**Тема занятия №24: Сетевое программирование и веб-запросы. Парсинг данных**

**1. Использование библиотеки selenium**

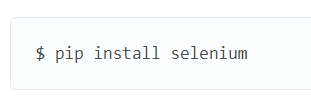
Python – один из самых популярных языков для веб-автоматизации с Selenium, поскольку в нем есть упрощенный синтаксис, который позволяет выполнять больше задач за меньшее количество строк кода! Таким образом, Python и Selenium создают идеальную комбинацию для автоматизированного тестирования в вебе.

Начнем с того, что Selenium – это открытый фреймворк, который в основном используется для автоматизации взаимодействия с веб-элементами в AUT (Application Under Test). Наряду с Python, Selenium также поддерживает Java, C#, JavaScript, Ruby и PHP. Однако, по своему опыту скажу, что Python – лучший вариант для веб-автоматизации в комбинации с Selenium.

Все больше разработчиков, как и я в свое время, осваивают Python, который считается третьим по популярности языком, если верить опросу Stack Overflow Developer Survey 2021. Поэтому шпаргалка по Selenium с Python может служить для ознакомления с полезным API для автоматизации веб-сайтов (или веб-приложений).

Если в Selenium вы новичок и вам интересно, что же это такое, мы рекомендуем ознакомиться с нашим руководством What is Selenium?

В этой статье мы разберемся с модулем в Python, который предоставляет нам функционал Selenium WebDriver. Для установки пакета Selenium WebDriver мы воспользуемся Python Package Index (PyPI). Выполните в терминале следующую команду, чтобы установить Selenium для Python:



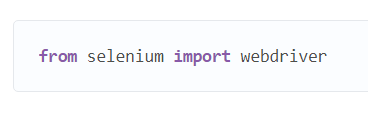
Если вы хотите запускать свои сценарии тестирования онлайн с Selenium Grid, воспользуйтесь LambdaTest для автоматизации тестирования.

Эта библиотека Python оборачивает Selenium WebDriver и предоставляет методы для автоматизации целого ряда задач, таких как заполнение форм, вход на сайт, нажатие на кнопки и многое другое. Кроме того, вы можете ознакомиться с руководством по Selenium с Python, в котором подробно рассматриваются неотъемлемые аспекты Selenium Python с точки зрения автоматизации веб-тестирования.

Команды, которые вы увидите в шпаргалке, можно использовать в качестве удобного ресурса для всех, кто играется с Selenium на Python для автоматизации веб-приложений. Если вам нужно краткое описание работы Selenium в Python, ознакомьтесь с руководством, в котором подробно рассматривается архитектура Selenium WebDriver и освещаются неотъемлемые аспекты, связанные с Selenium WebDriver в Python. Итак, давайте начнем нашу шпаргалку!

Импортируем библиотеку Selenium

Перед тем, как начать пользоваться командами Selenium, вам нужно импортировать пакет Selenium WebDriver.



Инициализация дайвера в Python

После загрузки соответствующего драйвера браузера вам необходимо запустить Selenium WebDriver и драйвер браузера (например, ChromeDriver для Chrome, GeckoDriver для Firefox и т.д.).

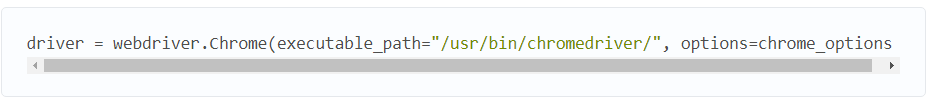


Если местоположения драйвера браузера нет в переменной PATH (или если его нет в System Path), нужно добавить следующие аргументы:

executable\_path: Путь к вашему веб-драйверу Selenium (бинарный файл)

options: Параметры, касающиеся выполнения веб-браузеров

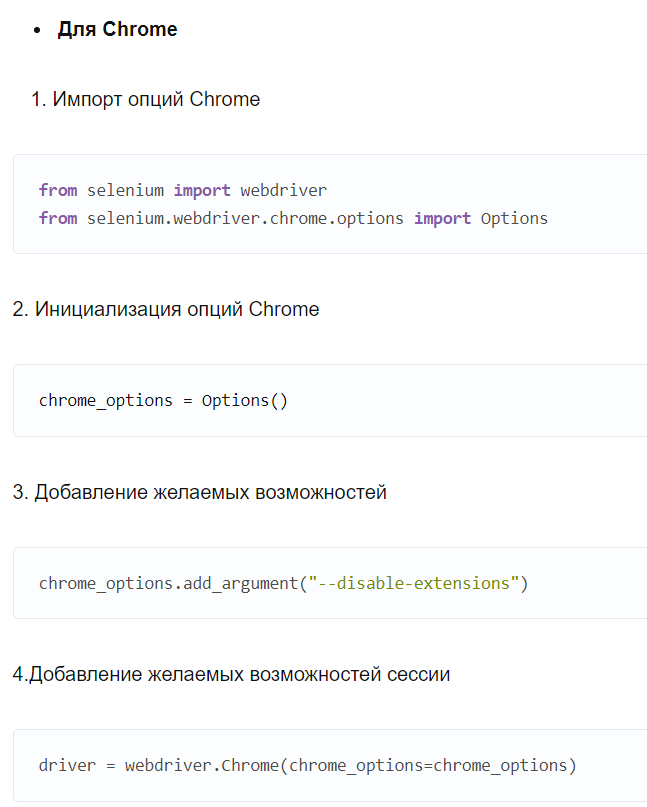
Пример:



