Selenium в Python

[Введение](#_Toc13867306)

[Selenium](#_Toc13867307)

[История создания](#_Toc13867308)

[Домашнее задание](#_Toc13867309)

[Используемая литература](#_Toc13867310)

# Введение

Рассмотрим одно из самых популярных средств автоматизации – Selenium.

Этот проект популярен, так как является условно бесплатным. Selenium предоставляет программные продукты, которые можно свободно скачать и установить. Но для оптимизации под задачи пользователя их необходимо дорабатывать, а это влечет затраты на оплату работы программистов.



# Selenium

Selenium – инструмент для автоматизации действий веб-браузера. Чаще всего используется для тестирования. Официальная страница проекта – http://docs.seleniumhq.org/

## История создания

В 2004 году разработчик Джейсон Хаггинс создал на языке JavaScript библиотеку JavaScriptTestRunner. Она служит для запуска тестов в браузере. Сегодня она известна как Selenium Core».

Хаггинс работал в компании ThoughtWorks в Стамбуле. Его проект заинтересовал коллег, и они активно включились в работу. Название «Selenium» закрепилось после того, как Джейсон Хаггинс с иронией высказался о конкурирующем проекте «Mercury Interactive QuickTest Professional». Так как «mercury» в переводе с английского означает «ртуть», разработчик нового ПО пошутил о том, что можно отравление ртутью лечат приемом селениума.

Код проекта открыли в конце 2014 года.

Программные продукты Selenium

В рамках которого разработана серия программных продуктов с открытым исходным кодом (open source):

* Selenium WebDriver;
* Selenium RC;
* Selenium Server;
* Selenium Grid;
* Selenium IDE.

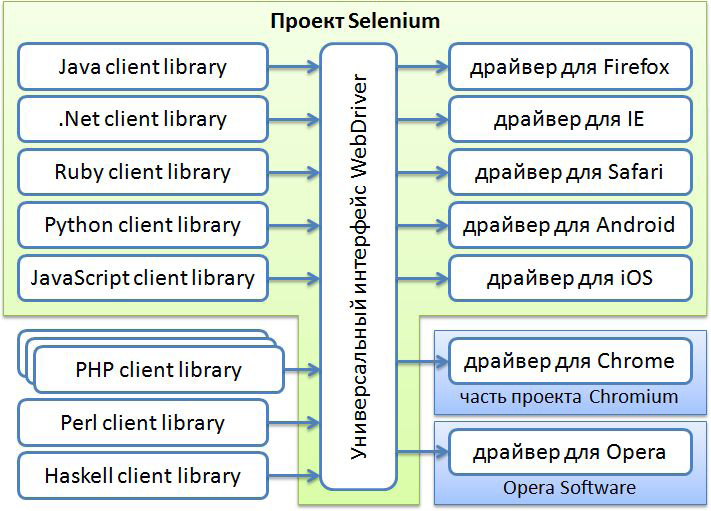
Рассмотрим каждый из них.

Selenium WebDriver

Selenium WebDriver (Selenium 2) – это программная библиотека для управления браузерами, основной продукт в рамках проекта Silenium. Она включает набор модулей для разработки ПО.

Selenium WebDriver состоит из набора драйверов и клиентских библиотек для таких браузеров, как Firefox, Internet Explorer и Safari, а также мобильных приложений, работающих с операционными системами Android и iOS.

Драйвер для Google Chrome разрабатывается в проекте Chromium, а для Opera (включая мобильные версии) разработку ведет компания Opera Software.



Сфера применения

Чаще всего Selenium WebDriver используется для тестирования функционала веб-сайтов/веб-ориентированных приложений. Автоматизированное тестирование удобно, потому что позволяет многократно запускать повторяющиеся тесты. Регрессионное тестирование, то есть, проверка, что старый код не перестал работать правильно после внесения новых изменений, является типичным примером, когда необходима автоматизация. WebDriver предоставляет все необходимые методы, обеспечивает высокую скорость теста и гарантирует корректность проверки (поскольку человеский фактор исключен). В официальной документации Selenium приводятся следующие плюсы автоматизированного тестирования веб-приложений:

* возможность проводить чаще регрессионное тестирование;
* быстрое предоставление разработчикам отчета о состоянии продукта;
* получение потенциально бесконечного числа прогонов тестов;
* обеспечение поддержки Agile и экстремальным методам разработки;
* сохранение строгой документации тестов;
* обнаружение ошибок, которые были пропущены на стадии ручного тестирования.

