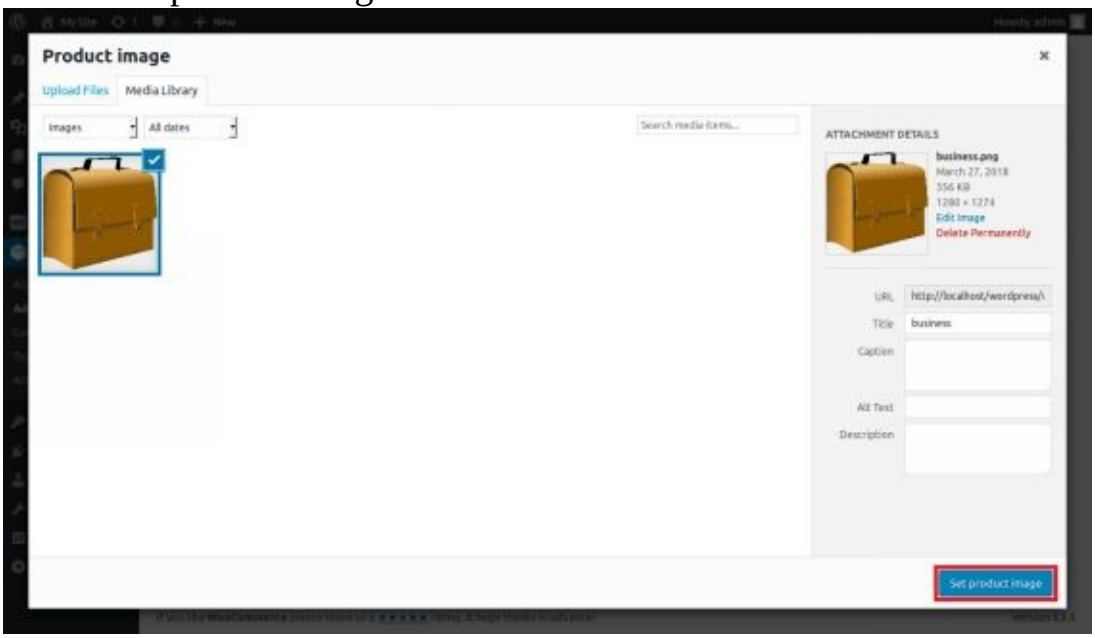
Click "Select Files" **(This button is not clicked when performing automatic operation)**



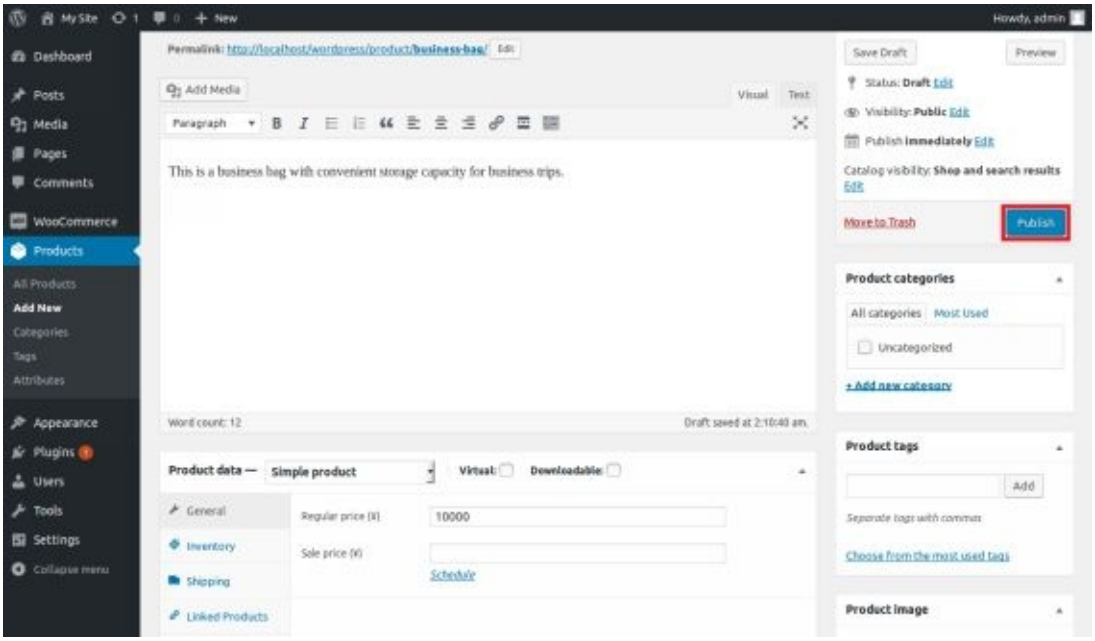
Select "business.png" and click the open button



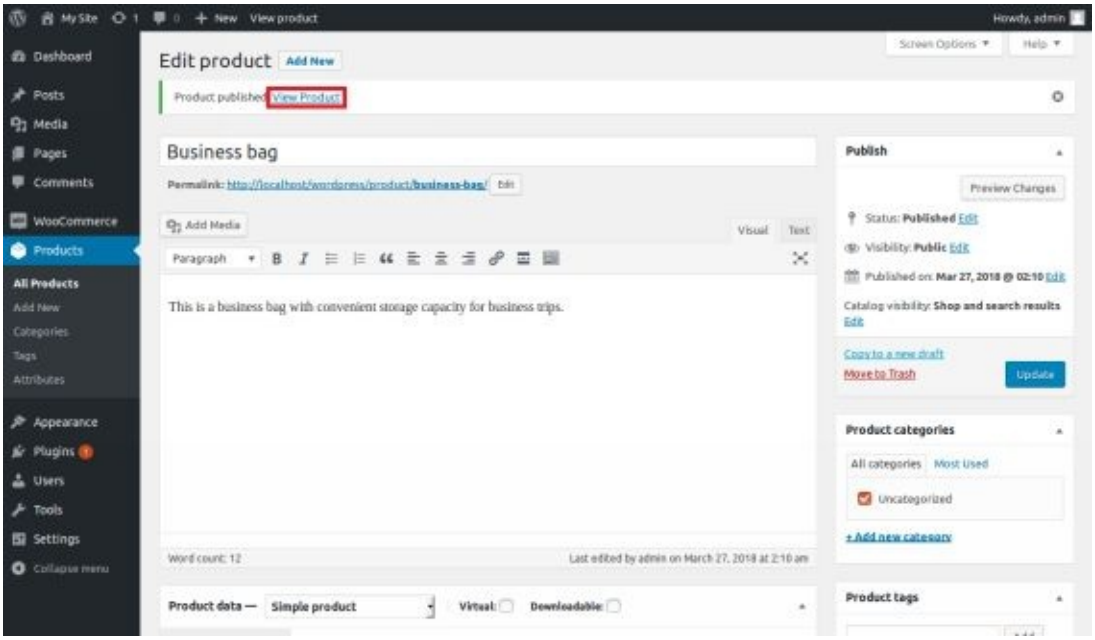
Click "Set product image"



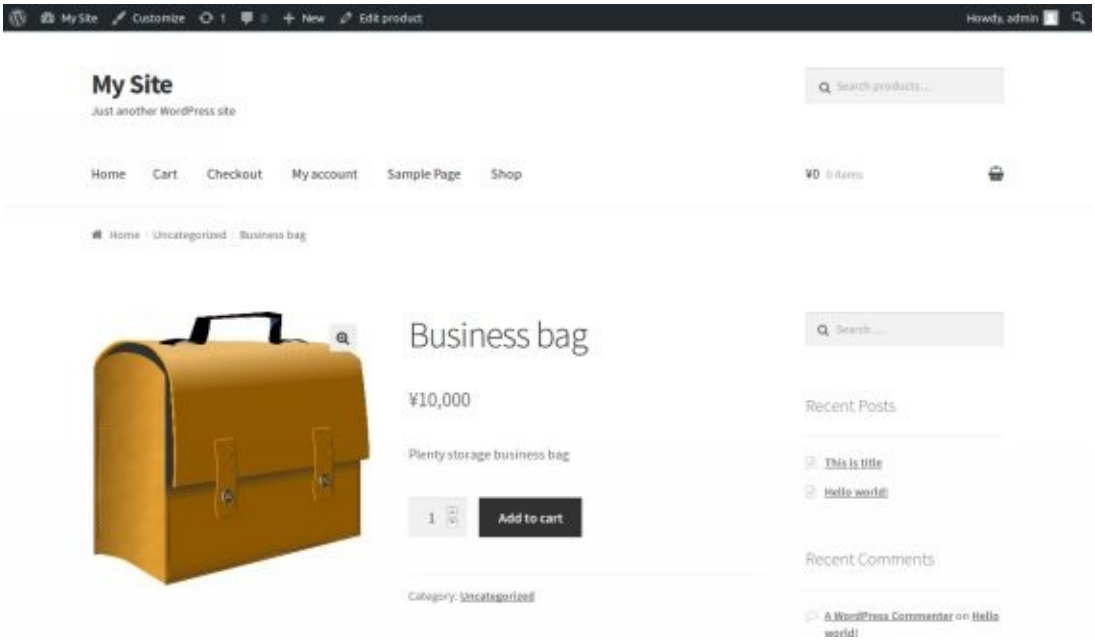
Click "Publish"



Click "View Product"



As a result, the product page is displayed.


My Site
Customize
1
New
Edit product
Howdy, admin
My Site
Just another WordPress site
Search products...
Home
Cart
Checkout
My account
Sample Page
Shop
¥0 0 items
Home / Uncategorized / Business bag
Business bag
¥10,000
Plenty storage business bag
1
Add to cart
Category: Uncategorized
Search ...
Recent Posts
This is title
Hello world!
Recent Comments
A WordPress Commenter on Hello world!

4-3-2 Create automatic operation code for operation target element

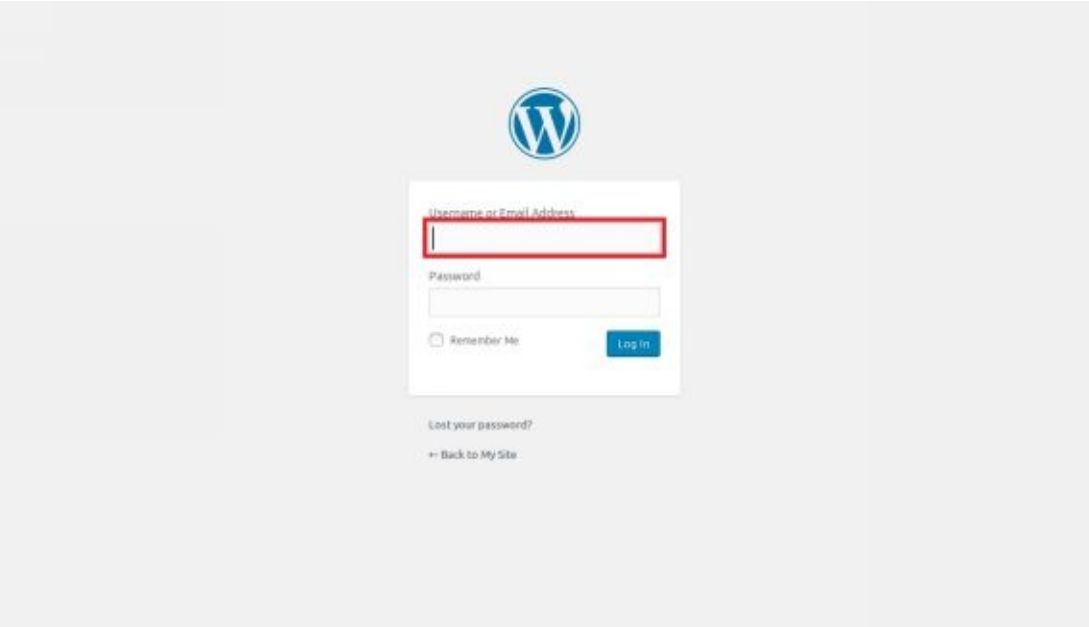
Here, we create automatic operation code from the HTML of the target element for each operation but please pay attention to the following points.
There are several "Add New" links in the page
In order to enter text data we need to set the value to innerHTML of body tag element after changing frame
HTML of "Select Files" button doesn’t have input tag and type ="file" attribute

Open "http://localhost/wordpress/wp-login.php" in the browser

Automatic operation code

```
driver.get("http://localhost/wordpress/wp-login.php")
```

Enter "admin" as Username or Email Address



HTML of operation target element

```
<input name="log" id="user_login" class="input" value="" size="20" type="text">
```

Automatic operation code

```
driver.find_element_by_id("user_login").send_keys("admin")
```

Enter "admin123" as password

HTML of operation target element

```
<input name="pwd" id="user_pass" class="input" value="" size="20" type="password">
```

Automatic operation code

```
driver.find_element_by_id("user_pass").send_keys("admin123")
```

Click "Log In"

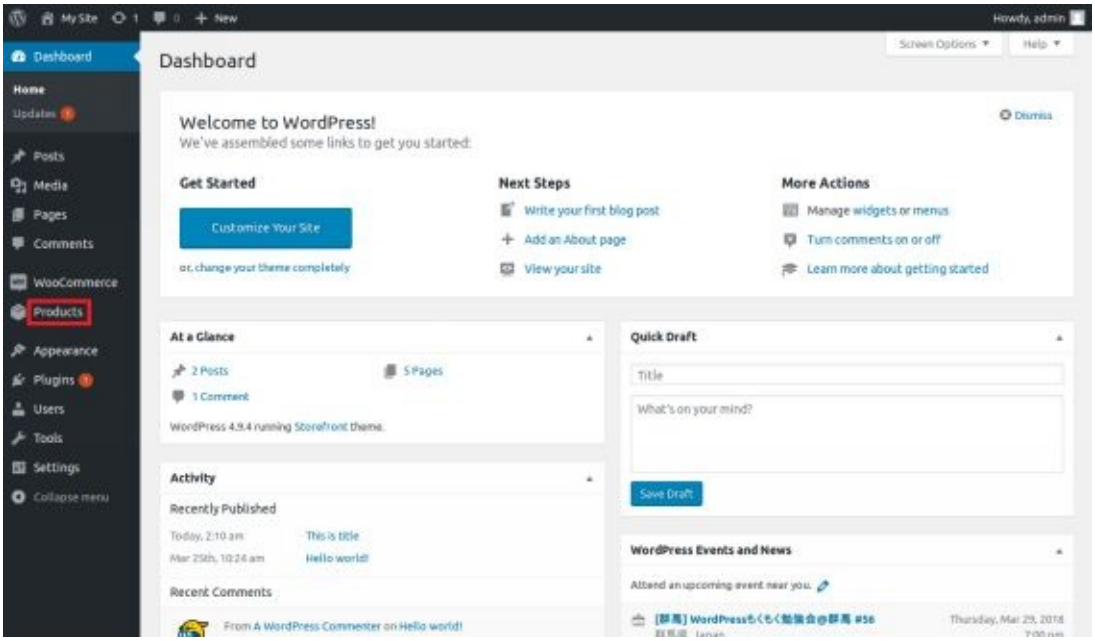
HTML of operation target element

```
<input name="wp-submit" id="wp-submit" class="button button-primary button-large" value="Log In" type="submit">
```

Automatic operation code

```
driver.find_element_by_id("wp-submit").click()
```

Click "Products"



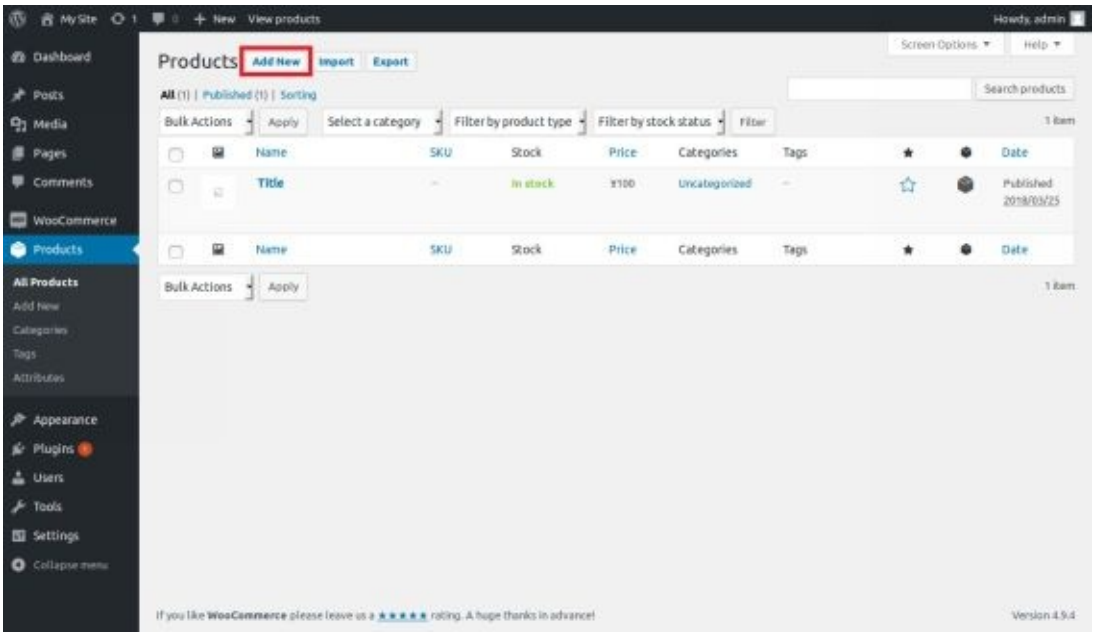
HTML of operation target element

```
<div class="wp-menu-name">Products</div>
```

Automatic operation code

```
driver.find_element_by_xpath("//div[text()='Products']").click()
```

Click "Add New"



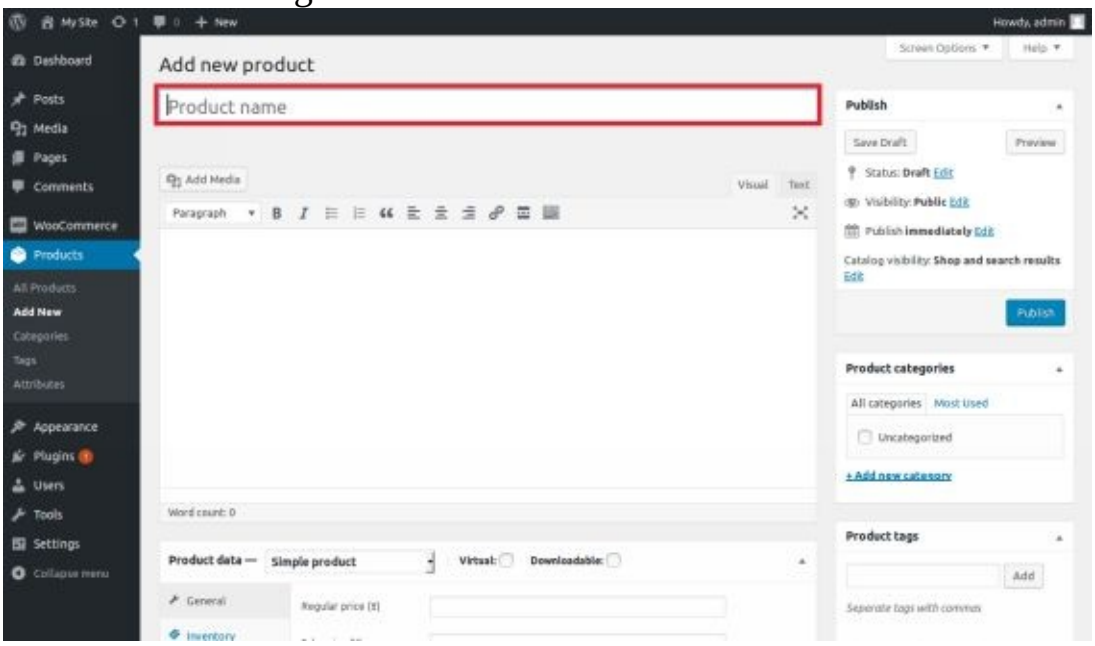
HTML of operation target element

```
<a href="http://localhost/wordpress/wp-admin/post-new.php?post_type=product" class="page-title-action">Add New</a>
```

Automatic operation code

```
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()
```

Enter "Business bag" in Product name



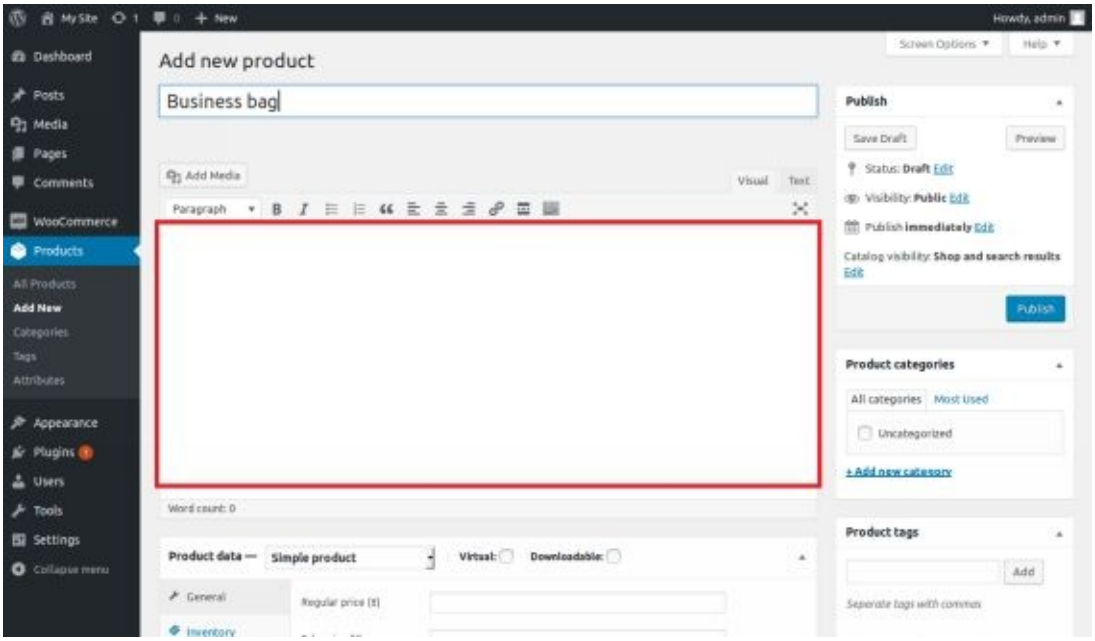
HTML of operation target element

```
<input name="post_title" size="30" value="" id="title" spellcheck="true" autocomplete="off" type="text">
```

Automatic operation code

```
driver.find_element_by_id("title").send_keys("Business bag")
```

Enter "This is a business bag with convenient storage capacity for business trips." in the description



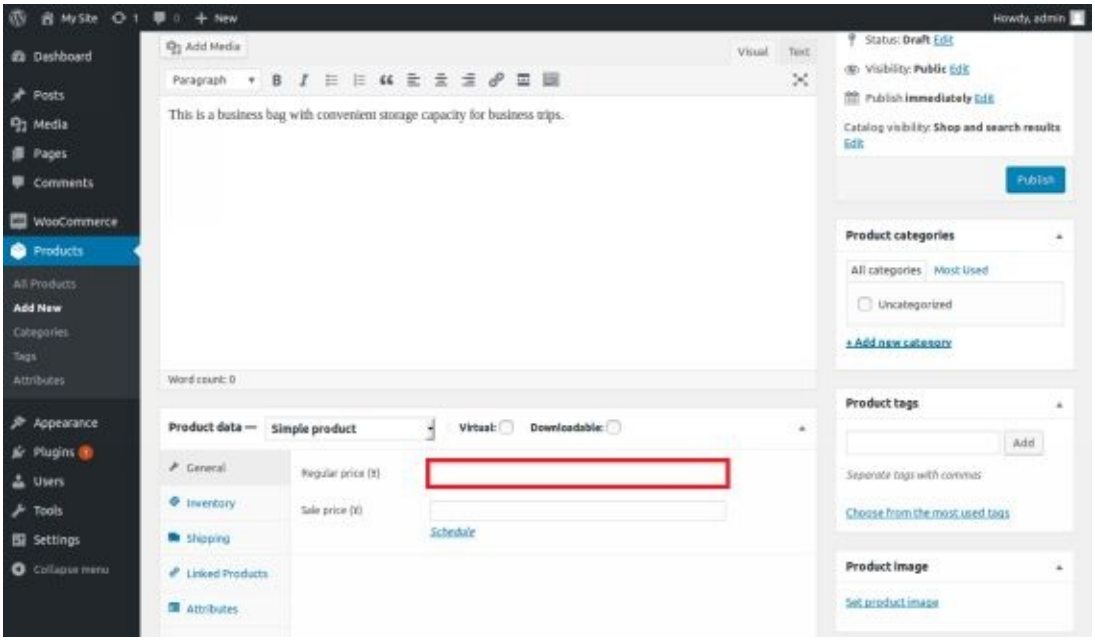
HTML of operation target element

```
<iframe id="content_ifr" ...> ...
<body id="tinymce" class="mce-content-body content post-type-product post-status-auto-draft page-template-default locale-ja mceContentBody wp-editor wp-autoresize html5-captions" data-id="content" style="overflow-y: hidden;" contenteditable="true">
<p><br data-mce-bogus="1"></p></body> ...
</iframe>
```

Automatic operation code

```
driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "This is a business bag with convenient storage capacity for business trips.";') driver.switch_to.default_content()
```

Enter "10000" in Regular price



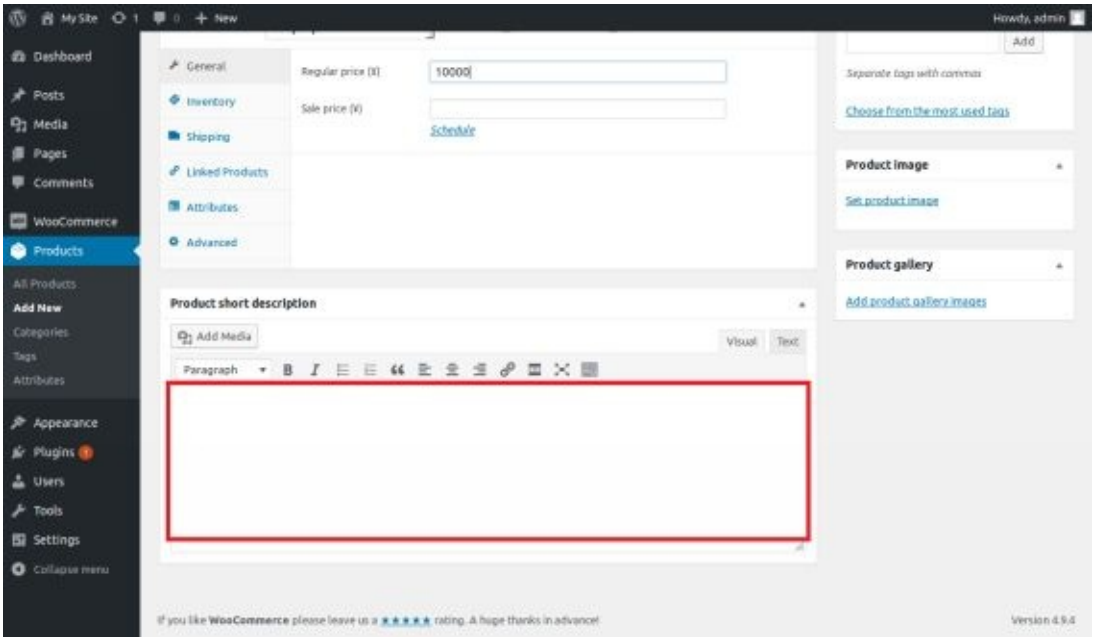
HTML of operation target element

```
<input class="short wc_input_price" style="" name="_regular_price" id="_regular_price" value="" placeholder="" type="text">
```

Automatic operation code

```
driver.find_element_by_id("_regular_price").send_keys("10000")
```

Enter "Plenty storage business bag" in Product short description



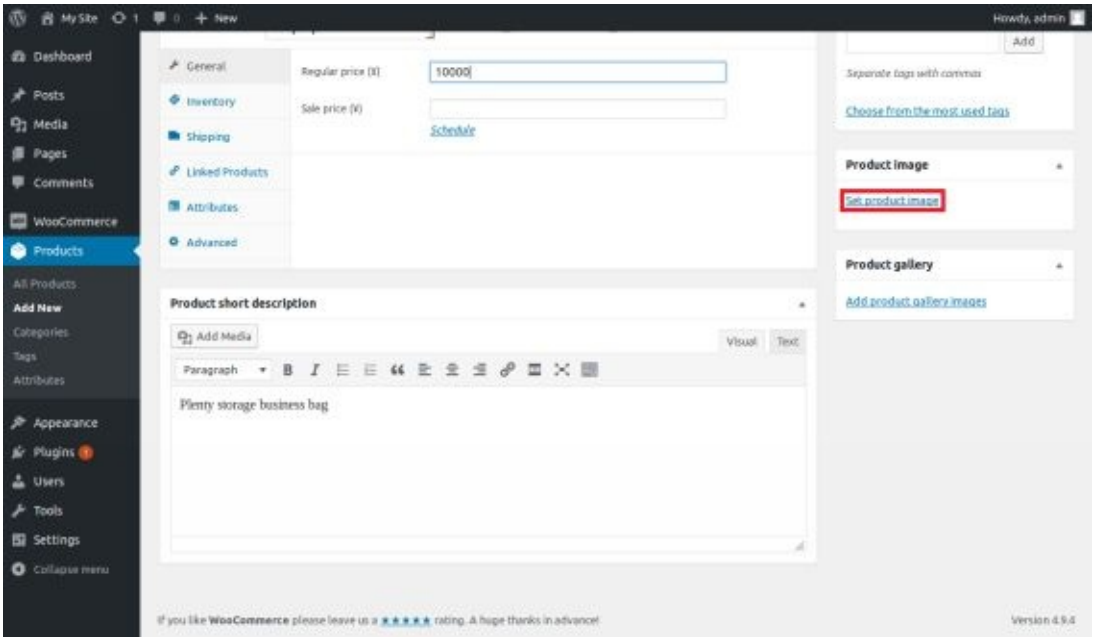
HTML of operation target element

```
<iframe id="excerpt_ifr" ...> ...
<body id="tinymce" class="mce-content-body excerpt post-type-product post-status-auto-draft page-template-default locale-ja mceContentBody wp-editor html5-captions" data-id="excerpt" contenteditable="true"><p><br data-mce-bogus="1"></p></body>
...
</iframe>
```

Automatic operation code

```
driver.switch_to.frame("excerpt_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "Plenty storage business bag";') driver.switch_to.default_content()
```

Click "Set product image"



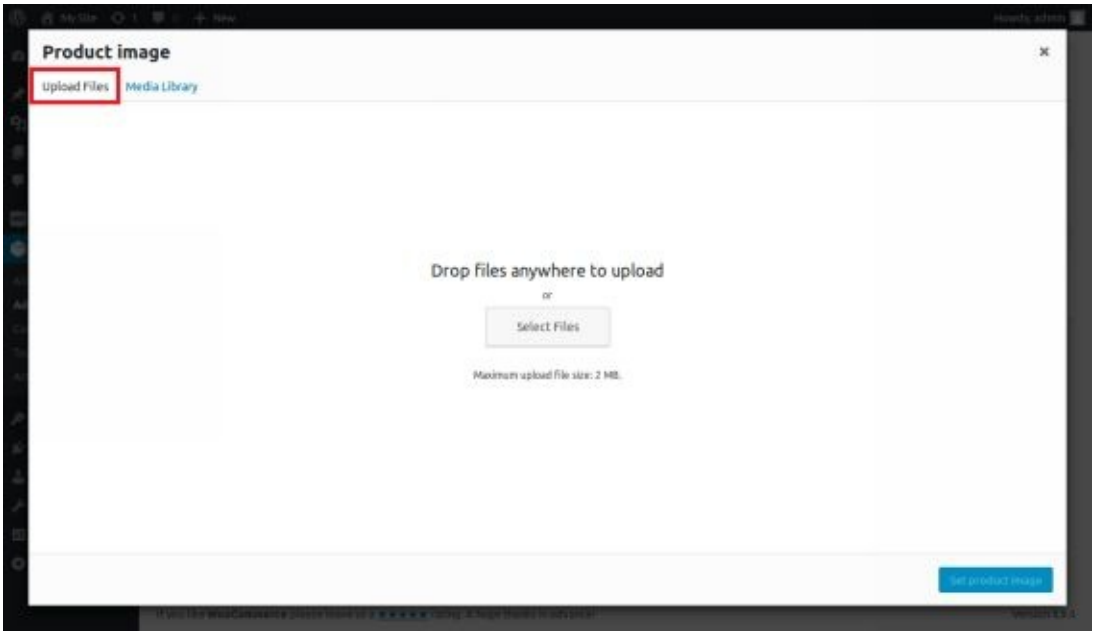
HTML of operation target element

```
<a href="http://localhost/wordpress/wp-admin/media-upload.php?post_id=19&amp;type=image&amp;TB_iframe=1" id="set-post-thumbnail" class="thickbox">Set product image</a>
```

Automatic operation code

```
driver.find_element_by_link_text("Set product image").click()
```

Click "Upload Files"



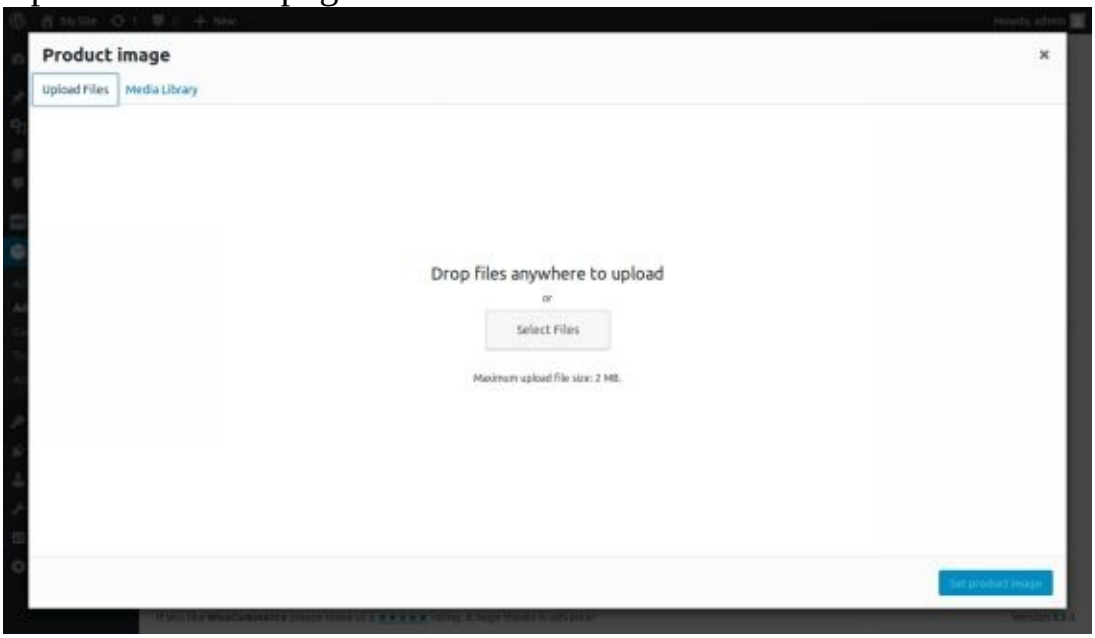
HTML of operation target element

```
<a href="#" class="media-menu-item">Upload Files</a>
```

Automatic operation code

```
driver.find_element_by_link_text("Upload Files").click()
```

Upload "business.png"



HTML of "Select Files" button

```
<button type="button" class="browser button button-hero" style="display: inline-block; position: relative; z-index: 1;" id="__wp-uploader-id-1">Select Files</button>
```

If we click the "Select Files" button here, file selection window opens, but since the window is not a browser, it can't be operated with Selenium WebDriver. So, when we do file uploading automatically, we need to use "sendKeys" instead of "click" method.

