

Оглавление

Документация и другие ресурсы вне курса	5
1 Начало	6
1.1 Введение в Selenium	6
1.2 Основные понятия в Selenium WebDriver	8
1.3 Преимущества Selenium	8
1.4 Недостатки Selenium	8
2 Установка Selenium и WebDriver	9
1) Установка WebDriver	9
2) Добавление webdriver в PATH в Windows:	9
3. Поиск элементов Selenium	12
3.1.1 Поиск элементов Selenium	12
<i>Два набора методов selenium</i>	12
Локаторы - Играют очень важную роль при работе с Selenium. Они обеспечивают путь к веб-элементам, которые необходимы для автоматизации определенных действий, таких как клик, ввод, установка флага и др.	12
3.1.2 Поиск элементов на странице	13
Поиск по <code>#id</code>	13
Поиск по <code>.class</code>	14
Поиск по имени тега	14
Поиск по значению атрибута <code>name="item"</code>	15
3.1.3 Поиск по составным селекторам	16
Потомки или дочерние элементы	16
Поиск по порядковому номеру дочернего элемента	17
Использование двух классов и более	17
3.1.4 Поиск элементов при помощи XPath	19
1. XPath всегда начинается с символа / или //	19
2. Символ [] - Это команда фильтрации	19
3. Символ * - Команда выбора всех элементов	20
4. Поиск по классу в XPath зависит от регистра	20
3.2 Работаем с браузером	22
Если код падает с ошибкой, весь код после <code>finally:</code> будет гарантированно выполнен	22
Но есть ещё третий способ, мой любимый, - это менеджер контекста with/as. С этим способом нам вообще не нужно думать о том, когда закрывать браузер, менеджер контекста делает это за нас в тот момент, когда это нужно.	22
Некоторые проблемы WebDriver (из сети и личного опыта):	23
3.3 Разница основных методов поиска элемента	24
<code>.find_element()</code> и <code>find_elements()</code>	24
Как всё таки быть, если нам нужен каждый второй или третий элемент на странице ?	25

3.4 Задачи по материалу	27
3.4.1	27
3.4.2.....	27
3.4.3 Сопоставьте значения из двух списков.....	27
3.4.4 Введите численный ответ	27
3.4.5 Введите численный ответ	28
3.4.6.....	28
3.4.7.....	29
3.4.8.....	29
3.4.9.....	30
3.4.10.....	31
3.4.11.....	32
4. Опции и аргументы.....	32
4.1 Запуск браузера с расширениями	32
4.2 Запуск браузера в скрытом режиме	36
Преимущества запуска браузера в фоновом режиме	36
4.3 Перенос профиля с основного браузера Chrome в браузер под управлением Selenium.....	37
4.4 Proxy и Selenium.....	38
5. Основные методы	40
5.1 Основные методы Selenium.....	40
5.2 Cookies	42
Куки чаще всего используются для:.....	42
Существует два вида cookie:.....	42
5.3 Cookies на практике	42
.get_cookies ()	42
.get_cookie(name_cookie)	43
5.4 Добавление cookie.....	45
5.5 Запуск JavaScript на странице.....	47
5.6 Задачи по материалу	49
5.6.1 Задача	49
5.6.2.....	49
5.6.3.....	49
5.6.4.....	50
5.6.5 Задача	50
5.6.6 Перекрытие элементов. Запуск JS.....	51
6. Скроллинг страниц	52
6.1 способ 1 <code>execute_script()</code> Прокрутка страницы	52
<code>document.body.scrollHeight</code>	52

<code>window.innerHeight</code>	53
<code>.scrollTo(0,document.body.scrollHeight)"</code>	53
6.2 способ 2 <code>Keys()</code> Прокрутка страницы.....	53
6.3 способ 3 <code>ActionChains()</code> Прокрутка содержимого страницы.....	57
6.4 Способ 4 <code>scroll_by_amount()</code>	59
6.5 Задачи по материалу.....	60
6.5.1.....	60
6.5.2.....	60
6.5.3.....	61
6.5.4.....	61
6.5.5.....	62
7. Окна и вкладки.....	64
7.1 Вкладки в браузере	64
Получаем title вкладки	67
7.2 Размеры окна браузера.....	68
<code>.set_window_size()</code>	68
<code>.get_window_size()</code>	68
7.3 Модальные окна.....	70
Виды модальных окон.	70
Модальное окно Alert.....	71
Модальное окно Prompt.....	71
Модальное окно Confirm.....	72
7.4 Задачи по материалу	72
7.4.1	72
7.4.2.....	73
7.4.3.....	73
7.4.4.....	74
7.4.5.....	74
7.4.6.....	75
7.4.7.....	76
7.4.8.....	76
8.Ожидания. Явное и неявное	78
8.1 Явное и неявное ожидание, Selenium Waits (Implicit Waits)	78
8.2 Методы ожиданий.....	80
8.3 Ожидание заголовка <code>.title_is(title)</code> и <code>.title_contains(title)</code>	82
<code>.title_is(title)</code>	82
<code>.title_contains(title)</code>	82

8.4 Задачи.....	83
8.4.1.....	83
8.4.2.....	83
8.4.3.....	84
8.4.4.....	84

Документация и другие ресурсы вне курса

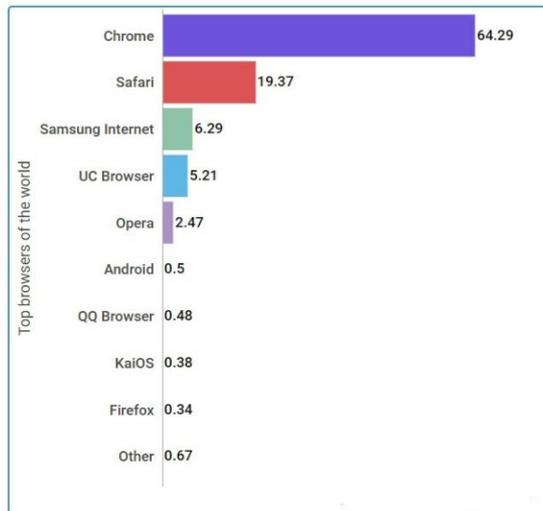
Неофициальная документация <https://selenium-python.readthedocs.io/getting-started.html>

Его перевод <https://habr.com/ru/post/250921/>

1.1 Введение в Selenium

Selenium - это бесплатная и открытая библиотека для автоматизированного тестирования веб-приложений и автоматизации действий в браузере.

Selenium - поддерживает все основные браузеры, но в этом курсе, мы будем говорить только про браузер **Chrome** т.к. он является самым популярным браузером на планете. К тому же использование других браузеров отличается только способом установки **webdriver**'а.

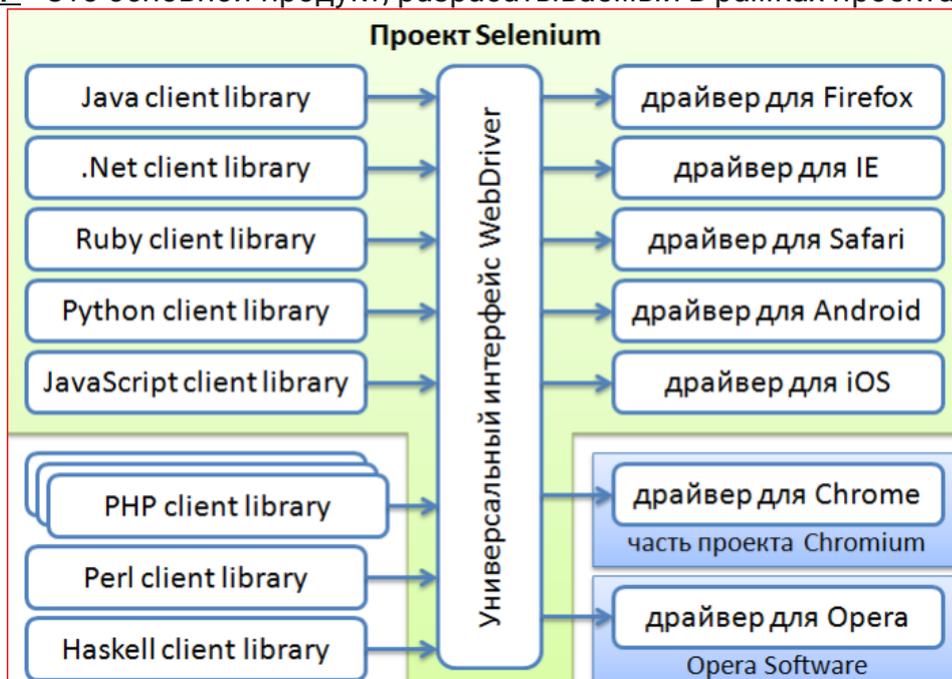


Парсеры на **Selenium** могут быть написаны на нескольких языках программирования, Java, C#, Python, JavaScript и Ruby и Kotlin. Нам же конечно интересен язык **Python**, с которым мы умеем обращаться.

Сценарные языки программирования, такие как **Ruby** и **Python** больше подходят для скриптов, чем компилируемые языки, такие как C# и Java.

Selenium - это программная библиотека для управления браузерами. Часто употребляется так же более короткое название **WebDriver**. На самом деле целое семейство драйверов для различных браузеров, а так же, набор библиотек на разных языках, позволяющих работать с этими драйверами.

Webdriver - Это основной продукт, разрабатываемый в рамках проекта Selenium.



В семейство Selenium входят такие вещи как.

- Selenium RC – это предыдущая версия библиотеки для управления браузерами.
- Selenium Server – это сервер, который **позволяет управлять браузером с удалённой** машины, по сети.
- Selenium Grid – это кластер, состоящий из нескольких Selenium-серверов.
- Selenium IDE – плагин к браузеру Firefox, который **может записывать действия пользователя**
- Selenium WebDriver – библиотека для управления браузерами.

Дополнительно (прогуглин, вне этого курса): <https://habr.com/ru/post/152653/>
[Как установить Селениум / Selenium IDE в браузер Хром](#)
[Видео 40. Автоматизация тестирования. Selenium IDE \(часть 1\)](#)
[Видео 40. Автоматизация тестирования. Selenium IDE \(часть 2\)](#)

К счастью, в этом курсе мы не будем говорить об этом. Нас интересует только последний пункт, **Selenium Webdriver** для браузера **Chrome**.

1.2 Основные понятия в Selenium Webdriver

- **Webdriver** - самая важная сущность, ответственная за управление браузером. Основной ход скрипта строится именно вокруг экземпляра этой сущности;
- **Webelement** - вторая важная сущность, представляющая собой абстракцию над веб-элементом (кнопки, ссылки, поля ввода и др.). **Webelement** ни что иное, как **DOM** объекты, находящиеся на веб странице;
- **By** - класс, содержащий статические методы для идентификации элементов, о которым подробнее мы будем говорить в следующих степах.

1.3 Преимущества Selenium

- Selenium** + бесплатный продукт с открытым исходным кодом;
- Selenium** + Очень гибкий инструмент, мы можем писать поистине большие скрипты, которые будут собирать информацию по заданным нами алгоритмам с множества сайтов;
- Selenium** + Разрабатывается с 2004 года. За это время он стал самым популярным инструментом для тестирования веб-приложений;
- Selenium** + Имеет огромное комьюнити, как в России, так и за её границами;
- Selenium** + В связи с тем, что инструмент очень популярен, по любому запросу в гугле есть масса информации и гайдов, но нигде больше нет таких задачек, как на этом курсе 😊.

1.4 Недостатки Selenium

- Selenium** - довольно сложная установка, если вы, как разработчик, всё делаете на автомате, то ваш клиент с фриланс биржи может не разобраться с тем, как обновить **webdriver**, при очередном автоматическом обновлении **Chrome**;
- Selenium** - подходит для работы только с веб-сайтами;
- Selenium** - во время работы используется **Chrome**, который может поглощать для своих нужд огромное количество оперативной памяти, на слабых машинах могут наблюдаться повисание. Поэтому важно учитывать это, и давать браузеру время прогрузить всё содержимое страницы;
- Selenium** - на собственном опыте замечено, что **один и тот же скрипт, может не запускаться с 1го раза**, при повторном запуске всё начинает работать как нужно. Но вот эту особенность тоже стоит иметь ввиду.

2 Установка Selenium и WebDriver

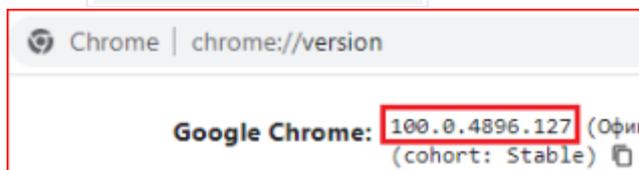
Установка Selenium ничем не отличается от установки других модулей в python.

Установка: `pip install selenium`

А вот что касается **WebDriver**, тут дела обстоят немного сложнее.

1) Установка WebDriver

Для начала нам необходимо выяснить версию вашего браузера **chrome**, для этого введите в строке поиска поиска `chrome://version/`



Версия браузера `100.0.4896.127`, далее переходим по [ссылке](#) чтобы скачать сам **WebDriver**

Важно! Скачать webdriver необходимо той версии что и ваш браузер.



Даже если у вас **windows x64** то качайте **chromedriver_win32.zip** - будет работать, главное чтоб версия браузера совпала.