Настройка параметров Selenium WebDriver

Класс Options в Selenium обычно используется в сочетании с желаемыми возможностями кастомизации Selenium WebDriver.

Так вы можете выполнять различные операции, такие как открытие браузера (Chrome, Firefox, Safari, IE, Edge и т.д.) в режиме увеличения, включение и отключение расширений браузера, отключение режима GPU, отключение всплывающих окон и многое другое. Поэтому важно хорошо разобраться в этом разделе шпаргалки по Selenium в Python, поскольку так вы сможете решить проблемы автоматизации, связанные с изменением свойств браузера, о которых мы говорили ранее.



Поиск элемента

В основном локаторы в Selenium (https://www.lambdatest.com/learning-hub/selenium-locators) используются для определения местоположения веб-элементов из DOM. Соответствующие взаимодействия (или действия) в дальнейшем выполняются относительно найденных веб-элементов. Несколько популярных локаторов в Selenium - ID, Name, Link Text, Partial Link Text, CSS Selectors, XPath, TagName и т.д.

Поиск элемента по атрибуту ID

В этом методе поиск элемента в DOM производится по ID. ID уникален для каждого элемента на странице. Таким образом, с помощью ID можно однозначно идентифицировать элемент. Например, ниже показано использование атрибута ID для поиска веб-элементов на странице входа в систему LambdaTest:

Так вы можете воспользоваться атрибутом ID в Selenium:



Поиск элемента по классу CSS

Элементы в HTML DOM еще можно найти по имени класса, который хранится в атрибуте класса HTML-тега. У класса может быть несколько экземпляров, но метод вернет первый элемент с соответствующим классом.



Поиск элемента по имени

Веб-элементы, такие как открывающий тег, имеют связанный с ними атрибут Name. В Selenium есть метод поиска веб-элементов с помощью атрибута NAME. Если у нескольких элементов одинаковое имя, он вернет первое совпадение.

Вот HTML-код, в котором есть элемент ввода имени Name.



Ниже показано использование метода Selenium Python для определения местоположения веб-элемента с помощью атрибута NAME:



Поиск элемента по тегу

Этот метод используется для поиска и выбора веб-элементов по HTML-тегу. Метод find\_element\_by\_tag\_name() используется для поиска таких тегов, как H1, DIV, INPUT и т.д. Если тег встречается несколько раз, он возвращает первый совпадающий.



**1. Использование библиотеки selenium**

Поиск элемента по XPath

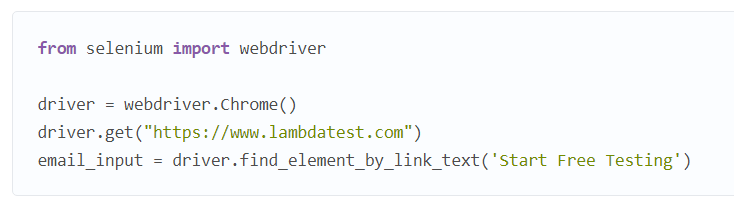
XPath использует путь для выбора узлов и определения местоположения необходимого веб-элемента. Метод find\_element\_by\_xpath() используется для поиска соответствующего элемента в документе с помощью XPath. Вы можете почитать руководство по XPath в Selenium, чтобы получить более глубокое представление об использовании XPath для поиска веб-элементов при автоматизации с Selenium.



Поиск элемента по тексту ссылки и частичному тексту ссылки

Этот метод ищет элемент по тексту ссылки (целому или части). При поиске частичного текста ссылки не ищется точное совпадение со строковым значением, поскольку ищется подмножество строк (в тексте ссылки).

Локатор текста ссылок в Selenium и частичного текста ссылок работает только со ссылками текущего веб-приложения.



Ниже показан поиск элемента на странице входа LambdaTest с помощью локатора частичного текста ссылок:



Различные методы поиска элементов

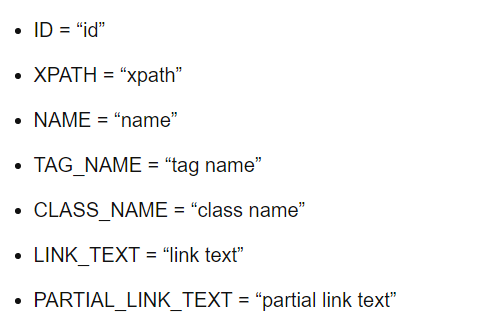
Есть два метода, которые могут быть полезны для поиска элементов страницы в сочетании с классом «By» для выбора атрибутов.