As for file uploading, at demo site in chapter 3, the HTML of the element to be operated has input tag and type ="file" attribute as shown below, but the above HTML doesn't have input tag and type ="file" attribute.

■HTML of uploading file at demo site

```
<input id="uploadfile" type="file">
```

So, I searched for a solution on the internet and found the following information.

How to upload files in a WordPress post using Selenium Python?
https://stackoverflow.com/questions/42974673/how-to-upload-files-in-a-wordpress-post-using-selenium-python

Information on the above site

Finally, I've got a solution for WP add media file upload. Actually, I have found that when the file selection window opens upon clicking the select files button, it generates a dynamic input field. Dynamic means, the ID for the input field is unique each time. Fortunately, the first part of the input field's ID remains same. For example, the ID is like html5_1bc7564i41pq5f7m1voce561a0e5. See the HTML below:
<input id="html5_1bc7564i41pq5f7m1voce561a0e5" style="font-size: 999px; opacity: 0; position: absolute; top: 0px; left: 0px; width: 100%; height: 100%;" multiple="" accept="" type="file"/>
So, what I did is, create an XPATH using the first part of the ID (html5_) and simply used send_keys and it worked perfectly. My XPATH is like:
input_file = "//input[starts-with(@id,'html5_')]"

The selenium code is:

driver.find_element_by_xpath(input_file).send_keys(file_path)

Here, file_path is the location of the file which I need to upload.

I missed the input field at the first place because it was not visible and did not realize that it is associated with the file upload. So I record the file upload steps with selenium IDE and found the dynamic input ID.

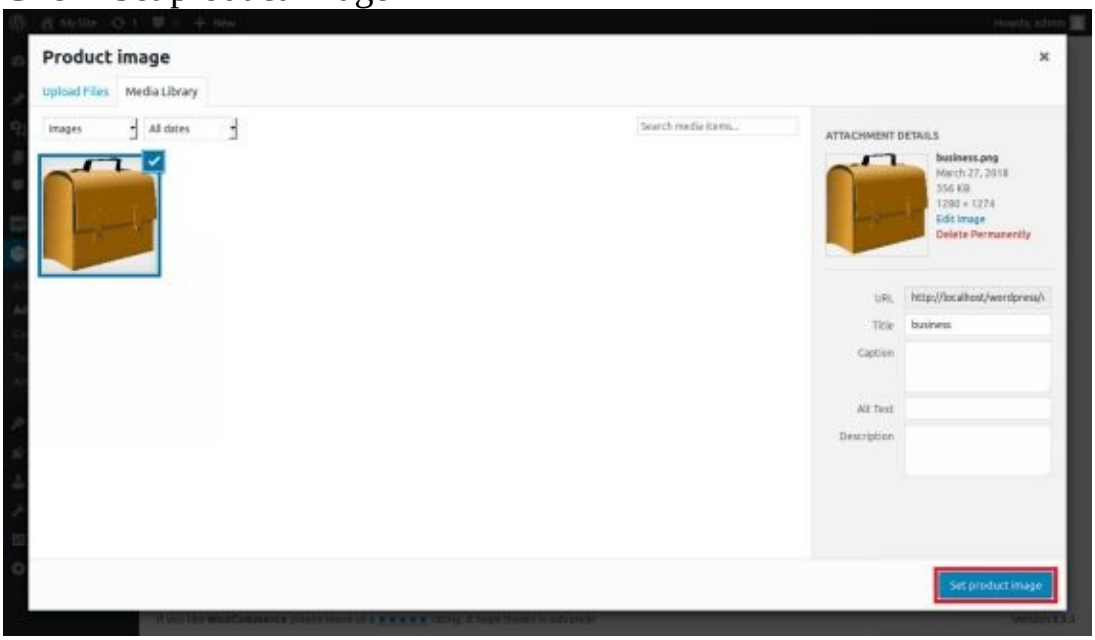
Based on above information, we use the following automatic operation code here.

Automatic operation code

```
time.sleep(2)
driver.find_element_by_xpath("//input[starts-
with(@id,'html5_')]").send_keys("/home/kenpapa/business.png")
time.sleep(2)
```

Since it takes time to upload file, in order to perform automatic operation using Selenium WebDriver, it is necessary to add wait processing. Here sleep method is used to have Selenium WebDriver wait for 2 seconds.

Click "Set product image"



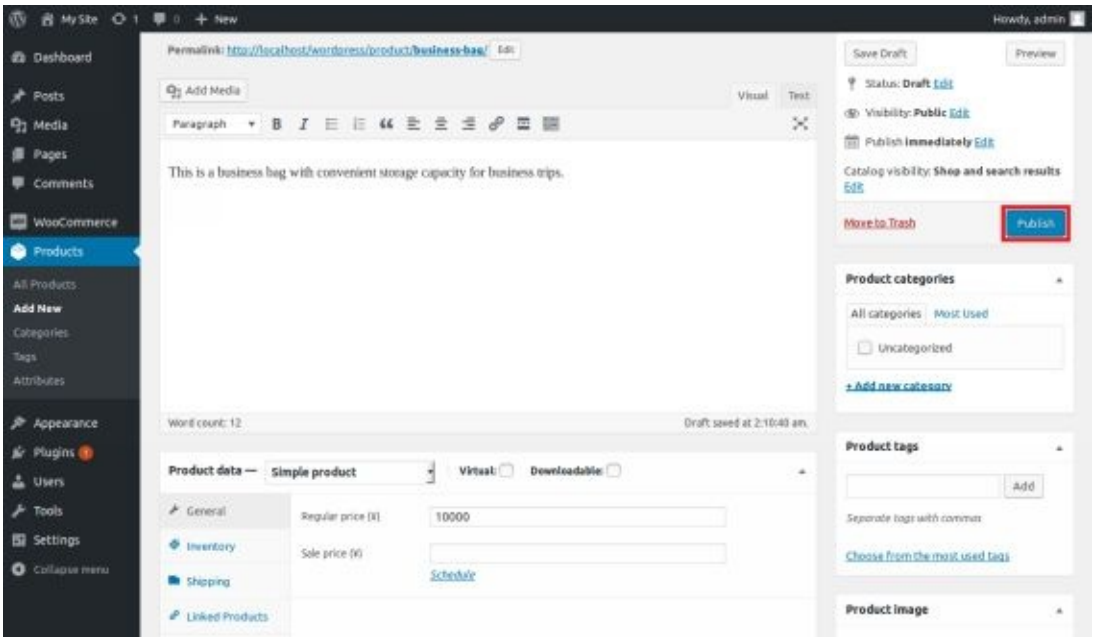
HTML of operation target element

```
<button type="button" class="button media-button button-primary button-large media-button-select">Set product image</button>
```

Automatic operation code

```
driver.find_element_by_xpath("//button[text()='Set product image']").click()
```

Click "Publish"



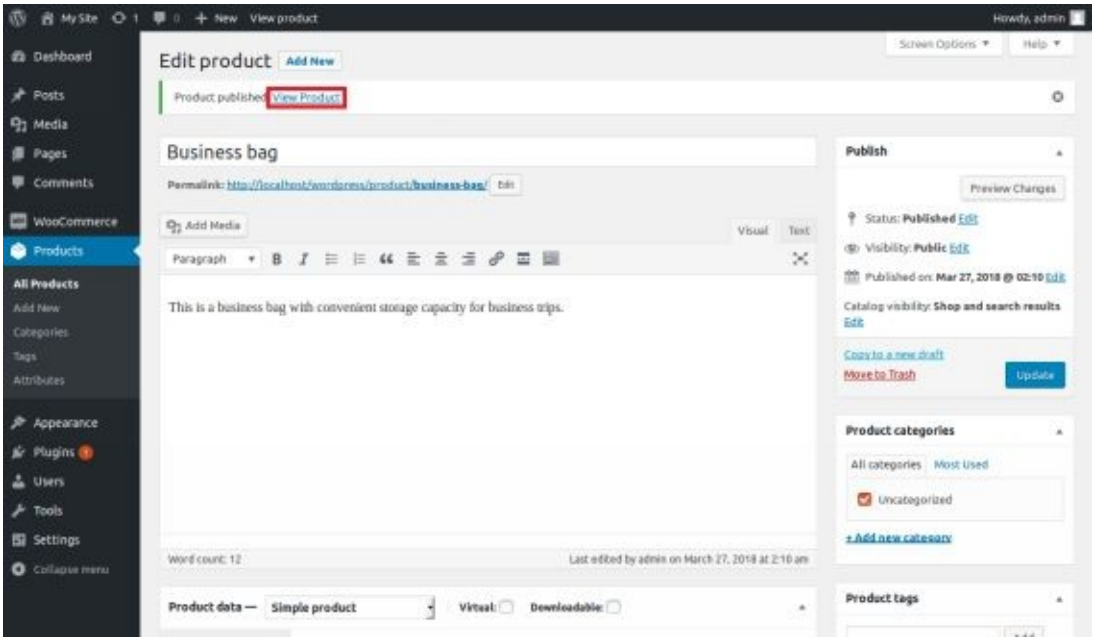
HTML of operation target element

```
<input name="publish" id="publish" class="button button-primary button-large" value="Publish" type="submit">
```

Automatic operation code

```
driver.find_element_by_id("publish").click()
```

Click "View Product"



HTML of operation target element

```
<a  href="http://localhost/wordpress/product/business-bag-2/">View Product</a>
```

Automatic operation code

```
driver.find_element_by_link_text("View Product").click()
```

4-3-3 Complete and execute automatic operation code

Since all the codes of automatic operation of product registration of the EC site have already been created, the following code completes the code of automatic operation of the product registration of the EC site in one file.

add_product_firefox.py

```
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Products']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()

driver.find_element_by_id("title").send_keys("Business bag")

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "This is a business bag with convenient storage capacity for business trips.";') driver.switch_to.default_content()

driver.find_element_by_id("_regular_price").send_keys("10000")

driver.switch_to.frame("excerpt_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "Plenty storage business bag";') driver.switch_to.default_content()

driver.find_element_by_link_text("Set product image").click()
driver.find_element_by_link_text("Upload Files").click()
```

```
time.sleep(2)
driver.find_element_by_xpath("//input[starts-
with(@id,'html5_')]").send_keys("/home/kenpapa/business.png")
time.sleep(2)

driver.find_element_by_xpath("//button[text()='Set product
image']").click()

driver.find_element_by_id("publish").click()

driver.find_element_by_link_text("View Product").click()
```

add_product_chrome.py (This code fails in the middle)

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")
driver.set_window_size(1280, 800)

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Products']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-
title-action']").click()

driver.find_element_by_id("title").send_keys("Business bag")

driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML
= "This is a business bag with convenient storage capacity for business
trips.";') driver.switch_to.default_content()

driver.find_element_by_id("_regular_price").send_keys("10000")
```

```
driver.switch_to.frame("excerpt_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML
= "Plenty storage business bag";') driver.switch_to.default_content()

driver.find_element_by_link_text("Set product image").click()
driver.find_element_by_link_text("Upload Files").click()

time.sleep(2)
driver.find_element_by_xpath("//input[starts-
with(@id,'html5_')]").send_keys("/home/kenpapa/business.png")
time.sleep(2)

driver.find_element_by_xpath("//button[text()='Set product
image']").click()

driver.find_element_by_id("publish").click()

driver.find_element_by_link_text("View Product").click()
```

The above code works fine for Firefox but fails for Chrome.

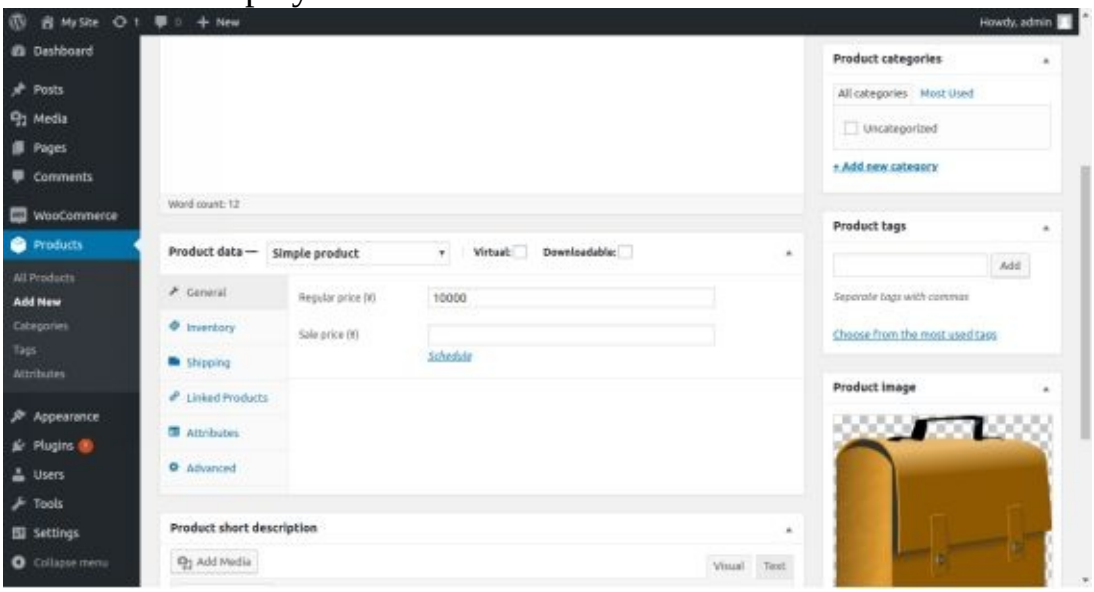
An error occurs in the following part of the code for Chrome.

```
driver.find_element_by_id("publish").click()
```

When executing the above code, error occurs as shown in the figure below.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ python add_product_chrome.py
Traceback (most recent call last):
  File "add_product_chrome.py", line 37, in <module>
    driver.find_element_by_id("publish").click()
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webdriv
er.py", line 312, in execute
    self.error_handler.check_response(response)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/errorha
ndler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.WebDriverException: Message: unknown error: Element <
input type="submit" name="publish" id="publish" class="button button-primary but
ton-large" value="Publish"> is not clickable at point (1188, 15). Other element
would receive the click: <span class="display-name">...</span>
  (Session info: chrome=65.0.3325.181)
  (Driver info: chromedriver=2.36.540471 (9c759b81a907e70363c6312294d30b6ccccc27
52),platform=Linux 4.13.0-37-generic x86_64)
```

The reason for this error is that Selenium tried to click the "Publish" button which is not displayed on the screen as below.

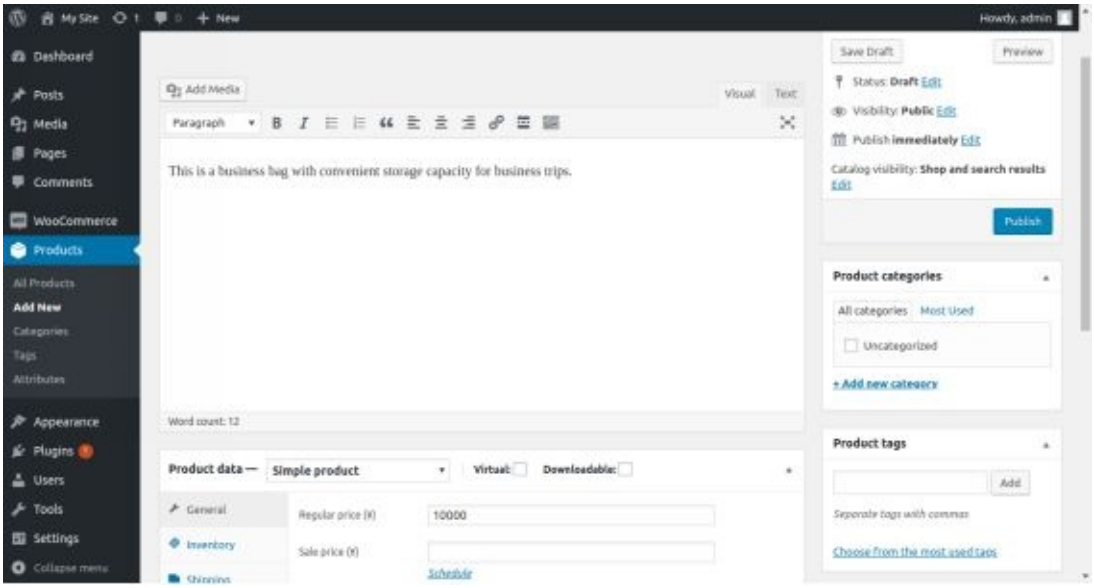


In Firefox, the "Publish" button is clicked even if it is not displayed on the screen, but it is not clicked in Chrome. For that reason, we add scrolling process so that the "Publish" button is displayed in Chrome.

Code of scrolling process

```
driver.execute_script('window.scrollTo(0,100);')
```

As a result, the "Publish" button is displayed so Selenium can click the button in Chrome.



The following code is the modified code. Scrolling process has been added to prevent error from occurring.

add_product_chrome.rb (Modified version)

```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/wp-login.php")
driver.set_window_size(1280, 800)

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

driver.find_element_by_xpath("//div[text()='Products']").click()
driver.find_element_by_xpath("//a[text()='Add New' and @class='page-title-action']").click()

driver.find_element_by_id("title").send_keys("Business bag")
```

```
driver.switch_to.frame("content_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "This is a business bag with convenient storage capacity for business trips.";') driver.switch_to.default_content()

driver.find_element_by_id("_regular_price").send_keys("10000")

driver.switch_to.frame("excerpt_ifr")
driver.execute_script('document.getElementById("tinymce").innerHTML = "Plenty storage business bag";') driver.switch_to.default_content()

driver.find_element_by_link_text("Set product image").click()
driver.find_element_by_link_text("Upload Files").click()

time.sleep(2)
driver.find_element_by_xpath("//input[starts-with(@id,'html5_')]").send_keys("/home/kenpapa/business.png")
time.sleep(2)

driver.find_element_by_xpath("//button[text()='Set product image']").click()

driver.execute_script('window.scrollTo(0,100);')
driver.find_element_by_id("publish").click()

driver.find_element_by_link_text("View Product").click()
```

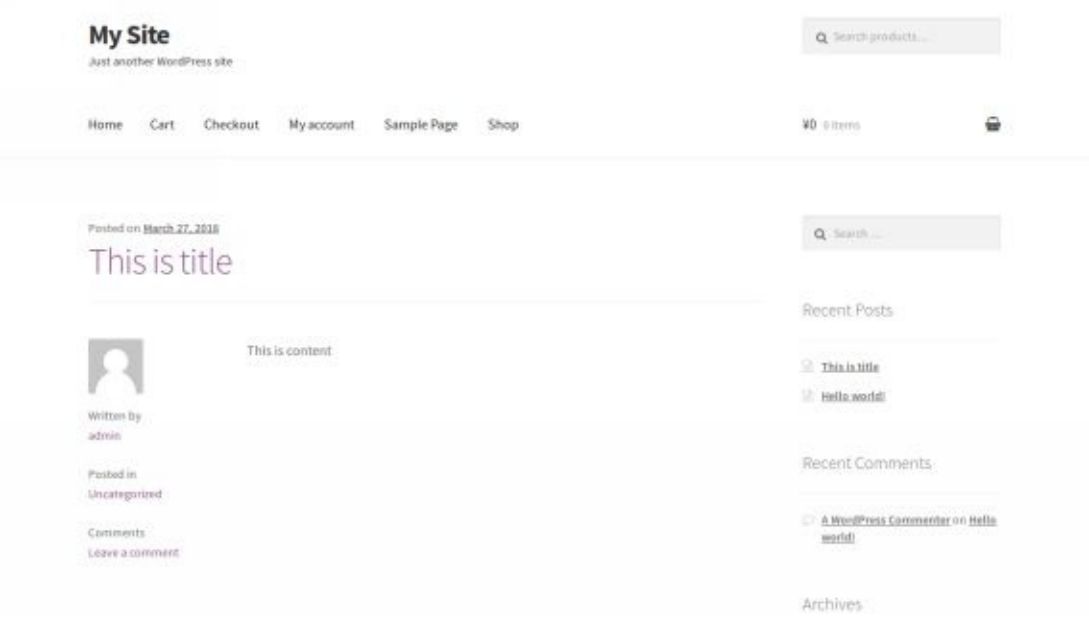
4-4 Purchase product at EC site

Here we automatically perform the following operations.
Open shop page
Select product
Enter purchaser information
Order

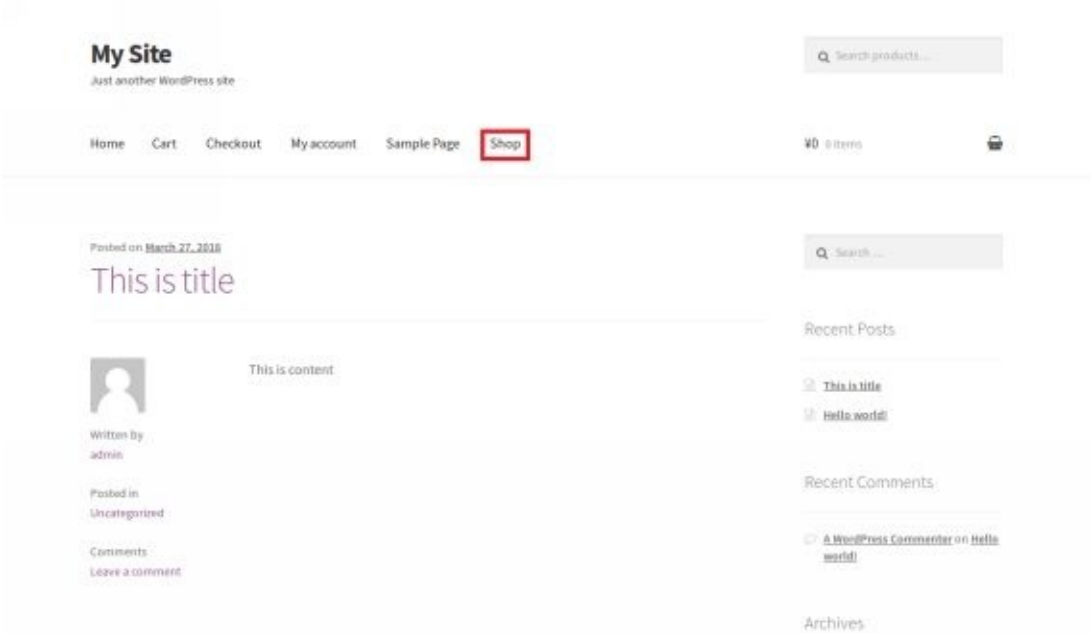
4-4-1 Flow of product purchase

The flow of product purchase at the EC site is as follows.