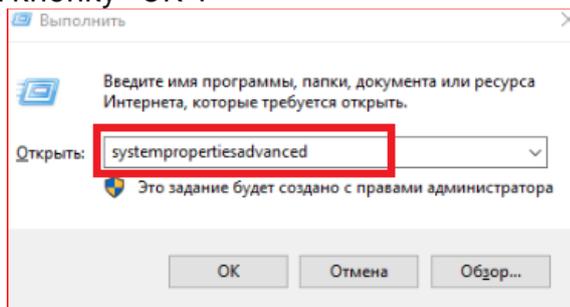
Итак, когда мы скачали **chromedriver_win32.zip** распакуем архив на диск **C:/** и переименуем, чтобы путь выглядел так **C:\chromedriver\chromedriver.exe**

2) Добавление webdriver в PATH в Windows:

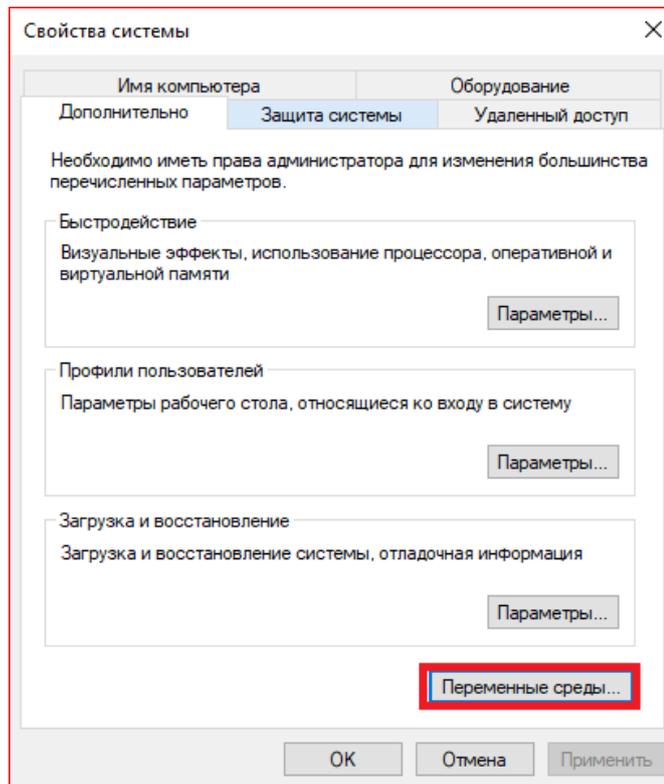
Чтобы добавить **webdriver** в переменные среды Windows:

1. Нажмите на клавиши «Win» + «R».

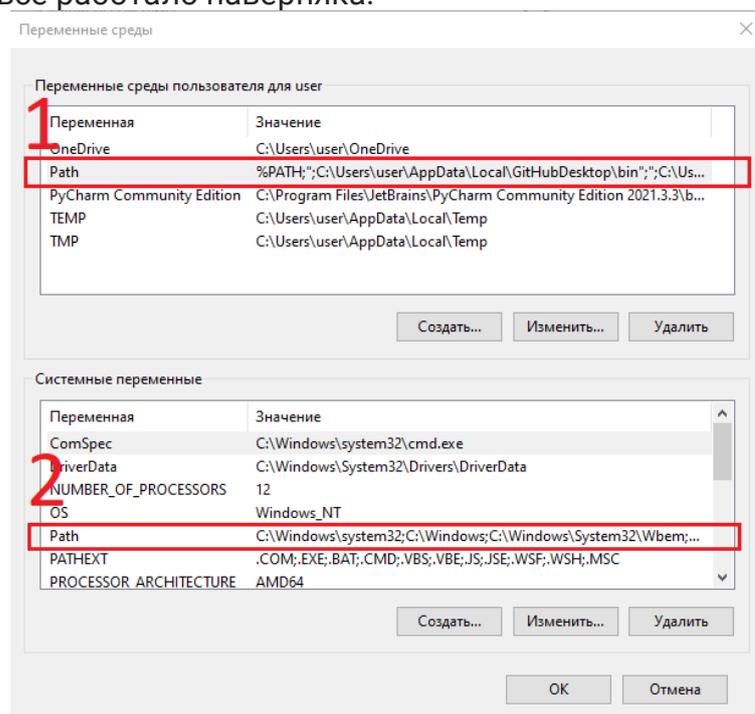
2. В окне «Выполнить» введите команду: «**systempropertiesadvanced**» (без кавычек), а затем нажмите на кнопку «OK».



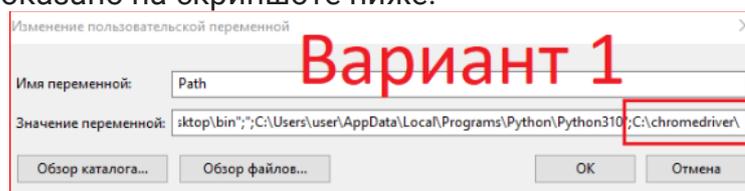
3. В окне «Свойства системы», во вкладке «Дополнительно» нажмите на кнопку «Переменные среды...»



4. В окне «Переменные среды» отображаются пользовательские переменные среды и системные переменные среды. Необходимо добавить путь к `C:\chromedriver\` в обе директории, чтобы всё работало наверняка.



5. Вариант 1) Добавить в конец списка после точки с запятой адрес к `C:\chromedriver\` как показано на скриншоте ниже.



6. Вариант 2) Нажмите кнопку Создать и укажите путь к `C:\chromedriver\` как показано на скриншоте ниже.

3. Поиск элементов [Selenium](#)

3.1.1 Поиск элементов Selenium

Вспоминаем урок [Поиск элементов на странице](#), (это из другого курса) Перечитайте его чтобы вспомнить содержимое, и пусть он будет открыт во второй вкладке чтобы подглядывать.

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

Два набора методов selenium

Для поиска нужного нам элемента на странице, мы можем использовать два набора методов selenium, я продемонстрирую оба, а **вы решите, какой вам больше нравится** (мое предпочтение второму набору методов).

- `.find_element_by_id("tag")` – поиск по уникальному атрибуту `id` элемента. Лучше использовать именно поиск по `id`, т.к. мы знаем что на странице может быть только 1 элемент с уникальным `id` этот поиск является самым стабильным;

- `.find_element_by_css_selector("tag")` – используйте этот способ, когда хотите получить элемент с использованием синтаксиса CSS-селекторов;

- `.find_element_by_xpath("path")` – поиск с помощью языка запросов XPath, позволяет выполнять очень гибкий поиск элементов, **одно из веских оснований использовать XPath заключено в наличии ситуаций, когда** на странице отсутствуют пригодные в качестве указателей атрибуты, такие как `id` или `name`;

- `.find_element_by_name("tag")` – используйте этот способ, когда известен атрибут `name` элемента. Результатом будет первый элемент с искомым значением атрибута `name`;

- `.find_element_by_tag_name("tag")` – поиск элемента по названию тега элемента;

- `.find_element_by_class_name("tag")` – поиск по значению атрибута `class`;

- `.find_element_by_link_text("tag")` – используйте этот способ, когда известен текст внутри тега;

- `.find_element_by_partial_link_text("tag")` – поиск ссылки на странице, если текст селектора совпадает с любой частью текста ссылки;

Перед тем как пользоваться этим набором методов, стоит отметить, что он немного устарел, хотя до сих пор всё работает, об устаревании намекает

```
PyCharm, button = browser.find_element_by_id("sale_button")
```

```
G:\Мой диск\stepik\selenium\main.py:12: DeprecationWarning: find_element_by_* commands are deprecated. Please use find_element() instead
button = browser.find_element_by_id("sale_button")
```

Второй набор методов поиска называется локаторами и полностью эквивалентен первому набору методов по функционалу.

Перед использованием локатора нам необходимо его импортировать:

```
from selenium.webdriver.common.by import By
```

Локаторы - Играют очень важную роль при работе с Selenium. Они обеспечивают путь к веб-элементам, которые необходимы для автоматизации определенных действий, таких как клик, ввод, установка флага и др.

- `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.

Локаторы мы используем с помощью двух универсальных методов `find_element()` который возвращает ровно один элемент который был найден первым, и второй `find_elements()` который возвращаем список найденных элементов.

Сравним два способа. Найдём на [странице](#), кнопку **Купить** с `id="sale_button"` и совершим по ней клик.

Вариант 1. `.find_element_by_id("sale_button")`

```
from selenium import webdriver
```

```
browser = webdriver.Chrome()
```

```
browser.get('http://parsinger.ru/html/watch/1/1_1.html')
```

```
button = browser.find_element_by_id("sale_button").click()
```

Вариант 2. `.find_element(By.ID, "sale_button")`

```
from selenium import webdriver
```

```
from selenium.webdriver.common.by import By
```

```
browser = webdriver.Chrome()
```

```
browser.get('http://parsinger.ru/html/watch/1/1_1.html')
```

```
button = browser.find_element(By.ID, "sale_button").click()
```

Думаю разница понятна, выбирайте какой из методов поиска выбрать лично вам, и поехали дальше.

3.1.2 Поиск элементов на странице

ps. Все примеры выполнены в браузере Chrome.

Элементы можно находить при помощи тегов, селекторов css, атрибутов и значения атрибута.

Например:

- `id="name_id"` - по идентификатору;
- `class="name_class"` - по имени класса;
- `div` - по имени тега;
- `href="link"` - по атрибуту;
- `name="item"` - по значению атрибута.

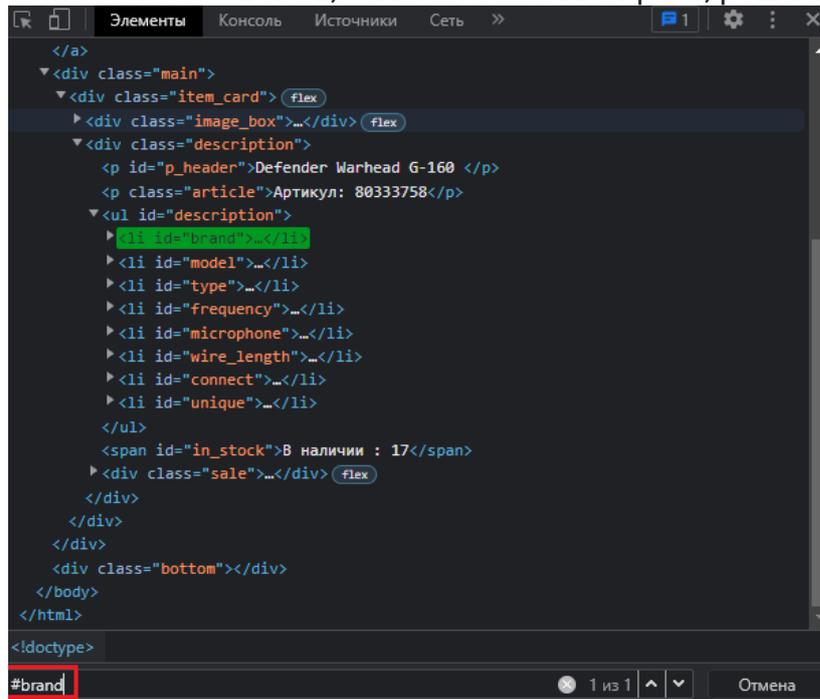
Откройте [страницу](#) в своём браузере и откройте инструмент разработчика **f12**, затем откройте панель поиска по дереву HTML командой **ctrl+f**, для закрепления знаний мы будем тренироваться искать элементы в браузере, затем в разделе selenium мы перейдём к написанию парсеров, используются знания из этого модуля.

Поиск по `#id`

`#` - базовый селектор, который помогаем выбирать элемент по значению атрибута `id`, не забывайте что `id` являются уникальным объектом на странице, т.е. `id` может быть использован только для одного тега в HTML документа.

У нас на странице есть определённое количество тегов с уникальными `id`, для поиска по `id` используется синтаксис `#значение_атрибута` т.е в нашем случае `#brand`. Потренируйтесь писать поиск по `id` самостоятельно на других тегах, так же можно использовать аналогичный синтаксис с квадратными

скобками, `[id='brand']` сделает то же самое что и `#brand`, только потребует написания большего количества символов, какой способ выбирать, решать вам.

