**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 "--".

**Задание: работа с выпадающим списком**

Для этой задачи мы придумали еще один вариант капчи для роботов. Придется немного переобучить нашего робота, чтобы он справился с новым заданием.

Напишите код, который реализует следующий сценарий:

1. Открыть страницу [https://suninjuly.github.io/selects1.html](http://suninjuly.github.io/selects1.html)
2. Посчитать сумму заданных чисел
3. Выбрать в выпадающем списке значение равное расчитанной сумме
4. Нажать кнопку "Submit"

Если все сделано правильно и достаточно быстро (в этой задаче тоже есть ограничение по времени), вы увидите окно с числом. Отправьте полученное число в качестве ответа для этого задания.

Когда ваш код заработает, попробуйте запустить его на странице [https://suninjuly.github.io/selects2.html](http://suninjuly.github.io/selects2.html). Ваш код и для нее тоже ﻿должен пройти успешно.

Подсказка: если вы получаете ошибку в духе "argument of type 'int' is not iterable", перепроверьте тип переменной, которую вы передаете в функцию поиска. Нужно передать строку!

## 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');")

## Пример задачи для execute\_script

Давайте теперь рассмотрим реальную ситуацию, когда пользователь должен кликнуть на элемент, который внезапно оказывается перекрыт другим элементом на странице.

Для клика в WebDriver мы используем метод click(). Если элемент оказывается перекрыт другим элементом, то наша программа вызовет следующую ошибку:

selenium.common.exceptions.WebDriverException: Message: unknown error: Element <button type="submit" class="btn btn-default" style="margin-bottom: 1000px;">...</button> is not clickable at point (87, 420). Other element would receive the click: <p>...</p>

Из описания ошибки можно понять, что указанный нами элемент нельзя кликнуть в данной точке, т.к. клик произойдёт на другом элементе с тегом <p>.

Чтобы увидеть пример данной ошибки, запустите следующий скрипт:

from selenium import webdriver

from selenium.webdriver.common.by import By

browser = webdriver.Chrome()

link = "https://SunInJuly.github.io/execute\_script.html"

browser.get(link)

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

button.click()

Теперь вы можете сами посмотреть на эту [страницу](https://suninjuly.github.io/execute_script.html) и увидеть, что огромный футер действительно перекрывает нужную нам кнопку. Футером (footer) называется нижний блок, который обычно одинаков для всех страниц сайта. Чтобы понять, как решить эту проблему, нужно разобраться, 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.  
Делается это с помощью следующего скрипта:

"return arguments[0].scrollIntoView(true);"

Мы дополнительно передали в метод scrollIntoView аргумент true, чтобы элемент после скролла оказался в области видимости. Другие возможные параметры метода можно посмотреть здесь: <https://developer.mozilla.org/ru/docs/Web/API/Element/scrollIntoView>

В итоге, чтобы кликнуть на перекрытую кнопку, нам нужно выполнить следующие команды в коде:

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

browser.execute\_script("return arguments[0].scrollIntoView(true);", button)

button.click()

В метод execute\_script мы передали текст js-скрипта и найденный элемент button, к которому нужно будет проскроллить страницу. После выполнения кода элемент button должен оказаться в верхней части страницы. Подробнее о методе см <https://developer.mozilla.org/ru/docs/Web/API/Element/scrollIntoView> .

Также можно проскроллить всю страницу целиком на строго заданное количество пикселей. Эта команда проскроллит страницу на 100 пикселей вниз:

browser.execute\_script("window.scrollBy(0, 100);")

!Важно. Мы не будем в этом курсе изучать, как работает JavaScript, и обойдемся только приведенным выше примером скрипта с прокруткой страницы. Для сравнения приведем скрипт на этом языке, который делает то же, что приведенный выше пример для WebDriver:

// javascript

button = document.getElementsByTagName("button")[0];

button.scrollIntoView(true);

Можете попробовать исполнить его в консоли браузера на странице <http://suninjuly.github.io/execute_script.html>. Для этого откройте инструменты разработчика в браузере, перейдите на вкладку **консоль (console)**, скопируйте туда этот код и нажмите Enter. Таким образом можно протестировать кусочки js кода прежде чем внедрять его в свои тесты на python.

Обратите внимание, что в коде в WebDriver нужно использовать ключевое слово **return**. Также его нужно будет использовать, когда вы захотите получить какие-то данные после выполнения скрипта. При этом при тестировании скрипта в консоли браузера слово **return** использовать не надо.

## Задание на execute\_script

В данной задаче вам нужно будет снова преодолеть капчу для роботов и справиться с ужасным и огромным футером, который дизайнер всё никак не успевает переделать. Вам потребуется написать код, чтобы:

1. Открыть страницу [https://SunInJuly.github.io/execute\_script.html](http://suninjuly.github.io/execute_script.html).
2. Считать значение для переменной x.
3. Посчитать математическую функцию от x.
4. Проскроллить страницу вниз.
5. Ввести ответ в текстовое поле.
6. Выбрать checkbox "I'm the robot".
7. Переключить radiobutton "Robots rule!".
8. Нажать на кнопку "Submit".

Если все сделано правильно и достаточно быстро (в этой задаче тоже есть ограничение по времени), вы увидите окно с числом. Отправьте полученное число в качестве ответа для этого задания.

Для этой задачи вам понадобится использовать метод execute\_script, чтобы сделать прокрутку в область видимости элементов, перекрытых футером.

## 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 своей 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)**.

## Задание: загрузка файла

В этом задании в форме регистрации требуется загрузить текстовый файл.

Напишите скрипт, который будет выполнять следующий сценарий:

1. Открыть страницу <http://suninjuly.github.io/file_input.html>
2. Заполнить текстовые поля: имя, фамилия, email
3. Загрузить файл. Файл должен иметь расширение .txt и может быть пустым
4. Нажать кнопку "Submit"

Если все сделано правильно и быстро, вы увидите окно с числом. Отправьте полученное число в качестве ответа для этого задания.