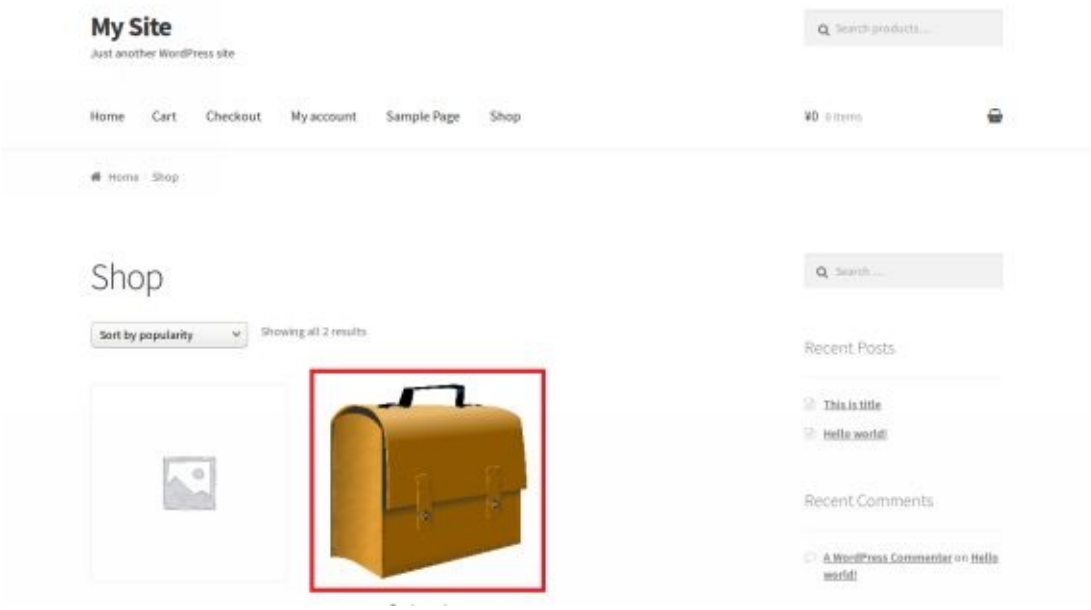
Open "http://localhost/wordpress/" in the browser



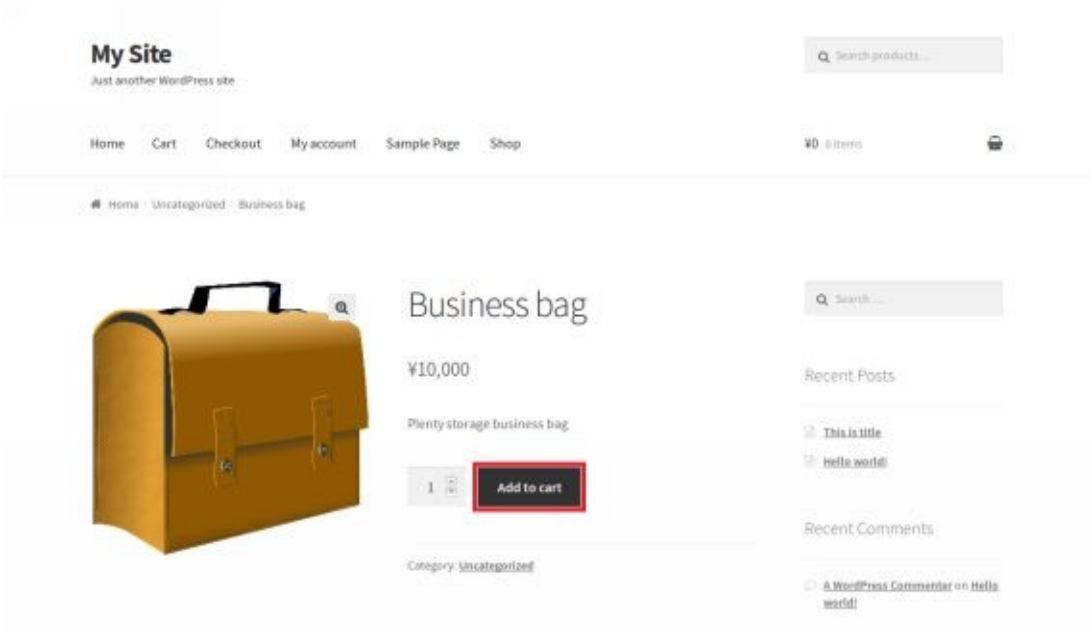
Click "Shop"



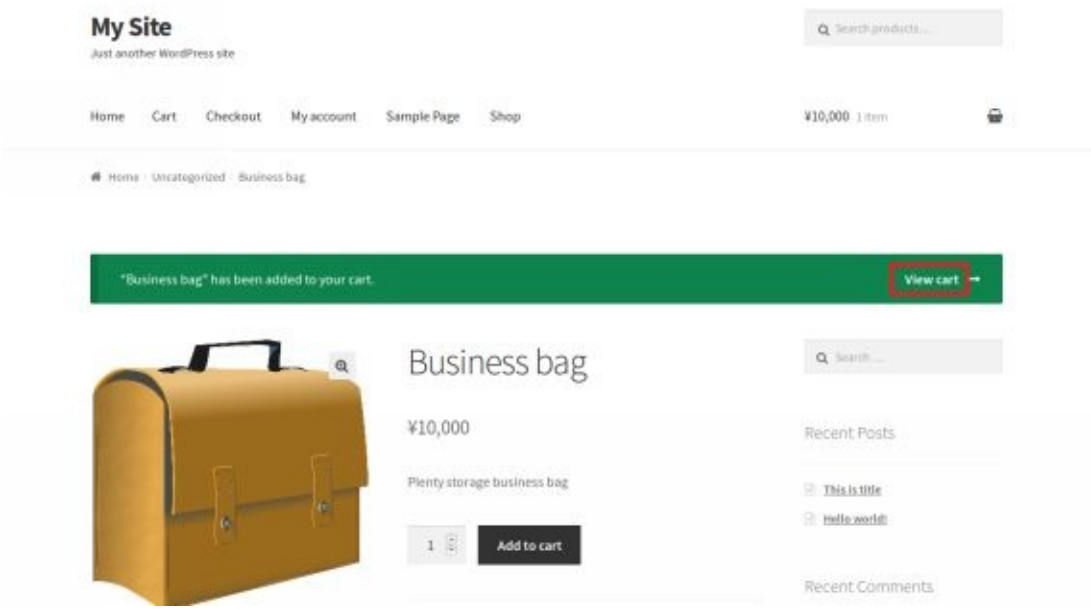
Click image



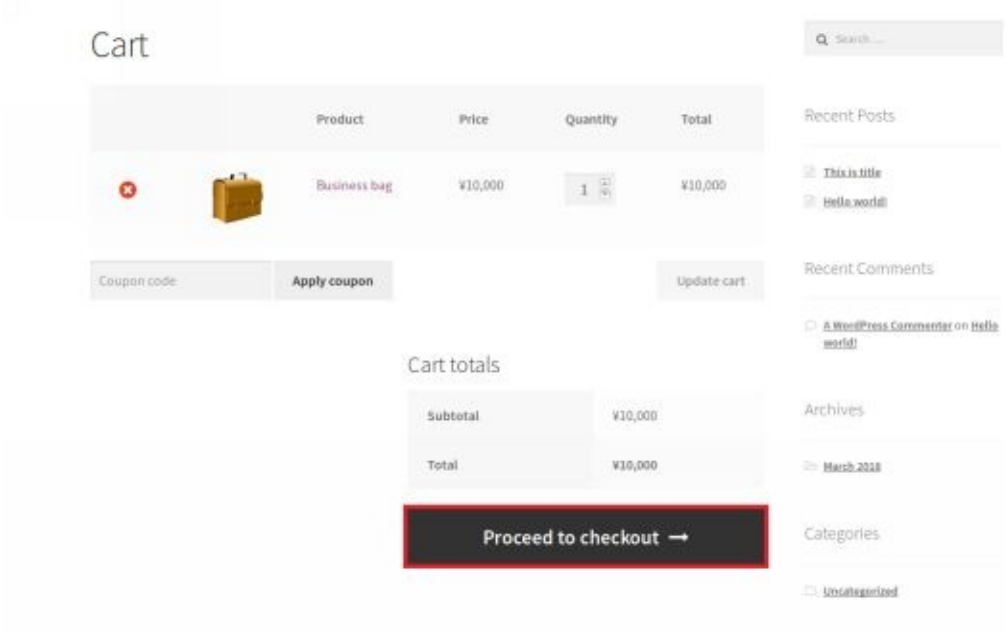
Click "Add to cart"



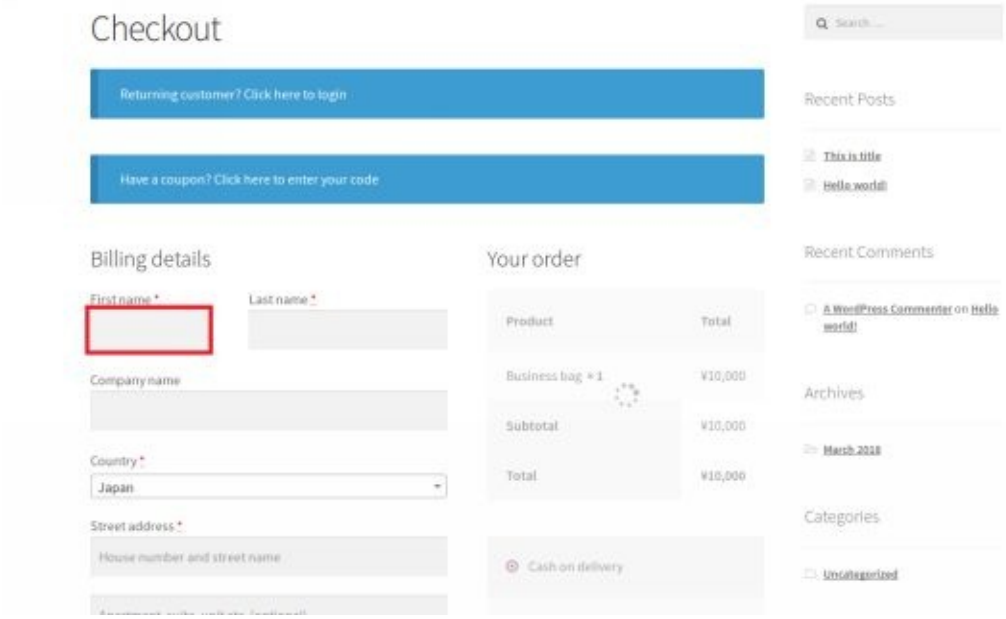
Click "View cart"



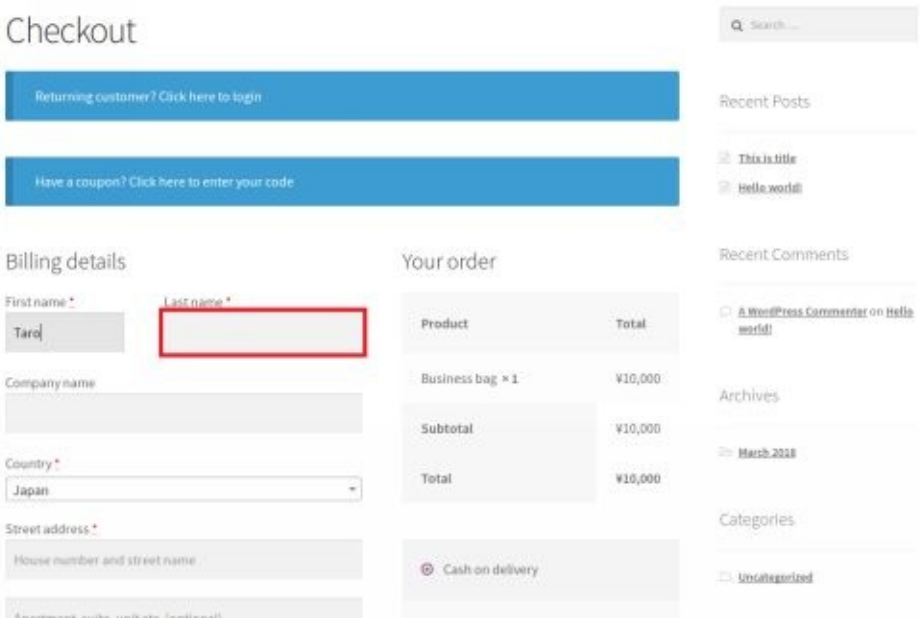
Click "Proceed to checkout"



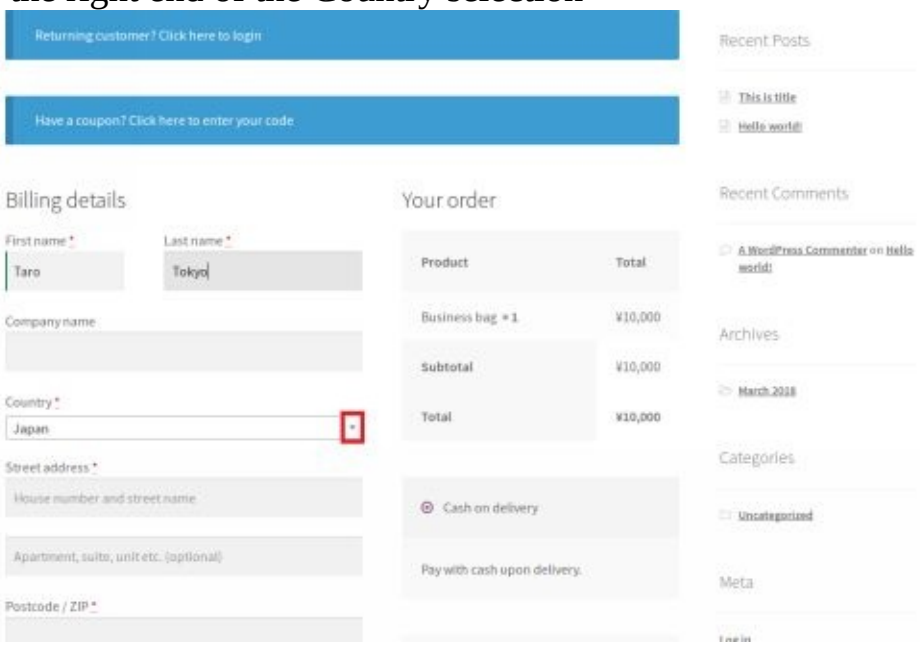
Enter "Taro" in First name



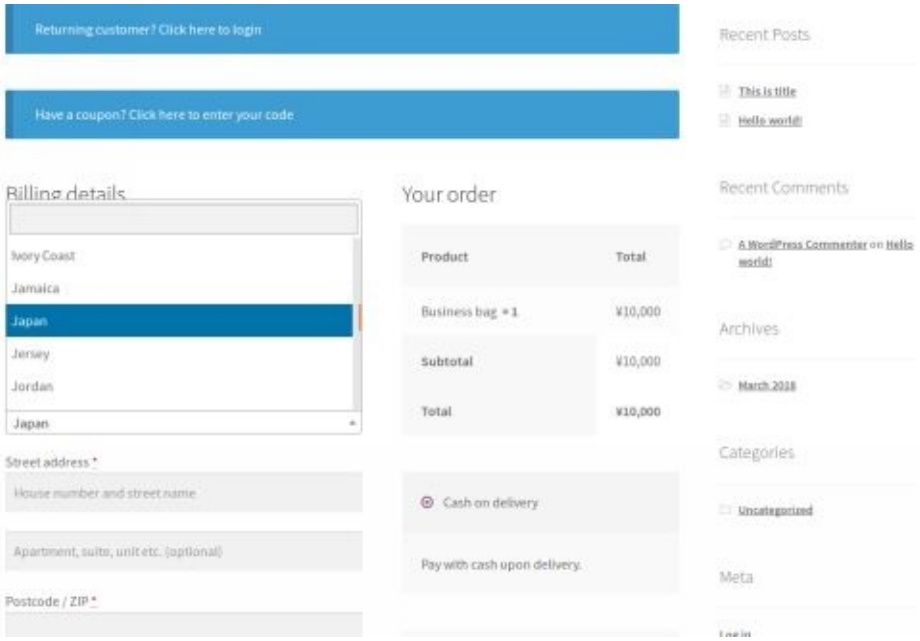
Enter "Tokyo" in Last name



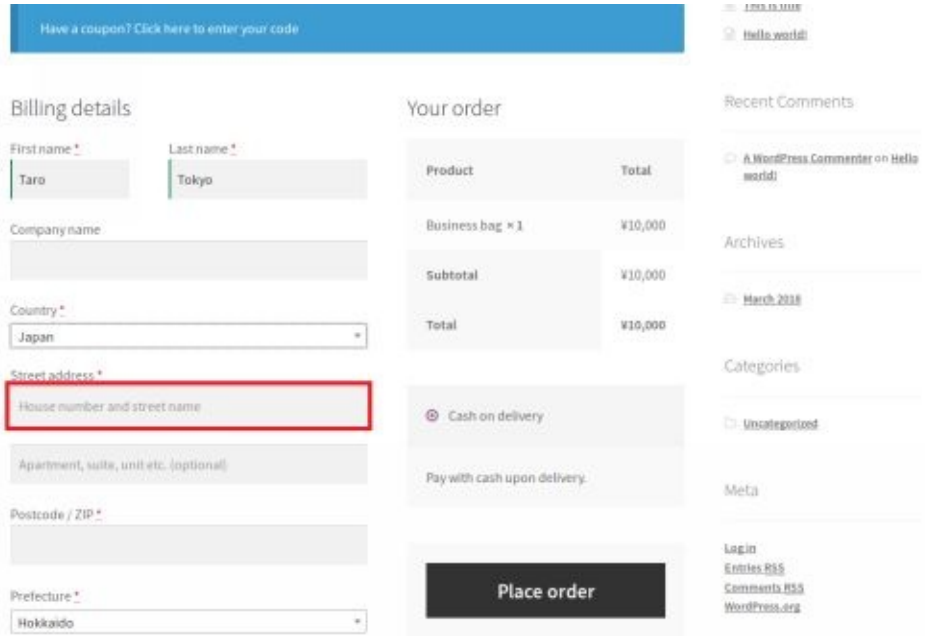
Click the right end of the Country selection



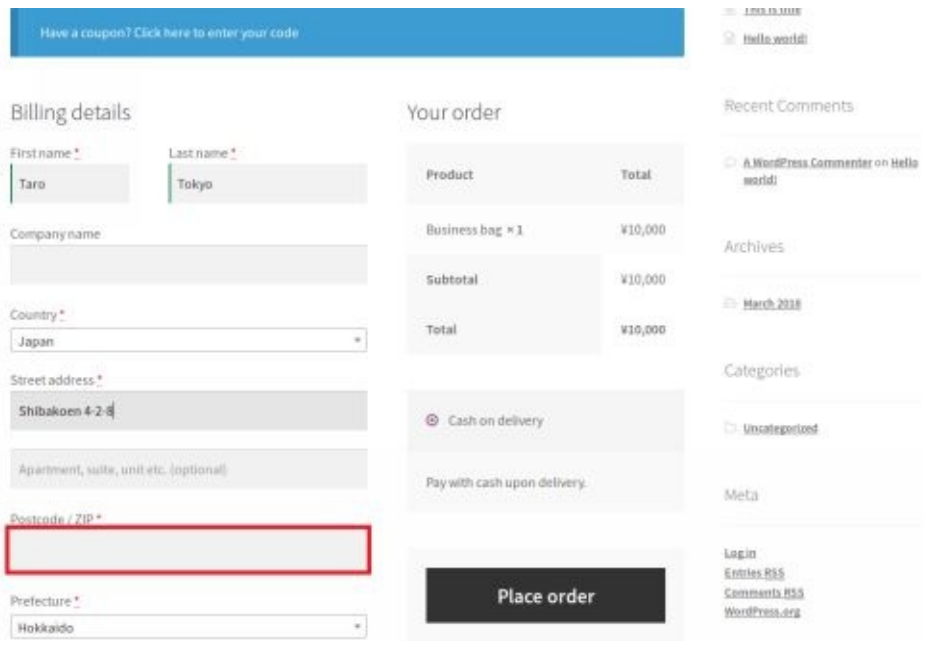
Select "Japan"



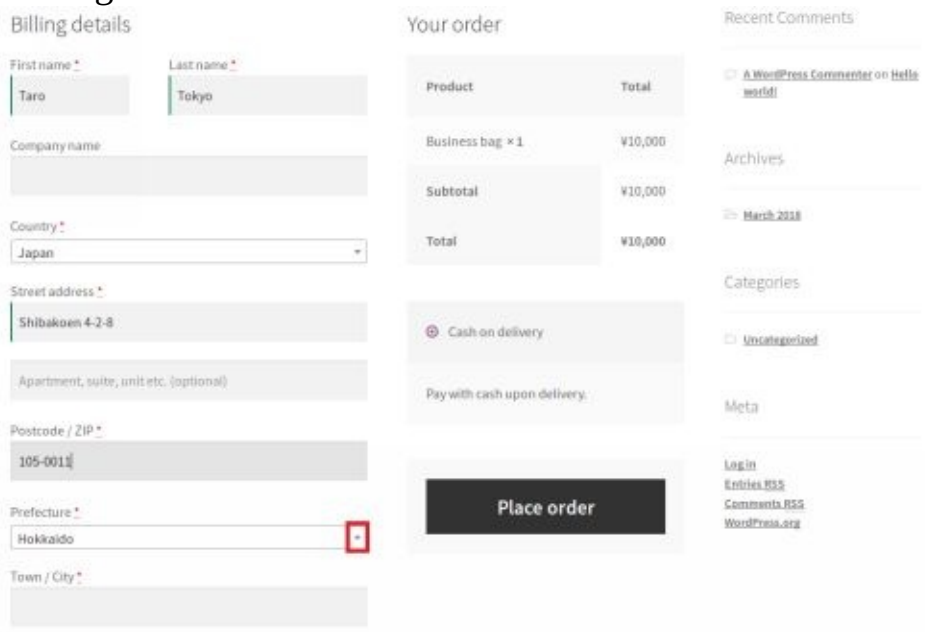
Enter "Shibakoen 4-2-8" in Street address



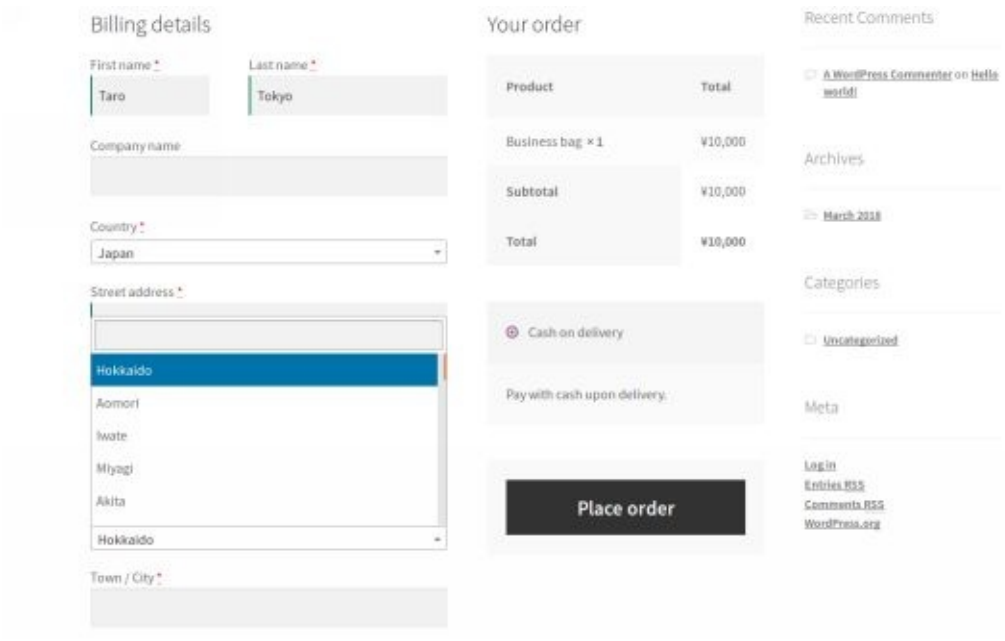
Enter "105-0011" in Postcode/ZIP



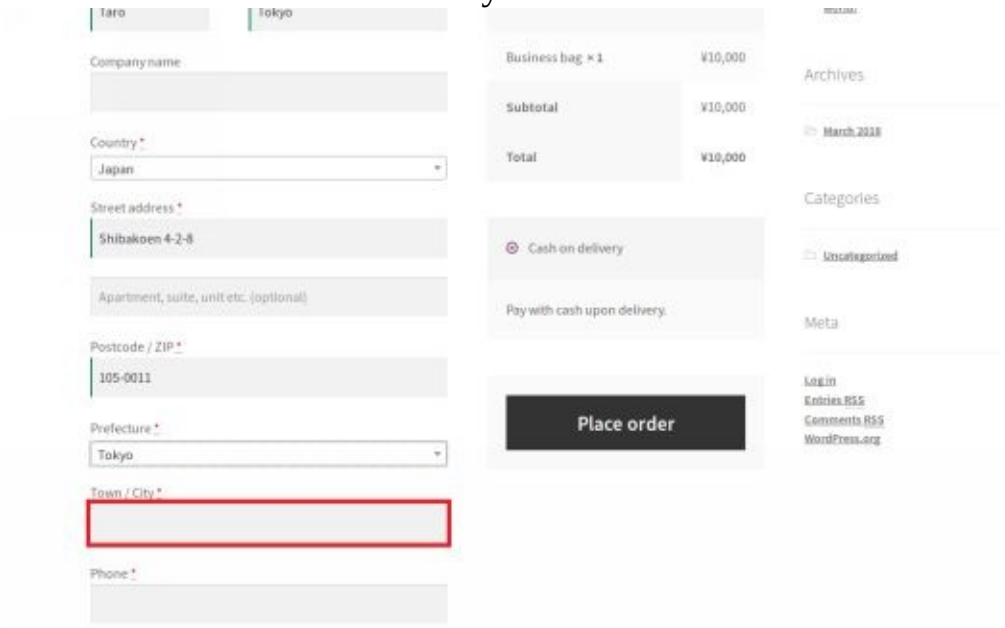
Click the right end of the Prefecture selection



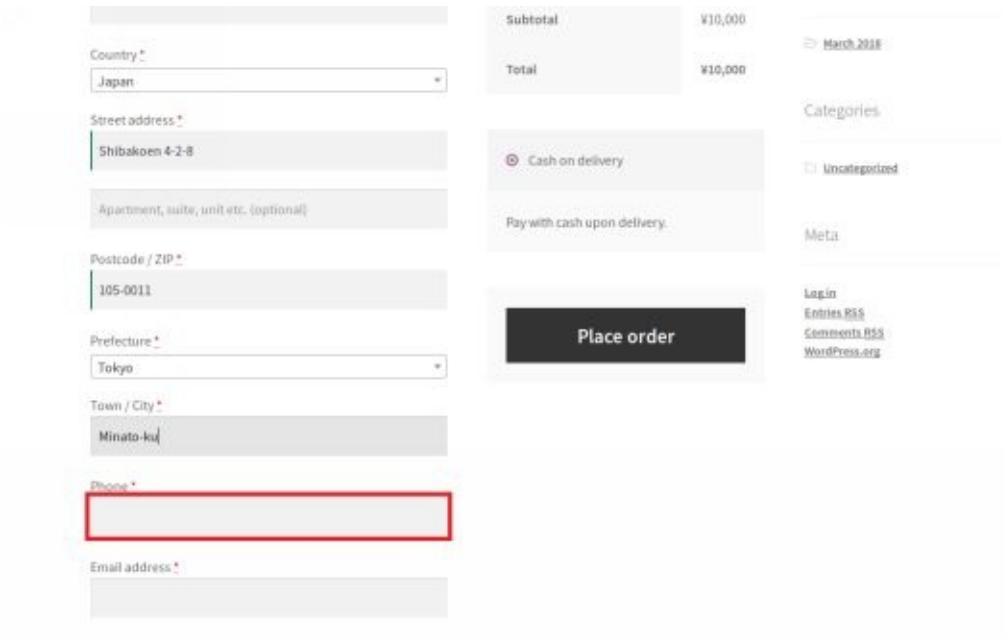
Select "Tokyo"



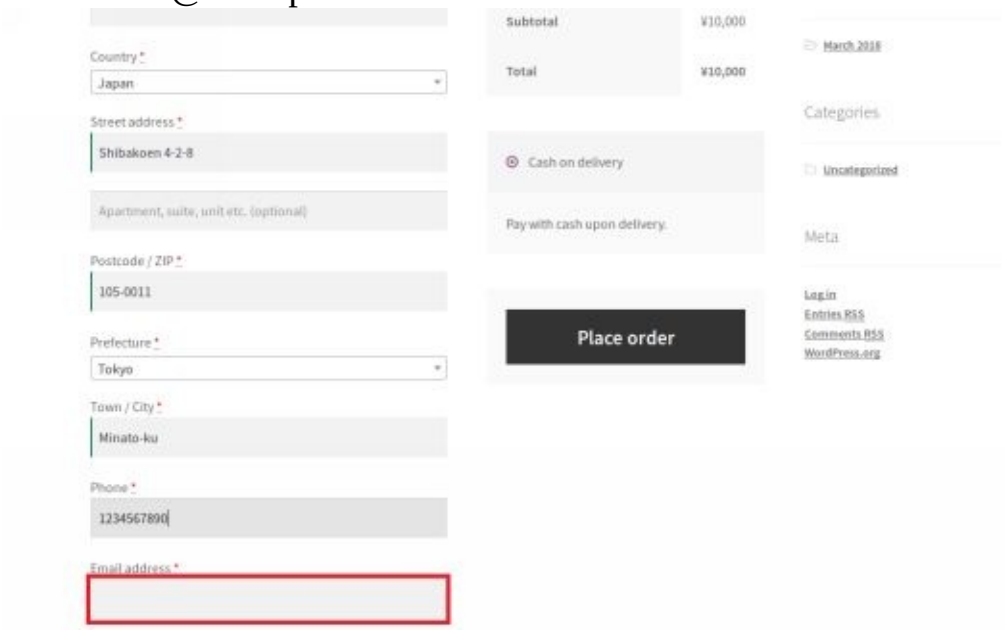
Enter "Minato-ku" in Town/City



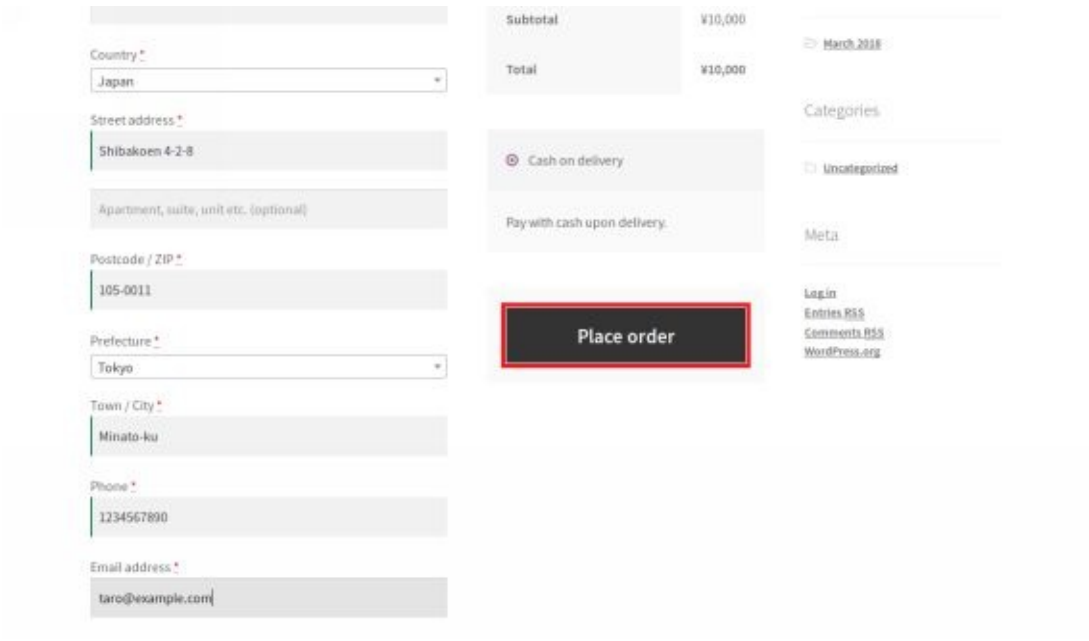
Enter "1234567890" in Phone



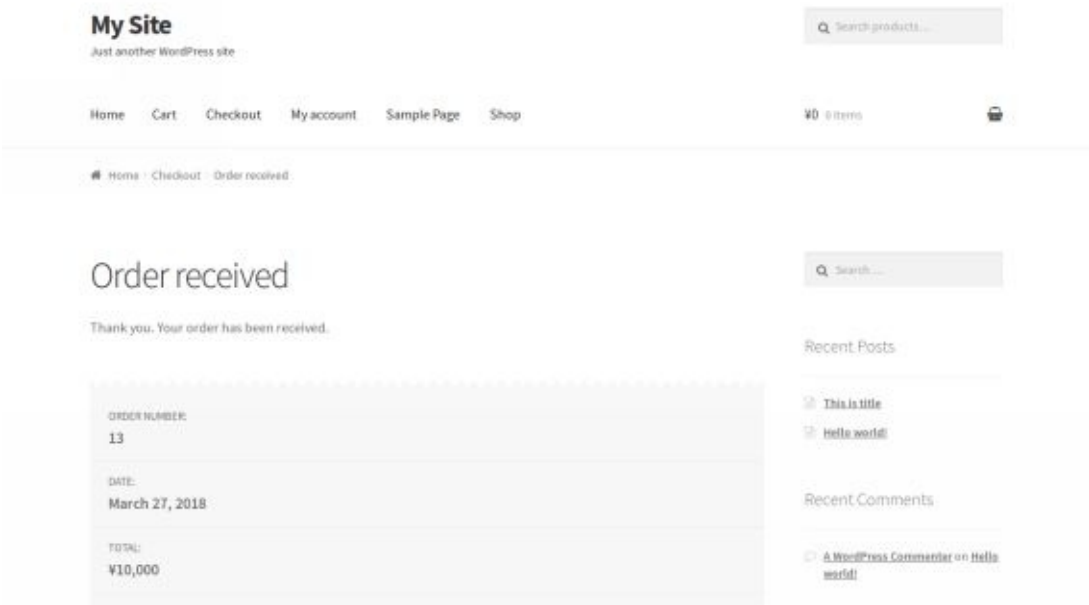
Enter "taro@example.com" in Email address



Click "Place order"



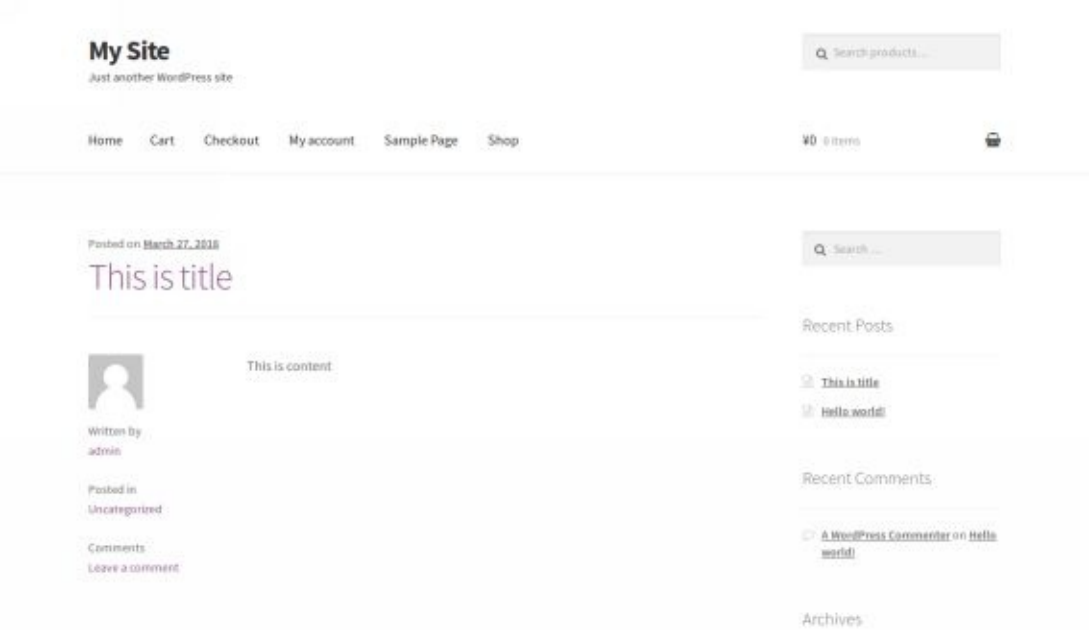
As a result, the order reception screen is displayed.