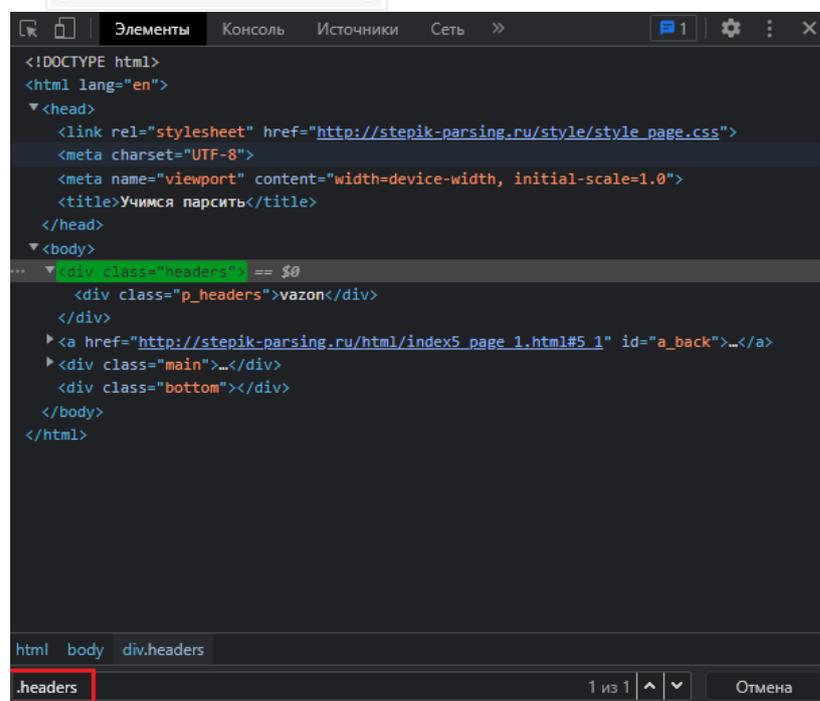


```
</a>
<div class="main">
  <div class="item_card"> flex
    <div class="image_box">...</div> flex
    <div class="description">
      <p id="p_header">Defender Warhead G-160 </p>
      <p class="article">Артикул: 80333758</p>
      <ul id="description">
        <li id="brand">...</li>
        <li id="model">...</li>
        <li id="type">...</li>
        <li id="frequency">...</li>
        <li id="microphone">...</li>
        <li id="wire_length">...</li>
        <li id="connect">...</li>
        <li id="unique">...</li>
      </ul>
      <span id="in_stock">В наличии : 17</span>
      <div class="sale">...</div> flex
    </div>
  </div>
  <div class="bottom"></div>
</body>
</html>
<!doctype>
#brand
```

Поиск по `.class`

`.` - базовый селектор который помогает выбрать элемент по имени класса
Аналогично мы ищем элементы по классу используя знак

точки. `.имя_класса` или `.description`, используя поиск по классу, мы находим все элементы с данным классом. Мы с таким же успехом можем использовать синтаксис с квадратными скобками `[class="headers"]`



```
<!DOCTYPE html>
<html lang="en">
  <head>
    <link rel="stylesheet" href="http://stepik-parsing.ru/style/style_page.css">
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Учимся парсить</title>
  </head>
  <body>
    <div class="headers"> == $0
      <div class="p_headers">vazon</div>
    </div>
    <a href="http://stepik-parsing.ru/html/index5_page_1.html#5_1" id="a_back">...</a>
    <div class="main">...</div>
    <div class="bottom"></div>
  </body>
</html>
html body div.headers
.headers
```

Поиск по имени тега

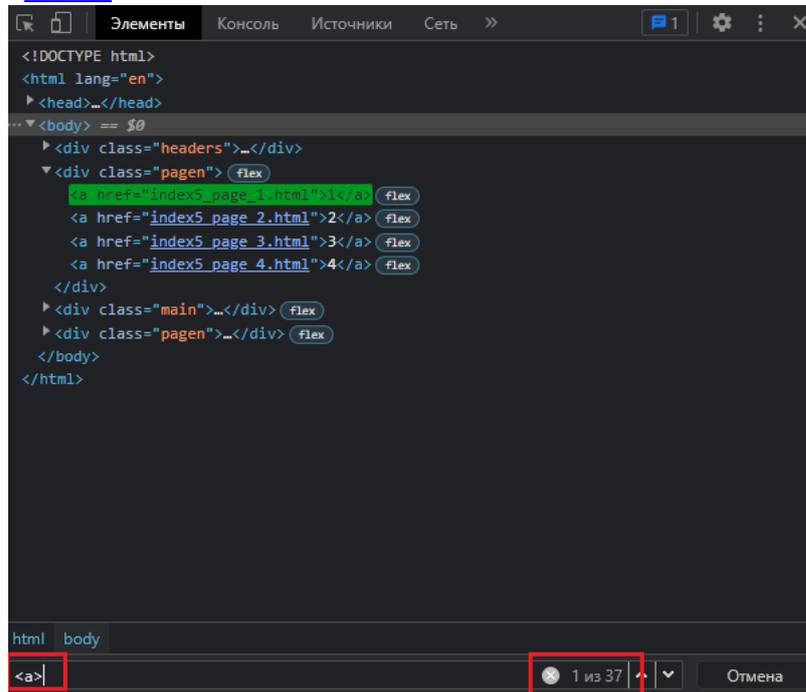
Самый простой способ найти любой тег просто написать его в строке поиска.

Например, `headers` или `div`, Такой способ мы можем использовать когда нам нужно собрать все элементы на странице разом, к примеру все абзацы. **Важное примечание:** Если

искать тег, который состоит из одного символа, к примеру тег `a` или тег `p` вы будете находить не только теги, но и все буквы которые есть на странице `<meta name="viewport"`

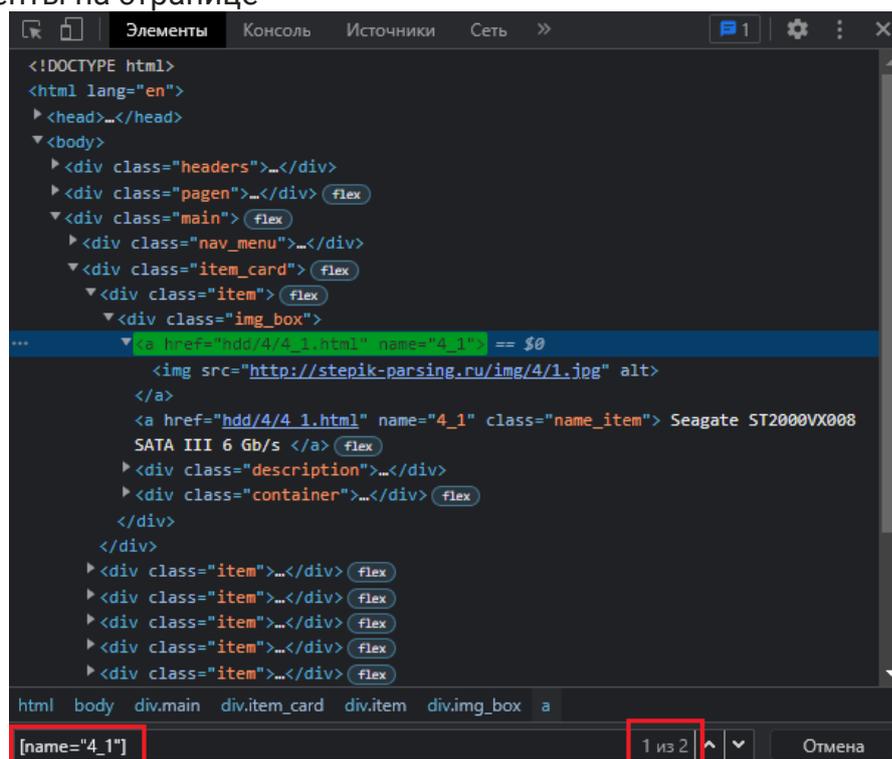
Чтобы действительно найти все ссылки на странице, нужно использовать поиск с угловыми скобками `<a>` или `<p>`

ps. Скрин ниже с [сайта](#)



Поиск по значению атрибута `name="item"`

В этом степе уже упоминалось о квадратных скобках, и показывал что можно искать `class` и `id` используя квадратные скобки, но на самом деле можно искать таким образом любые теги с любыми атрибутами и их значениями. Например, найти элемент с `name="4_1"` очевидно мы могли бы применить квадратные скобки `[name="4_1"]` и были бы совершенно правы. Но стоит помнить что поиск по значению атрибута, находит все искомые элементы на странице



3.1.3 Поиск по составным селекторам

Мы почти всегда можем обратиться к необходимому элементу напрямую и получить его данные. Но бывает к примеру есть два одинаковых элемента с одинаковым классом и с разными родителями, и мы хотим получить какой-то определённый элемент.

```
3 <div class="author">
4   <p class="text">Лев Толстой</p>
5 </div>
6
7 <div class="name_book">
8   <p class="text">Война и мир</p>
9 </div>
```

В этом примере оба тега `<p>` имеют одинаковый класс `"text"`. Для того чтобы получить тег у которого родитель с классом `"author"` нам и потребуется составной селектор.

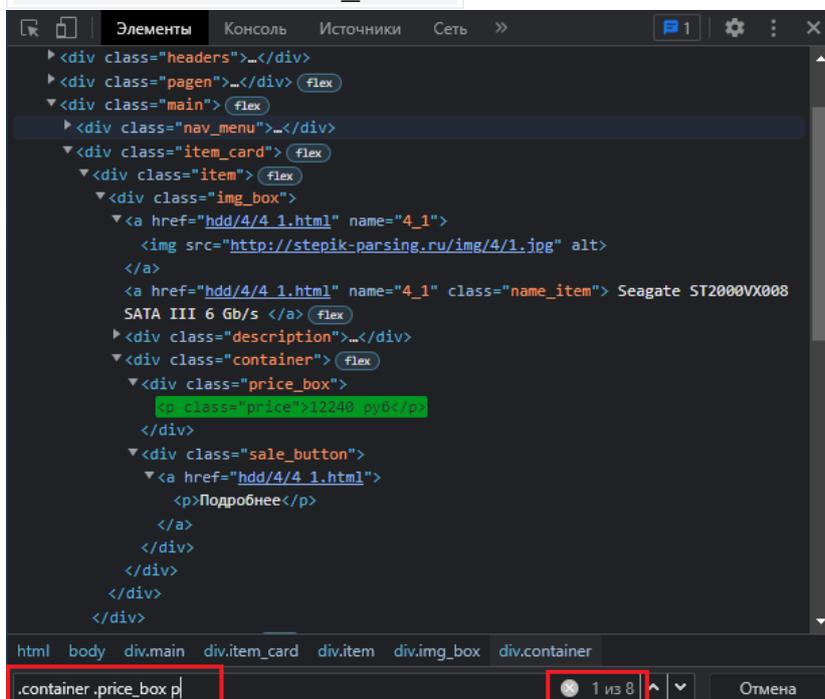
`' '` (пробел) выбирает элементы, которые находятся **внутри указанного элемента (вне зависимости от уровня вложенности)**.

`.author .text` - это означает что мы ищем `class` с значением атрибута `author`, а пробел указывает что нужно искать внутри этого тега, тег с классом `"text"`.

Элемент `.text` потомок элемента `.author`, потомок может находиться на любом уровне вложенности от элемента `.author` и между ними может быть сколько угодно других тегов.

Или второй пример, мы хотим извлечь конкретный элемент при помощи составного селектора

`.container .price_box p` [сайт](#)

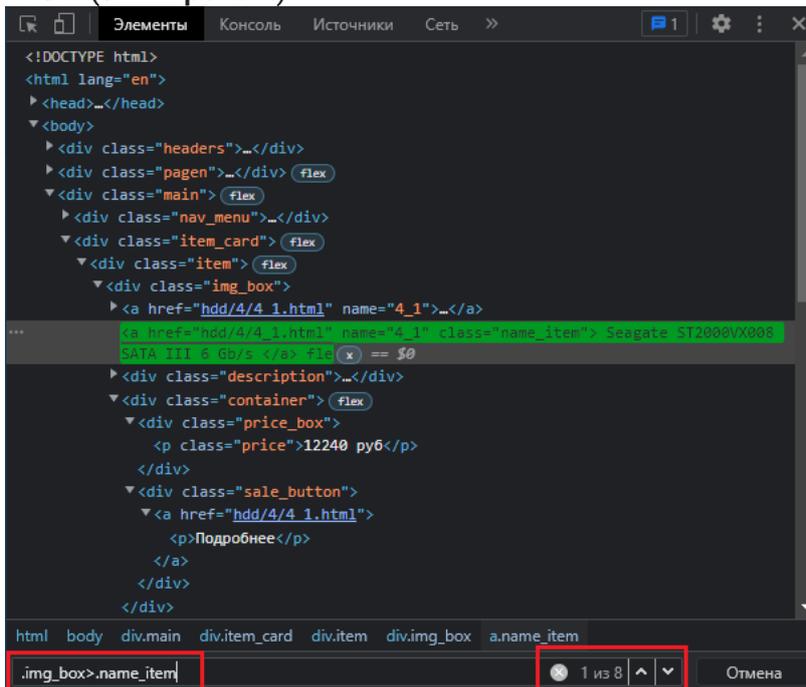


Как видим, мы получаем все элементы которые соответствуют критериям поиска. Глубина вложенного селектора может быть любой.

Потомки или дочерние элементы

`'>'` в отличие от пробела выбирает только те элементы, которые являются дочерними непосредственно по отношению к указанному элементу.