Функционал WebDriver позволяет использовать его не только для тестирования, но и для администрирования веб-сервисов, сократив до возможного предела количество действий, производимых вручную. Selenium WebDriver становится незаменимым помощником в случаях, когда, к примеру, ядро сайта устарело и требует от модераторов большого количества телодвижений для реализации маленький фич (например, загрузки галереи фото).

Также одной из незаменимых особенностей Selenium WebDriver является ожидание загрузки страницы. Сюда можно отнести случаи, когда парсинг данных на странице невозможен из-за страниц перенаправления или ожидания, содержащих примерно такой текст: «Подождите, страница загружается». Такие страницы, само собой разумеется, не является целью парсинга, однако обойти их часто не представляется возможным. Естественно, без Selenium WebDriver. Selenium WebDriver позволяет в таких случаях «ожидать», как ожидал бы человек, пока на странице, к примеру, не появится элемент с необходимым именем.

Еще один плюс Selenium заключен в том, что действия веб-драйвера видимы визуально и требуют минимального времени нахождения на странице, это позволяет с удобством демонстрировать функционал сайта, когда необходима презентация сервиса.

1. Установка

1.1. Начало работы

Привязка Selenium к Python предоставляет собой простой API Интерфейс программирования приложений для написания тестов функциональности/тестов соответствия требованиям с использованием веб-драйвера Selenium WebDriver. С помощью Selenium Python API вы можете интуитивно просто получить доступ ко всему функционалу Selenium WebDriver.

Привязка Python-Selenium предоставляет удобный API для доступа к таким веб-драйверам Selenium как Firefox, Ie, Chrome, Remote и других. На данный момент поддерживаются версии Python 2.7, 3.2, 3.3 – 3.7.

1.2. Загрузка Selenium для Python

Вы можете загрузить привязку Selenium к Python со [страницы пакета selenium на PyPI](http://pypi.python.org/pypi/selenium). Однако, лучшим способом будет использование модуля [pip](http://www.pip-installer.org/en/latest/installing.html). Python 3.4 содержит pip в [стандартной библиотеке](http://docs.python.org/3.4/installing/index.html). Используя pip, вы можете установить selenium следующей командой:

pip install selenium

**Пример использования**

Если вы установили привязку Selenium к Python, вы можете начать использовать ее с помощью интерпретатора Python.

from selenium import webdriver

from selenium.webdriver.common.keys import Keys

driver = webdriver.Chrome()

driver.get("http://www.python.org")

assert "Python" in driver.title

elem = driver.find\_element\_by\_name("q")

elem.send\_keys("pycon")

elem.send\_keys(Keys.RETURN)

assert "No results found." not in driver.page\_source

driver.close()

## **1.3. Пошаговый разбор примера**

Модуль selenium.webdriver предоставляет весь функционал WebDriver'а. На данный момент WebDriver поддерживает реализации Firefox, Chrome, Ie и Remote. Класс Keys обеспечивает взаимодействие с командами клавиатуры, такими как RETURN, F1, ALT и т.д…  
  
from selenium import webdriver

from selenium.webdriver.common.keys import Keys

Далее создается элемент класса Chrome WebDriver.  
  
driver = webdriver.Chrome()

Метод driver.get перенаправляет к странице URL в параметре. WebDriver будет ждать пока страница не загрузится полностью (то есть, событие “onload” игнорируется), прежде чем передать контроль вашему тесту или скрипту. Стоит отметить, что если страница использует много AJAX-кода при загрузке, то WebDriver может не распознать, загрузилась ли она полностью:

driver.get("http://www.python.org")