Обратите внимание, что нет никакой разницы между методами find\_element\_by\_tag и find\_element(By.tag). По умолчанию метод find\_element\_by\_tag вызывает find\_element(By.tag).

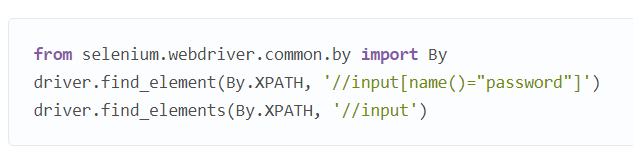
find\_element возвращает первый экземпляр из нескольких веб-элементов с определенным атрибутом в DOM. Метод вызывает исключение NoSuchElementException, если ни один элемент не соответствует требуемому локатору. Прочитайте статью о популярных исключениях Selenium, чтобы лучше понять, из-за чего они возникают.

find\_elements возвращает список всех экземпляров веб-элементов, соответствующих определенному атрибуту. Список будет пустым, если в DOM нет нужных элементов.

Ниже вы увидите список атрибутов, доступных для класса By:



Дальше показан пример использования метода find\_element, который использует локатор XPath для поиска нужного веб-элемента:

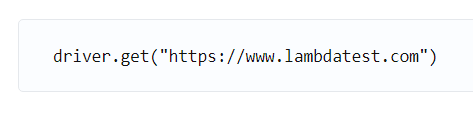


Открытие ссылки или документа

Перед выполнением любых операций с веб-элементами, присутствующими на странице, важно открыть целевой URL-адрес (или тестовый URL-адрес). Далее вы увидите несколько способов открыть URL-адрес в Selenium с Python:

driver.get(URL)

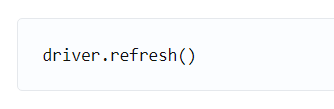
Метод driver.get() переходит на страницу, которая передается методу в параметре. Selenium WebDriver будет ждать, пока страница не загрузится полностью, после чего он запустит событие «onload» и вернет управление сценарию тестирования. В статье Selenium Waits in Python вы можете найти дополнительную информацию про обработку ожиданий в Selenium.



Обновление страницы

Бывают сценарии, в которых нужно обновлять содержимое страницы. Метод обновления страниц в Selenium WebDriver используется для обновления веб-страниц.

Метод driver.refresh() обновляет текущую веб-страницу. Он не принимает никаких аргументов и не возвращает никаких значений.



Ввод текста в веб-элемент

Метод send\_keys() в Python используется для ввода текста в текстовый элемент. Такой текст передается методу в качестве аргумента. Тот же метод можно использовать для имитации нажатия клавиш в любом поле (например, в полях ввода формы).

Пример использования send\_keys(), в котором адрес электронной почты передается текстовому элементу на странице регистрации LambdaTest:



Удаление текста в веб-элементе

Метод element.clear() в Selenium используется для удаления текста из полей, таких как поля ввода формы и т.д.

Пример использования метода очистки содержимого поля ввода электронной почты на домашней странице LambdaTest:



Нажатие на веб-элемент

Метод element.click() в Selenium используется для нажатия на элемент, такой как ссылка-якорь, кнопка и т.д.

Вот так с помощью метода click() можно нажать на кнопку на домашней странице LambdaTest:

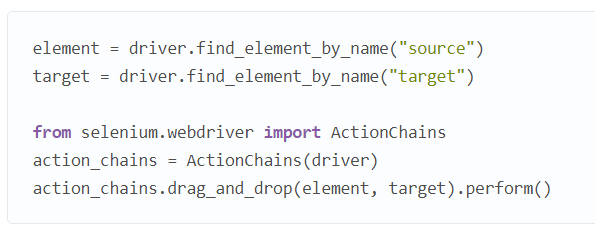


Перетаскивание веб-элемента

Перетаскивание объектов является одним из широко используемых сценариев в популярных приложениях (или программах), таких как Canvas, Google Drive, Trello, Asana и т.д. Метод drag\_and\_drop(элемент, цель) в Selenium помогает автоматизировать функционал перетаскивания веб-элементов из источника и переноса их в целевую область (или элемент).

У класса Actions в Selenium есть два метода, с помощью которых вы можете выполнять операции перетаскивания при тестировании кроссбраузерной совместимости. Обязательно ознакомьтесь с нашим подробным руководством, в котором есть информация о том, как выполнять перетаскивание в Selenium.

Вот простой пример использования метода drag\_and\_drop():



Получение кода страницы

Метод page\_source() в Selenium WebDriver используется для получения кода страницы. 