`.img_box>.name_item` - такой способ помогает найти дочерний элемент от его родителя. Т.е. Элемент с классом `.name_item` дочерний элементу `.img_box`, (англ. **child**), а `.img_box` для `.name_item` является родительским элементом (англ. **parent**)

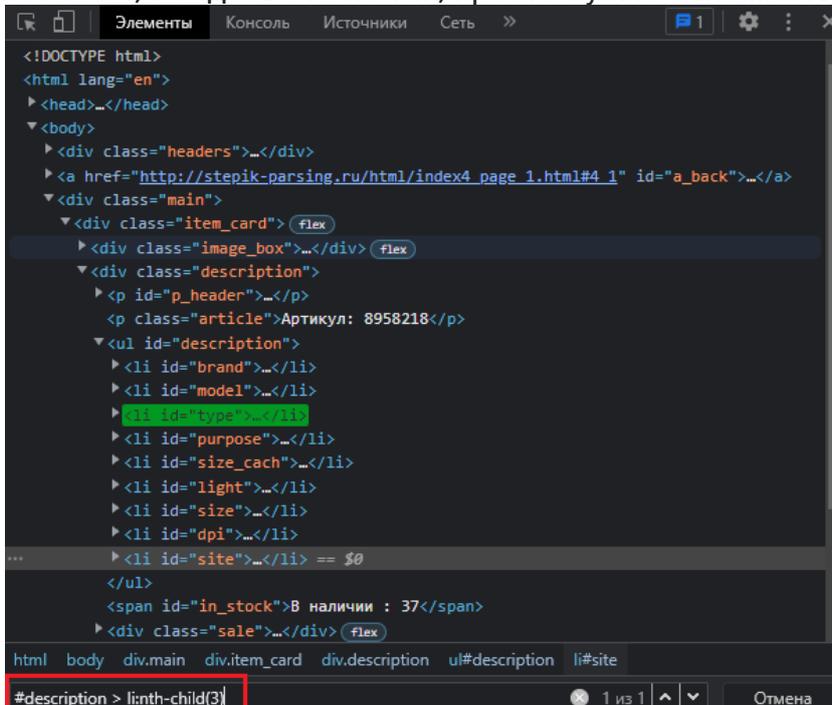


```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="headers">...</div>
    <div class="pagen">...</div>
    <div class="main">
      <div class="nav_menu">...</div>
      <div class="item_card">
        <div class="item">
          <div class="img_box">
            <a href="hdd/4/4 1.html" name="4_1">...</a>
            <a href="hdd/4/4 1.html" name="4_1" class="name_item"> Seagate ST2000VX008
            SATA III 8 6B/s </a>
          </div>
          <div class="description">...</div>
          <div class="container">
            <div class="price_box">
              <p class="price">12240 руб</p>
            </div>
            <div class="sale_button">
              <a href="hdd/4/4 1.html">
                <p>Подробнее</p>
              </a>
            </div>
          </div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Можете сами в этом убедиться, откройте [сайт](#) и поэкспериментируйте.

Поиск по порядковому номеру дочернего элемента

`#description > li:nth-child(3)` ([сайт](#)) - такой способ весьма полезен. `:nth-child()` может находить элемент основываясь на его позиции в списке элементов, в примере ниже мы нашли элемент под номером 3, возможно вы привыкли что счёт начинается с 0, то здесь это не так, просто нужно запомнить это.



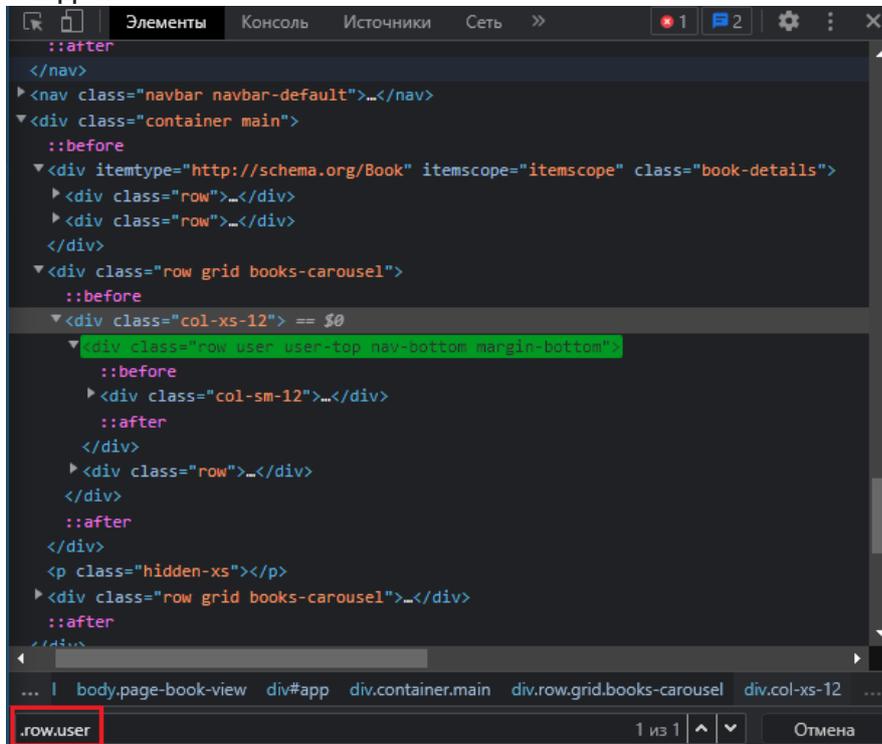
```
<!DOCTYPE html>
<html lang="en">
  <head>...</head>
  <body>
    <div class="headers">...</div>
    <a href="http://stepik-parsing.ru/html/index4_page 1.html#4 1" id="a_back">...</a>
    <div class="main">
      <div class="item_card">
        <div class="image_box">...</div>
        <div class="description">
          <p id="p_header">...</p>
          <p class="article">Артикул: 8958218</p>
          <ul id="description">
            <li id="brand">...</li>
            <li id="model">...</li>
            <li id="type">...</li>
            <li id="purpose">...</li>
            <li id="size_cach">...</li>
            <li id="light">...</li>
            <li id="size">...</li>
            <li id="dpi">...</li>
            <li id="site">...</li>
          </ul>
          <span id="in_stock">В наличии : 37</span>
          <div class="sale">...</div>
        </div>
      </div>
    </div>
  </body>
</html>
```

Использование двух классов и более

В примере ниже родительские теги имеют сразу 2 класса, их может быть и больше. Чтобы найти элементы у которых есть нужные нам классы, мы можем использовать: `.main.author` - такой составной селектор используется до поиска элемента с двойным классом.

```
2
3 <div class="main author">
4   <p class="text">Лев Толстой</p>
5 </div>
6
7 <div class="main2 name_book">
8   <p class="text">Война и мир</p>
9 </div>
```

Если классов будет больше чем мы указали в составном селекторе, то элемент будет найден.



(этот пример не работает на сайте тренажёре, т.к. размещён тут для наглядности.)

Искомый элемент имеет 5 классов `row user user-top nav-bottom margin-bottom`, мы находим элемент всего по двум классам. Важно помнить об этой особенности и указывать более конкретный составной селектор, чтобы в наш поиск не попадали лишние элементы.

3.1.4 Поиск элементов при помощи XPath

Когда мы не можем найти уникальный селектор для необходимого нам элемента, на помощь приходит поиск по XPath, поиск атрибута по его пути.

XPath - очень мощный и гибкий инструмент, **в то же время и очень опасный**.

1. Во-первых, **XPath** помогает писать сложные запросы поиска элементов.
2. Во-вторых, если разработчик сайта вдруг решил изменить порядок элементов на странице, то весь код парсера может сломаться, т.к. искомый элемент будет не найден.

XPath - оспользует и древовидную структуру документа. Проверять **XPath** запросы можно точно так же как и простые **CSS** селекторы - в консоли разработчика.

1. **XPath** всегда начинается с символа / или //

Символ / аналогичен символу > в CSS-селекторе, а символ // – пробелу.

- element1/element2 – выбирает элементы element2, являющиеся прямыми потомками element1;
- element1//element2 – выбирает элементы element2, являющиеся потомками element1 любой степени вложенности.

Разница состоит в том, что в **XPath**, когда мы начинаем запрос с символа /, мы должны указать элемент, являющийся корнем нашего документа. Корнем всегда будет элемент с тегом `<html>`. Пример: `/html/body/header`

Мы можем начинать запрос и с символа //. Это будет означать, что мы хотим найти всех потомков корневого элемента без указания корневого элемента. В этом случае для поиска того же хедера, мы можем выполнить запрос `//header`, так как других заголовков у нас нет.

2. Символ [] - Это команда фильтрации

Если по запросу найдено несколько элементов, то будет произведена фильтрация по правилу, указанному в скобках.

Правил фильтрации очень много:

- по любому атрибуту, будь то `id`, `class`, `title` (или любой другой).

Например, мы хотим найти ссылку с, для этого можно выполнить запрос на [сайте](#) `//a[@id='a_back']`

```
a href="http://stepik-parsing.ru/html/index1_page_1.html#1_1" id="a_back"
```

- по порядковому номеру. Допустим, мы хотим выбрать пятый по порядку элемент `li`. Для этого найдем элемент с классом `ul` и возьмем его пятого потомка: `//ul/li[5]`

```
<ul id="description"> == $0
  <li id="brand">...</li>
  <li id="model">...</li>
  <li id="type">...</li>
  <li id="display">...</li>
  <li id="material_frame">...</li>
  <li id="material_bracer">...</li>
  <li id="size">...</li>
  <li id="site">...</li>
</ul>
```

- по полному совпадению текста. Да, **XPath** – это **единственный способ найти элемент на сайте по внутреннему тексту**. Если мы хотим найти кнопку купить, можно воспользоваться **XPath** селектором `//button[text()='Купить']`. Такой селектор вернет элемент, только если текст **полностью совпадает**. Здесь важно сказать, что не всегда поиск по тексту – это хорошая практика, особенно в случае если сайт мультязычный.

```
<span id="price">2310 руб</span>
<span id="old_price">3550 руб</span>
...
<button id="sale_button">Купить</button> == $0
</div>
html body div.main div.item_card div.description div.sale button
//button[text]="Купить"]
```

• по **частичному совпадению** текста или атрибута. Для этого нужна функция `contains`. Запрос `//p[contains(text(), "Артикул")]` вернет нам все абзацы текста, которые содержат слово `Артикул`. Точно так же можно искать по частичному совпадению других атрибутов, это удобно, если у элемента несколько классов. Посмотрите код [сайта](#) и найдите тег `ul`, а в нём тег `li` со значением атрибута `material_frame` и `material_bracer` их можно найти селектором `//li[contains(@id, "material")]`

```
<ul id="description">
  <li id="brand">...</li>
  <li id="model">...</li>
  <li id="type">...</li>
  <li id="display">...</li>
  <li id="material_frame">...</li>
  <li id="material_bracer">...</li> == $0
  <li id="size">...</li>
  <li id="site">...</li>
</ul>
html body div.main div.item_card div.description ul#description li#material_bracer
//li[contains(@id, "material")] 1 из 2
```

3. Символ * - Команда выбора всех элементов

• Например хотим найти текст на [сайте](#) в заголовке с ценой `//div/*[@class="price"]`. Это может быть удобно, когда мы не знаем точно тег элемента, который ищем. Будет найдено 8 элементов соответствующих нашему поиску.

```
<div class="price_box">
  <p class="price">2310 руб</p>
</div>
<div class="sale_button">...</div>
html body div.main div.item_card
//div/*[@class="price"] 1 из 8 Отмена
```

4. Поиск по классу в XPath зависит от регистра

Так же как и в случае поиска по CSS-селектором будьте внимательными к регистру при поиске по классам:

`//div/*[@class="Price"]` не найдет элемент на нашей странице.

```
<div class="price_box">
  <p class="price">2310 руб</p>
</div>
<div class="sale_button">...</div>
html body div.main div.item_card
//div/*[@class="Price"] 0 из 0 Отмена
```

Что важно знать про **XPath**, чтобы пользоваться им, безболезненно:

- Не используйте селекторы вида `//div[1]/div[2]/div[3]` без крайней нужды: по такому селектору невозможно с первого раза понять, что за элемент вы ищете. А когда структура страницы хоть немного изменится, то ваш парсер с большой вероятностью перестанет работать.
- Если есть возможность использовать CSS-селекторы: `class`, `id` или `name` — лучше использовать их вместо поиска по **XPath**.
- Можно искать по полному или частичному совпадению текста или любого атрибута.
- Можно удобно перемещаться по структуре документа (переходить к потомкам и к родителям);
- Подойдет, когда у сайта всё плохо с атрибутами.
- Не стоит использовать разные расширения для браузеров по поиску **XPath**: они подбирают нечитабельные и переусложненные селекторы. Лучше потратить немного времени и разобраться в синтаксисе самостоятельно, тем более, что он не очень сложный.

3.2 Работаем с браузером

Когда наш парсер отработает, мы бы хотели, чтобы он закрылся сам и тем самым корректно завершил свою работу. Но этого **может не произойти** по множеству причин. Поэтому мы должны указать браузеру на то, что он должен закрыть окно после завершения работы, командой `browser.quit()`. Важно закрывать окно, потому что при создании `webdriver.Chrome()` в то же время создаётся процесс в ОС, который продолжит висеть. Вместо того, чтобы закрывать окно браузера вручную и не засорять оперативную память, **следует закрывать окно командой** `.quit()`.

Расставим таймауты, чтобы видеть процесс выполнения кода и чтобы браузер не закрывался за мгновение.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

browser = webdriver.Chrome()
browser.get('http://parsinger.ru/html/watch/1/1_1.html')
button = browser.find_element(By.ID, "sale_button")
time.sleep(2)
button.click()
time.sleep(2)
browser.quit()
```

Если ошибка произойдёт во время выполнения кода до команды `.quit()`, сеанс **WebDriver** не будет закрыт должным образом и файлы не будут удалены из памяти.

А для того, чтобы гарантированно код завершил свою работу командой `browser.quit()`, мы будем использовать конструкцию `try/finally`.

Если код падает с ошибкой, весь код после `finally:` будет гарантированно выполнен.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

try:
    browser = webdriver.Chrome()
    browser.get('http://parsinger.ru/html/watch/1/1_1.html')
    button = browser.find_element(By.ID, "sale_button")
    time.sleep(2)
    button.click()
    time.sleep(2)
finally:
    browser.quit()
```

Но есть ещё **третий способ**, мой любимый, - это менеджер контекста `with/as`. С этим способом нам вообще не нужно думать о том, когда закрывать браузер, менеджер контекста делает это за нас в тот момент, когда это нужно.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
```

```
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/html/watch/1/1_1.html')
    button = browser.find_element(By.ID, "sale_button")
    time.sleep(2)
    button.click()
    time.sleep(2)
```

Выполните эти три примера у себя в терминале, чтобы посмотреть как всё работает. И вы увидите что всё работает одинаково, и не будет никакой разницы, поэтому выбирайте понравившийся вариант.