Следующая строка — это утверждение (англ. assertion), что заголовок содержит слово “Python” [assert позволяет проверять предположения о значениях произвольных данных в произвольном месте программы. По своей сути assert напоминает констатацию факта, расположенную посреди кода программы. В случаях, когда произнесенное утверждение не верно, assert возбуждает исключение. Такое поведение позволяет контролировать выполнение программы в строго определенном русле. Отличие assert от условий заключается в том, что программа с assert не приемлет иного хода событий, считая дальнейшее выполнение программы или функции бессмысленным:

assert "Python" in driver.title

WebDriver предоставляет ряд способов получения элементов с помощью методов find\_element\_by\_\*. Для примера, элемент ввода текста input может быть найден по его атрибуту name методом find\_element\_by\_name.   
  
elem = driver.find\_element\_by\_name("q")

После этого мы посылаем нажатия клавиш (аналогично введению клавиш с клавиатуры). Специальные команды могут быть переданы с помощью класса Keys импортированного из selenium.webdriver.common.keys:

elem.send\_keys("pycon")

elem.send\_keys(Keys.RETURN)

После ответа страницы, вы получите результат, если таковой ожидается. Чтобы удостовериться, что мы получили какой-либо результат, добавим утверждение:  
  
assert "No results found." not in driver.page\_source

В завершение, окно браузера закрывается. Вы можете также вызывать метод quit вместо close. Метод quit закроет браузер полностью, в то время как close закроет одну вкладку. Однако, в случае, когда открыта только одна вкладка, по умолчанию большинство браузеров закрывается полностью:

driver.close()

# 2. Навигация

Предположительно, первое, что вы захотите сделать с помощью WebDriver, будет переход по ссылке. Обычно для таких целей используют метод get:

driver.get("http://www.google.com")

Метод driver.get перенаправляет к странице URL, поданной в параметре. WebDriver будет ждать пока страница не загрузится полностью (то есть, событие “onload” игнорируется), прежде чем передать контроль вашему тесту или скрипту. Стоит отметить, что если страница использует много AJAX-кода при загрузке, то WebDriver может не распознать, загрузилась ли она полностью. Если необходимо гарантировать полную загрузку страниц, вы можете использовать ожидания (англ. waits).

## **3.1. Взаимодействие со страницей**

Сама по себе возможность переходить по ссылке не так уж полезна. Что действительно хочется сделать, так это взаимодействовать со страницей, или, если быть точным, с элементами HTML на странице. Прежде всего необходимо найти их. WebDriver предоставляет ряд способов для поиска элементов. К примеру, на странице есть элемент, определенный таким образом:

<input type="text" name="passwd" id="passwd-id" />

Его можно найти, используя любой из следующих методов:

element = driver.find\_element\_by\_id("passwd-id")

element = driver.find\_element\_by\_name("passwd")

element = driver.find\_element\_by\_xpath("//input[@id='passwd-id']")

Также вы можете искать адрес гиперссылки по тексту гиперссылки, но будьте бдительны: текст должен совпадать в точности. Также будьте внимательны при использовании XPATH в WebDriver. Если существует больше одного элемента, удовлетворяющего условиям запроса, вернется только первый найденный. Если ничего не будет найдено, будет возбуждено исключение NoSuchElementException.

WebDriver имеет “Объекто-ориентированное” API. Если вы вызовете метод, используя его неправильно (например, используете “setSelected()” для “мета” тэга, WebDriver возбудит исключение.

