лабораторная РАБОТА №23

Инструкционно-технологическая карта

Тема: «Автоматизация взаимодействия с элементами веб-страницы с использование Selenium»

Цель работы: научиться взаимодействовать с элементами на веб-странице.

Время выполнения: 2 часа

1. **Теоретические сведения**

## Поиск элементов с помощью Selenium

Для поиска элементов на странице в Selenium WebDriver используются несколько стратегий, позволяющих искать по атрибутам элементов, текстам в ссылках, CSS-селекторам и XPath-селекторам. Существуют следующие методы поиска элементов:

* **find\_element\_by\_id** – поиск по уникальному атрибуту id элемента. Если разработчики проставляют всем элементам в приложении уникальный id, то лучше всего будет использовать этот метод, так как он наиболее стабильный;
* **find\_element\_by\_css\_selector** – поиск элемента с помощью правил на основе CSS. Это универсальный метод поиска, так как большинство веб-приложений использует CSS для вёрстки и задания оформления страницам. Если find\_element\_by\_id вам не подходит из-за отсутствия id у элементов, то скорее всего лучше использовать именно этот метод в тестах;
* **find\_element\_by\_xpath** – поиск с помощью языка запросов XPath, позволяет выполнять очень гибкий поиск элементов;
* **find\_element\_by\_name** – поиск по атрибуту name элемента;
* **find\_element\_by\_tag\_name** – поиск элемента по названию тега элемента;
* **find\_element\_by\_class\_name** – поиск по значению атрибута class;
* **find\_element\_by\_link\_text** – поиск ссылки на странице по полному совпадению;
* **find\_element\_by\_partial\_link\_text** – поиск ссылки на странице, если текст селектора совпадает с любой частью текста ссылки.

Например, необходимо найти кнопку со значением id="submit\_button":

from selenium import webdriver

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/simple\_form\_find\_task.html")

button = browser.find\_element\_by\_id("submit")

Если страница загрузилась, но дальше ничего не происходит, вернитесь обратно в консоль, в которой запускали скрипт. Скорее всего, увидите там ошибку **NoSuchElementException**. Она будет выглядеть следующим образом:

selenium.common.exceptions.NoSuchElementException: Message: no such element: Unable to locate element: {"method":"id","selector":"submit"}

Ошибка очевидна: неправильно указан локатор — значит, кнопки с таким id на странице нет. Исправим локатор, чтобы код проходил без ошибок:

from selenium import webdriver

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/simple\_form\_find\_task.html")

button = browser.find\_element\_by\_id("submit\_button")

Есть второй способ для поиска элементов с помощью универсального метода **find\_element**() и полей класса By из библиотеки selenium. Пример:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

browser.get("http://suninjuly.github.io/simple\_form\_find\_task.html")

button = browser.find\_element(By.ID, "submit\_button")

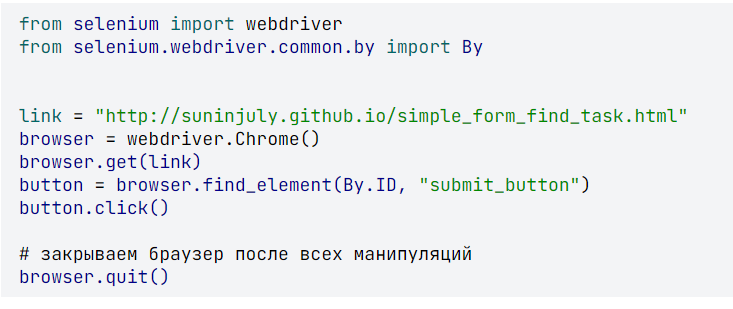
Можно использовать те же стратегии поиска, что и в первом способе. Второй способ более удобен для оформления архитектуры тестовых сценариев с помощью подхода Page Object Model. Пока же предлагаем пользоваться первым методом с явным указанием способа поиска, так как он кажется более удобным, но ничто не мешает пользоваться и тем, и другим. Поля класса By, которые можно использовать для поиска:

* **By.ID** – поиск по уникальному атрибуту id элемента;
* **By.CSS\_SELECTOR** – поиск элементов с помощью правил на основе CSS;
* **By.XPATH** – поиск элементов с помощью языка запросов XPath;
* **By.NAME** – поиск по атрибуту name элемента;
* **By.TAG\_NAME** – поиск по названию тега;
* **By.CLASS\_NAME** – поиск по атрибуту class элемента;
* **By.LINK\_TEXT** – поиск ссылки с указанным текстом. Текст ссылки должен быть точным совпадением;
* **By.PARTIAL\_LINK\_TEXT** – поиск ссылки по частичному совпадению текста.

