**Как работать с элементами типа checkbox и radiobutton?**

Checkbox (чекбокс или флажок) и radiobutton (радиобаттон или переключатель) — часто используемые в формах элементы. Основная разница между ними состоит в том, что флажки позволяют выбирать/отключать любой из представленных вариантов, а переключатели позволяют выбрать только один из вариантов. Далее мы будем называть эти элементы на англоязычный манер: checkbox и radiobutton.

Оба этих элемента создаются при помощи тега input со значением атрибута type равным checkbox или radio соответственно. В html-коде страницы вы увидите:

<input type="checkbox">

<input type="radio">

Если checkbox или radiobutton выбран, то у элемента появится новый атрибут checked без значения. Часто атрибут checked уже установлен для одного из элементов по умолчанию.

<input type="checkbox" checked>

<input type="radio" checked>

Radiobuttons объединяются в группу, где все элементы имеют одинаковые значения атрибута name, но разные значения атрибута value:

<input type="radio" name="language" value="python" checked>

<input type="radio" name="language" value="selenium">

Чтобы снять/поставить галочку в элементе типа checkbox или выбрать опцию из группы radiobuttons, надо указать WebDriver метод поиска элемента и выполнить для найденного элемента метод click():

option1 = browser.find\_element(By.CSS\_SELECTOR, "[value='python']")

option1.click()

Также вы можете увидеть тег label рядом с input. Этот тег используется, чтобы сделать кликабельным текст, который отображается рядом с флажком. Этот текст заключен внутри тега label. Элемент label связывается с элементом input с помощью атрибута for, в котором указывается значение атрибута id для элемента input:

<div>

<input type="radio" id="python" name="language" checked>

<label for="python">Python</label>

</div>

<div>

<input type="radio" id="java" name="language">

<label for="java">Java</label>

</div>

В этом случае можно также отметить нужный пункт с помощью WebDriver, выполнив метод click() на элементе label.

option1 = browser.find\_element(By.CSS\_SELECTOR, "[for='java']")

option1.click()

## Метод get\_attribute

Мы уже знаем, как найти нужный элемент на странице и как получить видимый пользователю текст. Для более детальных проверок в тесте нам может понадобиться узнать значение атрибута элемента. Атрибуты могут быть стандартными свойствами, которые понимает и использует браузер для отображения и вёрстки элементов или для хранения служебной информации, например, name, width, height, color и многие [другие](https://www.w3schools.com/tags/ref_attributes.asp). Также атрибуты могут быть созданы разработчиками проекта для задания собственных стилей или правил.

Значение атрибута представляет собой строку. Если значение атрибута отсутствует, то это равносильно значению атрибута равному "false". Давайте еще раз взглянем на страницу <http://suninjuly.github.io/math.html>. На ней есть radiobuttons, для которых выбрано значение по умолчанию. В автотесте нам может понадобиться проверить, что для одного из radiobutton по умолчанию уже выбрано значение. Для этого мы можем проверить значение атрибута checked у этого элемента. Вот HTML-код элемента:

<input class="check-input" type="radio" name="ruler" id="peopleRule" value="people" checked>

Найдём этот элемент с помощью WebDriver:

people\_radio = browser.find\_element(By.ID, "peopleRule")

Найдём атрибут "checked" с помощью встроенного метода get\_attribute и проверим его значение:

people\_checked = people\_radio.get\_attribute("checked")

print("value of people radio: ", people\_checked)

assert people\_checked is not None, "People radio is not selected by default"

Т.к. у данного атрибута значение не указано явно, то метод get\_attribute вернёт "true". Возможно, вы заметили, что "true" написано с маленькой буквы, — все методы WebDriver взаимодействуют с браузером с помощью JavaScript, в котором булевые значения пишутся с маленькой буквы, а не с большой, как в Python.

Мы можем написать проверку другим способом, сравнив строки:

assert people\_checked == "true", "People radio is not selected by default"

Если атрибута нет, то метод get\_attribute вернёт значение **None**. Применим метод get\_attribute ко второму radiobutton, и убедимся, что атрибут отсутствует.

robots\_radio = browser.find\_element(By.ID, "robotsRule")

robots\_checked = robots\_radio.get\_attribute("checked")

assert robots\_checked is None

Так же мы можем проверять наличие атрибута disabled, который определяет, может ли пользователь взаимодействовать с элементом. Например, в предыдущем задании на странице с капчей для роботов JavaScript устанавливает атрибут disabled у кнопки **Submit**, когда истекает время, отведенное на решение задачи.