Итак, мы получили элемент. Что вы можете с ним сделать? Первым делом вы захотите ввести какой-нибудь текст в текстовое поле:  
  
element.send\_keys("some text")

Также вы можете имитировать нажатие клавиш-стрелок клавиатуры с помощью класса “Keys”:

element.send\_keys(" and some", Keys.ARROW\_DOWN)

Метод send\_keys можно вызвать для любого элемента, который позволяет проверить сочетания клавиш, такие как те, что используются в GMail. Существует побочный эффект, заключающийся в том, что ввод в текстовое поле не очищает его автоматически. Вместо этого то, что вы набираете на клавиатуре, будет дописываться к уже вписанному в поле. Очистить содержимое тектового поля или текстовой области textarea легко — с помощью метода clear:

element.clear()

## **3.2. Заполнение форм**

Мы уже рассмотрели ввод текста в текстовую область или текстовое поле, а как быть с другими элементами? Вы можете попробовать раскрыть раскрывающийся список, после чего можно использовать “setSelected” для выделения тэгов вроде OPTION. Работать с тэгами SELECT не так уж сложно:

element = driver.find\_element\_by\_xpath("//select[@name='name']")

all\_options = element.find\_elements\_by\_tag\_name("option") for option

in all\_options: print("Value is: %s" % option.get\_attribute("value"))

option.click()

Такой код найдет первый элемент “SELECT” на странице, и в цикле пройдет по всем тэгам OPTION поочередно, сообщая их значения и поочередно выделяя их.

Как можно заметить, это не самый быстрый способ работы с элементами SELECT. Поддерживаемые вебдрайвером классы содержат один, называющийся “Select”, он предоставляет более удобные способы взаимодействия:

from selenium.webdriver.support.ui import Select

select = Select(driver.find\_element\_by\_name('name'))

select.select\_by\_index(index)

select.select\_by\_visible\_text("text")

select.select\_by\_value(value)

Также WebDriver предоставляет возможность снятия выделения со всех элементов выпадающего списка:

select = Select(driver.find\_element\_by\_id('id'))

select.deselect\_all()

Этот код снимает выделение со всех тегов OPTION первого тега SELECT на странице.  
  
Допустим, для теста вам необходим список всех выделенных по умолчанию опций. Класс Select предоставляет такое свойство (возвращает список):

select = Select(driver.find\_element\_by\_xpath("xpath"))

all\_selected\_options = select.all\_selected\_options

Для получения всех доступных опций используйте:  
  
options = select.options

После того как с заполнением формы будет покончено, вы, вероятно, захотите “сохранить” изменения [submit — отправить, передать, подтвердить — Прим. пер.]. Одним из способов сделать это — найти кнопку “submit” и кликнуть по ней:

*# Предположим, ID кнопки равен "submit"*

driver.find\_element\_by\_id("submit").click()

В качестве альтернативы первому методу можно использовать метод “submit”, доступный для каждого элемента. Если вызвать его для элемента внутри формы, WebDriver пробежится по всей структуре DOM, пока не найдет закрывающийся тег формы, и затем вызовет для нее submit. Если элемент находится не в форме, тогда возбудится исключение NoSuchElementException:

element.submit()

## **3.3. Перетаскивание**

Есть два варианта «перетаскивания» элементов: перемещение элемента на определенную величину, либо перетаскивание его на другой элемент:

element = driver.find\_element\_by\_name("source")

target = driver.find\_element\_by\_name("target")

from selenium.webdriver import ActionChains

action\_chains = ActionChains(driver)

action\_chains.drag\_and\_drop(element, target)

## **3.4. Переключение между окнами и фрэймами**

Современные веб-приложения редко обходятся без фреймов (frame) и редко, когда ограничиваются одним окном. WebDriver поддерживает переключение между именованными окнами с помощью метода “switch\_to\_window”:

driver.switch\_to\_window("windowName")

Все вызовы, начинающиеся с driver теперь будут истолкованы как обращенные к полученному окну. Но откуда вам знать имя окна? Взгляните на код javascript или ссылку, которые открывают окно:

<a href="кое\_где.html" target="имяОкна">

Нажмите сюда, чтобы открыть новое окно

</a>

Также вы можете послать “дескриптор окна” методу “switch\_to\_window()”. Пользуясь этой особенностью, вы можете использовать цикл для перебора всех открытых окон, к примеру, так:

for handle in driver.window\_handles:

driver.switch\_to\_window(handle)

Еще вы можете переходить между фрэймами (frame или iframes):  
  
driver.switch\_to\_frame("frameName")