!**Важно**. Можно столкнуться с ситуацией, когда на странице будет несколько элементов, подходящих под заданные параметры поиска. В этом случае WebDriver вернет вам **только первый элемент**, который встретит во время поиска по HTML. Если нужен не первый, а второй или следующие элементы, нужно либо задать более точный селектор для поиска, либо использовать методы **find\_elements\_by**

## Задание 1.

1. Запустить виртуальное окружение
2. Набрать в текстовом редакторе и сохранить следующий скрипт

****

**Примечание:** Каждый раз при открытии браузера

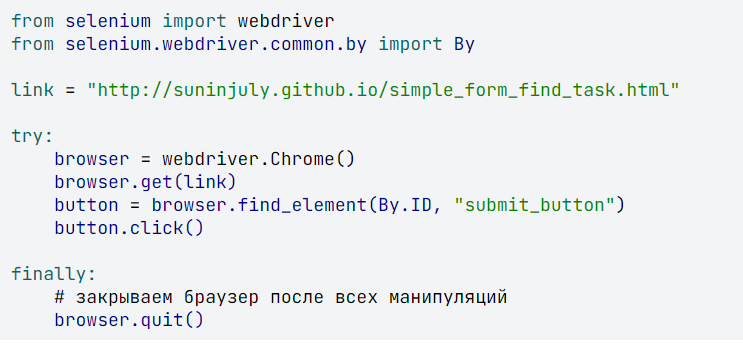
browser = webdriver.Chrome()

в системе создается процесс, который останется висеть, если вы вручную закроете окно браузера. Чтобы не остаться без оперативной памяти после запуска нескольких скриптов, всегда добавляйте к своим скриптам команду закрытия

browser.quit()

Для того чтобы гарантировать закрытие, даже если произошла ошибка в предыдущих строках, проще всего использовать конструкцию **try/finally.** Если в коде внутри блока **try** произойдет какая-то ошибка, то код внутри блока **finally** выполнится в любом случае.

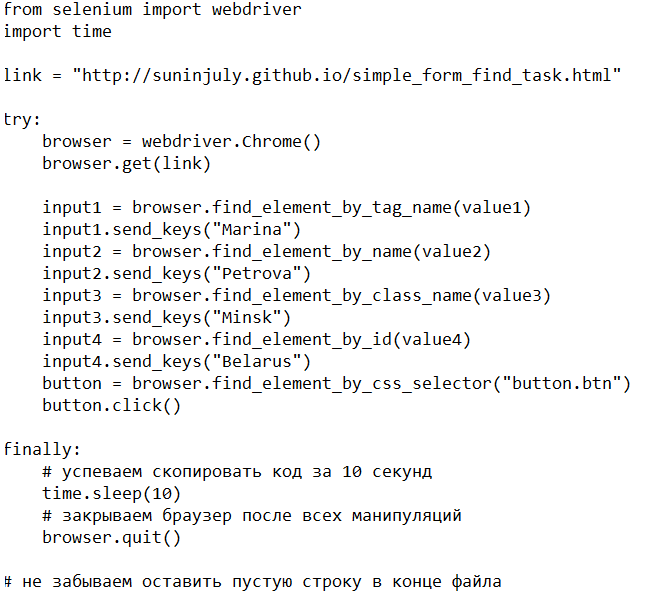
1. Изменить предыдущий скрипт, добавив конструкцию **try/finally:**



## Задание 2.

1. Запустить виртуальное окружение
2. Открыть страницу по [ссылке](http://suninjuly.github.io/simple_form_find_task.html) и заполнить форму на этой странице с помощью Selenium.
3. Набрать в текстовом редакторе и сохранить следующий скрипт.