<button type="submit" class="btn btn-default" disabled>Submit</button>

## Работа со списками

На веб-страницах мы также встречаем раскрывающиеся (выпадающие) списки. У таких списков есть несколько важных особенностей:

1. У каждого элемента списка обычно есть уникальное значение атрибута value
2. В списках может быть разрешено выбирать как только один, так и несколько вариантов, в зависимости от типа списка
3. Визуально списки могут различаться тем, что в одном случае все варианты скрыты в выпадающем меню (<http://suninjuly.github.io/selects1.html>), а в другом все варианты или их часть видны (<http://suninjuly.github.io/selects2.html>)

Но для взаимодействия с любым вариантом списка мы будем использовать одни и те же методы Selenium.

Посмотрим, как выглядит html для списка:

<label for="dropdown">Выберите язык программирования:</label>

<select id="dropdown" class="custom-select">

 <option selected>--</option>

 <option value="1">Python</option>

 <option value="2">Java</option>

 <option value="3">JavaScript</option>

</select>

Варианты ответа задаются тегом option, значение value может отсутствовать. Можно отмечать варианты с помощью обычного метода click(). Для этого сначала нужно применить метод click() для элемента с тегом select, чтобы список раскрылся, а затем кликнуть на нужный вариант ответа:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

browser.get(link)

browser.find\_element(By.TAG\_NAME, "select").click()

browser.find\_element(By.CSS\_SELECTOR, "option:nth-child(2)").click()

Последняя строчка может выглядеть и так:

browser.find\_element(By.CSS\_SELECTOR, "[value='1']").click()

Это не самый удобный способ, так как нам приходится делать лишний клик для открытия списка.

Есть более удобный способ, для которого используется специальный класс **Select** из библиотеки WebDriver. Вначале мы должны инициализировать новый объект, передав в него WebElement с тегом select. Далее можно найти любой вариант из списка с помощью метода **select\_by\_value(value):**

from selenium.webdriver.support.ui import Select

select = Select(browser.find\_element(By.TAG\_NAME, "select"))

select.select\_by\_value("1") # ищем элемент с текстом "Python"

Можно использовать еще два метода: **select.select\_by\_visible\_text("text")** и **select.select\_by\_index(index)**. Первый способ ищет элемент по видимому тексту, например,**select.select\_by\_visible\_text("Python")** найдёт "Python" для нашего примера.

Второй способ ищет элемент по его индексу или порядковому номеру. Индексация начинается с нуля. Для того чтобы найти элемент с текстом "Python", нужно использовать **select.select\_by\_index(1)**, так как опция с индексом 0 в данном примере имеет значение по умолчанию равное "--".

## Метод execute\_script

Рассмотрим еще один очень полезный и мощный метод, но он требует хотя бы минимальных знаний JavaScript. С помощью метода execute\_script можно выполнить программу, написанную на языке JavaScript, как часть сценария автотеста в запущенном браузере. Зачем это может понадобиться, если в автотестах мы стараемся взаимодействовать с интерфейсом сайта как обычный пользователь, нажимая кнопки, выбирая пункты меню и вводя текст в текстовые поля?

Дело в том, что стандартные методы, доступные в Selenium, не могут покрыть всех возможных ситуаций работы с веб-приложением. Сайты в интернете могут решать самые разные задачи, начиная от простого блога до сложных финансовых или графических приложений. Разработчики имеют доступ к огромному количеству различных библиотек для решения бизнес-сценариев, что приводит к появлению на веб-странице нестандартных редакторов текстов, уникальных меню, оригинальных видео-плееров и т.д. Порой это приводит к тому, что для нажатия вроде бы обычной кнопки тестировщику понадобится писать настоящий JavaScript-сценарий. Если вы столкнулись с такой ситуацией, то в первую очередь обратитесь за помощью к вашим фронтенд-разработчикам, чтобы они подсказали  пример нужного скрипта. Прежде чем использовать данный скрипт в тестах, вы можете проверить, как он работает прямо в браузере, выполнив код в консоли браузера. Затем можете добавить его в ваш автотест с помощью execute\_script(javascript\_code).