4-4-2 Create automatic operation code for operation target element

Here, we create automatic operation code from the HTML of the target element for each operation but please pay attention to the following points.
There are several "View cart" links in the page
andhave the same HTML

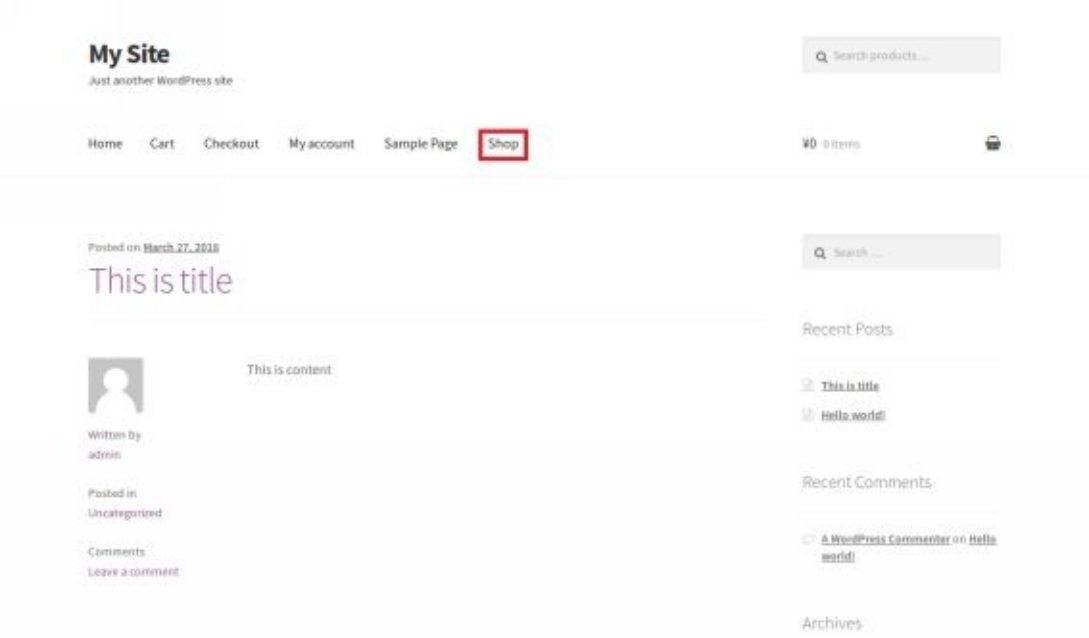
Open "http://localhost/wordpress/" in the browser



Automatic operation code

```
driver.get("http://localhost/wordpress/")
```

Click "Shop"



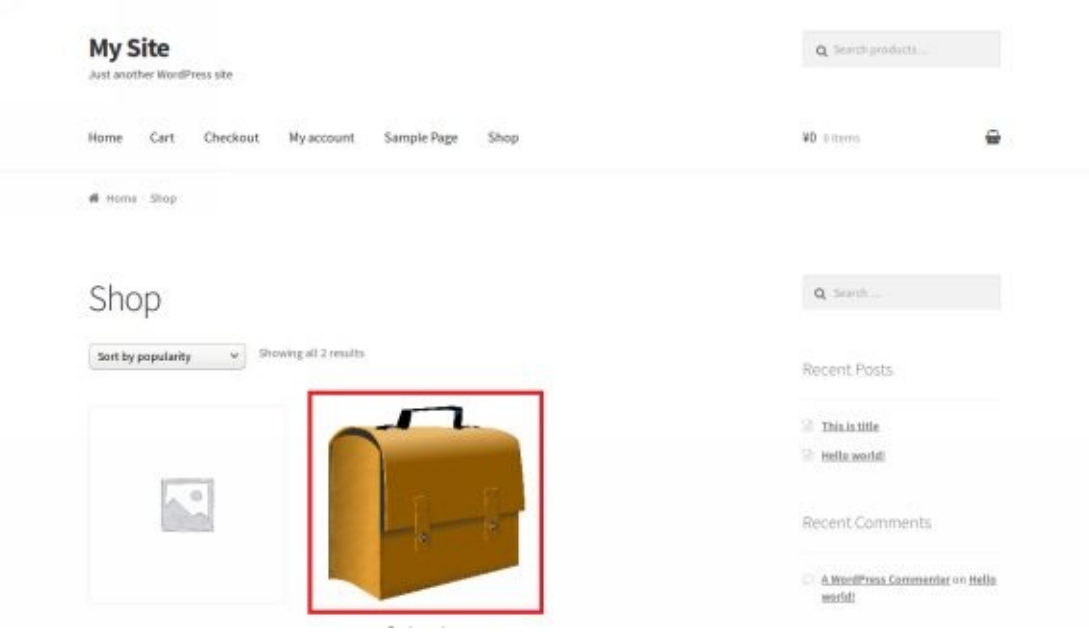
HTML of operation target element

```
<a href="http://localhost/wordpress/shop/">Shop</a>
```

Automatic operation code

```
driver.find_element_by_link_text("Shop").click()
```

Click image



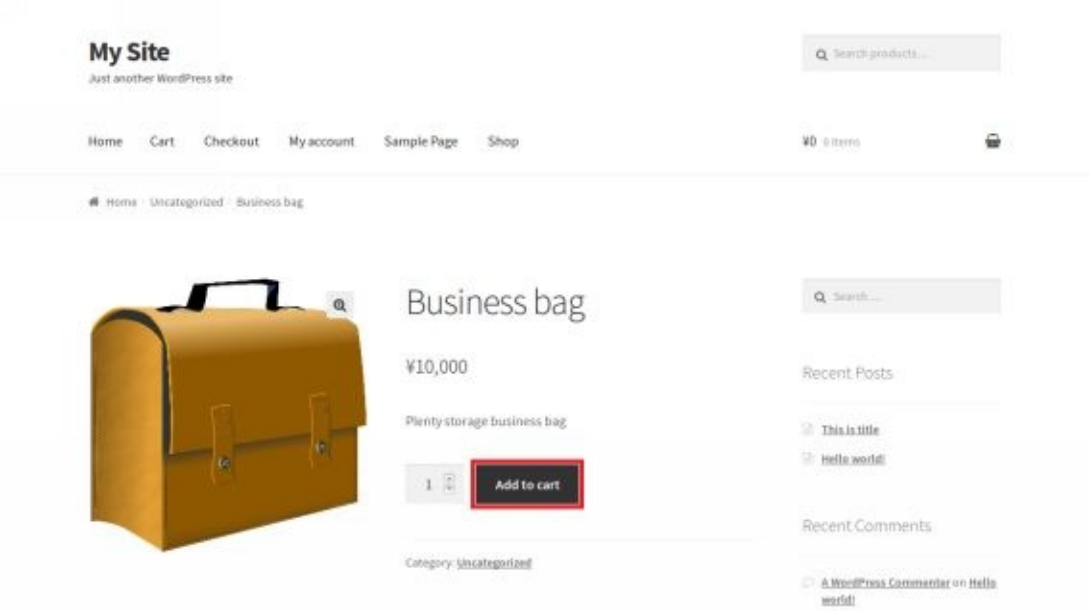
HTML of operation target element

```
<img src="//localhost/wordpress/wp-content/uploads/2018/03/business.png" class="attachment-woocommerce_thumbnail size-woocommerce_thumbnail wp-post-image" alt="" width="324" height="322">
```

Automatic operation code

```
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
```

Click "Add to cart"



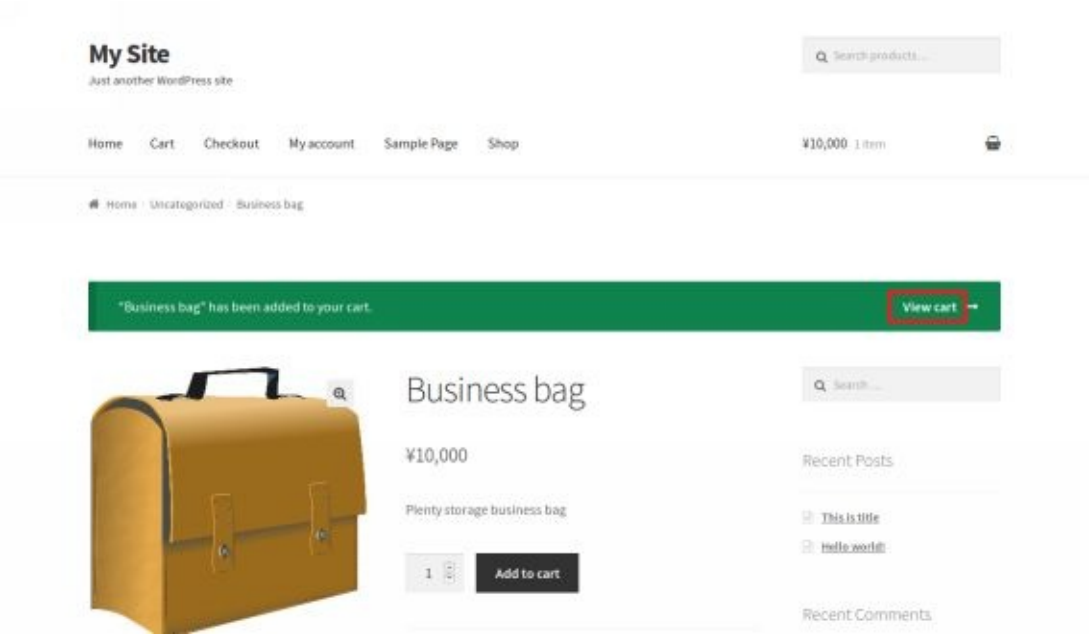
HTML of operation target element

```
<button type="submit" name="add-to-cart" value="11" class="single_add_to_cart_button button alt">Add to cart</button>
```

Automatic operation code

```
driver.find_element_by_name("add-to-cart").click()
```

Click "View cart"



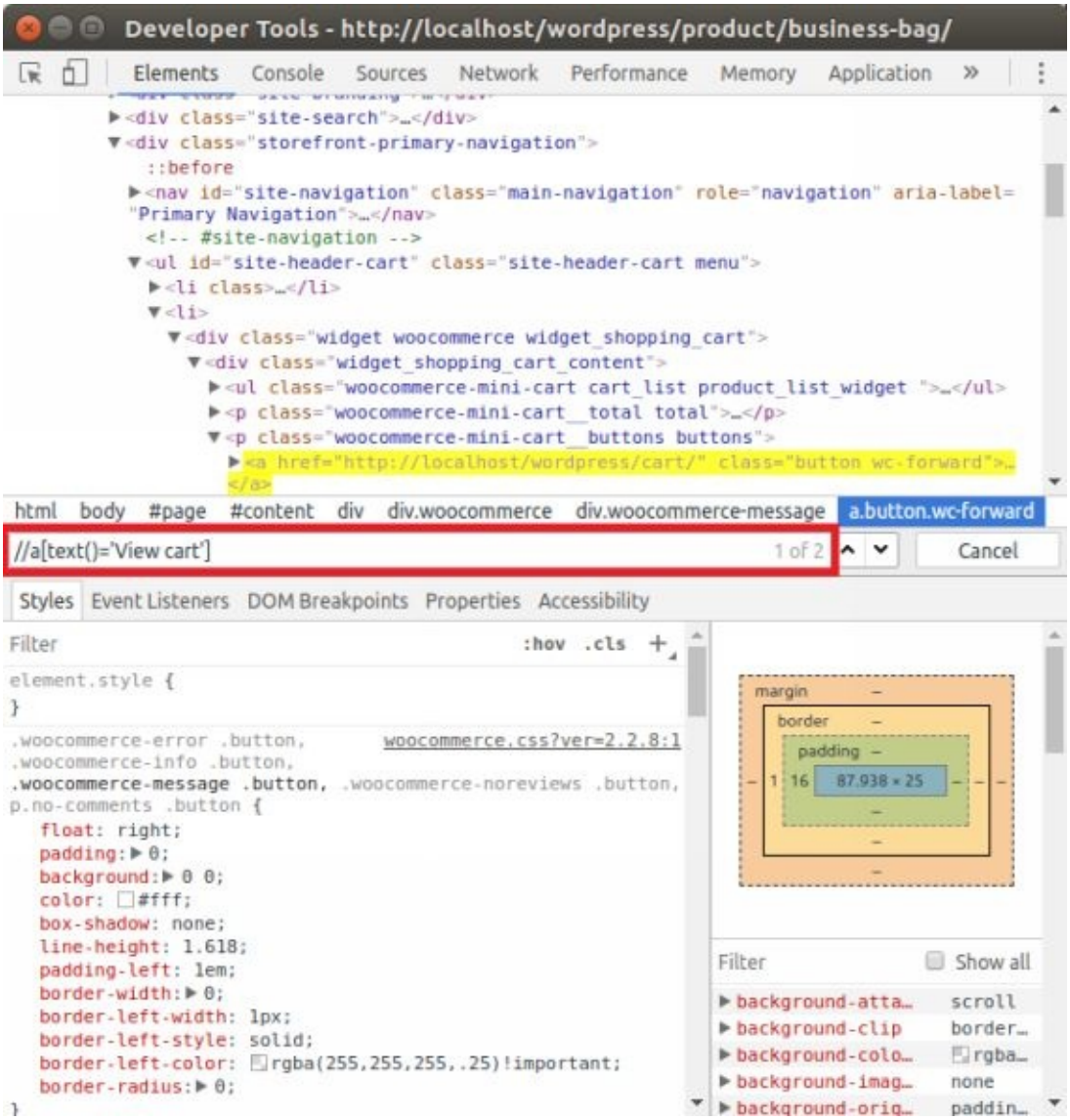
HTML of operation target element

```
<div class="woocommerce-message">
<a href="http://localhost/wordpress/cart/" class="button wc-forward">View cart</a> “Business bag” has been added to your cart.
</div>
```

Let's check the following xpath in the search box.

```
//a[text()='View cart']
```

The search result is shown below.

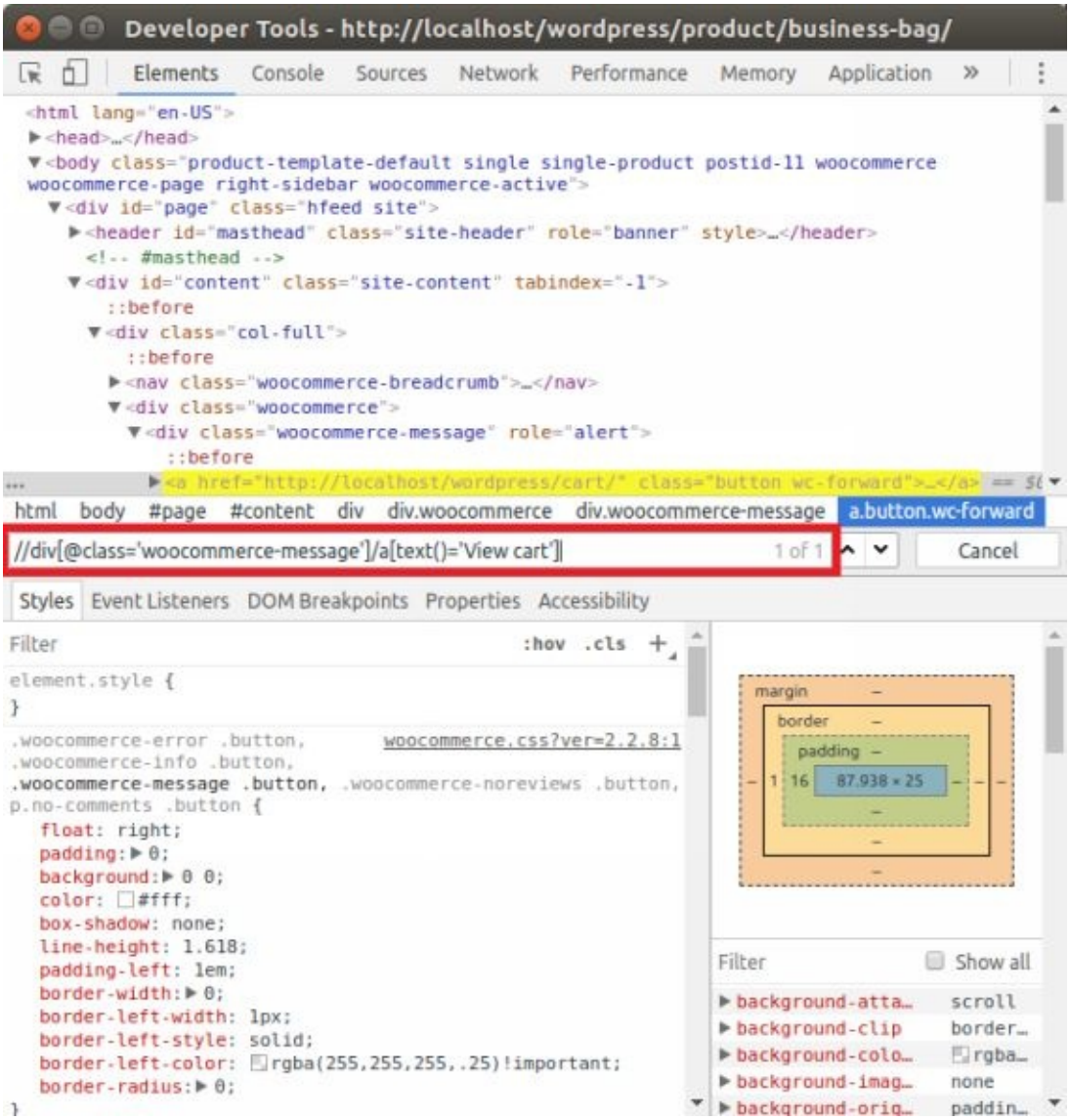


Since the result says "1 of 2", we can see that there are two elements which are identified by the same xpath in this page.

So, next, let's check the following xpath in the search box.

```
//div[@class='woocommerce-message']/a[text()='View cart']
```

The search result is shown below.

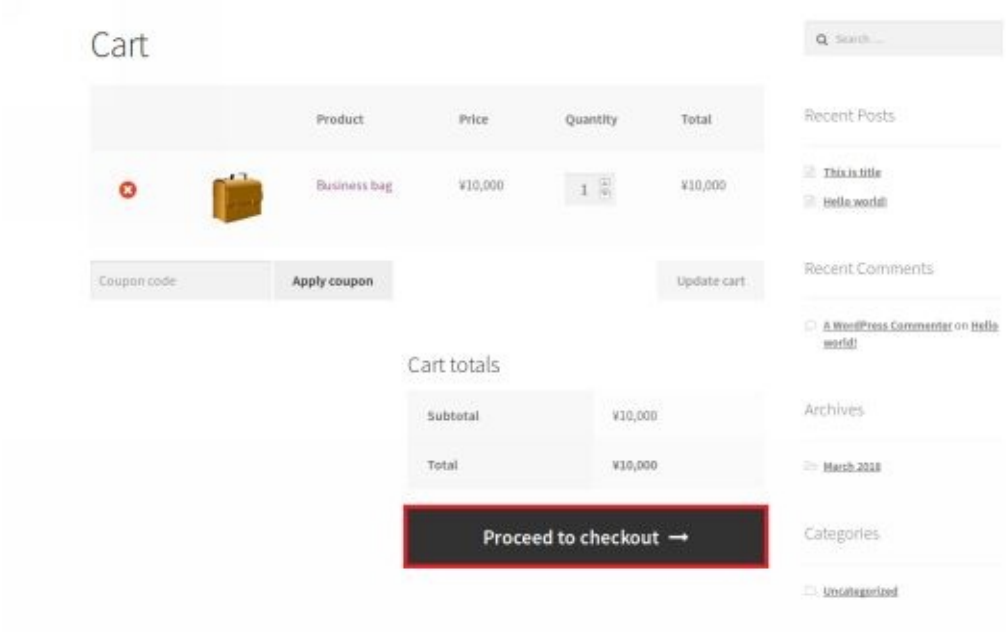


Since the result says "1 of 1", we can see that the element can be identified as one. Therefore, we use this xpath to create code for automatic operation.

Automatic operation code

```
driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()='View cart']").click()
```

Click "Proceed to checkout"



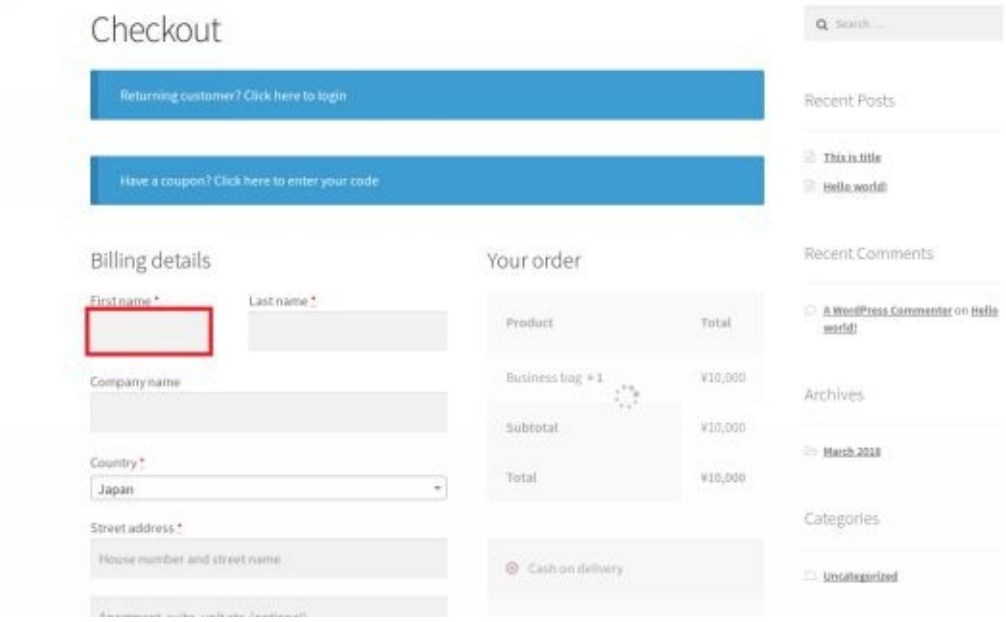
HTML of operation target element

```
<a href="http://localhost/wordpress/checkout/" class="checkout-button button alt wc-forward">Proceed to checkout</a>
```

Automatic operation code

```
driver.find_element_by_link_text("Proceed to checkout").click()
```

Enter "Taro" in First name



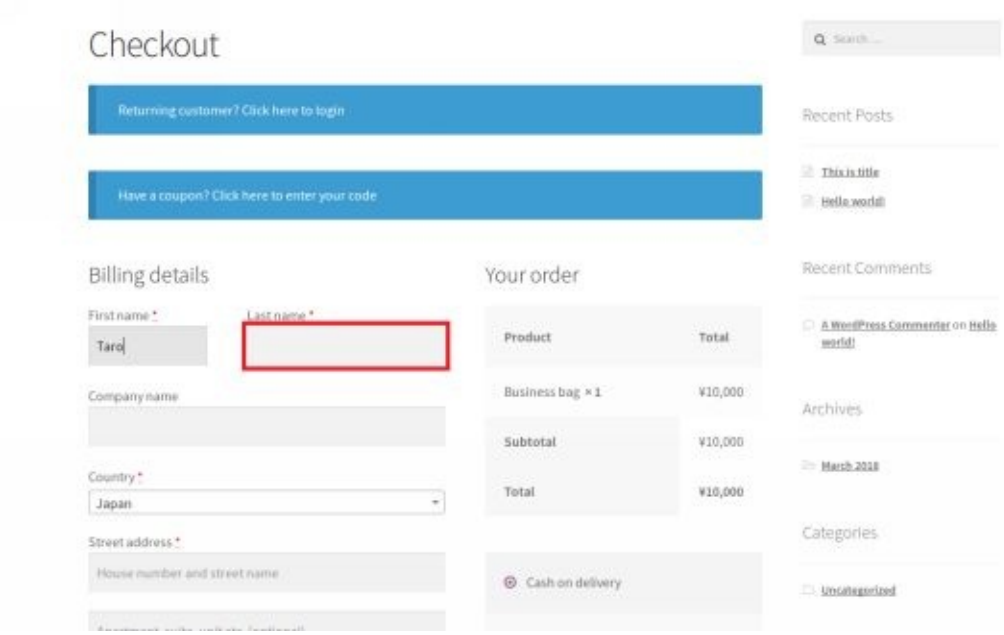
HTML of operation target element

```
<input class="input-text " name="billing_first_name" id="billing_first_name" placeholder="" value="" autocomplete="given-name" autofocus="autofocus" type="text">
```

Automatic operation code

```
driver.find_element_by_id("billing_first_name").send_keys("Taro")
```

Enter "Tokyo" in Last name



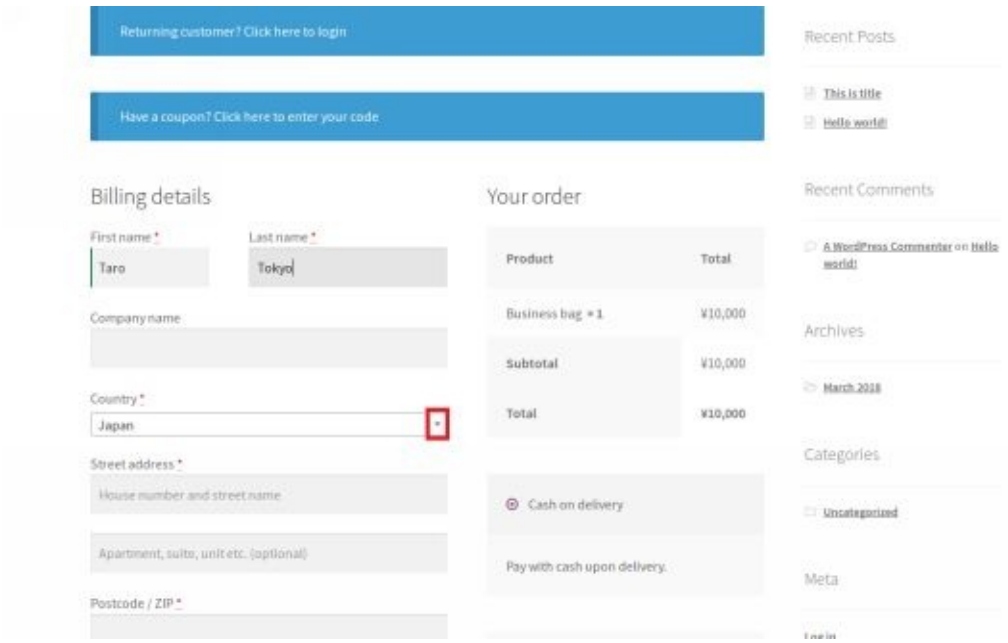
HTML of operation target element

```
<input class="input-text " name="billing_last_name" id="billing_last_name" placeholder="" value="" autocomplete="family-name" type="text">
```

Automatic operation code

```
driver.find_element_by_id("billing_last_name").send_keys("Tokyo")
```

Click the right end of the Country selection



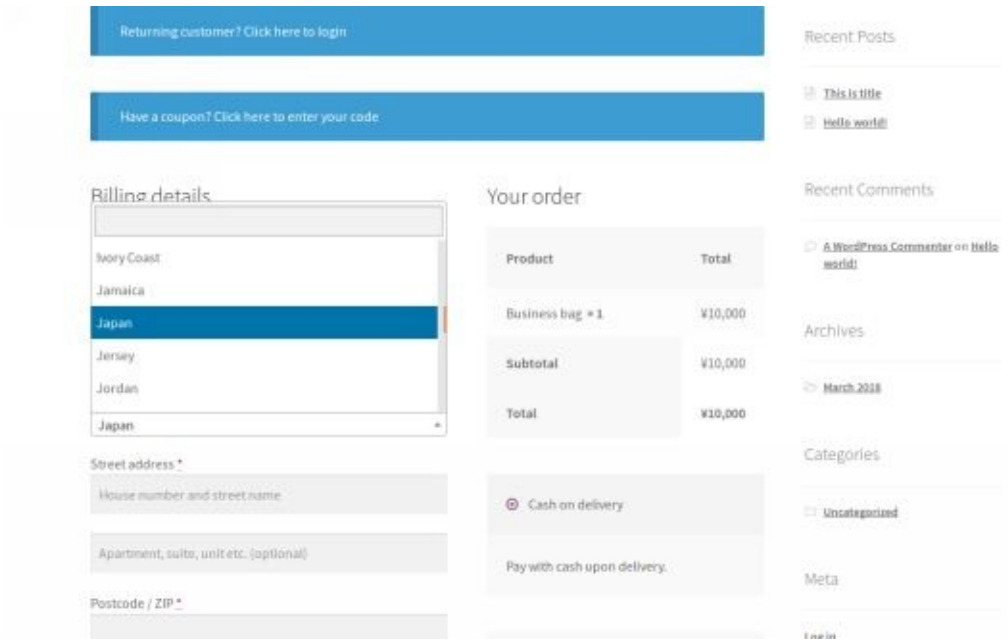
HTML of operation target element

```
<span class="select2-selection__arrow" role="presentation"><b role="presentation"></b></span>
```

Automatic operation code

```
driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").click()
```

Select "Japan"



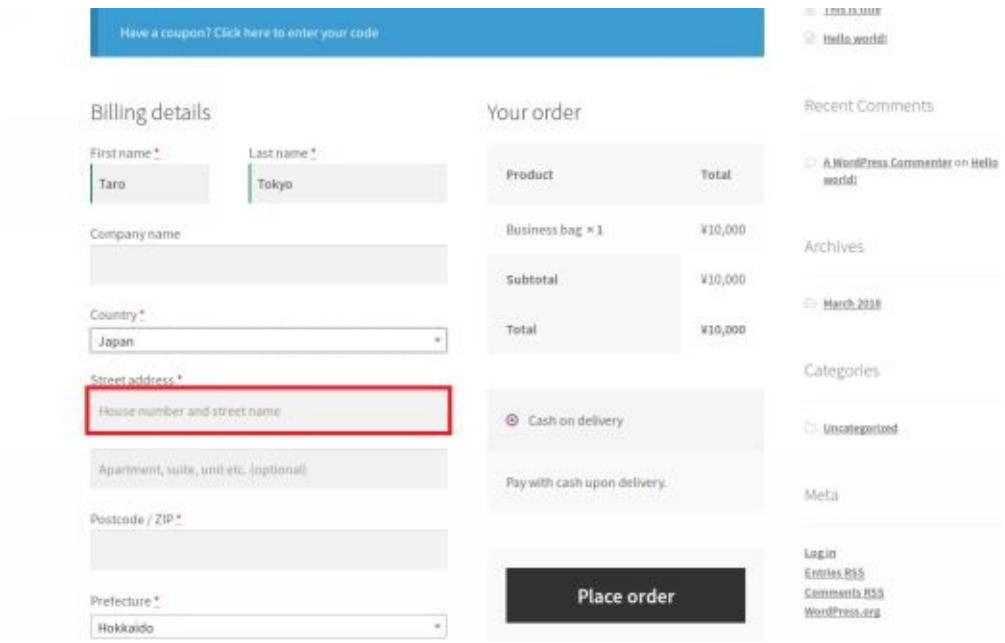
HTML of operation target element

```
<li class="select2-results__option" id="select2-billing_country-result-lmzt-JP" role="option" data-selected="true" tabindex="-1" aria-selected="false">Japan</li>
```