**Замечание**: значения value1, value2, value3 и value4 необходимо определить через инструменты разработчика в браузере



**Примечание**: пустая строка необходима для операционной системы UNIX/Linux. Если в скрипте ее не будет, то последняя строчка, содержащая код, может не выполниться.

1. Проверить работу скрипта в интерпретаторе

python имя\_task2.py

1. Если всё сделано правильно, то вы увидите окно с проверочным кодом. Это число сохранить в комментарии внутри скрипта.

## Поиск элемента по тексту в ссылке

В этом задании попробуем искать элементы по тексту ссылки, для этого воспользуемся методом find\_element\_by\_link\_text:

link = browser.find\_element\_by\_link\_text(text)

В качестве аргумента в метод передается такой текст, ссылку с которым мы хотим найти. Это тот самый текст, который содержится между открывающим и закрывающим тегом

<a> вот тут text </a>

Если хотим найти элемент по полному соответствию текста, то нам подойдет такой код:

link = browser.find\_element\_by\_link\_text("Text\_examples")

link.click()

А если хотим найти элемент со ссылкой по подстроке, то нужно написать следующий код:

link = browser.find\_element\_by\_partial\_link\_text("examples")

link.click()

## Задание 3.

1. Запустить виртуальное окружение
2. Открыть скрипт, сохраненный в задании 2.
3. Добавьте в самый верх своего кода import math
4. Поменяйте адрес в строке, где задается открытие страницы: <http://suninjuly.github.io/find_link_text>
5. Добавьте команду, которая найдет ссылку с текстом. Текст ссылки, который нужно найти, зашифрован формулой:

str(math.ceil(math.pow(math.pi, math.e)\*10000))

(можно вставить данное выражение в свой код, а можно выполнить в интерпретаторе, скопировать оттуда результат и уже его использовать в коде)

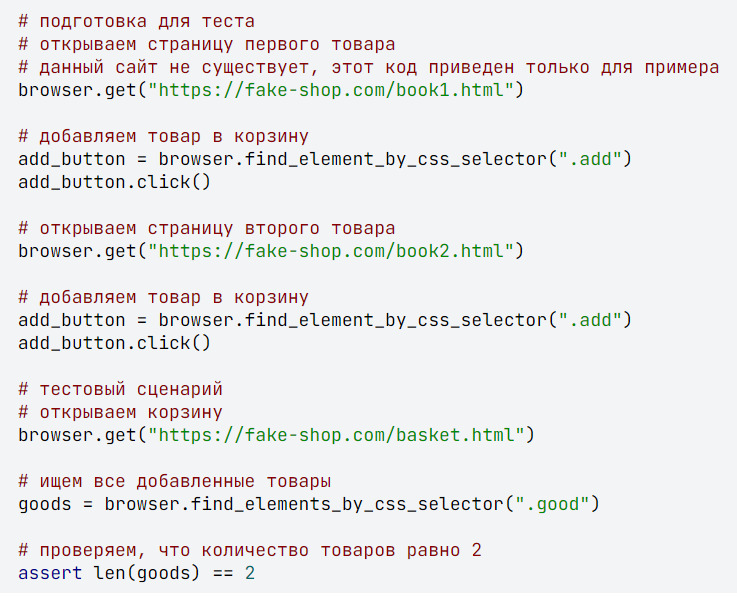
1. Добавьте команду для клика по найденной ссылке: она перенесет на форму регистрации
2. Оставьте скрипт заполнения формы так же, как делали в предыдущем задании
3. Проверить работу скрипта в интерпретаторе

python имя\_task3.py

1. Если всё сделано правильно, то вы увидите окно с проверочным кодом. Это число сохранить в комментарии внутри скрипта.

## Поиск всех необходимых элементов с помощью find\_elements\_by

Метод find\_element\_by возвращает только **первый** из всех элементов, которые подходят под условия поиска. Иногда возникает ситуация, когда есть несколько одинаковых по сути объектов на странице, например, иконки товаров в корзине интернет-магазина. В тесте нужно проверить, что отображаются все выбранные для покупки товары. Для этого существуют методы find\_element**s**\_by, которые в отличие от find\_element\_by вернут список всех найденных элементов по заданному условию. Проверив длину списка, можно удостовериться, что в корзине отобразилось правильное количество товаров. Пример кода (код приведен только для примера, сайта fake-shop.com скорее всего не существует):



Набор стратегий здесь такой же, как и в случае с **find\_element\_by**:

* find\_element**s**\_by\_css\_selector;
* find\_element**s**\_by\_xpath;
* find\_element**s**\_by\_name;
* find\_element**s**\_by\_tag\_name;
* find\_element**s**\_by\_class\_name;
* find\_element**s**\_by\_link\_text;
* find\_element**s**\_by\_partial\_link\_text.