Так же есть два похожих метода, которые новички **иногда путают между собой**, о которых самое время поговорить. это `browser.close()` и `browser.quit()`, не глядя дальше по тексту сможете ответить в чём отличие?

`browser.close()` - Закрывает текущее окно браузера если во время работы вы открыли новое окно\вкладку

`browser.quit()` - Закрывает все окна, вкладки, процессы вебдрайвера которые были запущены во время сессии.

Некоторые проблемы WebDriver (из сети и личного опыта):

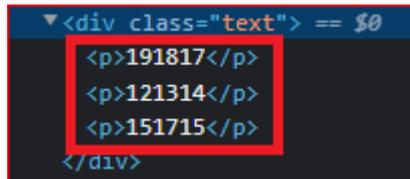
- Поведение Selenium может отличаться в разных браузерах;
- Иногда возникают сложности с поиском элементов (XPath и другие методы иногда просто не работают, хотя должны);
- Необъяснимые падения драйвера прямо посреди работы скрипта;
- Взаимодействие возможно только с активной вкладкой браузера, драйвер **позволяет открывать новые вкладки и новые окна, но не позволяет одновременно** в них работать.

3.3 Разница основных методов поиска элемента

`.find_element()` и `find_elements()`

Методы `.find_element()` и `find_elements()` вы будете использовать всегда при написании ваших парсеров с помощью `selenium`. Поэтому нужно понимать как с ними работать.

У нас есть [сайт](#), у которого структура дерева HTML очень простая. На странице есть 100 блоков `<div="text">`, в котором 3 тега `<p>` которые не имеют ни `class`, ни `id`. Мы решили что нам необходимо собрать каждый первый элемент `<p>`. мы могли бы пройти в цикле и использовать срезы, как мы делаем с простыми списками, наверняка подумали мы.



Давайте разбираться почему срезы не сработают.

`.find_element()` вернёт нам объект веб драйвера который не поддерживает срезы. Выполним следующий код.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'http://parsinger.ru/selenium/3/3.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    link = browser.find_element(By.CLASS_NAME, 'text')
    print(type(link))
```

```
>>> <class 'selenium.webdriver.remote.webelement.WebElement'>
```

Мы видим что возвращаемый тип объекта это экземпляр класса, который содержит в себе список элементов `<p>` в количестве 3шт, как показано на 1м скриншоте. Но это не простой список.

Давайте посмотрим на возвращаемый объект.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'http://parsinger.ru/selenium/3/3.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    link = browser.find_element(By.CLASS_NAME, 'text')
    print(link)
```

```
>>> <selenium.webdriver.remote.webelement.WebElement
(session="a7cb2c979d456b7cae336e5eafa4ad6b", element="c2651e50-25a3-41bb-ad8d-5a259929bbfe")>
```

Мы видим что возвращаемый объект, это элемент `selenium`, который между прочим, не поддерживает срезы. Давайте попытаемся получить элемент `[0]` у этого объекта, и посмотрим на результат.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
```

```
url = 'http://parsinger.ru/selenium/3/3.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    link = browser.find_element(By.CLASS_NAME, 'text')
    print(link[0])
```

```
>>> Traceback (most recent call last):
  File "G:\Мой диск\stepik\stepik_like\task_2.py", line 26, in <module>
    print(link[0])
TypeError: 'WebElement' object is not subscriptable
```

Получаем ошибку, которая нам говорит, что объект не может быть итерирован, так происходит потому что **все элементы** `<p>` **которые мы храним в этом объекте, являются одним целым**. Так происходит потому что Selenium не предоставляет методов работы с определённой частью элементов, в том числе и срезы.

А вот извлечь из этого объекта текст очень просто, достаточно применить к нему метод `.text`

```
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'http://parsinger.ru/selenium/3/3.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    link = browser.find_element(By.CLASS_NAME, 'text')
    print(link.text)
```

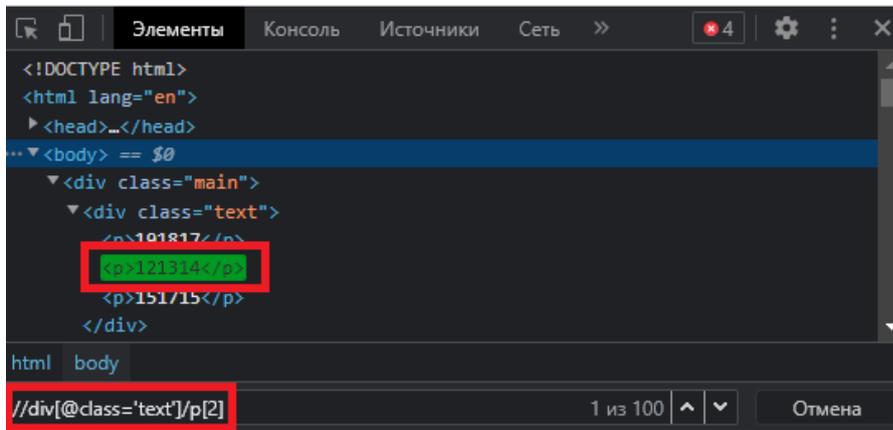
```
>>> 191817
    121314
    151715
```

`.find_element()` - Возвращает **первый** найденный элемент соответствующий нашим критериям поиска. (имеется ввиду элемент веб-драйвера, который содержит элемент DOM)

`.find_elements()` - Возвращает **все** найденные элементы, соответствующие критериям поиска и сохраняет результат в список `<class 'list'>`, список будет наполнен **НЕ** элементами `<p>`, а элементами веб драйвера, которые будут содержать в себе элементы DOM

Как всё таки быть, если нам нужен каждый второй или третий элемент на странице ? Мы всегда можем решить эту задачу при помощи XPath.

`.find_element(By.XPATH, "//div[@class='text']/p[2]")` - вернёт нам второй элемент `<p>`, первого найденного элемента `<div class="text">`.



`.find_elements(By.XPATH, "//div[@class='text']/p[2]")` - Соответственно вернёт все найденные элементы `<p>` расположенные на вторых позиции, во всех найденных `<div class="text">`



3.4 Задачи по материалу

3.4.1

Сопоставьте значения из двух списков

✓ Всё получилось!

By.ID	Поиск по атрибуту id элемента
By.CSS_SELECTOR	Поиск по css селектору
By.XPATH	Поиск элемента языком 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 точному совпадению текста
By.PARTIAL_LINK_TEXT	Поиск ссылки по частичному совпадению текста

3.4.2

Вставьте правильный импорт локатора `By`.
from selenium.webdriver.common.by import By

3.4.3 Сопоставьте значения из двух списков

browser.close()	Закрывает текущее окно браузера
browser.quit()	Закрывает все окна, вкладки, процессы

3.4.4 Введите численный ответ

Самое время выполнить простую задачку для закрепление пройденного материала. Суть задачи проста (у вас будет всего 5 секунд для того чтобы получить результат, поэтому подумайте над кодом)

1. Открыть [сайт](#) с помощью selenium;
2. Заполнить все существующие поля;
3. Нажмите на кнопку;
4. Скопируйте результат который появится рядом с кнопкой в случае если вы уложились в 5 секунд;
5. Вставьте результат в поле ниже.

Для заполнения полей вам потребуется метод `.send_keys("Текст")`, который мы применяем к каждому полю `input`, помните про универсальный метод `.find_elements()`, который возвращает список найденных элементов. Используйте этот метод для поиска всех полей.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By
with webdriver.Chrome() as browser:
    browser.get('https://stepik-parsing.ru/selenium/1/1.html')
```

```
input_form = browser.find_element(By.CLASS_NAME,
'form').send_keys('Text')
time.sleep(5)
```

Для решения задачи используйте цикл, чтобы обойти найденные элементы, методом `.find_element_s()`, и на каждой итерации к каждому полю применяйте метод `.send_keys("text")` для его заполнения, а метод `.click()` используйте чтобы нажать на кнопку. Не забудьте про модуль `time` чтобы установить задержки.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'http://parsinger.ru/selenium/2/2.html'

with webdriver.Chrome() as browser:
    browser.get(url)
    browser.find_element(By.LINK_TEXT, '16243162441624').click()
    result = browser.find_element(By.ID, 'result').text
    print(resu
```

3.4.5 Введите численный ответ

Вторая задачка тоже довольно проста. Вспоминаем метод `By.PARTIAL_LINK_TEXT` или `By.LINK_TEXT`, который ищет ссылку по частичному или полному совпадению текста.

Суть задачи.

1. Открываем [сайт](#) при помощи selenium;
2. Применяем метод `By.PARTIAL_LINK_TEXT` или `By.LINK_TEXT`;
3. Кликаем по ссылке с текстом **16243162441624**;
4. Результат будет ждать вас в теге `<p id="result"></p>` `<p id="result">`;
5. Скопируйте найденный результат в поле ниже.

p.s. Вы конечно можете вручную найти ссылку, при помощи простого поиска, но кого вы обманите?

РЕШЕНИЕ

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'https://parsinger.ru/selenium/2/2.html'

try:
    browser = webdriver.Chrome()
    browser.get(url)
    link_to_click = browser.find_element(By.PARTIAL_LINK_TEXT, '16243162441624')
    link_to_click.click()
    result = browser.find_element(By.ID, 'result')
    print(result.text)

finally:
    browser.quit()
```

3.4.6

1. Откройте [сайт](#);

2. Извлеките данные из каждого тега `<p>`;
3. Сложите все значения, их всего 300 шт;
4. Напишите получившийся результат в поле ниже.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('https://stepik-parsing.ru/selenium/3/3.html')

    result = [int(x.text) for x in browser.find_elements(By.XPATH,
"//div[@class='text']/p")]
    print(sum(result))
```

3.4.7

1. Откройте [сайт](#);
2. Извлеките данные из каждого **второго** тега `<p>`;
3. Сложите все значения, их всего 100 шт;
4. Напишите получившийся результат в поле ответа.

РЕШЕНИЕ

```
from selenium.webdriver.chrome.options import Options
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('https://stepik-parsing.ru/selenium/3/3.html')
    print(sum([int(x.text) for x in browser.find_elements(By.XPATH,
"//div[@class='text']/p[2]")])))
```

3.4.8

1. Откройте [сайт](#);
2. Установите все чек боксы в положение **checked** при помощи selenium и метода `click()`;
3. Когда все чек боксы станут активны, нажмите на кнопку;
4. Скопируйте число которое появится на странице;
5. Результат появится в `<p id="result">Result</p>`;
6. Вставьте число в поле для ответа.

РЕШЕНИЕ

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/selenium/4/4.html')
    [x.click() for x in browser.find_elements(By.CLASS_NAME, 'check')]
    browser.find_element(By.CLASS_NAME, 'btn').click()
    print(browser.find_element(By.ID, 'result').text)
    time.sleep(5)
```

3.4.9

Когда вам необходимо получить значение атрибута. Вы можете использовать метод `.get_attribute('attribute')`, где **attribute** - это имя требуемого вам атрибута.

```
▼ <div class="content"> flex == $0
  <input type="checkbox" class="check" value="1">
  <input type="checkbox" class="check" value="2">
  <input type="checkbox" class="check" value="3">
```

Если необходимо получить значение `value=""` мы напишем следующий код.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/selenium/5/5.html')
    checkbox = browser.find_elements(By.CLASS_NAME, 'check')
    for item in checkbox:
        print(item.get_attribute('value'))

>>>1,2,3,4 ... 520
```

Задача

1. Откройте [сайт](#);
2. Установите все чек боксы в положение **checked** при помощи selenium и метода `click()`;
3. Должны быть установлены те чек боксы, значение `value=""` которых, есть в списке `numbers`;
4. Когда все необходимые чек боксы станут **checked**, кнопка станет активной, нажмите на неё и скопируйте результат
5. Результат появится в `<p id="result">Result</p>`;
6. Вставьте число в поле для ответа.