Automatic operation code

```
driver.find_element_by_xpath("//li[text()='Japan']").click()
```

Enter "Shibakoen 4-2-8" in Street address



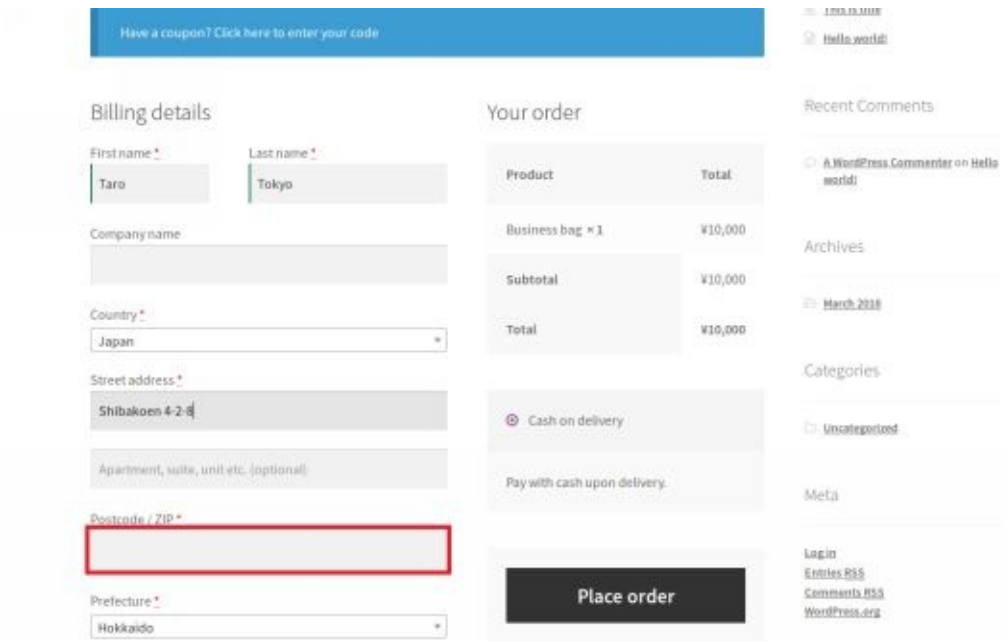
HTML of operation target element

```
<input class="input-text " name="billing_address_1" id="billing_address_1" placeholder="House number and street name" value="" autocomplete="address-line1" type="text">
```

Automatic operation code

```
driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")
```

Enter "105-0011" in Postcode/ZIP



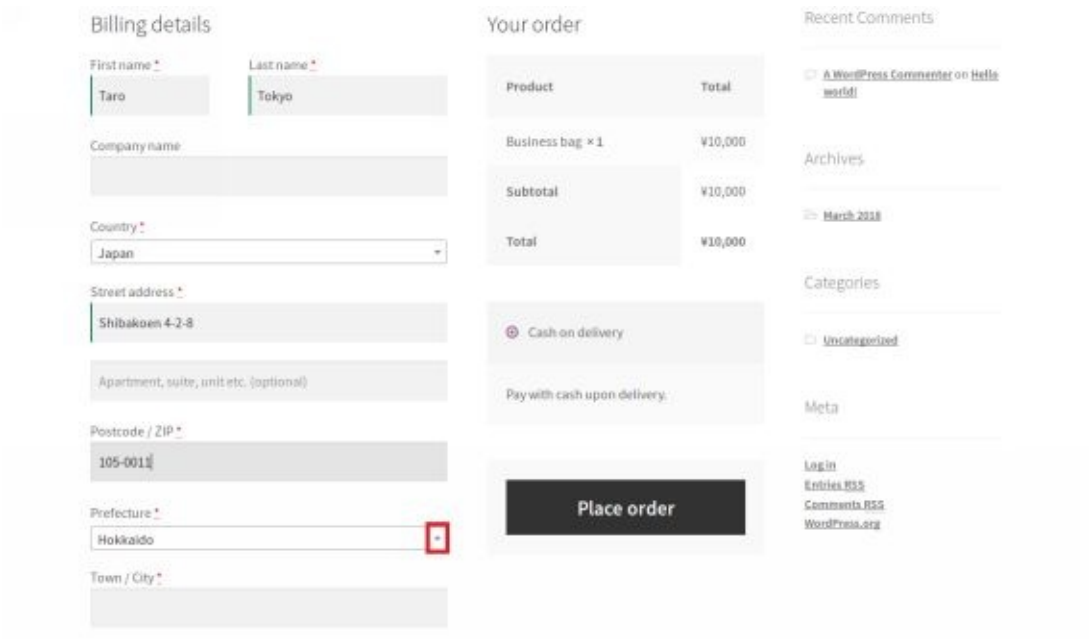
HTML of operation target element

```
<input class="input-text " name="billing_postcode" id="billing_postcode" placeholder="" value="" autocomplete="postal-code" type="text">
```

Automatic operation code

```
driver.find_element_by_id("billing_postcode").send_keys("105-0011")
```

Click the right end of the Prefecture selection



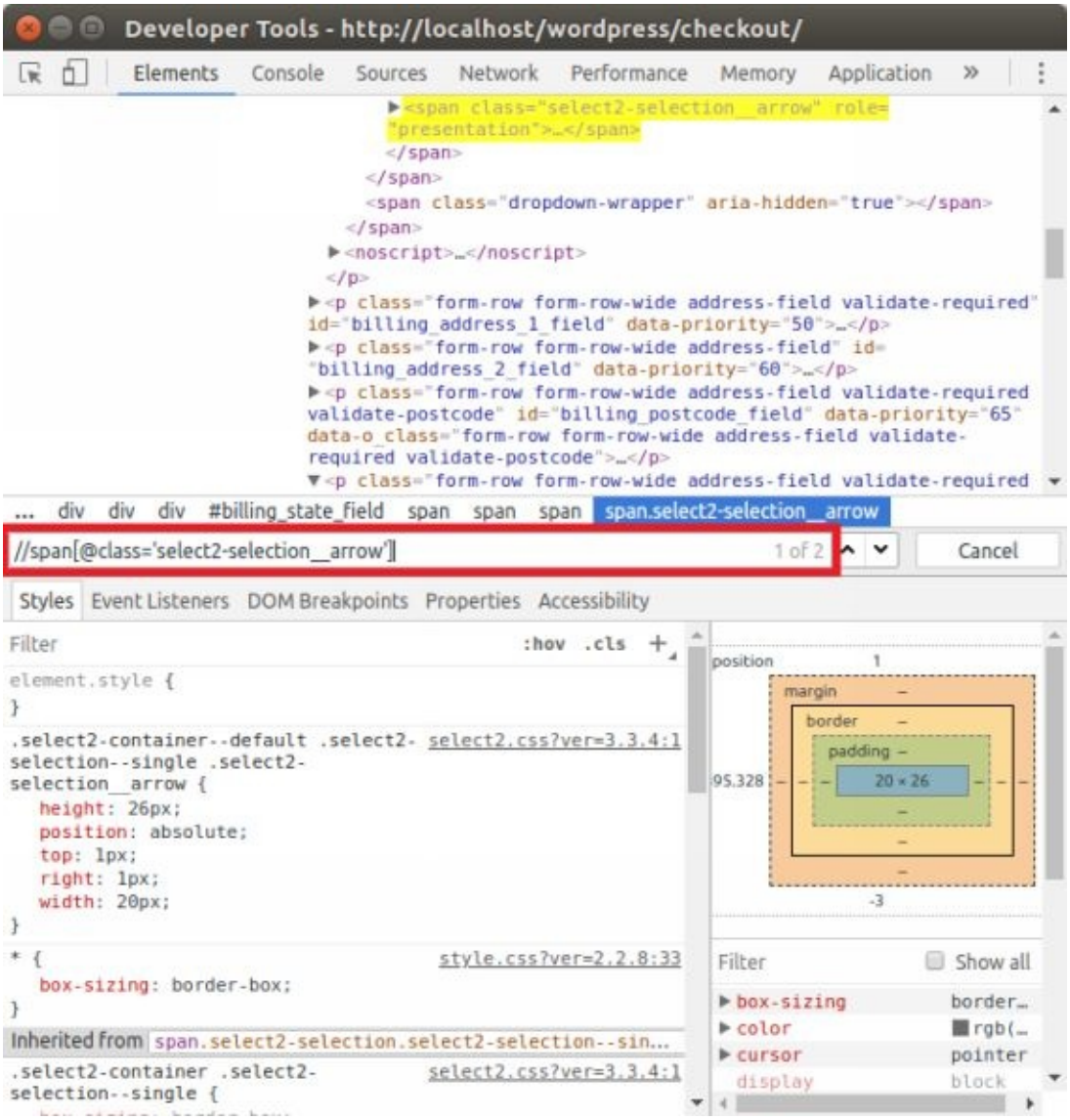
HTML of operation target element

```
<span class="select2-selection__arrow" role="presentation"><b role="presentation"></b></span>
```

Let's check the following xpath in the search box.

```
//span[@class='select2-selection__arrow']
```

The search result is shown below.



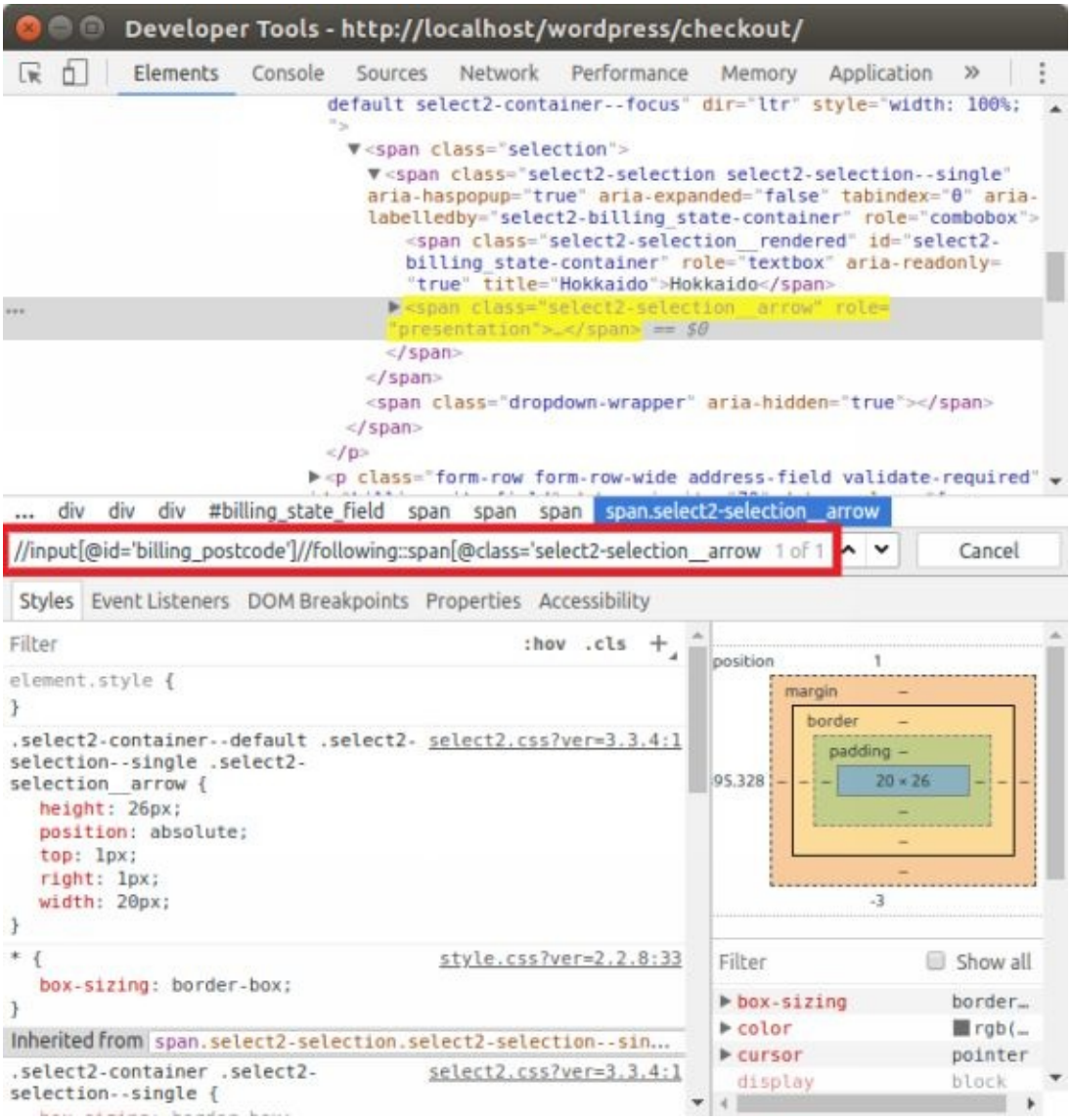
Since the result says "1 of 2", we can see that there are two elements which are identified by the same xpath in this page. (The other is the element on the right end of the country selection in)
So, in order to identify the element as one, we pay attention to the relation with "Postcode / ZIP" element in .

Since the "Postcode / ZIP" element is followed by the "right end of the prefecture selection" element, we try to write xpath using "following".

So, next, let's check the following xpath in the search box.

```
//input[@id='billing_postcode']//following::span[@class='select2-selection__arrow']
```

The search result is shown below.

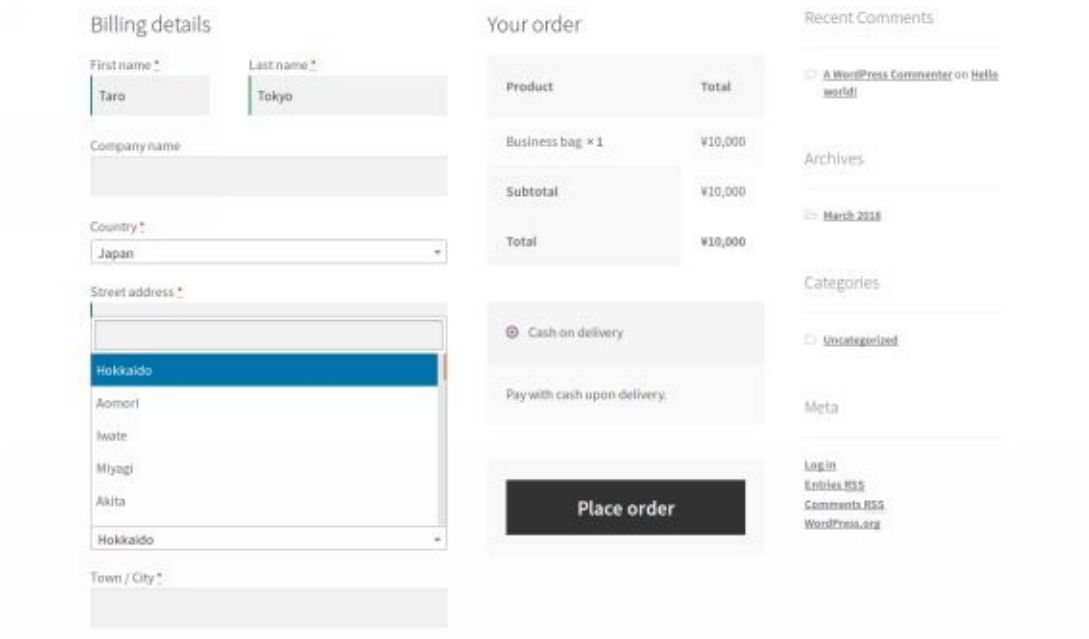


Since the result says "1 of 1", we can see that the element can be identified as one. Therefore, we use this xpath to create code for automatic operation.

Automatic operation code

```
driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click()
```

Select "Tokyo"



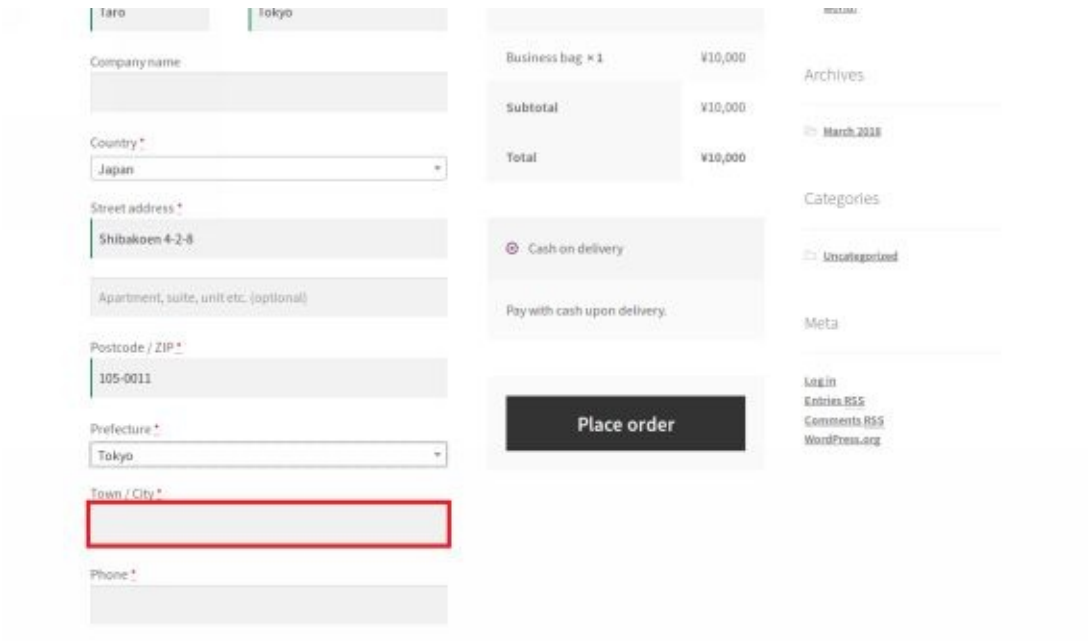
HTML of operation target element

```
<li class="select2-results__option" id="select2-billing_state-result-i9x9-JP13" role="option" data-selected="false" tabindex="-1" aria-selected="false">Tokyo</li>
```

Automatic operation code

```
driver.find_element_by_xpath("//li[text()='Tokyo']").click()
```

Enter "Minato-ku" in Town/City



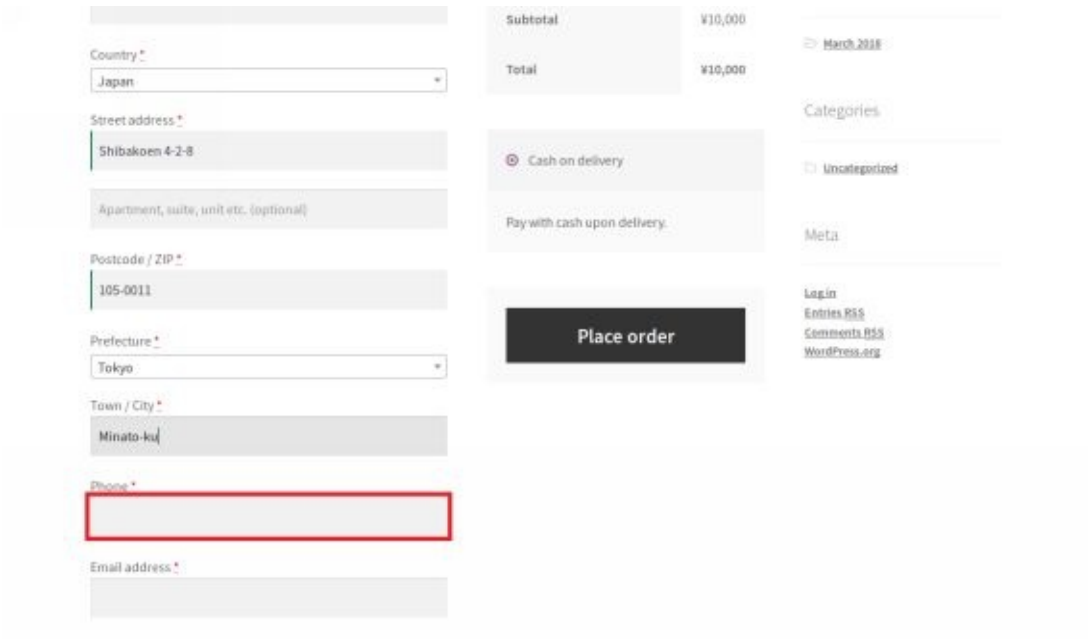
HTML of operation target element

```
<input class="input-text " name="billing_city" id="billing_city" placeholder="" value="" autocomplete="address-level2" type="text">
```

Automatic operation code

```
driver.find_element_by_id("billing_city").send_keys("Minato-ku")
```

Enter "1234567890" in Phone



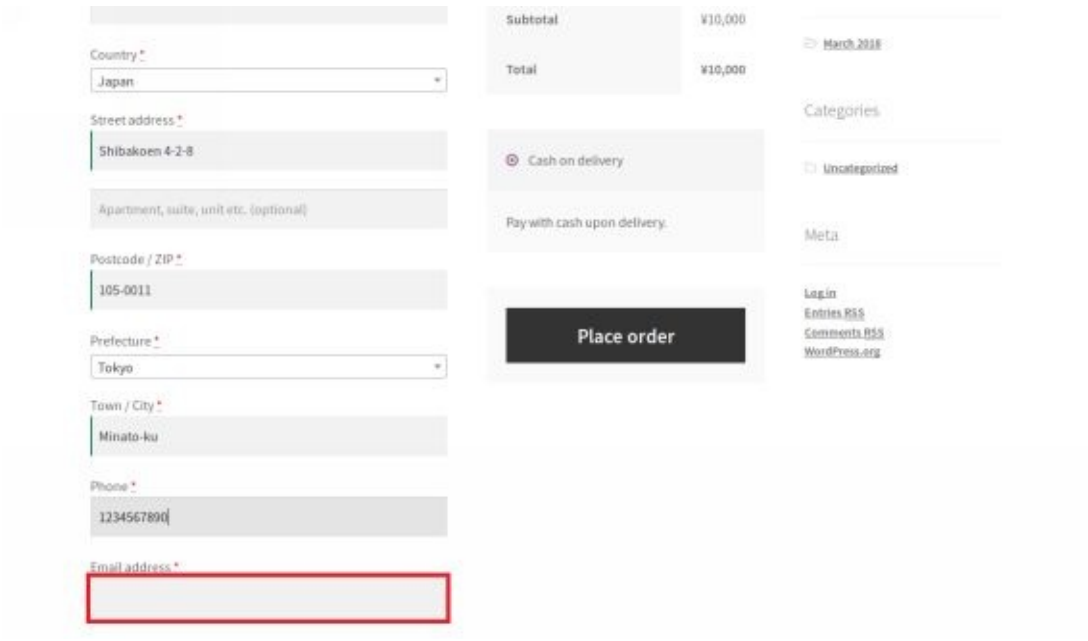
HTML of operation target element

```
<input class="input-text " name="billing_phone" id="billing_phone" placeholder="" value="" autocomplete="tel" type="tel">
```

Automatic operation code

```
driver.find_element_by_id("billing_phone").send_keys("1234567890")
```

Enter "taro@example.com" in Email address



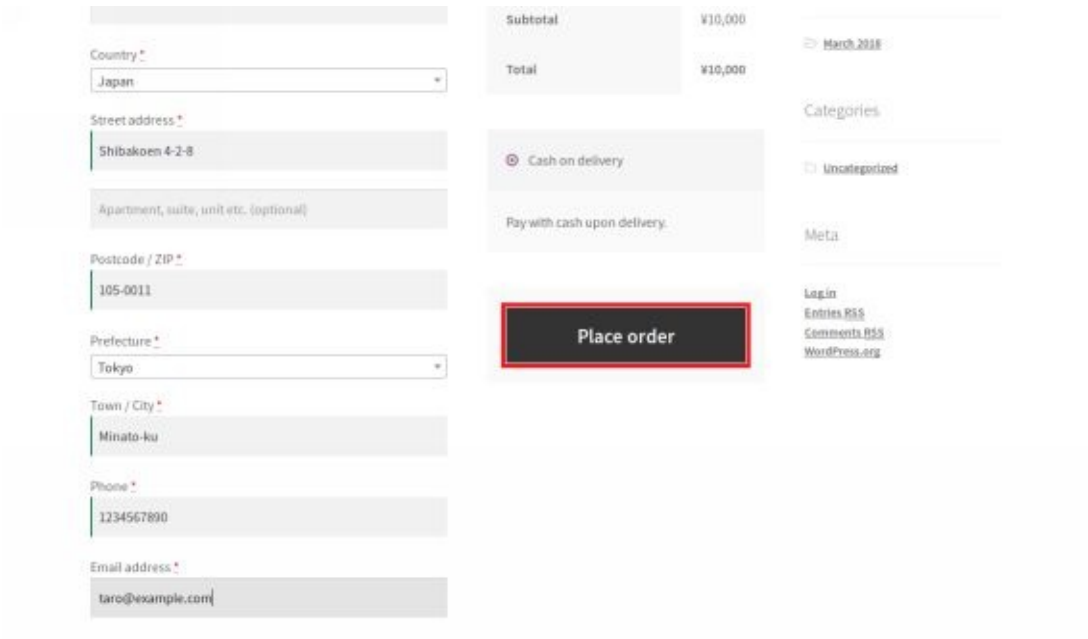
HTML of operation target element

```
<input class="input-text " name="billing_email" id="billing_email" placeholder="" value="" autocomplete="email username" type="email">
```

Automatic operation code

```
driver.find_element_by_id("billing_email").send_keys("taro@example.com")
```

Click "Place order"



HTML of operation target element

```
<input class="button alt" name="woocommerce_checkout_place_order" id="place_order" value="Place order" data-value="Place order" type="submit">
```

Automatic operation code

```
driver.find_element_by_id("place_order").click()
```

4-4-3 Complete and execute automatic operation code

Since all the codes of automatic operation of product purchase of the EC site have already been created, the following code completes the code of automatic operation of the product purchase of the EC site in one file.

buy_product_firefox.py (This code fails in the middle)

```
from selenium import webdriver

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/")

driver.find_element_by_link_text("Shop").click()
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
driver.find_element_by_name("add-to-cart").click()

driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()=
cart']").click()

driver.find_element_by_link_text("Proceed to checkout").click()

driver.find_element_by_id("billing_first_name").send_keys("Taro")

driver.find_element_by_id("billing_last_name").send_keys("Tokyo")

driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").clic
driver.find_element_by_xpath("//li[text()='Japan']").click()

driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")

driver.find_element_by_id("billing_postcode").send_keys("105-0011")

driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click() driver.find_element_by_xpath("//li[text()='Tokyo']")

driver.find_element_by_id("billing_city").send_keys("Minato-ku")
```

```
driver.find_element_by_id("billing_phone").send_keys("1234567890")

driver.find_element_by_id("billing_email").send_keys("taro@example.com")

driver.find_element_by_id("place_order").click()
```

buy_product_chrome.py

```
from selenium import webdriver

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/")
driver.set_window_size(1280, 800)

driver.find_element_by_link_text("Shop").click()
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
driver.find_element_by_name("add-to-cart").click()

driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()=
cart']").click()

driver.find_element_by_link_text("Proceed to checkout").click()

driver.find_element_by_id("billing_first_name").send_keys("Taro")

driver.find_element_by_id("billing_last_name").send_keys("Tokyo")

driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").clic
driver.find_element_by_xpath("//li[text()='Japan']").click()

driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")

driver.find_element_by_id("billing_postcode").send_keys("105-0011")

driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click() driver.find_element_by_xpath("//li[text()='Tokyo']")

driver.find_element_by_id("billing_city").send_keys("Minato-ku")
```

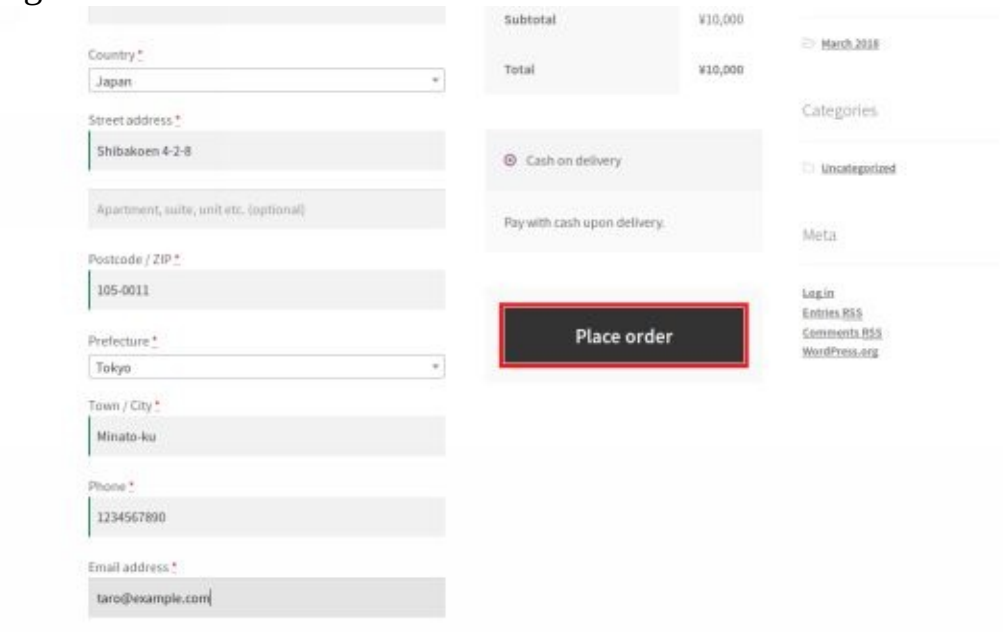
```
driver.find_element_by_id("billing_phone").send_keys("1234567890")

driver.find_element_by_id("billing_email").send_keys("taro@example.com")

driver.find_element_by_id("place_order").click()
```

The above code works fine for Chrome but fails for Firefox.

An error occurs in the following part of the code for Firefox.

```
driver.find_element_by_id("place_order").click()
```

This code performs the operation of clicking the "Place order" button in the figure below.