Давайте попробуем вызвать alert в браузере с помощью WebDriver. Пример сценария:

from selenium import webdriver

browser = webdriver.Chrome()

browser.execute\_script("alert('Robots at work');")

Обратите внимание, что исполняемый JavaScript нужно заключать в кавычки (двойные или одинарные). Если внутри скрипта вам также понадобится использовать кавычки, а для выделения скрипта вы уже используете двойные кавычки, то в скрипте следует поставить одинарные:

browser.execute\_script("document.title='Script executing';")

Такой формат записи тоже будет работать:

browser.execute\_script('document.title="Script executing";')

Можно с помощью этого метода выполнить сразу несколько инструкций, перечислив их через точку с запятой. Изменим сначала заголовок страницы, а затем вызовем alert:

browser.execute\_script("document.title='Script executing';alert('Robots at work');")

## Метод location\_once\_scrolled\_into\_view

как работает метод **click()**.

В первую очередь WebDriver проверит, что ширина и высота элемента больше 0, чтобы по нему можно было кликнуть.

Затем, если элемент находится за границей окна браузера, WebDriver автоматически проскроллит страницу, чтобы элемент попал в область видимости, то есть не находился за границей экрана. Но это не гарантирует того, что элемент не перекрыт другим элементом, который тоже находится в области видимости.

А в какую точку элемента будет происходить клик? Selenium рассчитывает координаты центра элемента и производит клик в вычисленную точку. Это тоже приведёт к ошибке, если часть элемента всё-таки видна, но элемент перекрыт больше чем на половину своей высоты или ширины.

Если мы столкнулись с такой ситуацией, мы можем заставить браузер дополнительно проскроллить нужный элемент, чтобы он точно стал видимым.

## Загрузка файлов

﻿Если нам понадобится загрузить файл на веб-странице, мы можем использовать уже знакомый нам метод send\_keys. Только теперь нам нужно в качестве аргумента передать путь к нужному файлу на диске вместо простого текста.

Чтобы указать путь к файлу, можно использовать стандартный модуль Python для работы с операционной системой — **os**. В этом случае ваш код не будет зависеть от операционной системы, которую вы используете. Добавление файла будет работать и на Windows, и на Linux, и даже на MaсOS.

Пример кода, который позволяет указать путь к файлу**'file.txt**', находящемуся в той же папке, что и скрипт, который вы запускаете:

import os

current\_dir = os.path.abspath(os.path.dirname(\_\_file\_\_)) # получаем путь к директории текущего исполняемого файла

file\_path = os.path.join(current\_dir, 'file.txt') # добавляем к этому пути имя файла

element.send\_keys(file\_path)

Попробуйте добавить в файл отдельно команды **print(os.path.abspath(\_\_file\_\_))** и **print(os.path.abspath(os.path.dirname(\_\_file\_\_)))** и посмотрите на разницу. Подробнее о методах модуля **os** можете почитать самостоятельно в документации: <https://docs.python.org/3/library/os.path.html>. Обратите внимание, что это будет работать только при запуске кода из файла, в интерпретаторе не сработает.

Если совсем непонятно что происходит, пример:

Допустим, мы написали код скрипта и сохранили код в lesson2\_step7.py в свой локальной папке D:\stepik\_homework. Активируем виртуальное окружение и запускаем его **python lesson2\_step7.py.**В таком случае конструкция **os.path.abspath(os.path.dirname(\_\_file\_\_))**вернет нам путь до директории файла с кодом, то есть D:\stepik\_homework**.**В эту же папку кладем файл, который хотим прикрепить, то есть file.txt. Тогда, после выполнения команды:

file\_path = os.path.join(current\_dir, 'file.txt')

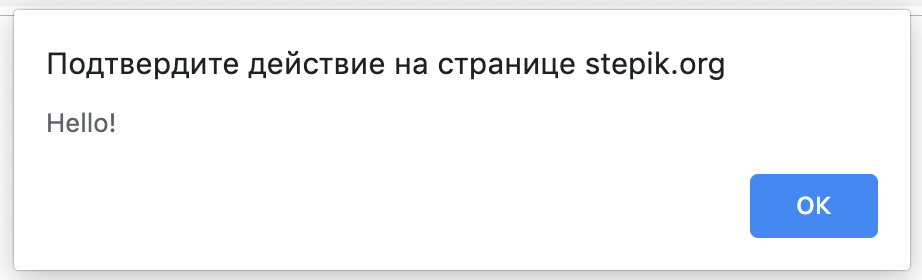
В переменной file\_path будет полный путь к файлу '**D:\stepik\_homework\file.txt'**. Фишка в том, что если мы файлы lesson2\_step7.py вместе с file.txt перенесем в другую папку, или на компьютер с другой ОС, то такой код без правок заработает и там.

Элемент в форме, который выглядит, как кнопка добавления файла, имеет атрибут**type="file"**. Мы должны сначала найти этот элемент с помощью селектора, а затем применить к нему метод **send\_keys(file\_path)**.