Можно получить доступ к подчиненным фреймам, подавая путь, разделяемый точкой, или можно получить фрейм по индексу:

driver.switch\_to\_frame("frameName.0.child")

Следующий код перенаправит к фрейму с именем “child”, который в свою очередь принадлежит первому подчиненному фрейму фрейма “frameName”. **Пути к фреймам описываются полностью — от верхнего уровня**:

driver.switch\_to\_frame("frameName.0.child")

Когда работа с фреймами будет завершена, необходимо переключиться обратно, к главному фрейму, что может быть сделано следующим образом:  
  
driver.switch\_to\_default\_content()

## **3.5. Всплывающие окна**

Selenium WebDriver из упаковки поддерживает управление всплывающими диалоговыми окнами. После того как вы инициируете запуск, откроется окно, управлять им можно так:  
  
alert = driver.switch\_to\_alert()

Код вернет объект текущего открытого окна. С этим объектом вы можете принять, отклонить вопрос окна, прочитать его содержимое или даже ввести текст по приглашению окна. Интерфейс взаимодействия со всплывающими окнами работает одинаково хорошо как для предупреждений (alerts), так и для запросов к подтверждению (confirms) и приглашений к вводу (prompts). За дополнительной информацией обратитесь к документации API.

## **3.6. Навигация: история и локация**

Чуть раньше мы упомянули о навигации по ссылке с использованием команды “get” (driver.get(«[www.example.com](http://www.example.com/)») ). Как вы уже могли заметить, WebDriver для отдельных случаев предоставляет узконаправленные, специализированные интерфейсы взаимодействия, и навигация не исключение. Чтобы перейти по ссылке, вы можете воспользоваться методом get:

driver.get("http://www.example.com")

Чтобы перейти вперед или назад по истории вкладки:

driver.forward()

driver.back()

Имейте в виду, что этот функционал полностью зависит от используемого драйвера. Вы можете получить непредвиденный результат, если привыкли к поведению какого-либо конкретного браузера, а работаете с другим.

## **3.7. Куки (cookies)**

Прежде, чем мы завершим эту главу, вам возможно будет интересно узнать как использовать куки. Прежде всего, вам необходим домен, использующий куки:

*# Перейти на необходимый домен*

driver.get("http://www.example.com")

*# Установить куки. Следующий cookie действителен для всего домена*

cookie = {"ключ": "значение"}

driver.add\_cookie(cookie)

*# И теперь получим все доступные куки для текущего адреса URL*

all\_cookies = driver.get\_cookies()

for cookie\_name, cookie\_value in all\_cookies.items():

print("%s -> %s", cookie\_name, cookie\_value)

4. Поиск элементов

Существует ряд способов поиска элементов на странице. Вы вправе использовать наиболее уместные для конкретных задач. Selenium предоставляет следующие методы поиска элементов на странице:

* find\_element\_by\_id
* find\_element\_by\_name
* find\_element\_by\_xpath
* find\_element\_by\_link\_text
* find\_element\_by\_partial\_link\_text
* find\_element\_by\_tag\_name
* find\_element\_by\_class\_name
* find\_element\_by\_css\_selector

**Чтобы найти все элементы, удовлетворяющие условию поиска, используйте следующие методы (возвращается список):**

* find\_elements\_by\_name
* find\_elements\_by\_xpath
* find\_elements\_by\_link\_text
* find\_elements\_by\_partial\_link\_text
* find\_elements\_by\_tag\_name
* find\_elements\_by\_class\_name
* find\_elements\_by\_css\_selector

Как вы могли заметить, во втором списке отсутствует поиск по id. Это обуславливается особенностью свойства id для элементов HTML: идентификаторы элементов страницы всегда уникальны.

Помимо общедоступных (public) методов, перечисленных выше, существует два приватных (private) метода, которые при знании указателей объектов страницы могут быть очень полезны: find\_element and find\_elements.