Также для поиска нескольких элементов можно использовать универсальный метод find\_element**s**вместе с атрибутами класса **By**:

from selenium.webdriver.common.by import By

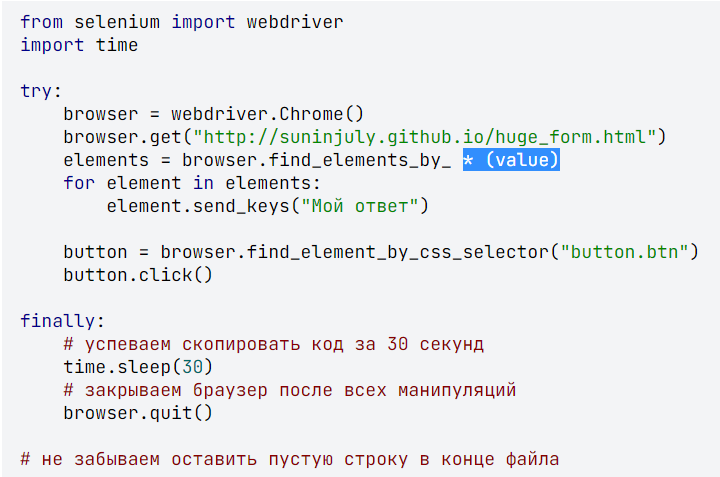
browser.find\_elements(By.CSS\_SELECTOR, "button.submit")

!**Важно**. Обратите внимание на важную разницу в результатах, которые возвращают методы find\_element и find\_element**s**. Если первый метод не смог найти элемент на странице, то он вызовет ошибку NoSuchElementException, которая прервёт выполнение вашего кода. Второй же метод всегда возвращает валидный результат: если ничего не было найдено, то он вернёт пустой список и ваша программа перейдет к выполнению следующего шага в коде.

## Задание 4.

1. Необходимо заполнить форму (<http://suninjuly.github.io/huge_form.html>). В награду за заполнение формы становится доступен код на скидку. В форме 100 обязательных полей и ограничено время на ее заполнение. Используйте WebDriver и подходящий метод find\_elements\_by..
2. Используйте приведенный ниже шаблон: в цикле for можно последовательно взять каждый элемент из найденного списка текстовых полей и отправить произвольный текст в каждое поле. Если скрипт не успевает заполнить форму, выберите текст покороче.

**Замечание**: значения метода **find\_element\_by\_\***  и соответствующего **value**, необходимо определить через инструменты разработчика в браузере.



1. Проверить работу скрипта в интерпретаторе

python имя\_task4.py

Если всё сделано правильно, то вы увидите окно с проверочным кодом. Это число сохранить в комментарии внутри скрипта.

## Задание 5.

На странице <http://suninjuly.github.io/find_xpath_form>  находится такая же форма регистрации, как в задании 2, при этом в нее добавилась куча одинаковых кнопок отправки. Но сработает только кнопка с текстом "Submit", и задача нажать в коде именно на неё.

Для выполнения задания необходимо использовать метод поиска элемента по **XPath.**

1. В коде из задания 2 замените ссылку на  <http://suninjuly.github.io/find_xpath_form>.
2. Подберите уникальный XPath-селектор так, чтобы он находил *только* кнопку с текстом **Submit**. XPath можете формулировать как угодно (по тексту, по структуре, по атрибутам) - главное, чтобы он работал.
3. Модифицируйте код из задания 2 таким образом, чтобы поиск кнопки происходил с помощью XPath.
4. Проверить работу скрипта в интерпретаторе

python имя\_task5.py

Если всё сделано правильно, то вы увидите окно с проверочным кодом. Это число сохранить в комментарии внутри скрипта.