## Alerts и как с ними жить

Мы уже встречали alert в нашем курсе, когда получали число-ответ в задачах. Также мы узнали, что можно самостоятельно вызвать alert с помощью JavaScript:

alert('Hello!');



Теперь рассмотрим ситуацию, когда в сценарии теста возникает необходимость не только получить содержимое alert, но и нажать кнопку OK, чтобы закрыть alert. **Alert** является модальным окном: это означает, что пользователь не может взаимодействовать дальше с интерфейсом, пока не закроет alert. Для этого нужно сначала переключиться на окно с alert, а затем принять его с помощью команды **accept()**:

alert = browser.switch\_to.alert

alert.accept()

Чтобы получить текст из alert, используйте свойство text объекта alert:

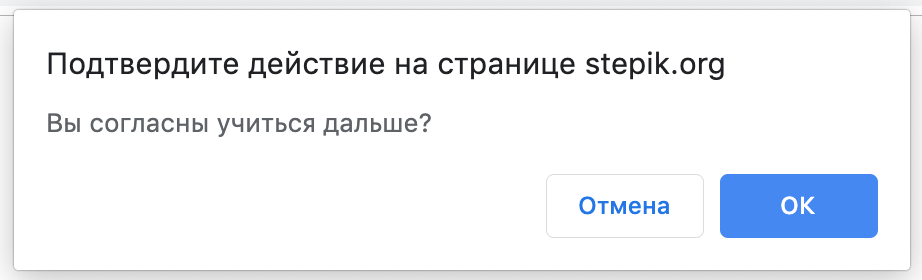
alert = browser.switch\_to.alert

alert\_text = alert.text

Другой вариант модального окна, который предлагает пользователю выбор согласиться с сообщением или отказаться от него, называется **confirm**. Для переключения на окно confirmиспользуется та же команда, что и в случае с alert:

confirm = browser.switch\_to.alert

confirm.accept()



Для confirm**-**окон можно использовать следующий метод для отказа:

confirm.dismiss()

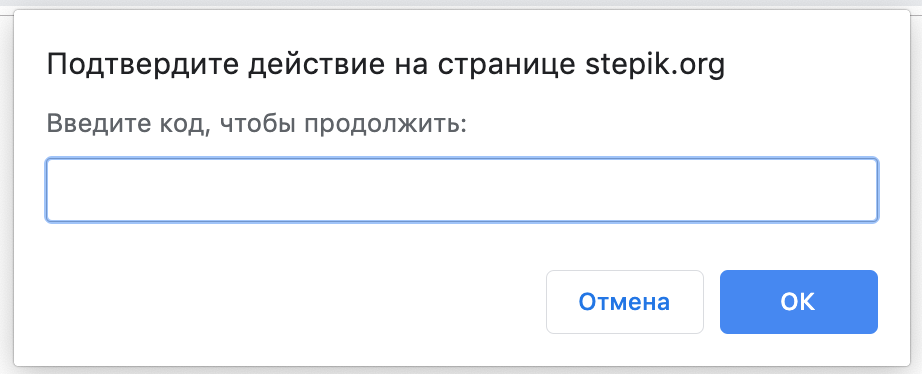
То же самое, что и при нажатии пользователем кнопки "Отмена".

Третий вариант модального окна — **prompt**— имеет дополнительное поле для ввода текста. Чтобы ввести текст, используйте метод **send\_keys()**:

prompt = browser.switch\_to.alert

prompt.send\_keys("My answer")

prompt.accept()



## Переход на новую вкладку браузера

При работе с веб-приложениями приходится переходить по ссылкам, которые открываются в новой вкладке браузера. WebDriver может работать только с одной вкладкой браузера. При открытии новой вкладки WebDriver продолжит работать со старой вкладкой. Для переключения на новую вкладку надо явно указать, на какую вкладку мы хотим перейти. Это делается с помощью команды switch\_to.window:

browser.switch\_to.window(window\_name)

Чтобы узнать имя новой вкладки, нужно использовать метод window\_handles, который возвращает массив имён всех вкладок. Зная, что в браузере теперь открыто две вкладки, выбираем вторую вкладку:

new\_window = browser.window\_handles[1]

Также мы можем запомнить имя текущей вкладки, чтобы иметь возможность потом к ней вернуться:

first\_window = browser.window\_handles[0]

После переключения на новую вкладку поиск и взаимодействие с элементами будут происходить уже на новой странице.