For Firefox, "StaleElementReferenceException" error occurs as shown in the figure below.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ python buy_product_firefox.py
Traceback (most recent call last):
  File "buy_product_firefox.py", line 35, in <module>
    driver.find_element_by_id("place_order").click()
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 80, in click
    self._execute(Command.CLICK_ELEMENT)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webelem
ent.py", line 628, in _execute
    return self._parent.execute(command, params)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/webdriv
er.py", line 312, in execute
    self.error_handler.check_response(response)
  File "/usr/local/lib/python2.7/dist-packages/selenium/webdriver/remote/errorha
ndler.py", line 242, in check_response
    raise exception_class(message, screen, stacktrace)
selenium.common.exceptions.StaleElementReferenceException: Message: The element
reference of <button id="place_order" class="button alt" name="woocommerce_check
out_place_order" type="submit"> is stale; either the element is no longer attach
ed to the DOM, it is not in the current frame context, or the document has been
refreshed

kenpapa@kenpapa-VirtualBox:~/python$
```

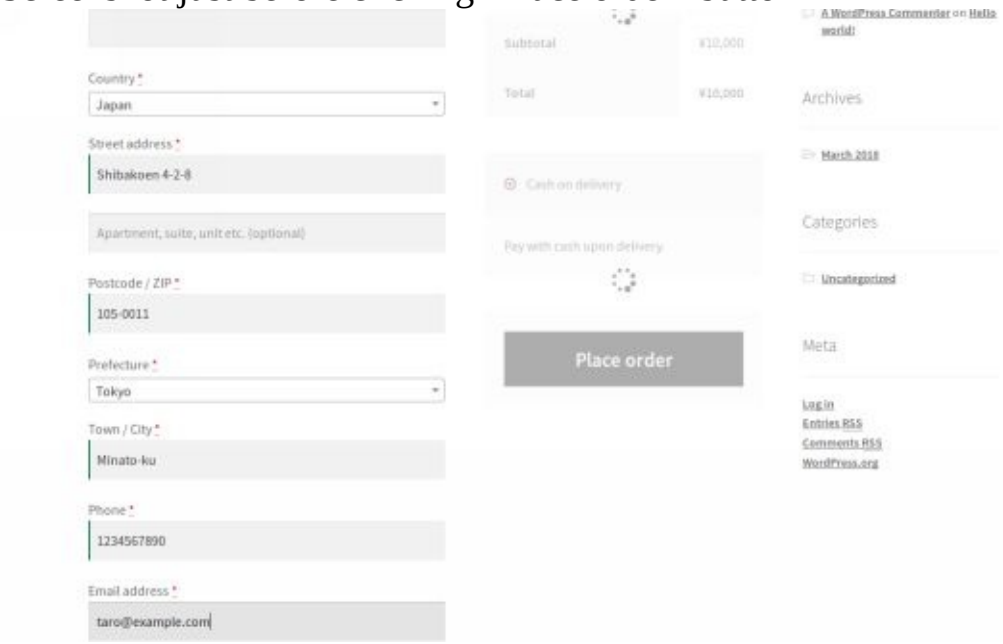
Therefore, in order to check the difference between Firefox and Chrome, we take two screenshots, the one just before clicking the "Place order" button and another taken after 2 seconds from the first screenshot.

Modified code to take screenshots for Firefox

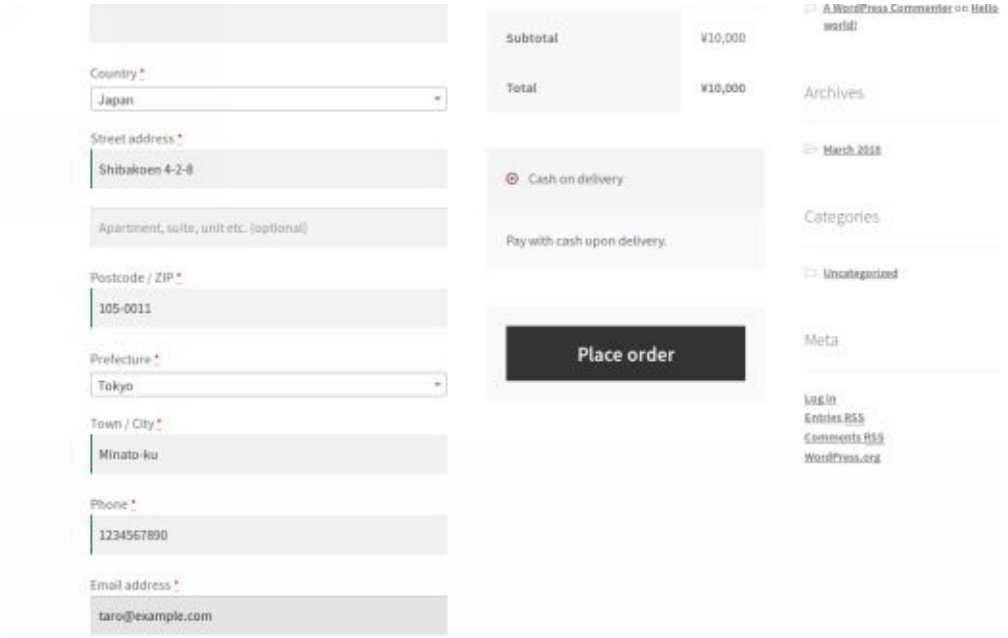
```
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/")

driver.find_element_by_link_text("Shop").click()
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
driver.find_element_by_name("add-to-cart").click()

driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()=
cart']").click()

driver.find_element_by_link_text("Proceed to checkout").click()

driver.find_element_by_id("billing_first_name").send_keys("Taro")
```

```
driver.find_element_by_id("billing_last_name").send_keys("Tokyo")

driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").clic
driver.find_element_by_xpath("//li[text()='Japan']").click()

driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")

driver.find_element_by_id("billing_postcode").send_keys("105-0011")

driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click() driver.find_element_by_xpath("//li[text()='Tokyo']")

driver.find_element_by_id("billing_city").send_keys("Minato-ku")

driver.find_element_by_id("billing_phone").send_keys("1234567890")

driver.find_element_by_id("billing_email").send_keys("taro@example.com")

driver.save_screenshot("/home/kenpapa/before_order_firefox.png")
time.sleep(2)
driver.save_screenshot("/home/kenpapa/before_order_plus_sleep_firefox.p
#driver.find_element_by_id("place_order").click()
```

Screenshot just before clicking "Place order" button



Screenshot after 2 seconds from the first screenshot

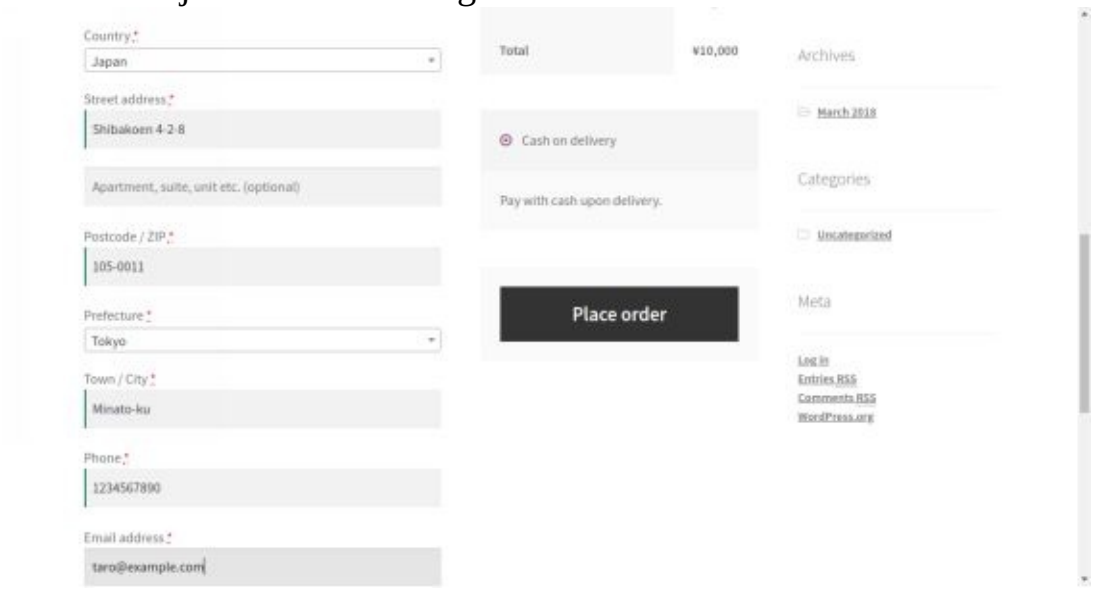


Modified code to take screenshots for Chrome

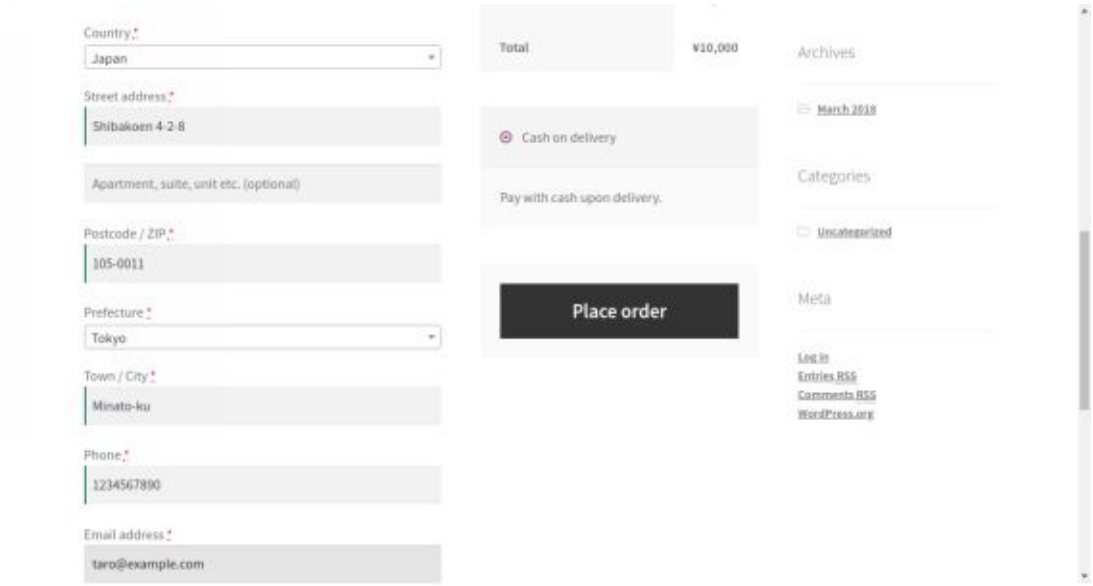
```
from selenium import webdriver
import time

driver = webdriver.Chrome()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/")
driver.set_window_size(1280, 800)

driver.find_element_by_link_text("Shop").click()
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
driver.find_element_by_name("add-to-cart").click()

driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()=
cart']").click()

driver.find_element_by_link_text("Proceed to checkout").click()

driver.find_element_by_id("billing_first_name").send_keys("Taro")

driver.find_element_by_id("billing_last_name").send_keys("Tokyo")
```

```
driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").clic
driver.find_element_by_xpath("//li[text()='Japan']").click()

driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")

driver.find_element_by_id("billing_postcode").send_keys("105-0011")

driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click() driver.find_element_by_xpath("//li[text()='Tokyo']")

driver.find_element_by_id("billing_city").send_keys("Minato-ku")

driver.find_element_by_id("billing_phone").send_keys("1234567890")

driver.find_element_by_id("billing_email").send_keys("taro@example.com")

driver.save_screenshot("/home/kenpapa/before_order_chrome.png")
time.sleep(2)
driver.save_screenshot("/home/kenpapa/before_order_plus_sleep_chrome.p
#driver.find_element_by_id("place_order").click()
```

Screenshot just before clicking "Place order" button



Screenshot after 2 seconds from the first screenshot



In Chrome there is no difference between the two screenshots (the page had been completed before the "Place order" button was clicked), but there is a difference in Firefox (the page hadn't been completed when the button was clicked). Therefore, it is thought that an error occurred in Firefox because Selenium tried to click the button before the page was completed.

The following code is the modified code. 2 seconds sleep was added so that automatic operation is performed after the page has been completed.

buy_product_firefox.rb (Modified version)

```
from selenium import webdriver
import time

driver = webdriver.Firefox()
driver.implicitly_wait(10)
driver.get("http://localhost/wordpress/")

driver.find_element_by_link_text("Shop").click()
driver.find_element_by_xpath("//img[contains(@src,'business')]").click()
driver.find_element_by_name("add-to-cart").click()

driver.find_element_by_xpath("//div[@class='woocommerce-message']/a[text()=
cart']").click()
```

```
driver.find_element_by_link_text("Proceed to checkout").click()

driver.find_element_by_id("billing_first_name").send_keys("Taro")

driver.find_element_by_id("billing_last_name").send_keys("Tokyo")

driver.find_element_by_xpath("//span[@class='select2-selection__arrow']").clic
driver.find_element_by_xpath("//li[text()='Japan']").click()

driver.find_element_by_id("billing_address_1").send_keys("Shibakoen 4-2-8")

driver.find_element_by_id("billing_postcode").send_keys("105-0011")

driver.find_element_by_xpath("//input[@id='billing_postcode']//following::span
selection__arrow']").click() driver.find_element_by_xpath("//li[text()='Tokyo']")

driver.find_element_by_id("billing_city").send_keys("Minato-ku")

driver.find_element_by_id("billing_phone").send_keys("1234567890")

driver.find_element_by_id("billing_email").send_keys("taro@example.com")

time.sleep(2)
driver.find_element_by_id("place_order").click()
```

As for "StaleElementReferenceException", there is an explanation on Selenium official website.

http://www.seleniumhq.org/exceptions/stale_element_reference.jsp

In Chapter 3, the same automatic operation code worked without problems for Firefox and Chrome. On the other hand, in this chapter, the behavior was different between browsers and it was necessary to modify the code.

Chapter 5 Automated testing with Test Framework

In this chapter, we execute automated testing of web application using Selenium WebDriver and Test Framework.

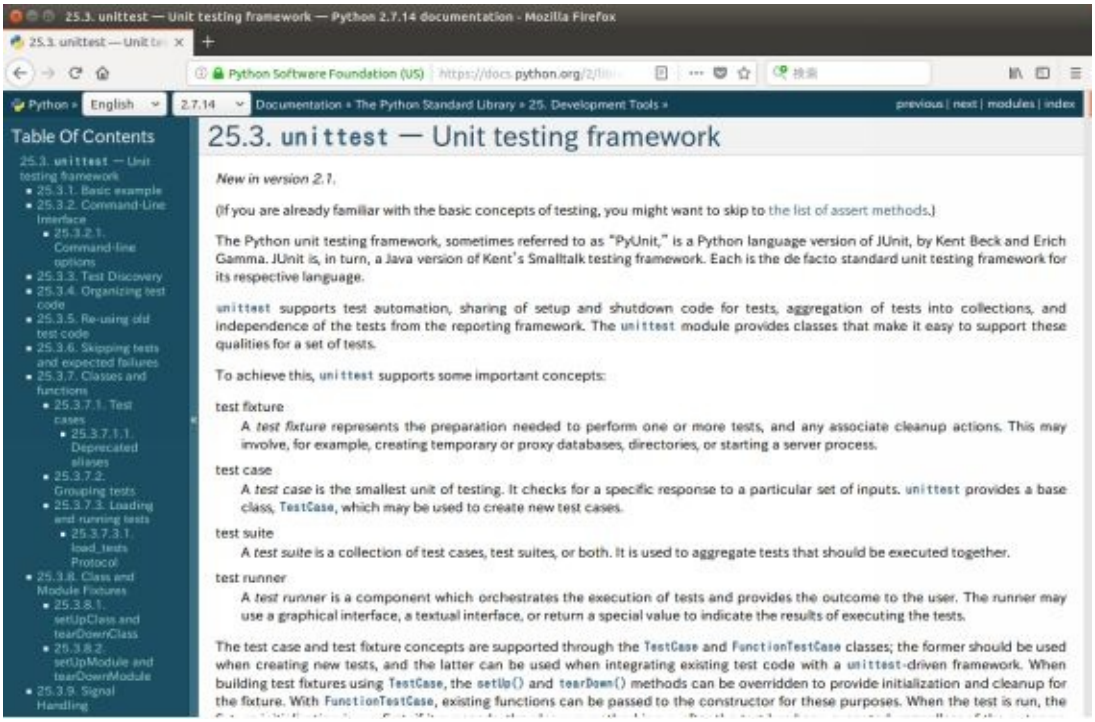
The advantages of automated testing are as follows.
Can be executed many times
There is no operational mistake like a human being
Execute faster than human
Reusable
The execution timing can be controlled with the related tool (Jenkins etc)

In this chapter, we execute the following two tests using Selenium WebDriver and Test Framework.
Google Search Test
WordPress Login Test

5-1 What is unittest ?

Python has a test framework called unittest. Multiple tests can be executed all at once by using test framework.

unittest official website



https://docs.python.org/2/library/unittest.html

The following code is a simple test code written using unittest. When running this code, two tests "test1" and "test2" are executed.

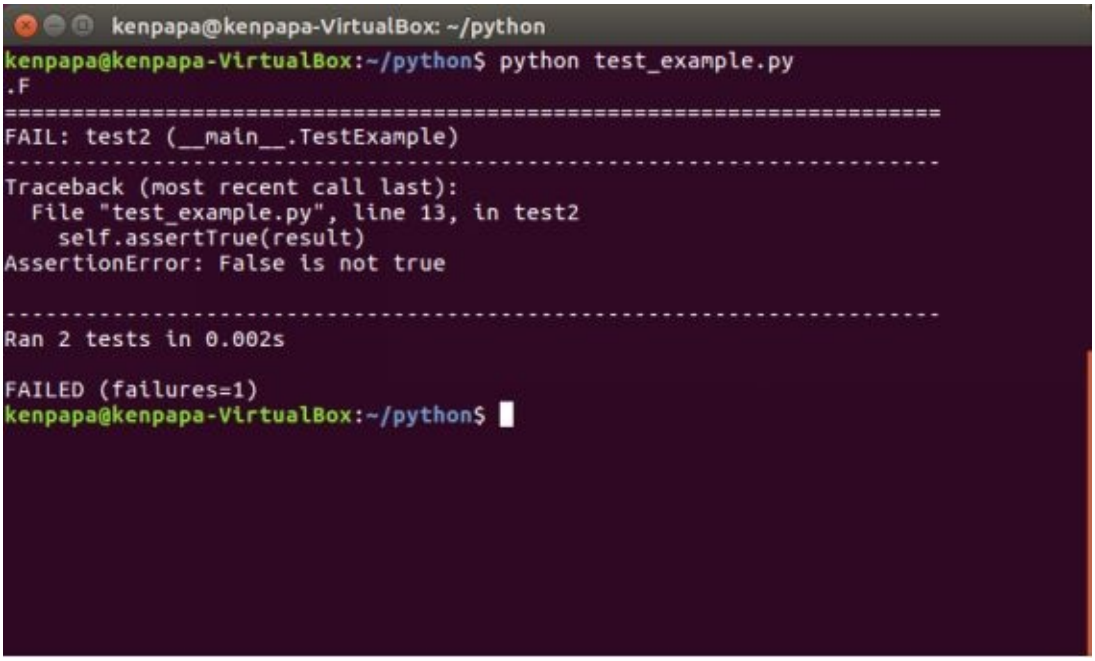
test_example.py

```
import unittest

class TestExample(unittest.TestCase):

    def test1(self):
    text = "hello world"
    result = "world" in text
    self.assertTrue(result)

    def test2(self):
    text = "hello world"
```

```
    result = "WORLD" in text
    self.assertTrue(result)

if __name__ == "__main__":
    unittest.main()
```

In each test, "assertTrue" method is used to judge whether the test is success or failure.

The test code can be executed with "python" command. The execution result is as follows.



In test1, "world" exists in the string "hello world", so the test has been successful.
In test2, "WORLD" does not exist in the string "hello world", so the test has failed.

As shown above, multiple tests can be executed all at once by using test framework.

5-2 Google Search Test

Google Search Test uses the title of the page as a judgment of whether the test is success or failure. So, first of all we practice to get the title of the page with Selenium WebDriver.

Let's create and execute the following three codes.

Code1 (google_search.py)

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()

print driver.title

driver.quit()
```

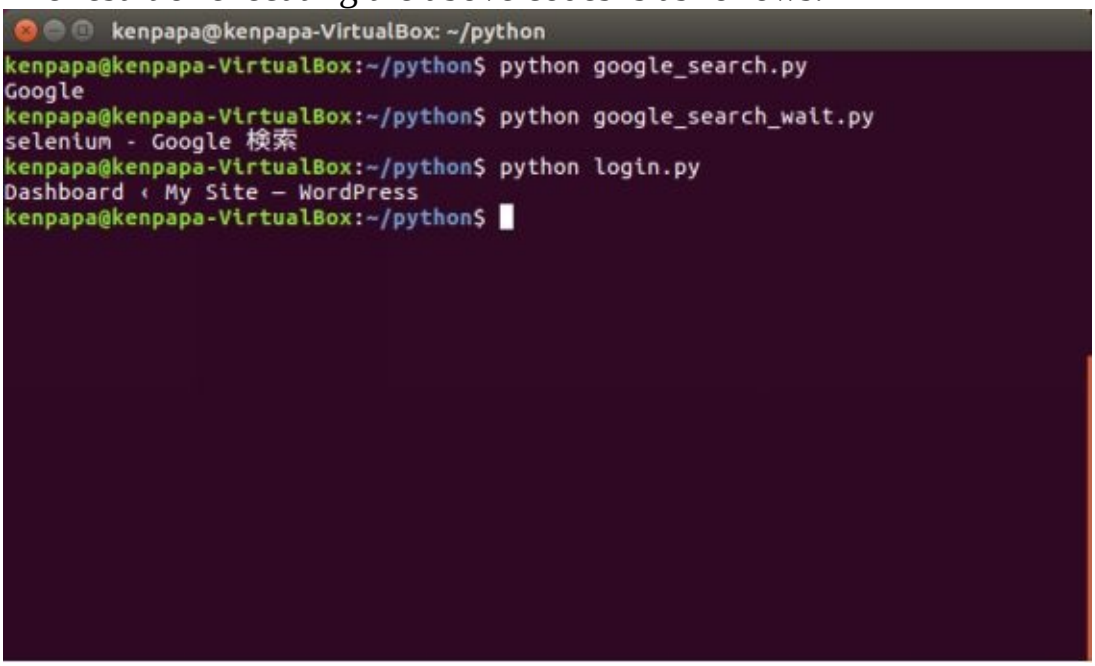
Code2 (google_search_wait.py)

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Firefox()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()

wait = WebDriverWait(driver, 10)
wait.until(EC.title_contains("selenium"))
print driver.title

driver.quit()
```

Code3 (login.py)

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://localhost/wordpress/wp-login.php")
driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

print driver.title

driver.quit()
```

The result of executing the above codes is as follows.



Code1 outputs "Google" which is the title of the page before search processing. **(Unexpected result)**
Code2 is the modified code of Code1. Code2 has wait processing and results in expected result.

Code3 outputs the title of the page after WordPress login. The correct title has been acquired without wait processing.

Like the result of Code1, we often encounter unexpected result when using Selenium.

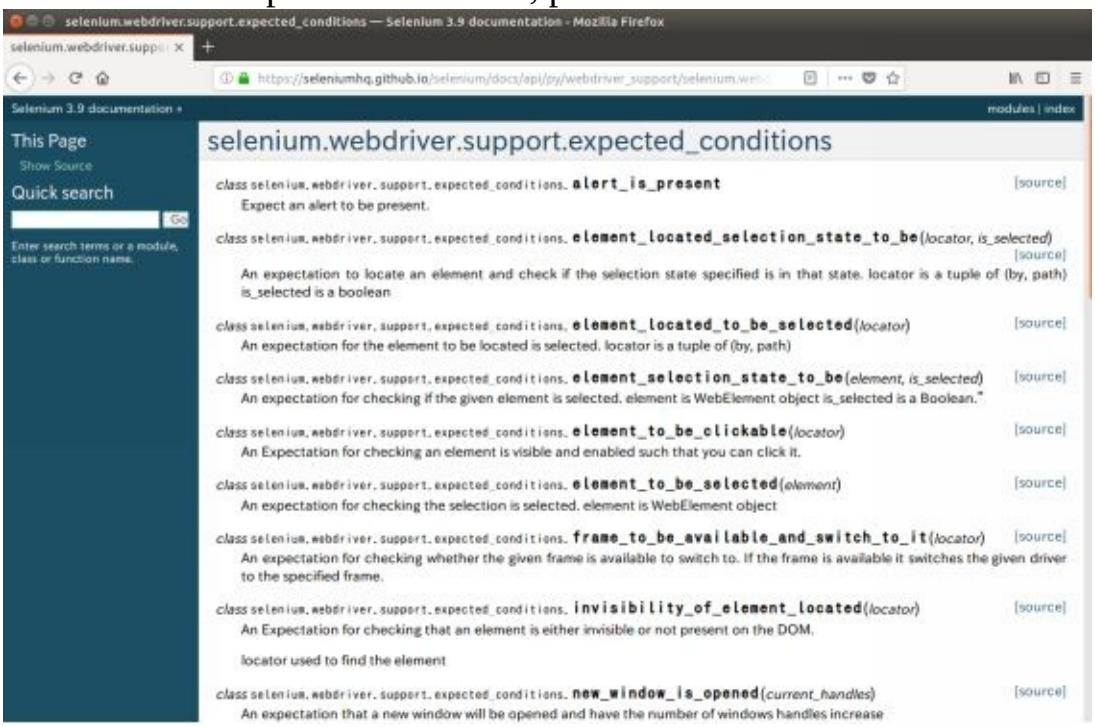
Therefore, wait processing is necessary. One of wait processing in Selenium WebDriver is Explicit Wait and it is used in Code2. Explicit Wait can be used to add wait processing for each operation individually.

Below is the code of Explicit Wait.

```
wait = WebDriverWait(driver, 10)
wait.until(EC.title_contains("selenium"))
```

As a condition of wait processing, "title_contains" which is one of ExpectedCondition is used.

For details on Expected Condition, please check API document.



https://seleniumhq.github.io/selenium/docs/api/py/api.html

The practice of acquiring the page title is finished.

Let's execute Google Search Test. The test code is as follows.

google_search_test.py

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import unittest

class GoogleSearchTest(unittest.TestCase):

    def test_google_search(self):
    driver = webdriver.Firefox()

    driver.get("http://www.google.com")
    element = driver.find_element_by_id("lst-ib")
    element.send_keys("selenium")
    element.submit()

    wait = WebDriverWait(driver, 10)
    wait.until(EC.title_contains("selenium"))
    title = driver.title
    result = "selenium" in title
    self.assertTrue(result)

    driver.quit()

if __name__ == "__main__":
    unittest.main()
```

The execution result is as follows.

```
kenpapa@kenpapa-VirtualBox: ~/python
kenpapa@kenpapa-VirtualBox:~/python$ python google_search_test.py
.
----------------------------------------------------------------------
Ran 1 test in 9.964s

OK
kenpapa@kenpapa-VirtualBox:~/python$
```

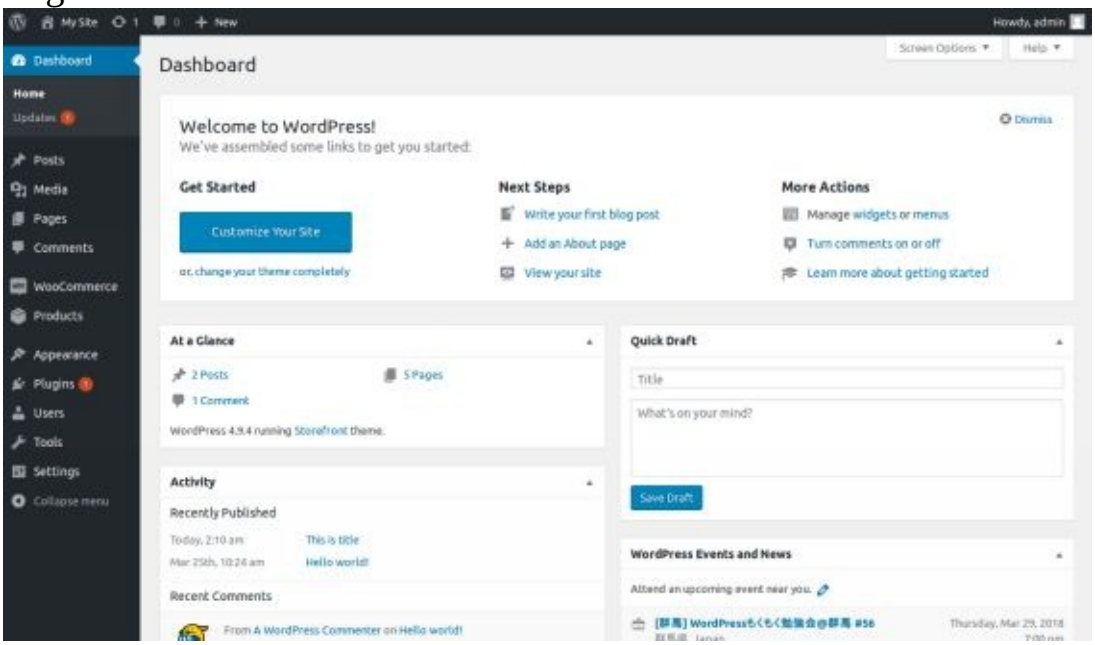
We can confirm that the test was successful.

5-3 WordPress Login Test

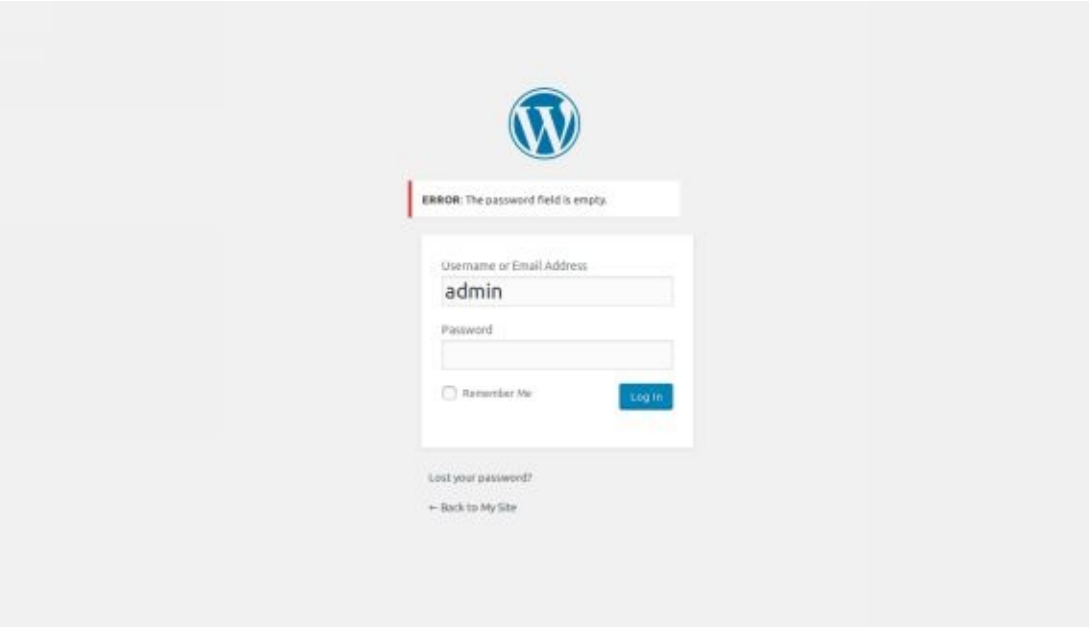
As a next test, we test to confirm that WordPress Login feature works correctly.

We execute the following four test patterns as a test case.
Login success
Login failure (no password)
Login failure (no user name)
Login failure (wrong password)

Login success screen

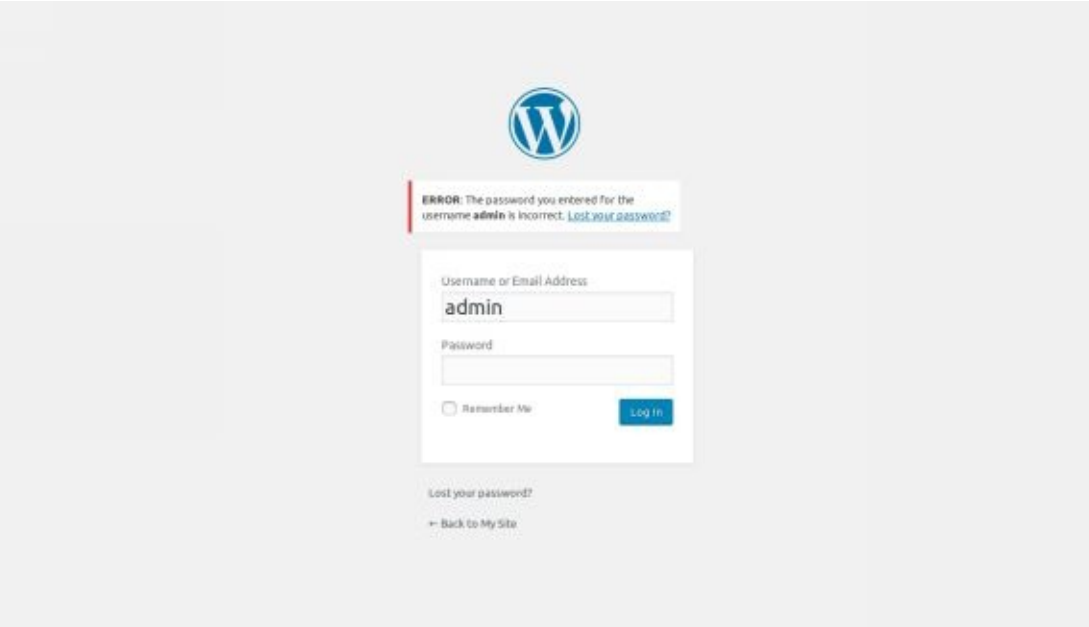


Login failure (no password) screen



Login failure (no user name) screen

Login failed (wrong password) screen



The test code is as follows.

login_test.py

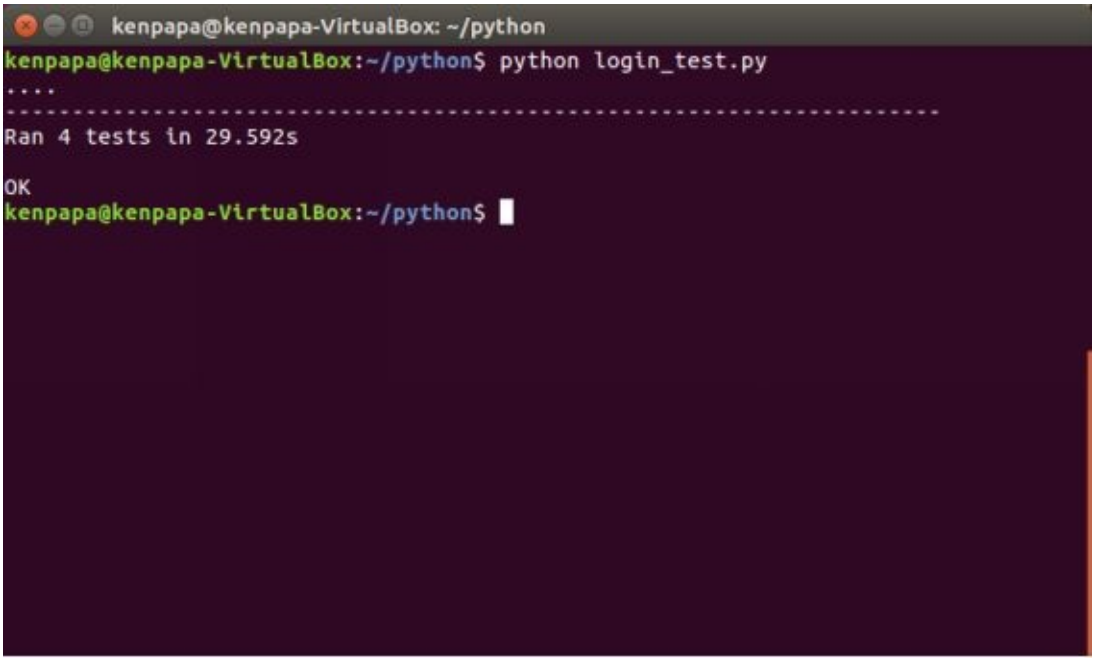
```
from selenium import webdriver
import unittest

class LoginTest(unittest.TestCase):

    def test_login_ok(self):
    driver = webdriver.Firefox()
    driver.get("http://localhost/wordpress/wp-login.php")

    driver.find_element_by_id("user_login").send_keys("admin")
    driver.find_element_by_id("user_pass").send_keys("admin123")
    driver.find_element_by_id("wp-submit").click()

    source = driver.page_source
    result = "Dashboard" in source
    self.assertTrue(result)

    driver.quit()

    def test_login_no_password(self):
```

```
driver = webdriver.Firefox()
driver.get("http://localhost/wordpress/wp-login.php")