```
numbers = [1, 2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16, 17, 22, 23, 28, 29,
33, 34, 38,
39, 43, 44, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 64, 68,
69, 73,
74, 78, 79, 83, 84, 88, 89, 91, 92, 97, 98, 101, 104, 108, 109, 113, 114,
118,
119, 123, 124, 128, 129, 131, 132, 137, 138, 140, 141, 144, 145, 148,
149, 153,
154, 158, 159, 163, 164, 165, 168, 169, 171, 172, 177, 178, 180, 181,
184, 185,
187, 188, 189, 190, 192, 193, 194, 195, 197, 198, 199, 200, 204, 205,
206, 207,
208, 209, 211, 212, 217, 218, 220, 221, 224, 225, 227, 228, 229, 230,
232, 233,
234, 235, 237, 238, 239, 240, 245, 246, 247, 248, 249, 251, 252, 253,
254, 255,
256, 257, 258, 260, 261, 264, 265, 268, 269, 273, 274, 278, 279, 288,
289, 291,
```

```
292, 293, 294, 295, 296, 297, 300, 301, 302, 303, 304, 305, 308, 309,  
313, 314,  
318, 319, 328, 329, 331, 332, 339, 340, 341, 342, 343, 344, 345, 346,  
348, 349,  
353, 354, 358, 359, 368, 369, 371, 372, 379, 380, 385, 386, 408, 409,  
411, 412,  
419, 420, 425, 426, 428, 429, 433, 434, 438, 439, 444, 445, 446, 447,  
448, 451,  
452, 459, 460, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 477,  
478, 479,  
480, 485, 486, 487, 488, 491, 492, 499, 500, 505, 506, 508, 509, 513,  
514, 518, 519]
```

РЕШЕНИЕ

```
import time  
from selenium.webdriver.chrome.options import Options  
from selenium import webdriver  
from selenium.webdriver.common.by import By
```

```
number = [1, 2, 3, 4, 8, 9, 11, 12, 13, 14, 15, 16, 17, 22, 23, 28, 29, 33, 34, 38,  
39, 43, 44, 48, 49, 51, 52, 53, 54, 55, 56, 57, 58, 61, 62, 63, 64, 68, 69, 73,  
74, 78, 79, 83, 84, 88, 89, 91, 92, 97, 98, 101, 104, 108, 109, 113, 114, 118,  
119, 123, 124, 128, 129, 131, 132, 137, 138, 140, 141, 144, 145, 148, 149, 153,  
154, 158, 159, 163, 164, 165, 168, 169, 171, 172, 177, 178, 180, 181, 184, 185,  
187, 188, 189, 190, 192, 193, 194, 195, 197, 198, 199, 200, 204, 205, 206, 207,  
208, 209, 211, 212, 217, 218, 220, 221, 224, 225, 227, 228, 229, 230, 232, 233,  
234, 235, 237, 238, 239, 240, 245, 246, 247, 248, 249, 251, 252, 253, 254, 255,  
256, 257, 258, 260, 261, 264, 265, 268, 269, 273, 274, 278, 279, 288, 289, 291,  
292, 293, 294, 295, 296, 297, 300, 301, 302, 303, 304, 305, 308, 309, 313, 314,  
318, 319, 328, 329, 331, 332, 339, 340, 341, 342, 343, 344, 345, 346, 348, 349,  
353, 354, 358, 359, 368, 369, 371, 372, 379, 380, 385, 386, 408, 409, 411, 412,  
419, 420, 425, 426, 428, 429, 433, 434, 438, 439, 444, 445, 446, 447, 448, 451,  
452, 459, 460, 465, 466, 467, 468, 469, 470, 472, 473, 474, 475, 477, 478, 479,  
480, 485, 486, 487, 488, 491, 492, 499, 500, 505, 506, 508, 509, 513, 514, 518, 519]
```

```
with webdriver.Chrome() as browser:  
    browser.get('http://parsinger.ru/selenium/5/5.html')  
    [x.click() for n, x in enumerate(browser.find_elements(By.CLASS_NAME, 'check'),  
start=1) for y in number if y == n]  
    time.sleep(5)
```

3.4.10

Выпадающие списки

Работа с выпадающими списками практически ничем не отличается от работы с чек боксами или полями.

Для того чтобы получить значение любого элемента из выпадающего списка, мы можем использовать привычные нам методы поиска элемента.

```
<select id="opt">  
  <option value="1">98016191141</option>  
  <option value="2">48026291242</option>
```

В нашем случае на нашем [сайте](#) все элементы выпадающего списка, могут быть спокойно найдены по тегу, метод `By.TAG_NAME` нам в этом поможет.

```
from selenium import webdriver
```

```
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/selenium/6/6.html')
    g = browser.find_element(By.TAG_NAME, 'option').text
    print(g)
```

```
>>> 98016191141
```

Задача

1. Открываем [сайт](#) с помощью selenium;
2. Получаем значения всех элементов выпадающего списка;
3. Суммируем(плюсуем) все значения;
4. Вставляем получившийся результат в поле на сайте;
5. Нажимаем кнопку и копируем длинное число;
6. Вставляем конечный результат в поле ответа.

РЕШЕНИЕ

```
from selenium.webdriver.common.by import By
from selenium import webdriver

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/selenium/7/7.html')
    result = 0
    for el in browser.find_elements(By.TAG_NAME, 'option'):
        result += int(el.text)
    browser.find_element(By.ID, 'input_result').send_keys(result)
    browser.find_element(By.CLASS_NAME, 'btn').click()
    print(browser.find_element(By.ID, 'result').text)
```

3.4.11

1. Откройте [сайт](#) при помощи selenium;
2. Решите уравнение на странице;
3. Найдите и выберите в выпадающем списке элемент с числом, которое у вас получилось после решения уравнения;
4. Нажмите на кнопку;
5. Скопируйте число и вставьте в поле ответа.

РЕШЕНИЕ

```
from selenium.webdriver.common.by import By
from selenium import webdriver

with webdriver.Chrome() as browser:
    n = 12434107696*6 + 1
    browser.get('http://parsinger.ru/selenium/6/6.html')
    browser.find_element(By.TAG_NAME, 'select').send_keys(str(n))
    browser.find_element(By.CLASS_NAME, 'btn').click()
    print(browser.find_element(By.ID, 'result').text)
```

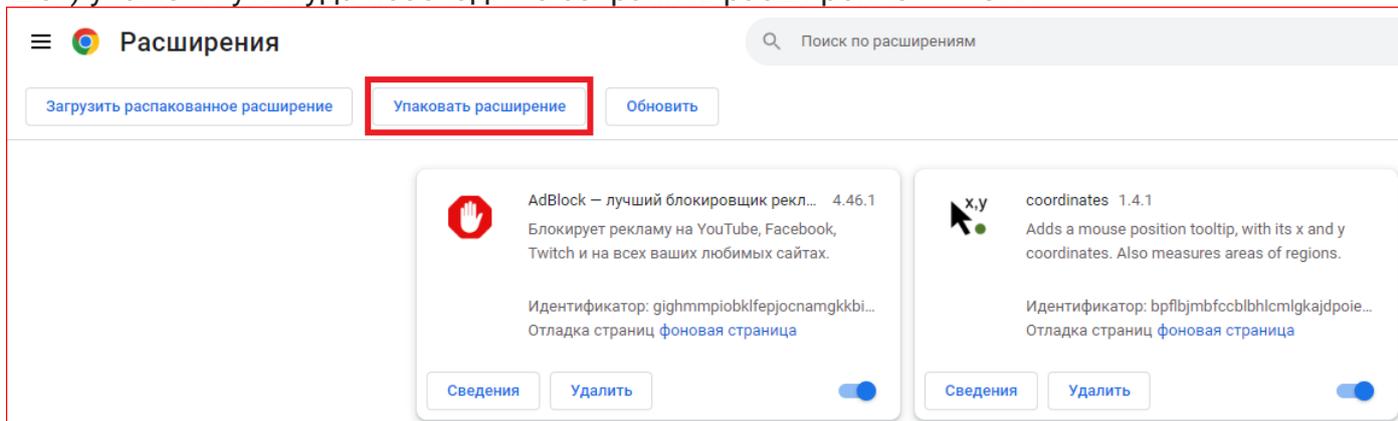
4. Опции и аргументы

4.1 Запуск браузера с расширениями

Чтобы запустить браузер с предустановленным на него расширением, нам для начала необходимо это расширение подготовить. В Chrome существует [магазин расширений](#), надеюсь об этом все знают и хоть раз в жизни пользовались им.

Давайте найдём расширение которое помогает определять координаты курсора. Которое кстати поможет решить одну из задач. Называется **coordinates** и скачать его можно по [ссылке](#), установите его привычным образом.

Для того чтобы подготовить расширение должным образом, нам **необходимо его упаковать**. Для этого запустим **Chrome** и перейдем в раздел расширений `chrome://extensions` ("Меню" → "Настройки" → "Расширения") → далее, отметим чекбокс (флажок) "Режим разработчика" → "Упакованное расширение" (скриншот) укажем путь куда необходимо сохранить расширение. → "ОК".

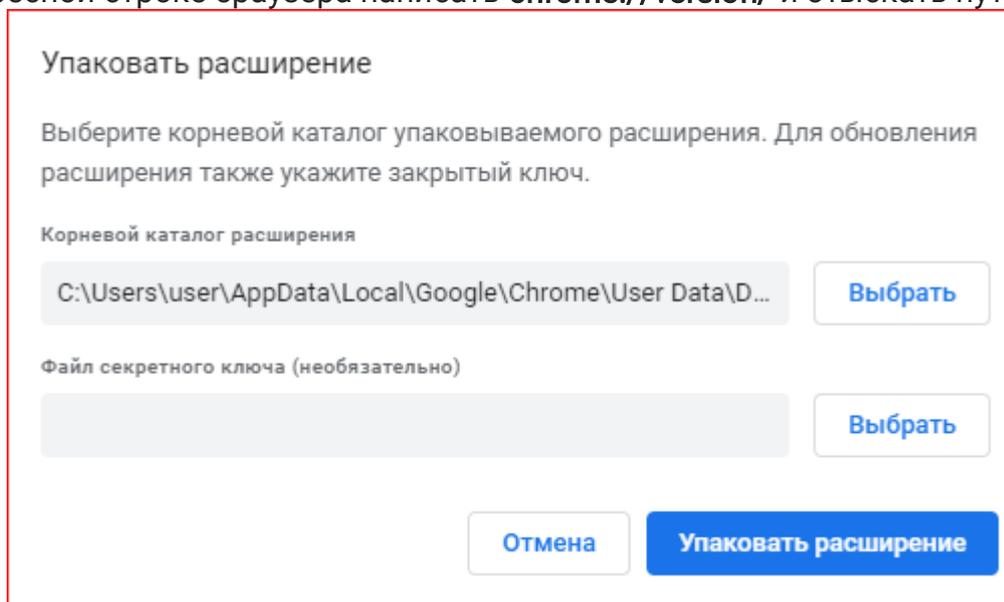


В появившемся окне, необходимо указать путь к нужному вам расширению.

Адрес по которому они находятся в скрытой

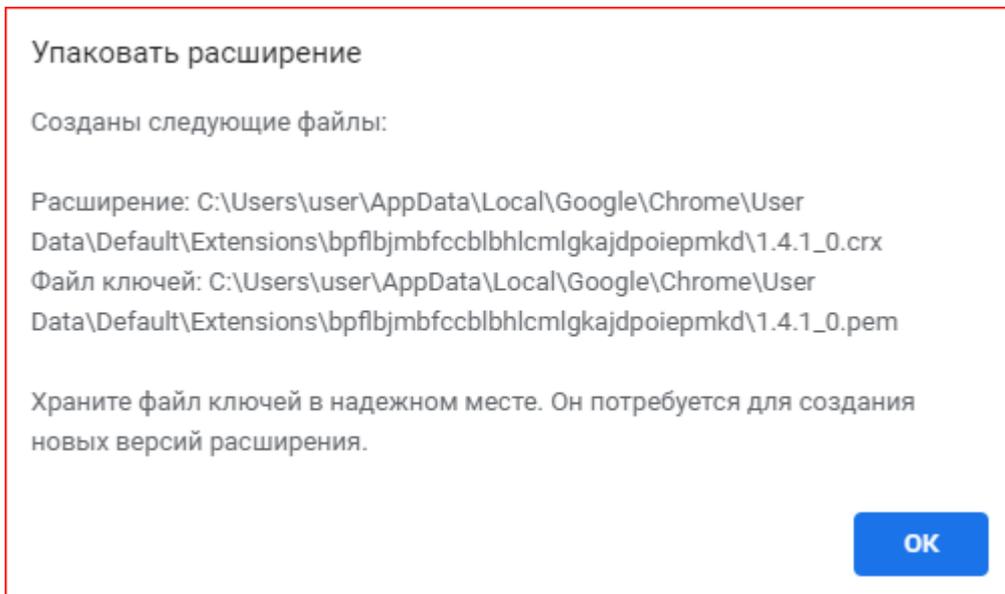
папке: `C:\Users\user\AppData\Local\Google\Chrome\User Data\Default\Extensions`

Или в адресной строке браузера написать `chrome://version/` и отыскать путь там.

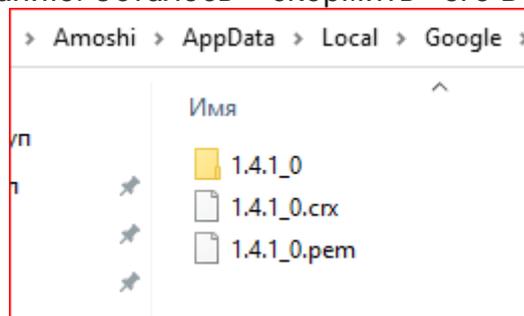


Обратите внимание что создаётся 2 файла

- `1.4.1_0.crx` - упакованное расширение;
- `1.4.1_0.pem` - файл ключей, который необходимо будет удалить если потребуется повторная упаковка расширения (имя будет соответствовать названия расширения).



В папке появится файл "1.4.1_0.crx" , это и есть наше упакованное расширение, теперь оно готово к использованию. Осталось "скормить" его в Selenium.



В методе `.add_extension('coordinates.crx')` мы указываем путь к нашему упакованному расширению, если расширение лежит в папке с проектом, достаточно указать его имя.

Так же необходимо не забыть передать опции в `webdriver`, делается это так `webdriver.Chrome(options=)`

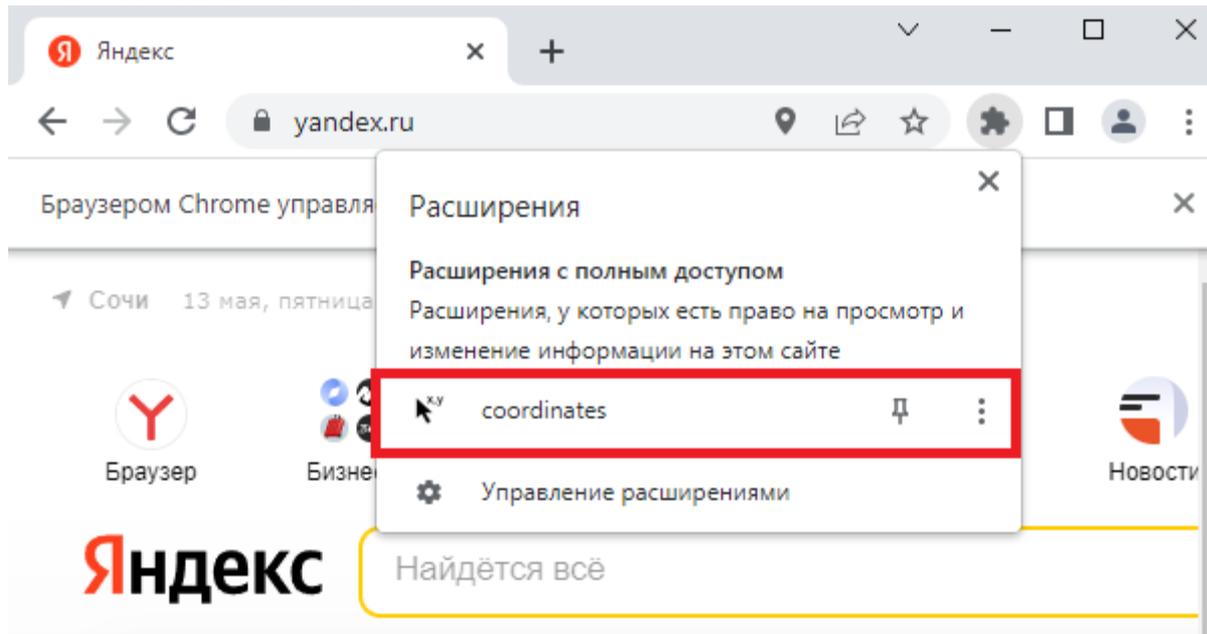
В этом примере я переименовал файл 1.4.1_0.crx => coordinates.crx