Пример использования:

from selenium.webdriver.common.by import By

driver.find\_element(By.XPATH, '//button[text()="Some text"]')

driver.find\_elements(By.XPATH, '//button')

Для класса By доступны следующие атрибуты:

ID = "id"

XPATH = "xpath"

LINK\_TEXT = "link text"

PARTIAL\_LINK\_TEXT = "partial link text"

NAME = "name"

TAG\_NAME = "tag name"

CLASS\_NAME = "class name"

CSS\_SELECTOR = "css selector"

4.1. Поиск по Id

Используйте этот способ, когда известен id элемента. Если ни один элемент не удовлетворяет заданному значению id, будет вызвано исключение NoSuchElementException.  
  
Для примера, рассмотрим следующий исходный код страницы:

<html>

<body>

<form id="loginForm">

<input name="username" type="text" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Login" />

</form>

</body>

<html>  
Элемент form может быть определен следующим образом:

login\_form = driver.find\_element\_by\_id('loginForm')

4.2. Поиск по Name

Используйте этот способ, когда известен атрибут name элемента. Результатом будет первый элемент с искомым значением атрибута name. Если ни один элемент не удовлетворяет заданному значению name, будет вызвано исключение NoSuchElementException.

Для примера, рассмотрим следующий исходный код страницы:

<html>

<body>

<form id="loginForm">

<input name="username" type="text" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Login" />

<input name="continue" type="button" value="Clear" />

</form>

</body>

<html>

Элементы с именами username и password могут быть определены следующим образом:

username = driver.find\_element\_by\_name('username')

password = driver.find\_element\_by\_name('password')

Следующий код получит кнопку “Login”, находящуюся перед кнопкой “Clear”:

continue = driver.find\_element\_by\_name('continue')

4.3. Поиск по XPath

XPath – это язык, использующийся для поиска узлов дерева XML-документа. Поскольку в основе HTML может лежать структура XML (XHTML), пользователям Selenium предоставляется возможность посредоством этого мощного языка отыскивать элементы в их веб-приложениях. XPath выходит за рамки простых методов поиска по атрибутам id или name (и в то же время поддерживает их), и открывает спектр новых возможностей, таких как поиск третьего чекбокса (checkbox) на странице, к примеру.

Одно из веских оснований использовать XPath заключено в наличии ситуаций, когда вы не можете похвастать пригодными в качестве указателей атрибутами, такими как id или name, для элемента, который вы хотите получить. Вы можете использовать XPath для поиска элемента как по абсолютному пути (не рекомендуется), так и по относительному (для элементов с заданными id или name). XPath указатели в том числе могут быть использованы для определения элементов с помощью атрибутов отличных от id и name.

Абсолютный путь XPath содержит в себе все узлы дерева от корня (html) до необходимого элемента, и, как следствие, подвержен ошибкам в результате малейших корректировок исходного кода страницы. Если найти ближайщий элемент с атрибутами id или name (в идеале один из элементов-родителей), можно определить искомый элемент, используя связь «родитель-подчиненный». Эти связи будут куда стабильнее и сделают ваши тесты устойчивыми к изменениям в исходном коде страницы.

Для примера, рассмотрим следующий исходный код страницы:  
  
<html>

<body>

<form id="loginForm">

<input name="username" type="text" />

<input name="password" type="password" />

<input name="continue" type="submit" value="Login" />

<input name="continue" type="button" value="Clear" />

</form>

</body>

<html>  
Элемент form может быть определен следующими способами:

login\_form = driver.find\_element\_by\_xpath("/html/body/form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[1]")

login\_form = driver.find\_element\_by\_xpath("//form[@id='loginForm']")

1. Абсолютный путь (поломается при малейшем изменении структуры HTML страницы)
2. Первый элемент form в странице HTML
3. Элемент form, для которого определен атрибут с именем id и значением loginForm

Элемент username может быть найден так:  
  
username = driver.find\_element\_by\_xpath("//form[input/@name='username']") username = river.find\_element\_by\_xpath("//form[@id='loginForm']/input[1]") username = driver.find\_element\_by\_xpath("//input[@name='username']")