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("wp-submit").click()

source = driver.page_source
result = "The password field is empty." in source
self.assertTrue(result)

driver.quit()

def test_login_no_user(self):
driver = webdriver.Firefox()
driver.get("http://localhost/wordpress/wp-login.php")

driver.find_element_by_id("user_pass").send_keys("admin123")
driver.find_element_by_id("wp-submit").click()

source = driver.page_source
result = "The username field is empty." in source
self.assertTrue(result)

driver.quit()

def test_login_wrong_password(self):
driver = webdriver.Firefox()
driver.get("http://localhost/wordpress/wp-login.php")

driver.find_element_by_id("user_login").send_keys("admin")
driver.find_element_by_id("user_pass").send_keys("admin456")
driver.find_element_by_id("wp-submit").click()

source = driver.page_source
result = "The password you entered for the username <strong>admin</strong> is incorrect." in source
self.assertTrue(result)

driver.quit()
```

```
if __name__ == "__main__":
    unittest.main()
```

Each test gets the source of the page with "driver.page_source" and determines whether it is success or failure depending on whether the expected string is included in the source or not.

The execution result is as follows.



Using preprocessing and postprocessing makes the code simple.

```
from selenium import webdriver
import unittest

class LoginTest(unittest.TestCase):

    def setUp(self):
    self.driver = webdriver.Firefox()
    self.driver.get("http://localhost/wordpress/wp-login.php")

    def tearDown(self):
    self.driver.quit()

    def test_login_ok(self):
```

```
    self.driver.find_element_by_id("user_login").send_keys("admin")
    self.driver.find_element_by_id("user_pass").send_keys("admin123")
    self.driver.find_element_by_id("wp-submit").click()

    source = self.driver.page_source
    result = "Dashboard" in source
    self.assertTrue(result)

    def test_login_no_password(self):
    self.driver.find_element_by_id("user_login").send_keys("admin")
    self.driver.find_element_by_id("wp-submit").click()

    source = self.driver.page_source
    result = "The password field is empty." in source
    self.assertTrue(result)

    def test_login_no_user(self):
    self.driver.find_element_by_id("user_pass").send_keys("admin123")
    self.driver.find_element_by_id("wp-submit").click()

    source = self.driver.page_source
    result = "The username field is empty." in source
    self.assertTrue(result)

    def test_login_wrong_password(self):
    self.driver.find_element_by_id("user_login").send_keys("admin")
    self.driver.find_element_by_id("user_pass").send_keys("admin456")
    self.driver.find_element_by_id("wp-submit").click()

    source = self.driver.page_source
    result = "The password you entered for the username <strong>admin</strong> is incorrect." in source
self.assertTrue(result)


if __name__ == "__main__":
    unittest.main()
```

Excluding login processing as a common function makes the code simpler.

```
from selenium import webdriver
import unittest

class LoginTest(unittest.TestCase):

    def setUp(self):
    self.driver = webdriver.Firefox()
    self.driver.get("http://localhost/wordpress/wp-login.php")

    def tearDown(self):
    self.driver.quit()

    def test_login_ok(self):
    self.login("admin", "admin123")

    source = self.driver.page_source
    result = "Dashboard" in source
    self.assertTrue(result)

    def test_login_no_password(self):
    self.login("admin", "")

    source = self.driver.page_source
    result = "The password field is empty." in source
    self.assertTrue(result)

    def test_login_no_user(self):
    self.login("", "admin123")

    source = self.driver.page_source
    result = "The username field is empty." in source
    self.assertTrue(result)

    def test_login_wrong_password(self):
    self.login("admin", "admin456")

    source = self.driver.page_source
    result = "The password you entered for the username <strong>admin</strong> is incorrect." in source self.assertTrue(result)
```

```
    def login(self, user, password):
    self.driver.find_element_by_id("user_login").send_keys(user)
self.driver.find_element_by_id("user_pass").send_keys(password)
self.driver.find_element_by_id("wp-submit").click()


if __name__ == "__main__":
    unittest.main()
```

Let’s add the process to take screenshots so that we can check the browser screen for each test.

```
from selenium import webdriver
import unittest

class LoginTest(unittest.TestCase):

    def setUp(self):
    self.driver = webdriver.Firefox()
    self.driver.get("http://localhost/wordpress/wp-login.php")

    def tearDown(self):
    self.driver.quit()

    def test_login_ok(self):
    self.login("admin", "admin123")

    source = self.driver.page_source
    result = "Dashboard" in source
    self.assertTrue(result)

    self.driver.save_screenshot("/home/kenpapa/test_login_ok.png")
    def test_login_no_password(self):
    self.login("admin", "")

    source = self.driver.page_source
    result = "The password field is empty." in source
    self.assertTrue(result)
```

```
    self.driver.save_screenshot("/home/kenpapa/test_login_no_password.png
    def test_login_no_user(self):
    self.login("", "admin123")

    source = self.driver.page_source
    result = "The username field is empty." in source
    self.assertTrue(result)

    self.driver.save_screenshot("/home/kenpapa/test_login_no_user.png")
    def test_login_wrong_password(self):
    self.login("admin", "admin456")

    source = self.driver.page_source
    result = "The password you entered for the username <strong>admin</strong
incorrect." in source self.assertTrue(result)

    self.driver.save_screenshot("/home/kenpapa/test_login_wrong_password.
    def login(self, user, password):
    self.driver.find_element_by_id("user_login").send_keys(user)
    self.driver.find_element_by_id("user_pass").send_keys(password)
    self.driver.find_element_by_id("wp-submit").click()


if __name__ == "__main__":
    unittest.main()
```

In this chapter, we executed automated testing of web application using Selenium WebDriver and Test Framework.

Chapter 6 Some notes on the use of Selenium WebDriver

Since Selenium WebDriver supports many programming languages and many browsers, we can use Selenium in our favorite environment but there are some notes.

In this chapter I mention some notes on the use of Selenium WebDriver based on my experience of Selenium.

6-1 Changes of selenium related softwares

Selenium related softwares (browser, browser driver, Selenium Client, Selenium Server) are updating every day. So, if you continue to use Selenium WebDriver, you may experience that the code which worked correctly in the past doesn't work properly in the new environment.

I introduce some of the problems which I experienced in the past for Selenium WebDriver.

File Upload didn't work on Firefox 55.
Local HTML file could not be operated on Firefox 55.
PHP WebDriver didn't work with latest Selenium Server.
The code of Node.js which worked in the Selenium 3.6.0 environment didn't work in the Selenium 4.0.0-alpha.1 environment.

As for 1, the problem had been discussed at the following site.
File upload no longer works with geckodriver 0.18.0 and Firefox 55
https://github.com/mozilla/geckodriver/issues/858

As for 2, the problem had been discussed at the following site.
Navigating to a local html file crashes the geckodriver with Firefox 55
https://github.com/mozilla/geckodriver/issues/861

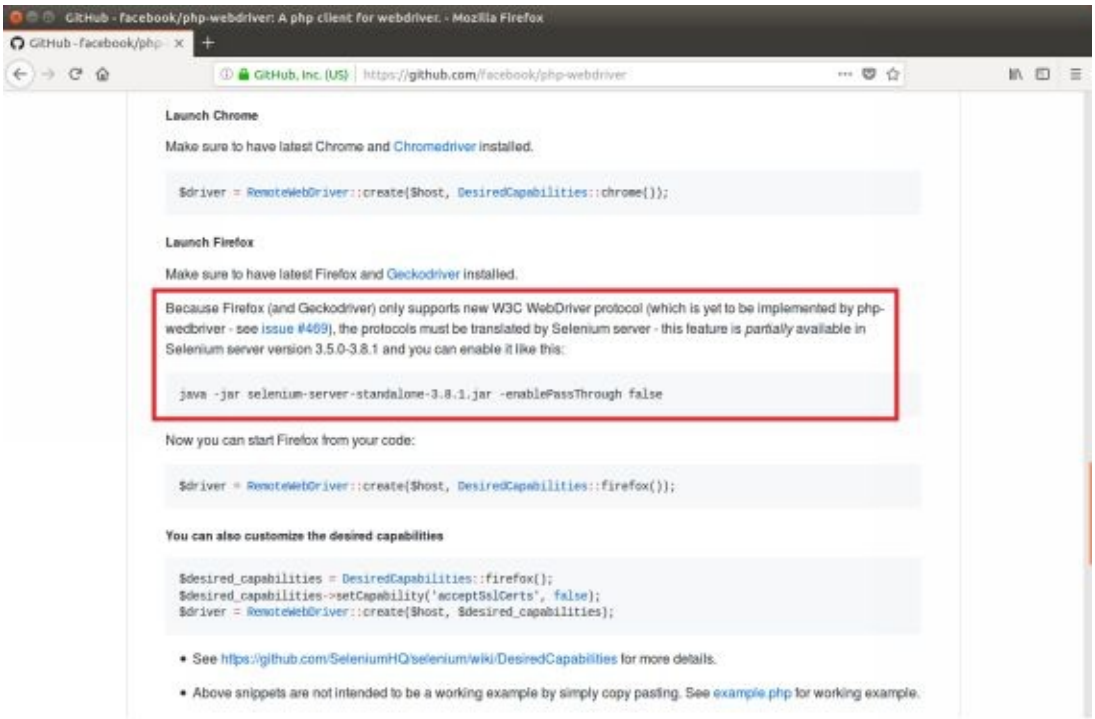
Problems 1 and 2 were solved in Firefox 56.

As for 3, the problem has been discussed at the following site.
W3C WebDriver protocol support #469
https://github.com/facebook/php-webdriver/issues/469

This problem has not been completely solved yet.

If Selenium Server version is between 3.5.0 and 3.8.1, we can tentatively solve this problem using "-enablePassThrough false" option when starting Selenium Server.

This problem is also mentioned on the official website of PHP WebDriver.



GitHub - facebook/php-webdriver: A php client for webdriver. - Mozilla Firefox

GitHub - facebook/php

GitHub, Inc. (US) https://github.com/facebook/php-webdriver

Launch Chrome

Make sure to have latest Chrome and Chromedriver installed.

```
$driver = RemoteWebDriver::create($host, DesiredCapabilities::chrome());
```

Launch Firefox

Make sure to have latest Firefox and Geckodriver installed.

Because Firefox (and Geckodriver) only supports new W3C WebDriver protocol (which is yet to be implemented by php-webdriver - see issue #469), the protocols must be translated by Selenium server - this feature is partially available in Selenium server version 3.5.0-3.8.1 and you can enable it like this:

```
java -jar selenium-server-standalone-3.8.1.jar -enablePassThrough false
```

Now you can start Firefox from your code:

```
$driver = RemoteWebDriver::create($host, DesiredCapabilities::firefox());
```

You can also customize the desired capabilities

```
$desired_capabilities = DesiredCapabilities::firefox();
$desired_capabilities->setCapability('acceptSslCerts', false);
$driver = RemoteWebDriver::create($host, $desired_capabilities);
```

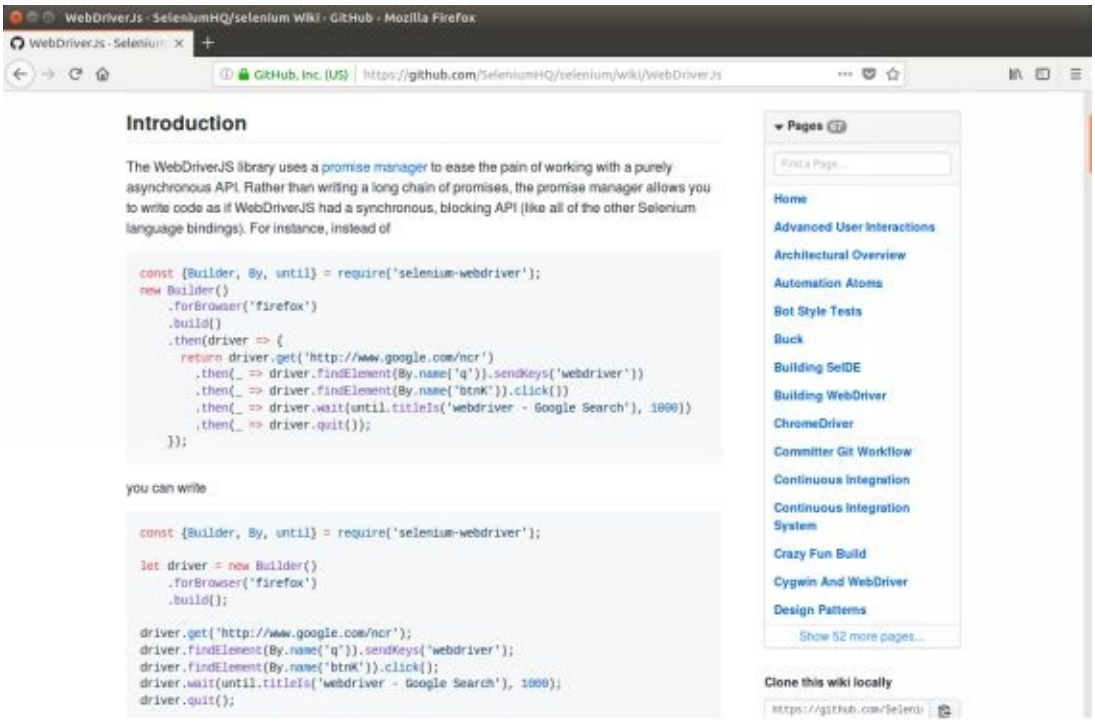
- See https://github.com/SeleniumHQ/selenium/wiki/DesiredCapabilities for more details.
- Above snippets are not intended to be a working example by simply copy pasting. See example.php for working example.

As for 4, Selenium WebDriver 4.0.0-alpha.1 of Node.js has a lot of changes from 3.6.0. (Please see below site for details) *https://github.com/SeleniumHQ/selenium/blob/master/javascript/node/selenium-webdriver/CHANGES.md*

One of the major changes is "**The core WebDriver API no longer uses promise manager**".

The promise manager allows us to write code as if WebDriverJS had a synchronous, blocking API (like all of the other Selenium language bindings).

As for the promise manager, it is described in detail in the following site.

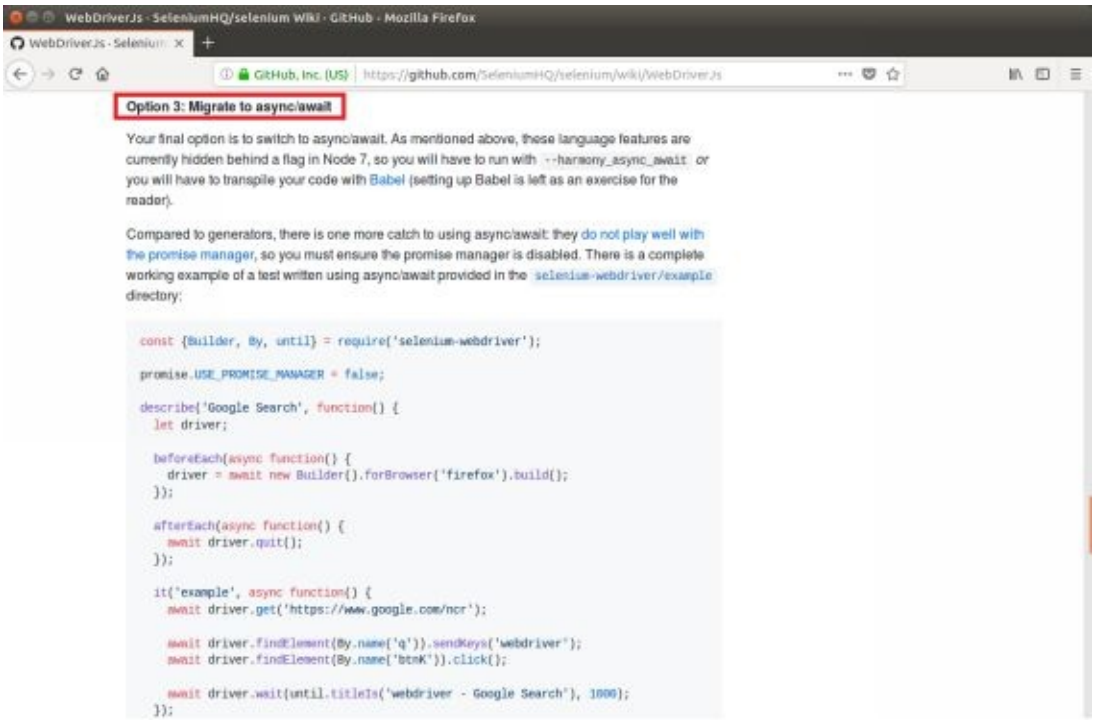


https://github.com/SeleniumHQ/selenium/wiki/WebDriverJs

There is no promise manager on 4.0.0-alpha.1, so it is necessary to add wait processing to the code of automatic operation. The following three options are introduced on the site shown above.

Option 1: Use classic promise chaining
Option 2: Migrate to Generators
Option 3: Migrate to async/await

Option 3 is used in this book.



WebDriverJs · SeleniumHQ/selenium Wiki · GitHub · Mozilla Firefox

Option 3: Migrate to async/await

Your final option is to switch to async/await. As mentioned above, these language features are currently hidden behind a flag in Node 7, so you will have to run with `--harmony_async_await` or you will have to transpile your code with Babel (setting up Babel is left as an exercise for the reader).

Compared to generators, there is one more catch to using async/await: they do not play well with the promise manager, so you must ensure the promise manager is disabled. There is a complete working example of a test written using async/await provided in the `selenium-webdriver/example` directory:

```
const {Builder, By, until} = require('selenium-webdriver');

promise.USE_PROMISE_MANAGER = false;

describe('Google Search', function() {
  let driver;

  beforeEach(async function() {
    driver = await new Builder().forBrowser('firefox').build();
  });

  afterEach(async function() {
    await driver.quit();
  });

  it('example', async function() {
    await driver.get('https://www.google.com/ncr');

    await driver.findElement(By.name('q')).sendKeys('webdriver');
    await driver.findElement(By.name('btnK')).click();

    await driver.wait(until.titleIs('webdriver - Google Search'), 1000);
  });
```

Code example for Selenium 3.6.0

```
var webdriver = require("selenium-webdriver");
var By = webdriver.By;

var driver = new webdriver.Builder().forBrowser("chrome").build();

driver.get("http://www.google.com");
element = driver.findElement(By.id("lst-ib"));
element.sendKeys("selenium");
element.submit();
```

Code example for Selenium 4.0.0-alpha.1

```
var webdriver = require("selenium-webdriver");
var By = webdriver.By;

var driver = new webdriver.Builder().forBrowser("chrome").build();

async function main() {
await driver.get("http://www.google.com"); element = await driver.findElement(By.id("lst-ib")); await element.sendKeys("selenium");
```

```
await element.submit();
}

main();
```

If you encounter any problem related to Selenium, it would be better to check the information on the internet. It is possible that some of the selenium users in the world have the same problem and discussing the problem.

6-2 Behavior difference between browsers

One of the advantages of Selenium WebDriver is
Support a lot of browsers such as Chrome, IE, Edge, Firefox, Opera, Safari

Therefore, when executing automated testing, it is possible to check operation with multiple browsers using Selenium WebDriver.

Ideally it would be better to use the same code with multiple browsers but the behaviors between browsers are actually different, so the code often needs to be modified due to the behavior difference between browsers.

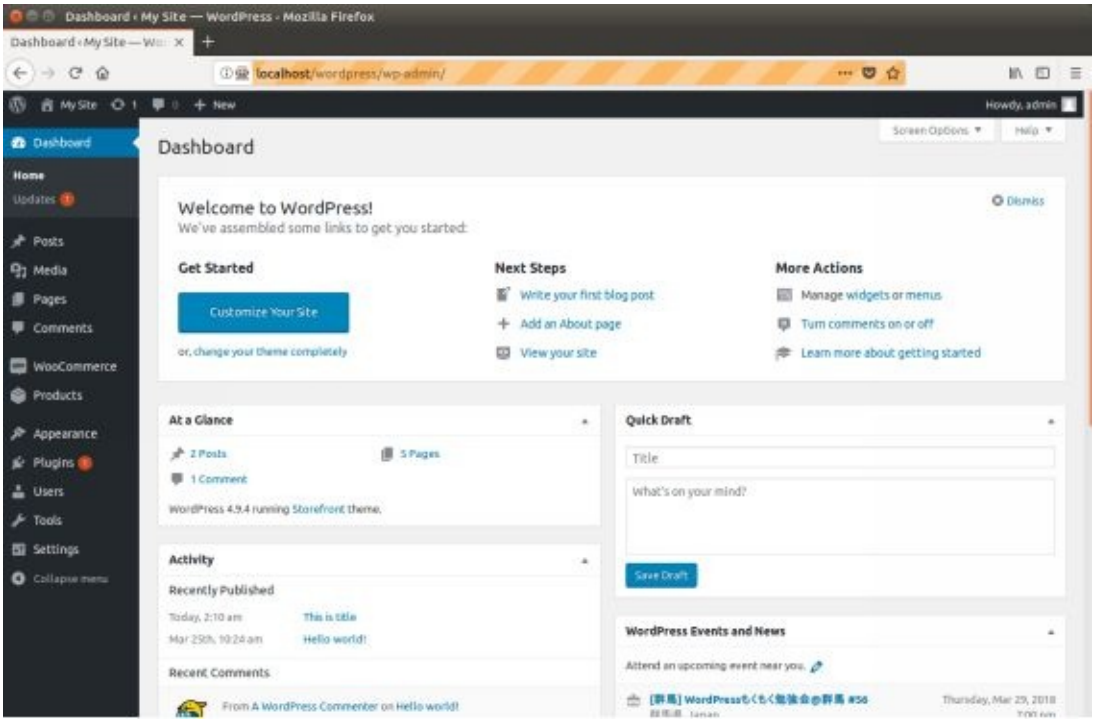
Actually, in Chapter 4, there was a slight difference in the behavior of Firefox and Chrome, so we had to change the code.

Let's recall the difference again.

6-2-1 Difference in default window size

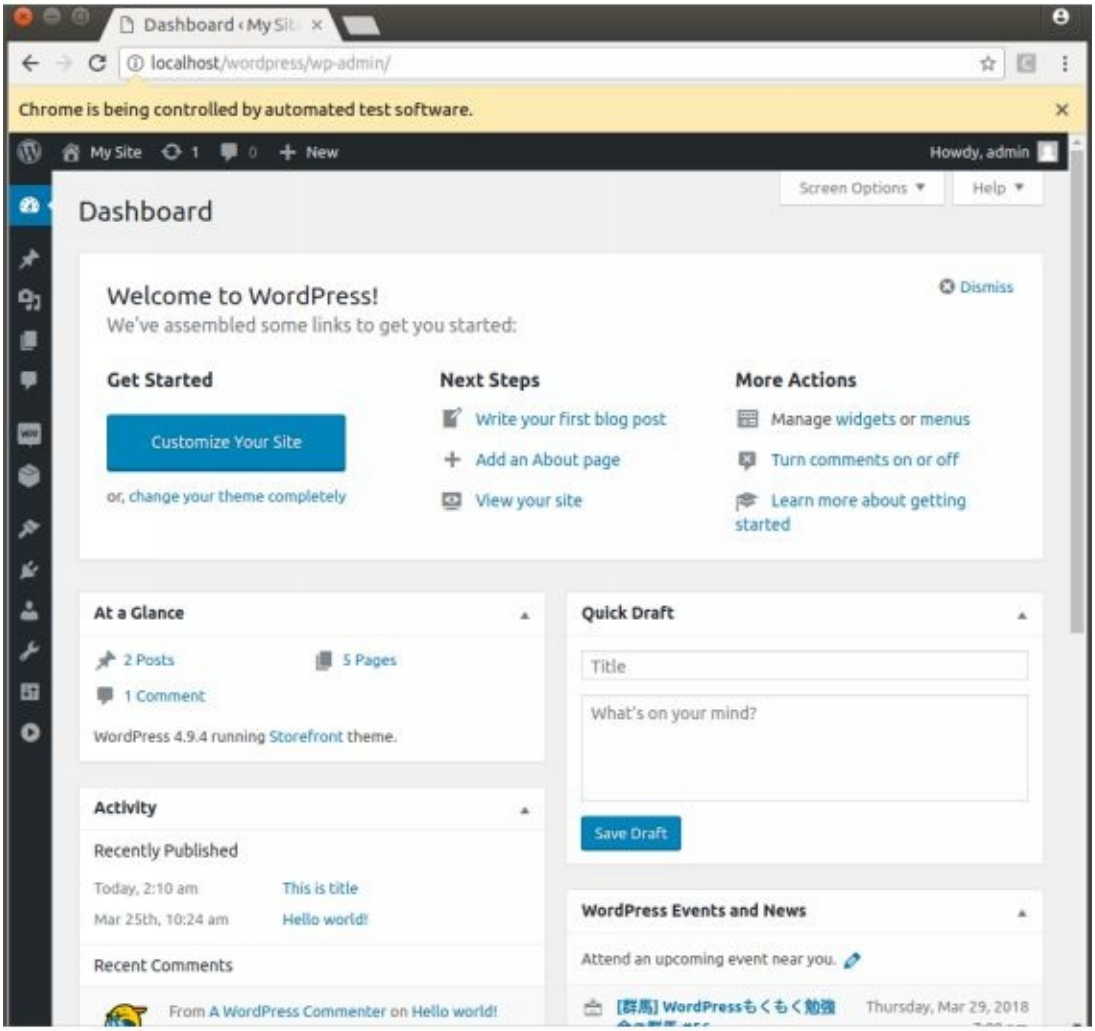
The default window size of Chrome is small and the title of the menu is hidden, so it is necessary to enlarge the window size to select the menu.

Default window size of Firefox



The text of the menu on the left is displayed.

Default window size of Chrome



The text of the menu on the left is not displayed.

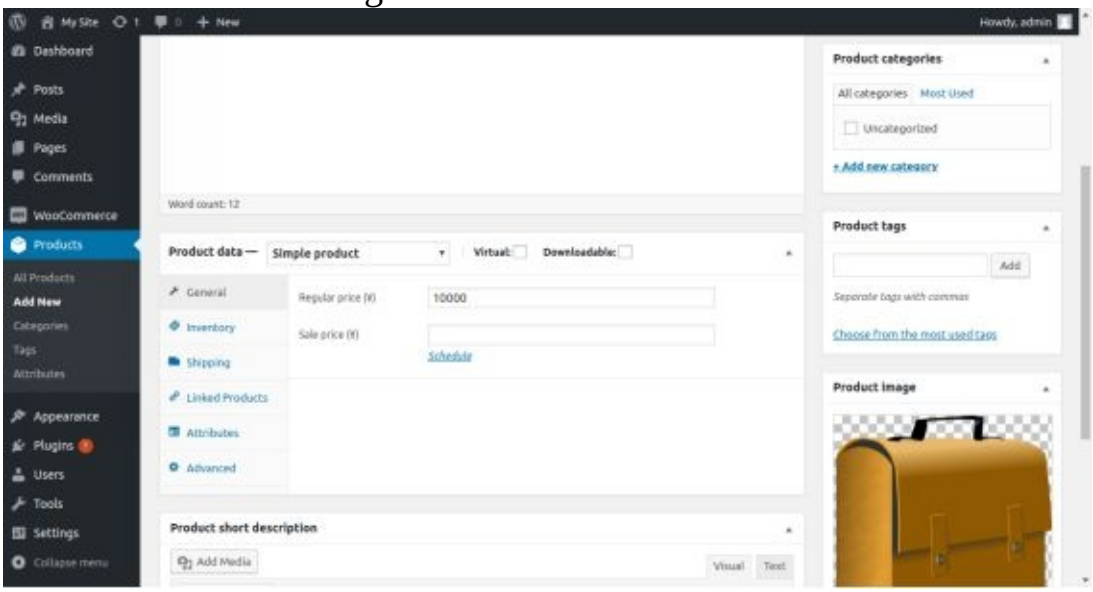
Therefore, the following code needs to be added in the code for Chrome.