```
import time
from selenium import webdriver
```

```
options_chrome = webdriver.ChromeOptions()
options_chrome.add_extension('coordinates.crx')
```

```
with webdriver.Chrome(options=options_chrome) as browser:
    url = 'https://yandex.ru/'
    browser.get(url)
    time.sleep(50)
```

Если всё сделано правильно, то при запуске браузера, в нём будет установлено расширение. Установите достаточную задержку чтобы запустить этот код у себя и убедитесь в этом самостоятельно.



4.2 Запуск браузера в скрытом режиме

Запуск браузера в фоновом режиме, очень прост, достаточно запомнить всего один параметр и синтаксис. Для этого нам **потребуется метод `.add_argument()` и передать ему параметр `--headless`**. Для примера откроем первую страницу Яндекса, и получим первую найденную ссылку.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

options_chrome = webdriver.ChromeOptions()
options_chrome.add_argument('--headless')

with webdriver.Chrome(options=options_chrome) as browser:
    url = 'https://yandex.ru/'
    browser.get(url)
    a = browser.find_element(By.TAG_NAME, 'a')
    print(a.get_attribute('href'))
```

```
>>>
```

```
https://passport.yandex.ru/auth?origin=home_yandexid&retpath=https%3A%2F%2Fyandex.ru&backpath=https%3A%2F%2Fyandex.ru
```

Преимущества запуска браузера в фоновом режиме.

- Отсутствует отрисовка содержимого, тем самым **потребуется меньше ресурсов**.
- **Работает быстрее**, т.к. потребляет меньше потому что ему ничего не нужно обрисовывать.
 - Не занимает место на экране, и не мешает вашей работе во время выполнения скрипта.
 - Использование `--headless` может значительно ускорить работу парсера, на относительно слабых машинах.

В некоторых гайдах вы можете встречать параметр `--disable-gpu`, который по сути выполняет то же самое что и `--headless`, запускает браузер "без головы".

4.3 Перенос профиля с основного браузера Chrome в браузер под управлением Selenium

Иногда вы будете хотеть перенести все настройки, закладки, историю с основного браузера, в браузер под управлением Selenium. Сейчас я вам покажу как это можно сделать.

- Определяем путь к папке с профилями `\User Data\` для этого напишите команду, в адресной строке браузера `chrome://version/` и ищите адрес в поле с заголовком "Путь к профилю:" У меня адрес выглядит вот так: `C:\Users\user\AppData\Local\Google\Chrome\User Data\`

- `'user-data-`

```
dir=C:\\Users\\user\\AppData\\Local\\Google\\Chrome\\User Data'
```

 Добавить путь к профилю в метод

```
.add_argument()
```

```
import time
```

```
from selenium import webdriver
```

```
options_chrome = webdriver.ChromeOptions()
```

```
options_chrome.add_argument('user-data-
```

```
dir=C:\\Users\\user\\AppData\\Local\\Google\\Chrome\\User Data')
```

```
with webdriver.Chrome(options=options_chrome) as browser:
```

```
    url = 'https://yandex.ru/'
```

```
    browser.get(url)
```

```
    time.sleep(10)
```

Если всё сделано правильно, то у вас запустится окно браузера с вашими параметрами, историей, закладками.

Если у вас возникает ошибка `"invalid argument: user data directory is already in use, please specify a unique value for"` Это означает что данный профиль уже используется, закройте основной браузер и повторите попытку. Если Вам необходимо основное окно браузера, то скопируйте полностью папку `\User Data\` в удобное место и укажите путь к ней

4.4 Proxy и Selenium

Работа с прокси в Selenium очень проста, намного проще чем в [requests](#), там мы создавали словарь, прописывали в ключах схемы, и затем передавали его в запросе.

Сайтом, на котором можно узнать свой IP у нас будет сайт [2ip.ru](#). Выполните код ниже, чтобы увидеть к принт в консоли с вашим IP адресом.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

url = 'https://2ip.ru/'
with webdriver.Chrome() as browser:
    browser.get(url)
    print(browser.find_element(By.ID,
'd_clip_button').find_element(By.TAG_NAME, 'span').text)
    time.sleep(5)

>>> 95.27.00.01
```

Теперь модифицируем данный код, чтобы запрос отправлялся через прокси. Прокси должен быть вида IP:PORT

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

proxy = ['8.210.83.33:80']
url = 'https://2ip.ru/'

chrome_options = webdriver.ChromeOptions()
chrome_options.add_argument('--proxy-server=%s' % proxy)

with webdriver.Chrome(options=chrome_options) as browser:
    browser.get(url)
    print(browser.find_element(By.ID,
'd_clip_button').find_element(By.TAG_NAME, 'span').text)
    time.sleep(5)
```

Посмотрите **внимательно** на код, и определите отличия в этих 2х примерах.

Первое что мы сделали это - передали параметр `'--proxy-server=%s' % proxy` методу `.add_argument()` в класс дополнительных опций `.ChromeOptions()` и передали сам прокси, который лежал в переменной `proxy`. Если этот прокси ещё живой, можете запустить этот код у себя в IDE, если прокси умер, то с помощью скрипта в первом абзаце, можете спарсить себе новый список.

Подобно [requests](#), мы можем установить `timeout=` для загрузки страницы, после истечению которого произойдёт, либо закрытие окна, либо переход к следующему прокси.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

proxy_list = ['8.210.83.33:80', '199.60.103.28:80',
'103.151.246.38:10001', '199.60.103.228:80', '199.60.103.228:80',
'199.60.103.28:80', ]
```

```

for PROXY in proxy_list:
    try:
        chrome_options = webdriver.ChromeOptions()
        chrome_options.add_argument('--proxy-server=%s' % PROXY)
        url = 'https://2ip.ru/'

        with webdriver.Chrome(options=chrome_options) as browser:
            browser.get(url)
            print(browser.find_element(By.ID,
'd_clip_button').find_element(By.TAG_NAME, 'span').text)

            browser.set_page_load_timeout(5)

        proxy_list.remove(PROXY)
    except Exception as _ex:
        print(f"Превышен timeout ожидания для - {PROXY}")
        continue

```

В этом примере, есть список прокси `proxy_list`, по которому мы итерируемая в цикле `for`, передавая на каждой итерации в переменную `PROXY`, следующий IP из этого списка. В этом примере мы применили конструкцию `try/except`, чтобы наш скрипт не падал с ошибкой, и продолжал работать. Если не обернуть код в `try/except`, мы после каждого истёкшего `timeout=` будем получать ошибку: `Message: unknown error:`

```
net::ERR_TUNNEL_CONNECTION_FAILED.
```

`timeout` в Selenium применяется методом `.set_page_load_timeout(5)` где цифра 5 длительность в секундах.

p.s. Если вы найдёте работающий прокси, то этот код напечатает вам его в консоль =)

5. Основные методы

5.1 Основные методы Selenium

В своём распоряжении Selenium имеет большое количество методов которые мы можем использовать. Здесь будут размещены почти все из них. Но как показывает практика, пользоваться вы будете малой их частью. Про одни методы вы должны просто знать что они существуют, про другие методы вам скорее всего даже вспоминать не придётся, а какие то методы очень полезны и мы порешаем задачи с их помощью, чтобы лучше закрепить их в памяти.

- `webdriver.back()` - вернуться назад, равнозначно стрелочке "назад" в браузере;

- `webdriver.forward()` - вернуться вперёд, равнозначно стрелочке "вперёд" в браузере;

- `webdriver.refresh()` - обновляет активную страницу в браузере, равнозначно стрелочке обновить;

- `webdriver.get_screenshot_as_file("../file_name.jpg")` - ожидает полной загрузки страницы и сохраняет скриншот в указанной папке. Возвращает **False**, если есть ошибка ввода-вывода, иначе возвращает **True**;
- `webdriver.save_screenshot("file_name.jpg")` - ожидает полной загрузки страницы и сохраняет скриншот в папке с проектом;
- `webdriver.get_screenshot_as_png()` - сохраняет скриншот в виде двоичных данных, которые можно передать или сохранить в файл в конструкторе `with/as`;
- `webdriver.get_screenshot_as_base64()` - после загрузки страницы, получает скриншот текущего окна в виде строки в кодировке **base64**. Полезно во встроенных изображениях в **HTML**;
- `webdriver.get("http://example_url.ru")` - метод получает ссылку которая откроется в браузере;
- `webdriver.quit()` - метод разрывает все соединения установленные браузеров, очищает после себя оперативную память;
- `webdriver.close()` - закрывает текущую вкладку;
- `webdriver.execute_script("script_code")` - исполняет на странице переданный JavaScript код;
- `webdriver.execute_async_script("script_code" , *args)` - асинхронно выполняет код JavaScript на странице;
- `webdriver.set_page_load_timeout()` - устанавливает timeout ожидания загрузки страницы после чего выбрасывает исключение;
- `webdriver.find_element("element" or "locator- By.")` - возвращает первый найденный элемент соответствующий локатору или элементу;
- `webdriver.find_elements("element" or "locator- By.")` - возвращает `resultSet` найденных элементов, с ним можно работать как со списком;
- `webdriver.get_window_position()` - возвращает позицию открытого окна браузера, возвращается словарь `{'x': 10, 'y': 50}`;

- `webdriver.maximize_window()` - разворачивает текущее окно;



- `webdriver.minimize_window()` - сворачивает текущее окно;



- `webdriver.fullscreen_window()` - максимизирует активное окно браузера,

аналогично нажатию клавиши **F11**;

- `webdriver.get_window_size()` - получает текущий размер окна браузера + рамки окна и панель управления браузером, возвращает словарь `{'width': 945, 'height': 1020}`;

- `webdriver.set_window_size(800, 600)` - устанавливает высоту и ширину

браузера;

- `webdriver.get_cookies()` - возвращает словарь с **cookies**;
- `webdriver.get_cookie(name_cookie)` - возвращает набор **cookie** по его имени;
- `webdriver.add_cookie(cookie_dict)` - добавляет **cookie** к вашему текущему

сеансу;

- `webdriver.delete_cookie(name_cookie)` - удаляет **cookie** с заданным именем;
- `webdriver.delete_all_cookies()` - удаляет все файлы **cookie** в рамках

текущего сеанса;

- `webdriver.implicitly_wait(10)` - устанавливает неявное ожидание поиска

элемента, или для команды завершения;

- `webdriver.click(webelement)` - совершает клик по выбранному элементу,

клик возможен только по интерактивному элементу.

5.2 Cookies

Куки (cookie, букв. — «печенье») - небольшой фрагмент данных, отправленный веб-сервером и хранимый на компе пользователя. Когда вы открываете сайт, сервер отправляет вашему браузеру данные которые хранятся в его памяти.

Куки чаще всего используются для:

- Аутентификации пользователя;
- Хранение личных настроек на сайте, к примеру тёмная тема или сохранение товаров в корзине если вы не залогинились на сайте ;
- Отслеживание состояния сеанса доступа пользователя;
- Сведения статистики о пользователях;
- Хранения информации о местоположении пользователя и IP-адресе;
- Клики и переходы;
- версию операционной системы и браузера;
- И многое другое.

«Cookies не являются персональными данными, так как в законе сказано что персональные данные — это информация, позволяющая идентифицировать человека. Даже фамилия, имя и отчество могут не являться персональными данными, если требуются дополнительные сведения, чтобы определить личность человека. Не говоря уже о cookies».

Существует два вида cookie:

- **Сессионные**(временные) - это те куки которые хранят в себе информацию которая актуальна ближайшее время, к примеру к таким данным можно отнести записи форм, полей, время пребывания на сайте. Чаще всего они существуют пока вы находитесь на сайте и удаляются как только вы его покидаете;
- **Постоянные** - это те куки которые могут храниться в вашем браузере очень долго, например логин от вашей учётной записи на сайте или другие данные которые связаны с вашей учётной записью. Это может быть данные о вашем местоположении в учётной записи гугла.

Что бы увидеть какие **cookie** сохраняет сайт в вашем браузере, вам нужно открыть инструмент разработчика клавишей **F12**;

Название	Значение	Domain	Path	Expires...	Размер	HttpO...	Secure	SameS...	SameP...	Partiti...	Prior...
._gat	1	.stepik.org	/	2022-0...	5						Medium
._ym_visorc	w	.stepik.org	/	2022-0...	11		✓	None			Medium
._ym_d	1653981236	.stepik.org	/	2023-0...	15		✓	None			Medium
._ym_uid	165398123646591758	.stepik.org	/	2023-0...	25		✓	None			Medium
._ym_jsad	2	.stepik.org	/	2022-0...	9		✓	None			Medium
._ga	GA1.2.1808854067.1653981236	.stepik.org	/	2024-0...	30						Medium
._gcl_au	1.1.525631717.1653981236	.stepik.org	/	2022-0...	31						Medium
.sessionid	rcwkv5gdzev1xhamioxv6o1xvsjqf...	.stepik.org	/	2022-0...	41	✓	✓	None			Medium
._gid	GA1.2.1262955418.1653981236	.stepik.org	/	2022-0...	31						Medium
._gat_UA-41633741-1	1	.stepik.org	/	2022-0...	19						Medium
.csrftoken	iKHTvTrNuh3kbc3zmOHb3IRIAF...	.stepik.org	/	2023-0...	73		✓	None			Medium

В **Selenium** мы можем получить доступ ко всем cookie сразу в виде словаря, либо к конкретному полю, об этом в следующем шаге.

5.3 Cookies на практике

`.get_cookies()`

В коде ниже использован метод `.get_cookies()` который получает список всех **cookie** на странице. Выполните код ниже у себя в терминале.