1. Первый элемент form с дочерним элементом input, для которого определен атрибут с именем name и значением username
2. Первый дочерний элемент input элемента form, для которого определен атрибут с именем id и значением loginForm
3. Первый элемент input, для которого определен атрибут с именем name и значением username

Кнопка “Clear” может быть найдена следующими способами:

clear\_button = driver.find\_element\_by\_xpath("//input[@name='continue'][@type='button']")

clear\_button = driver.find\_element\_by\_xpath("//form[@id='loginForm']/input[4]")

1. Элемент input, для которого заданы атрибут с именем name и значением continue и атрибут с именем type и значением button
2. Четвертый дочерний элемент input элемента form, для которого задан атрибут с именем id и значением loginForm

4.4. Поиск гиперссылок по тексту гиперссылки

Используйте этот способ, когда известен текст внутри анкер-тэга [anchor tag, анкер-тэг, тег «якорь» — тэг — Прим. пер.]. С помощью такого способа вы получите первый элемент с искомым значением текста тэга. Если никакой элемент не удовлетворяет искомому значению, будет вызвано исключение NoSuchElementException.  
  
Для примера, рассмотрим следующий исходный код страницы:

<html>

<body>

<p>Are you sure you want to do this?</p>

<a href="continue.html">Continue</a>

<a href="cancel.html">Cancel</a>

</body>

<html>  
Элемент-гиперссылка с адресом «continue.html» может быть получен следующим образом:

continue\_link = driver.find\_element\_by\_link\_text('Continue')

continue\_link = driver.find\_element\_by\_partial\_link\_text('Conti')

4.5. Поиск элементов по тэгу

Используйте этот способ, когда вы хотите найти элемент по его тэгу. Таким способом вы получите первый элемент с указанным именем тега. Если поиск не даст результатов, будет возбуждено исключение NoSuchElementException.  
  
Для примера, рассмотрим следующий исходный код страницы:  
  
<html>

<body>

<h1>Welcome</h1>

<p>Site content goes here.</p>

</body>

<html>

Элемент заголовка h1 может быть найден следующим образом:

heading1 = driver.find\_element\_by\_tag\_name('h1')

4.6. Поиск элементов по классу

Используйте этот способ в случаях, когда хотите найти элемент по значению атрибута class. Таким способом вы получите первый элемент с искомым именем класса. Если поиск не даст результата, будет возбуждено исключение NoSuchElementException.  
  
Для примера, рассмотрим следующий исходный код страницы:  
  
<html>

<body>

<p class="content">Site content goes here.</p>

</body>

<html>

Элемент “p” может быть найден следующим образом:  
  
content = driver.find\_element\_by\_class\_name('content')

4.7. Поиск элементов по CSS-селектору

Используйте этот способ, когда хотите получить элемент с использованием синтаксиса CSS-селекторов [CSS-селектор — это формальное описание относительного пути до элемента/элементов HTML. Классически, селекторы используются для задания правил стиля. В случае с WebDriver, существование самих правил не обязательно, веб-драйвер использует синтаксис CSS только для поиска — Прим. пер.]. Этим способом вы получите первый элемент удовлетворяющий CSS-селектору. Если ни один элемент не удовлетворяют селектору CSS, будет возбуждено исключение NoSuchElementException.

Для примера, рассмотрим следующий исходный код страницы:  
  
<html>

<body>

<p class="content">Site content goes here.</p>

</body>

<html>

Элемент “p” может быть определен следующим образом:  
  
content = driver.find\_element\_by\_css\_selector('p.content')

5. Ожидания

В наши дни большинство веб-приложений используют [AJAX](https://ru.wikipedia.org/wiki/AJAX) технологии. Когда страница загружена в браузере, элементы на этой странице могут подгружаться с различными временными интервалами. Это затрудняет поиск элементов, если элемент не присутствует в [DOM](https://ru.wikipedia.org/wiki/Document_Object_Model), возникает исключение ElementNotVisibleException. Используя ожидания, мы можем решить эту проблему. Ожидание дает некий временной интервал между произведенными действиями — поиске элемента или любой другой операции с элементом.