```
driver.set_window_size(1280, 800)
```

6-2-2 Difference in scroll processing

In the case of Chrome, if the target element is not displayed on the screen, it is necessary to scroll using Javascript and operate the element after it is displayed on the screen.

The state before clicking the "Publish" button

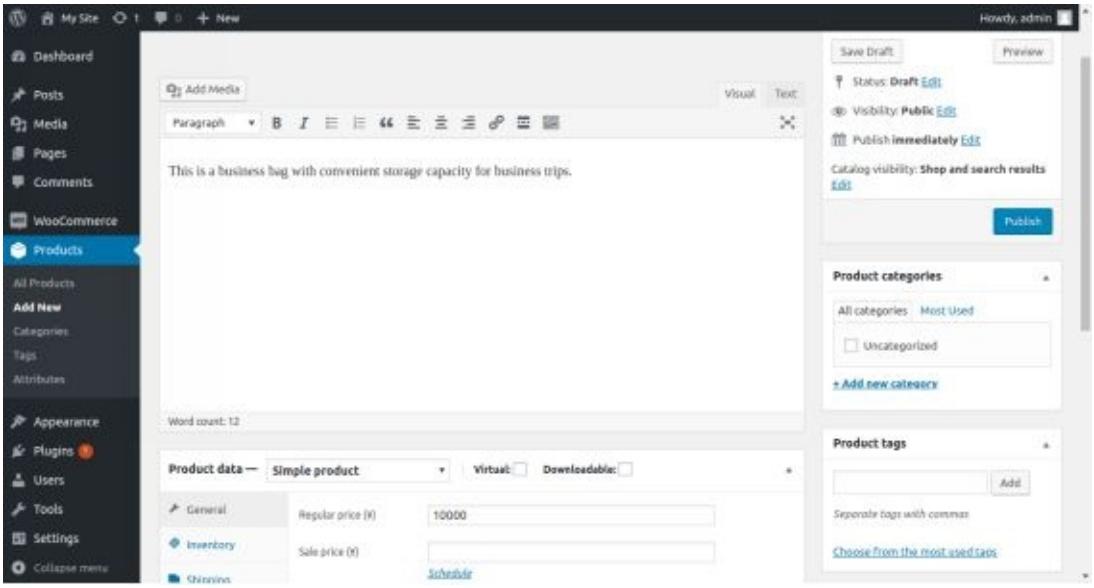


"Publish" button is not displayed.

Therefore, the following code needs to be added in the code for Chrome.

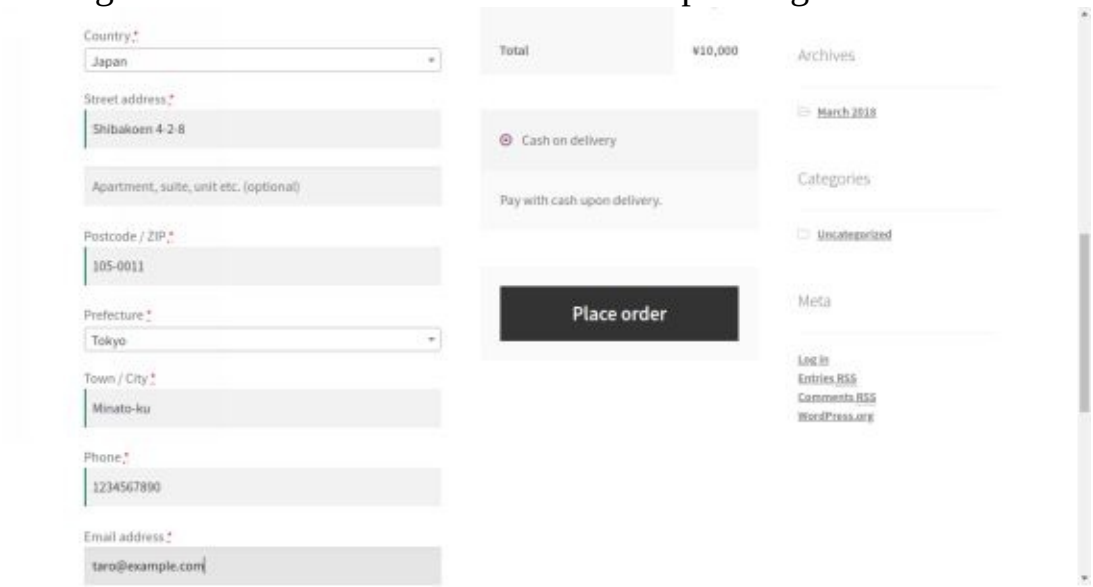
```
driver.execute_script('window.scrollTo(0,100);')
```

As a result, the "Publish" button is displayed so Selenium can click it.


My Site
New
Howdy, admin
Dashboard
Posts
Media
Pages
Comments
WooCommerce
Products
All Products
Add New
Categories
Tags
Attributes
Appearance
Plugins
Users
Tools
Settings
Collapse menu
Add Media
Visual
Text
Paragraph
This is a business bag with convenient storage capacity for business trips.
Word count: 12
Product data —
Simple product
Virtual:
Downloadable:
General
Inventory
Shipping
Regular price (₹)
10000
Sale price (₹)
Schedule
Save Draft
Preview
Status: Draft Edit
Visibility: Public Edit
Publish immediately Edit
Catalog visibility: Shop and search results Edit
Publish
Product categories
All categories
Most Used
Uncategorized
+ Add new category
Product tags
Add
Separate tags with commas
Choose from the most used tags

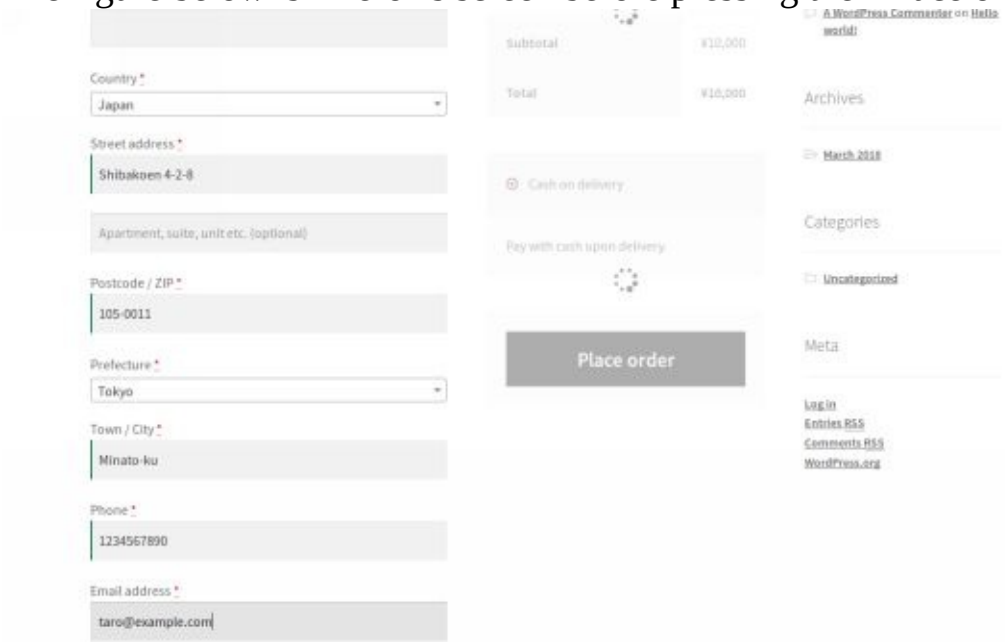
6-2-3 Processing timing difference

The figure below is Chrome's screen before pressing the "Place order" button.



"Place order" button can be pressed.

The figure below is Firefox's screen before pressing the "Place order" button.



"Place order" button can’t be pressed because the creation hasn't finished yet.

Therefore, the following code needs to be added in the code for Firefox.

```
time.sleep(2)
```

6-3 Implementation differences between programming languages

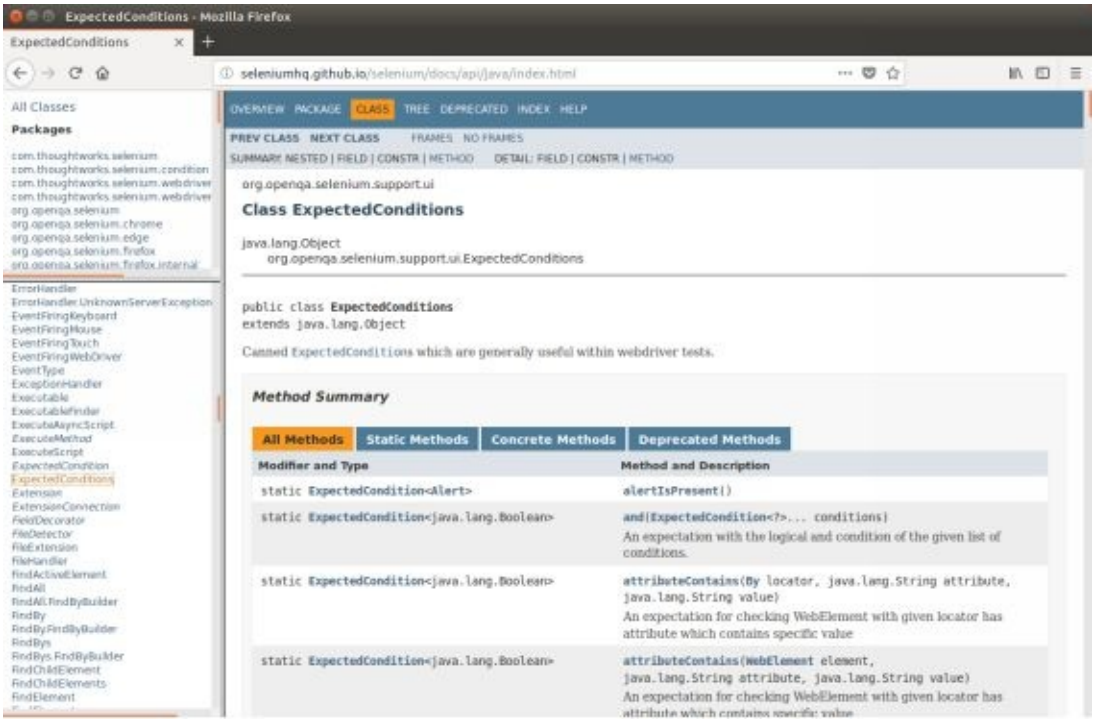
One of the advantages of Selenium WebDriver is
Support a lot of programming languages such as Java, C#, Ruby, Python, JavaScript(Node), PHP

Therefore, when executing automated testing of web application, we can write the code in our favorite language. However, in fact, there are some differences between programming languages as below.

6-3-1 ExpectedCondition

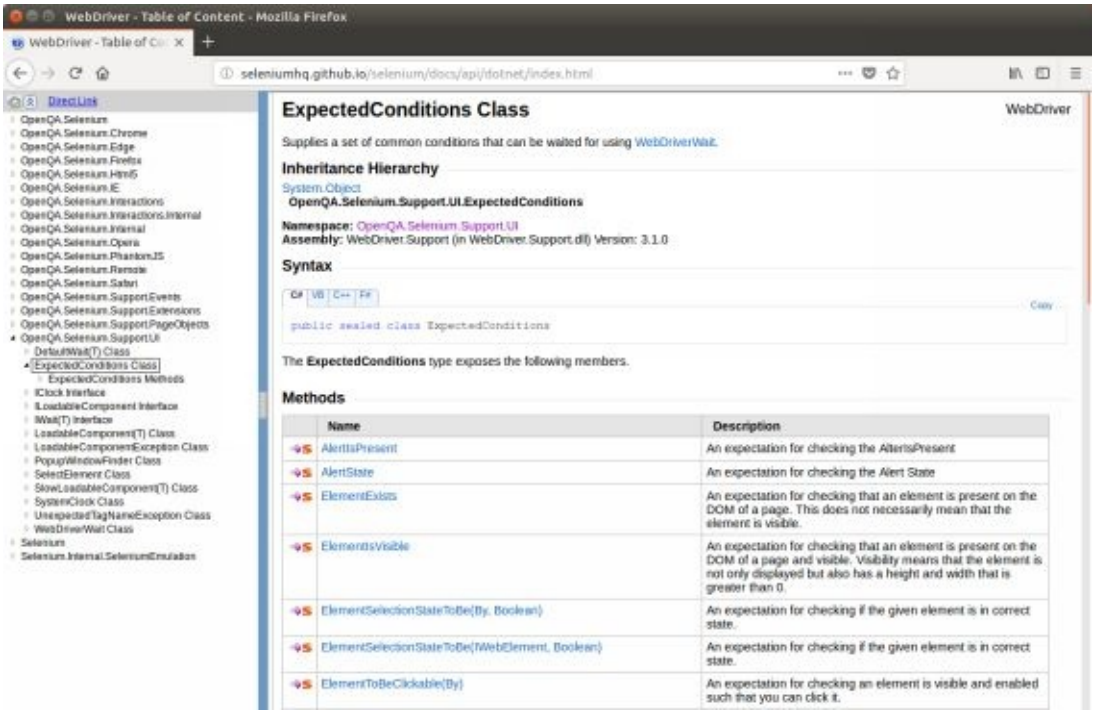
ExpectedCondition is not implemented in Ruby's Selenium WebDriver.

Java



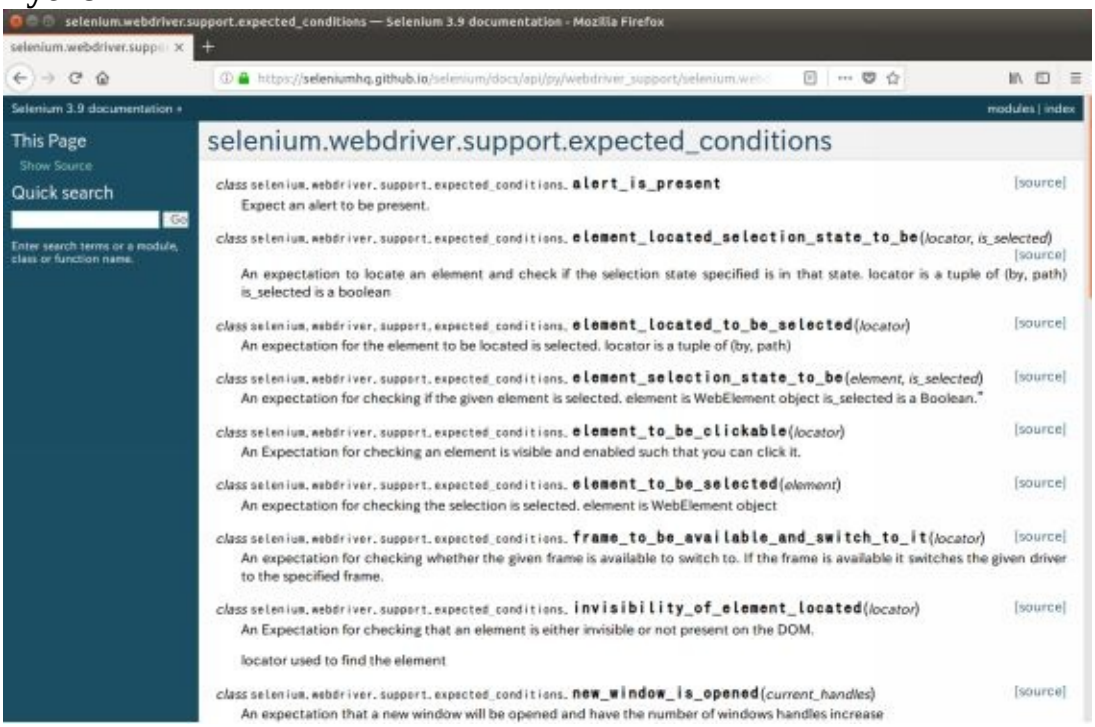
http://seleniumhq.github.io/selenium/docs/api/java/index.html

C#



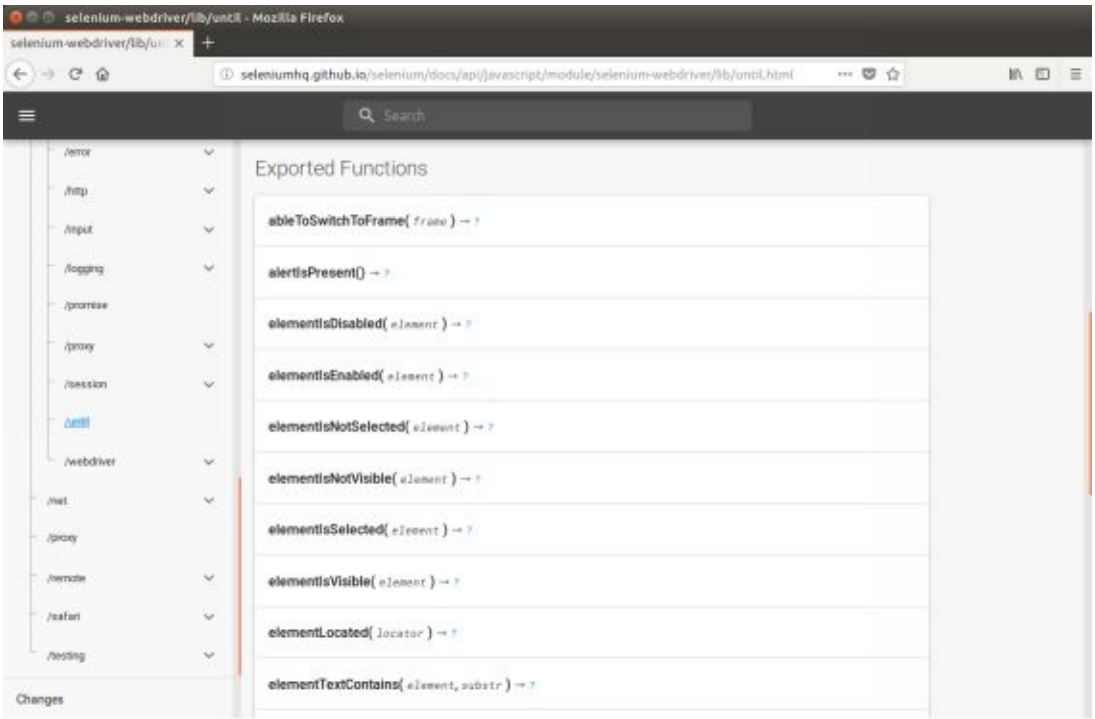
http://seleniumhq.github.io/selenium/docs/api/dotnet/index.html

Python



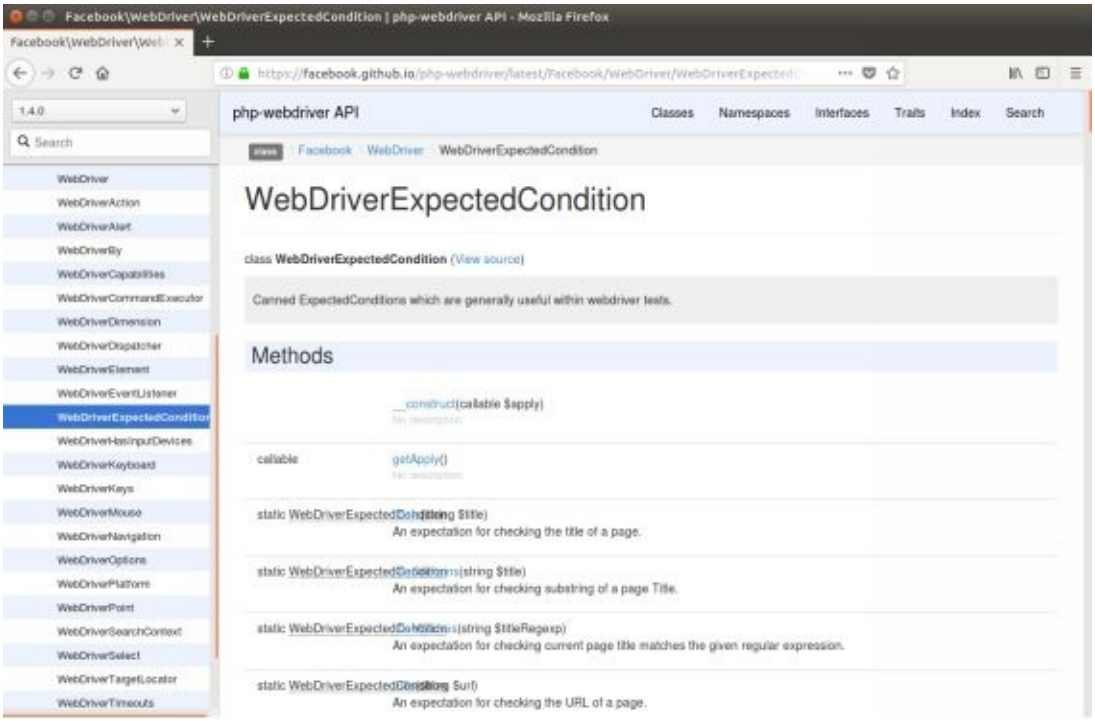
https://seleniumhq.github.io/selenium/docs/api/py/api.html

Node.js



http://seleniumhq.github.io/selenium/docs/api/javascript/index.html

PHP



https://facebook.github.io/php-webdriver/latest/

For example, the following 6 codes describe the same operation in different

programming languages. 5 codes except Ruby are implemented with ExpectedCondition, but the code in Ruby is implemented without ExpectedCondition.

GoogleSearchWait.java

```
import org.openqa.selenium.By;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.WebElement;
import org.openqa.selenium.firefox.FirefoxDriver;
import org.openqa.selenium.support.ui.ExpectedConditions;
import org.openqa.selenium.support.ui.WebDriverWait;

public class GoogleSearchWait {

   public static void main(String[] args) {
   WebDriver driver = new FirefoxDriver();

   driver.get("http://www.google.com");
   WebElement element = driver.findElement(By.id("lst-ib"));
   element.sendKeys("selenium");
   element.submit();

   WebDriverWait wait = new WebDriverWait(driver, 10);
   wait.until(ExpectedConditions.titleContains("selenium"));
System.out.println(driver.getTitle());

   driver.quit();
   }

}
```

GoogleSearchWait.cs

```
using System;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium;
using OpenQA.Selenium.Support.UI;
using SeleniumExtras.WaitHelpers;
```

```
public class GoogleSearchWait
{
    public static void Main(string[] args)
    {
    IWebDriver driver = new FirefoxDriver();

    driver.Navigate().GoToUrl("http://www.google.com");
    IWebElement element = driver.FindElement (By.Id("lst-ib"));
    element.SendKeys("selenium");
    element.Submit();

    WebDriverWait wait = new WebDriverWait(driver, TimeSpan.FromSeconds(
    wait.Until(SeleniumExtras.WaitHelpers.ExpectedConditions.TitleContains(
Console.WriteLine(driver.Title);

    driver.Quit();
    }
}
```

google_search_wait.py

```
from selenium import webdriver
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

driver = webdriver.Firefox()

driver.get("http://www.google.com")
element = driver.find_element_by_id("lst-ib")
element.send_keys("selenium")
element.submit()

wait = WebDriverWait(driver, 10)
wait.until(EC.title_contains("selenium")) print driver.title

driver.quit()
```

google_search_wait.js

```
var webdriver = require("selenium-webdriver");
```

```
var By = webdriver.By;
var until = webdriver.until;

var driver = new webdriver.Builder().forBrowser("firefox").build();

async function main() {
await driver.get("http://www.google.com");
await driver.findElement(By.id("lst-ib")).sendKeys("selenium");
await driver.findElement(By.name("btnK")).click();

await driver.wait(until.titleContains("selenium"), 10000); var title =
await driver.getTitle();
console.log(title);

await driver.quit();
}

main();
```

google_search_wait.php

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use Facebook¥WebDriver¥WebDriverBy;
use Facebook¥WebDriver¥Remote¥RemoteWebDriver;
use Facebook¥WebDriver¥Remote¥DesiredCapabilities;
use Facebook¥WebDriver¥WebDriverExpectedCondition;

$host = 'http://localhost:4444/wd/hub';
$driver = RemoteWebDriver::create($host,
DesiredCapabilities::firefox());

$driver->get("http://www.google.com");
$element = $driver->findElement(WebDriverBy::id("lst-ib"));
$element->sendKeys("selenium");
$element->submit();

$driver->wait(10)-
```

```
>until(WebDriverExpectedCondition::titleContains('selenium'));
print $driver->getTitle();
print "¥n";

$driver->quit();
```

google_search_wait.rb

```
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :firefox

driver.get("http://www.google.com")
element = driver.find_element(:id => "lst-ib")
element.send_keys("selenium")
element.submit

wait = Selenium::WebDriver::Wait.new(timeout: 10)
wait.until { driver.title.start_with? "selenium" }
puts driver.title

driver.quit
```

6-3-2 Node.js is asynchronous

One of the features of Node.js is "asynchronous". Therefore, even if the code is written in the same order as other programming languages, the behavior may be different.

For example, the results of the following 6 codes are as follows. Only Node.js shows different result.

(Java, C#, Ruby, Python, PHP)
Google
hello

(Node.js)
hello
Google

Java

```
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class GoogleTitle {

   public static void main(String[] args) {
   WebDriver driver = new FirefoxDriver();

   driver.get("http://www.google.com");
   System.out.println(driver.getTitle());
   driver.quit();

   System.out.println("hello");
   }

}
```

C#

```
using System;
using OpenQA.Selenium.Firefox;
```

```
using OpenQA.Selenium;

public class GoogleTitle
{
   public static void Main(string[] args)
   {
   IWebDriver driver = new FirefoxDriver();

   driver.Navigate().GoToUrl("http://www.google.com");
   Console.WriteLine(driver.Title);
   driver.Quit();

   Console.WriteLine("hello");
   }
}
```

Ruby

```
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :firefox

driver.get("http://www.google.com")
puts driver.title
driver.quit

puts "hello"
```

Python

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://www.google.com")
print driver.title
driver.quit()

print "hello"
```

PHP

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use Facebook¥WebDriver¥WebDriverBy;
use Facebook¥WebDriver¥Remote¥RemoteWebDriver;
use Facebook¥WebDriver¥Remote¥DesiredCapabilities;

$host = 'http://localhost:4444/wd/hub';
$driver = RemoteWebDriver::create($host,
DesiredCapabilities::firefox());

$driver->get("http://www.google.com");
print $driver->getTitle();
print "¥n";
$driver->quit();

print "hello¥n";
```

Node.js

```
var webdriver = require("selenium-webdriver");

var driver = new webdriver.Builder().forBrowser("firefox").build();

async function main() {
await driver.get("http://www.google.com");

var title = await driver.getTitle();
console.log(title);

await driver.quit();
}

main();

console.log("hello");
```

6-3-3 How to take a screenshot

Screenshot can be taken with Selenium WebDriver, but the ease of implementation depends on the programming language.

The easiest: Ruby, Python, PHP
Easy: C#, Node.js
A little difficult: Java

Ruby

```
require 'selenium-webdriver'

driver = Selenium::WebDriver.for :firefox

driver.get("http://www.google.com")
driver.save_screenshot("/home/kenpapa/screenshotdemo.png")
driver.quit
```

Python

```
from selenium import webdriver

driver = webdriver.Firefox()

driver.get("http://www.google.com")
driver.save_screenshot("/home/kenpapa/screenshotdemo.png")
driver.quit()
```

PHP

```
<?php
require_once __DIR__ . '/vendor/autoload.php';

use Facebook¥WebDriver¥WebDriverBy;
use Facebook¥WebDriver¥Remote¥RemoteWebDriver;
use Facebook¥WebDriver¥Remote¥DesiredCapabilities;

$host = 'http://localhost:4444/wd/hub';
$driver = RemoteWebDriver::create($host,
DesiredCapabilities::firefox());
```

```
$driver->get("http://www.google.com");
$driver->takeScreenshot("/home/kenpapa/screenshotdemo.png");
$driver->quit();
```

C#

```
using System;
using OpenQA.Selenium.Firefox;
using OpenQA.Selenium;

public class ScreenShotDemo
{
   public static void Main(string[] args)
   {
   IWebDriver driver = new FirefoxDriver();

   driver.Navigate().GoToUrl("http://www.google.com");
   Screenshot ss = ((ITakesScreenshot)driver).GetScreenshot();
   ss.SaveAsFile("/home/kenpapa/screenshotdemo.png",
ScreenshotImageFormat.Png); driver.Quit();
   }
}
```

Node.js

```
var webdriver = require("selenium-webdriver");
var fs = require('fs');

var driver = new webdriver.Builder().forBrowser("firefox").build();

async function main() {
await driver.get("http://www.google.com");

var data = await driver.takeScreenshot();
await   fs.writeFile('/home/kenpapa/screenshotdemo.png',   data,
'base64', (err) => {
   if (err) throw err;
});
```

```
await driver.quit();
}

main();
```

Java

```
import java.io.File;
import java.io.IOException;

import org.apache.commons.io.FileUtils;
import org.openqa.selenium.OutputType;
import org.openqa.selenium.TakesScreenshot;
import org.openqa.selenium.WebDriver;
import org.openqa.selenium.firefox.FirefoxDriver;

public class ScreenShotDemo {

   public static void main(String[] args) {
   WebDriver driver = new FirefoxDriver();

   driver.get("http://www.google.com");

   File scrFile =
((TakesScreenshot)driver).getScreenshotAs(OutputType.FILE);
try {
   FileUtils.copyFile(scrFile, new
File("/home/kenpapa/screenshotdemo.png"));
Thread.sleep(2000);
   } catch (IOException e) {
   e.printStackTrace();
   } catch (InterruptedException e) {
   e.printStackTrace();
   }

   driver.quit();
   }
```

```
}
```

In the case of Java, it is necessary to prepare an extension library called Commons IO to specify the file storage location. Also, when quit processing is done after getting the screenshot, the screenshot was not created successfully, so "Thread.sleep (2000);" is inserted in the above code.

Appendix

Appendix A: About VirtualBox

The official website of VirtualBox is below. The VirtualBox installer is available from this site.

VirtualBox official website
https://www.virtualbox.org/

When using VirtualBox for the first time, the following sites will be useful.

An overview of VirtualBox is explained in detail in the following site.

Chapter 1. First steps
https://www.virtualbox.org/manual/ch01.html

The procedure for installing VirtualBox on Windows 10 is described in detail in the following site, for example.

INSTALLING VIRTUALBOX ON WINDOWS 10
https://websiteforstudents.com/installing-virtualbox-windows-10/

Appendix B: About Ubuntu

The official website of Ubuntu is below.

Ubuntu official website
https://www.ubuntu.com/

The iso image of Ubuntu can be obtained from the following URL of the official website.
(In this book, iso image of Ubuntu 16.04.4 LTS is used)

Download Ubuntu Desktop
https://www.ubuntu.com/download/desktop

When using Ubuntu for the first time, the following sites will be useful.

An overview of Ubuntu is explained in detail in the following site.

Ubuntu: A Beginner's Guide
https://www.makeuseof.com/tag/ubuntu-an-absolute-beginners-guide/

The procedure for installing Ubuntu in VirtualBox is described in detail in the following sites, for example.

Install Ubuntu on Oracle VirtualBox
https://linus.nci.nih.gov/bdge/installUbuntu.html

Install Ubuntu-16.04 LTS on Virtual Box (Desktop version)
https://medium.com/@tushar0618/install-ubuntu-16-04-lts-on-virtual-box-desktop-version-30dc6f1958d0