Selenium WebDriver предоставляет два типа ожиданий — неявное (implicit) и явное (explicit). Явное ожидание заставляет WebDriver ожидать возникновение определенного условия до произведения действий. Неявное ожидание заставляет WebDriver опрашивать DOM определенное количество времени, когда пытается найти элемент.  
  
5.1 Явные ожидания

Явное ожидание — это код, которым вы определяете какое необходимое условие должно произойти для того, чтобы дальнейший код исполнился. Худший пример такого кода — это использование команды time.sleep(), которая устанавливает точное время ожидания. Существуют более удобные методы, которые помогут написать вам код, ожидающий ровно столько, сколько необходимо. WebDriverWait в комбинации с ExpectedCondition является одним из таких способов.

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected\_conditions as EC

driver = webdriver.Firefox() driver.get("http://somedomain/url\_that\_delays\_loading")

try:

element = WebDriverWait(driver, 10).until( EC.presence\_of\_element\_located((By.ID, "myDynamicElement"))

)

finally:

driver.quit()

Этот код будет ждать 10 секунд до того, как отдаст исключение TimeoutException или если найдет элемент за эти 10 секунд, то вернет его. WebDriverWait по умолчанию вызывает ExpectedCondition каждые 500 миллисекунд до тех пор, пока не получит успешный return. Успешный return для ExpectedCondition имеет тип Boolean и возвращает значение true, либо возвращает not null для всех других ExpectedCondition типов.

**Ожидаемые условия.** Существуют некие условия, которые часто встречаются при автоматизации веб-сайтов. Ниже перечислены реализации каждого. Связки в Selenium Python предоставляют некоторые удобные методы, так что вам не придется писать класс expected\_condition самостоятельно или же создавать собственный пакет утилит.

* title\_is
* title\_contains
* presence\_of\_element\_located
* visibility\_of\_element\_located
* visibility\_of
* presence\_of\_all\_elements\_located
* text\_to\_be\_present\_in\_element
* text\_to\_be\_present\_in\_element\_value
* frame\_to\_be\_available\_and\_switch\_to\_it
* invisibility\_of\_element\_located
* element\_to\_be\_clickable — it is Displayed and Enabled.
* staleness\_of
* element\_to\_be\_selected
* element\_located\_to\_be\_selected
* element\_selection\_state\_to\_be
* element\_located\_selection\_state\_to\_be
* alert\_is\_present

from selenium.webdriver.support import expected\_conditions as EC

wait = WebDriverWait(driver, 10)

element = wait.until(EC.element\_to\_be\_clickable((By.ID,'someid')))

Модуль expected\_conditions уже содержит набор предопределенных условий для работы с WebDriverWait.

5.2 Неявные ожидания

Неявное ожидание указывает WebDriver'у опрашивать DOM определенное количество времени, когда пытается найти элемент или элементы, которые недоступны в тот момент. Значение по умолчанию равно 0. После установки, неявное ожидание устанавливается для жизни экземпляра WebDriver объекта.

from selenium import webdriver

driver = webdriver.Firefox()

driver.implicitly\_wait(10) *# seconds* driver.get("http://somedomain/url\_that\_delays\_loading")

myDynamicElement = driver.find\_element\_by\_id("myDynamicElement")

# Домашнее задание

# 1) Написать программу, которая собирает входящие письма из своего или тестового почтового ящика и сложить данные о письмах в базу данных (от кого, дата отправки, тема письма, текст письма) 2) Написать программу, которая собирает «Хиты продаж» с сайтов техники mvideo, onlinetrade и складывает данные в БД. Магазины можно выбрать свои. Главный критерий выбора: динамически загружаемые товары

# Используемая литература

1. <https://www.seleniumhq.org/docs/>
2. <https://selenium2.ru/docs.html>
3. <https://www.seleniumhq.org/selenium-ide/>
4. <https://python-scripts.com/web-automation-with-python-and-selenium#zapusk-brauzera-headless>