```
from pprint import pprint
from selenium import webdriver

with webdriver.Chrome() as webdriver:
    webdriver.get('https://yandex.ru/')
    cookies = webdriver.get_cookies()
    pprint(cookies)
```

```
>>>
[{'domain': '.yandex.ru',
  'expiry': 1685518907,
  'httpOnly': False,
  'name': '_ym_d',
  'path': '/',
  'sameSite': 'None',
  'secure': True,
  'value': '1653982908'},
 ...
 {'domain': '.yandex.ru',
  'expiry': 1656574906,
  'httpOnly': False,
  'name': 'yandex_gid',
  'path': '/',
  'sameSite': 'None',
  'secure': True,
  'value': '239'}]
```

```
.get_cookie(name_cookie)
```

В отличие от первого метода, этот метод находит и возвращает **cookie** по его имени, для того чтобы определить имя есть 2 способа.

- Способ №1 - этот способ не очень надёжен, т.к. с "живого" браузера данные в **cookies** могут отличаться в зависимости от открытой сессии. Но если ваш код не зависит от параметров сессии то можно получить имена cookie именно в браузере;

Название	Значение	Domain
yp	1656575405.ygu.1#1669751408.s...	.yandex.ru
_ym_d	1653983235	.yandex.ru
_ym_isad	2	.yandex.ru
_ym_uid	1653983235333764251	.yandex.ru
_yasc	YN9w1vNafnGdX9F5YSJf2wjw5b...	.yandex.ru
gdpr	0	.yandex.ru
my	YwA=	.yandex.ru
yabs-sid	350786901653983405	mc.yandex.ru
yabs-sid	322393641653983407	zen-desktop-
i	9gpNs0IEVMuoX0popqj5JE0GW4...	.yandex.ru
yuidss	1978530571653983405	.yandex.ru
is_gdpr	0	.yandex.ru
yabs-frequency	/5/0000000000000000/1LmOhY6...	.yandex.ru
is_gdpr_b	CMrGBxCCdigC	.yandex.ru
yandexuid	1978530571653983405	.yandex.ru
mda	0	.yandex.ru
ymex	1969343405.yrts.1653983405	.yandex.ru
yandex_gid	239	.yandex.ru

• Способ №2 - мы можем в цикле `for/in` итерироваться по списку `cookie` который мы получили с помощью метода `.get_cookies()`. Этим способом мы можем получить не только имя `cookie` но и его значение;

```

• from selenium import webdriver
•
• with webdriver.Chrome() as webdriver:
•     webdriver.get('https://yandex.ru/')
•     cookies = webdriver.get_cookies()
•     for cookie in cookies:
•         print(cookie['name']) # или cookie['value'] чтобы получить их

```

значение

```

• >>>
• _ym_d, _ym_isad, _ym_uid, my, gdpr, _yasc, i, is_gdpr, yuidss
• yabs-frequency, is_gdpr_b, yandexuid, yp, mda, ymex, yandex_gid
•
•

```

• Когда мы знаем имена всех `cookie` на странице мы можем получить нужные нам данные по ключу. Мы помним что `.get_cookies()` возвращает список словарей. Если вы посмотрите на первый пример с кодом, вы увидите что в `cookie` хранится время экспирации `'expiry': 1685518907`, т.е. истечения срока жизни `cookie`. Пример кода ниже, поможет нам извлечь конкретное значение из `cookie`.

```

• from selenium import webdriver
•
• with webdriver.Chrome() as webdriver:
•     webdriver.get('https://yandex.ru/')
•     print(webdriver.get_cookie('_ym_uid')['expiry'])
•
• >>>1685520499

```

5.4 Добавление cookie

```
webdriver.add_cookie(cookie_dict)
```

Все мы когда то чистили браузер от печенек (*cookie*), наверное каждый начинающий программист знает, что если очистить **cookie** то настройки сайта слетят, слетит учётная запись, тёмная тема перестанет загружаться по умолчанию, сайт вдруг перестанет приветствовать вас по имени. Всем понятно зачем чистить куки, а вот **зачем их добавлять в браузер?** Добавляем мы их по обратной причине. Когда мы работаем с **Selenium** мы **не всегда можем использовать один профиль браузера в нескольких скриптах**. В таких случаях мы можем либо создать ещё несколько профилей, либо использовать одни **cookie** на все скрипты.

`.add_cookie(cookie_dict)` - это метод который добавляет **cookie** в ваш браузер.

Принимает словарь, но с определёнными ограничениями, мы **не можем передать в cookie что попало**. В браузере есть заранее подготовленные поля в которые мы можем передавать данные.

Доступные поля для **cookie**, посмотреть их можно в любом браузере, эти поля доступны для передаче их в словаре, как формировать словарь мы поговорим ниже.

- `"name"` - устанавливает имя **cookie** файла;
- `"value"` - устанавливает значение **cookie**, это значение может либо идентифицировать пользователя, либо содержать любую другую служебную информацию;
- `"expires"` и `"max-age"` - определяет срок жизни **cookie**, после истечения этого срока **cookie** будет удалён из памяти браузера, если не указывать эти значения, содержимое **cookie** будет удалено после закрытия браузера ;
- `"path"` - указывает путь к директории на сервере, для которой будут доступны `cookie`, чтобы **cookie** были доступны по всему домену необходимо указать `/` ;
- `"domain"` - хранит в себе информацию о домене или поддомене которые имеет доступ к этой **cookie**, если необходимо чтоб **cookie** были доступны по всему домену и всем поддоменом, указывается базовый домен, к примеру `www.example.ru`;
- `"secure"` - указывает серверу что **cookie** должны передавать только по защищённому **https**-соединению;
- `"httponly"` - параметр запрещает доступ к **cookie** посредством API браузера
- `document.cookie` . Предотвращает кражи **cookie** посредством **XSS**-атак. Если флаг установлен **True**, вы сможете получить доступ к этой **cookie** **только через браузер** в том числе и через **Selenium**;
- `"samesite"` - ограничивает передачу **cookie** между сайтами, предотвращает кражу **cookie** посредством **XSS**-атак. Имеет 3 состояния;
- `samesite=none` - на передачу **cookie** нет никаких ограничений;
- `samesite=lax` - разрешает передачу только безопасным HTTP методом;
- `samesite=strict` или `samesite` - самое строгое состояние, которое запрещает отправку **cookie** на другие сайты.

Запустите код ниже у себя в терминале, поиграйтесь с параметрами, посмотрите на результат, найдите изменения в браузере.

```
import time
from pprint import pprint
from selenium import webdriver
```

```
cookie_dict = {
    'name': 'any_name_cookie',      # Любое имя для cookie
    'value': 'any_value_cookie',    # Любое значение для cookie
    'expiry': 2_000_000_000,        # Время жизни cookie в секундах
    'path': '/',                    # Директория на сервере для которой
```

будут доступны cookie

```

'domain': 'parsinger.ru', # Информация о домене и поддомене для
которых доступны cookie
'secure': True, # or False # Сигнал браузера о том что передать
cookie только по защищённому HTTPS
'httpOnly': True, # or False # Ограничивает доступ к cookie по
средствам API
'sameSite': 'Strict', # or lax or none # Ограничение на передачу
cookie между сайтами
}

```

```

with webdriver.Chrome() as webdriver:
    webdriver.get('https://parsinger.ru/methods/4/index.html')
    webdriver.add_cookie(cookie_dict)
    pprint(webdriver.get_cookies())
    time.sleep(100)

```

Все ключи словаря `cookie_dict` соответствуют полям `cookie` в браузере, поэтому вы не можете менять ключи в этом словаре, если вы это сделаете, ничего не произойдёт. Словарь просто не запишется в `cookie` браузера. Вы можете изменять только значения этого словаря, и только следуя определённым правилам. Вы можете изменять как вы хотите только значения ключей `"name"` и `"value"`, остальные значения в ключах имеют строгие правила.

Имя	Значение	Домен	Путь	Срок действия / Макс.	П.	HttpOnly	Secure	SameSite	SameSite	Partiti...	Prio...
yabs-sid	2683222041654009785	mc.yandex.ru	/	Сессия				None			Medi...
yp	1656601785.ygu.1#1669777788...	.yandex.ru	/	2032-05-28T...	7...		✓	None			Medi...
yabs-vdrf	BBvPYDW267m000	.an.yandex.ru	/	2022-06-07T...	2...		✓	None			Medi...
_ym_d	1654009612	.yandex.ru	/	2023-05-31T...	1...		✓	None			Medi...
_yasc	dRzcbWyKdAe1qUb9/pY9evVZ...	.yandex.ru	/	2022-06-30T...	6...		✓	None			Medi...
gdpr	0	.yandex.ru	/	Сессия	5		✓	None			Medi...
my	YwA=	.yandex.ru	/	2032-05-28T...	6		✓	None			Medi...
yabs-sid	482634921654009787	zen-desktop-mor...	/	Сессия	2...		✓	None			Medi...
i	k2qGnh963oX01EGOG3LSazq...	.yandex.ru	/	2024-05-30T...	9...	✓	✓	None			Medi...
_ym_uid	1654009612717306938	.yandex.ru	/	2023-05-31T...	2...		✓	None			Medi...
yuidss	3028091221654009784	.yandex.ru	/	2032-05-28T...	2...		✓	None			Medi...
is_gdpr	0	.yandex.ru	/	2024-05-30T...	8		✓	None			Medi...
yabs-frequency	/5/0000000000000000/LVTwO9...	.yandex.ru	/	2022-09-08T...	7...		✓	None			Medi...
is_gdpr_b	CMrGBxCKdigC	.yandex.ru	/	2024-05-30T...	2...		✓	None			Medi...
yandexuid	3028091221654009784	.yandex.ru	/	2032-05-28T...	2...		✓	None			Medi...
mda	0	.yandex.ru	/	2025-05-30T...	4		✓	None			Medi...
ymex	1969369785.yrts.1654009785	.yandex.ru	/	2023-05-31T...	3...		✓	None			Medi...
yandex_gid	239	.yandex.ru	/	2022-06-30T...	1...		✓	None			Medi...

5.5 Запуск JavaScript на странице

`.execute_script()`

Синтаксис `webdriver.execute_script(script, *args)`.

В `.execute_script()` можно использовать следующие полезные параметры.

Посмотреть все события можно [тут](#) и [тут](#), ниже приведены те, которые чаще всего используются при написании парсеров.

- `.execute_script("return arguments[0].scrollIntoView(true);", element)` - прокручивает родительский контейнер элемента таким образом, чтобы `element` для которого вызывается `scrollIntoView`, был виден пользователю;
- `.execute_script("window.open('http://parsinger.ru', 'tab2');")` - создаст новую вкладку с именем "tab2";
- `.execute_script("return document.body.scrollHeight")` - вернёт значение высоты элемента `<body>`;
- `.execute_script("return window.innerHeight")` - вернёт значение высоты окна браузера;
- `.execute_script("return window.innerWidth")` - вернёт значение ширину окна браузера;
- `.execute_script("window.scrollTo(X, Y)")` - прокручивает документ на заданное число пикселей;
 - `X` - смещение в пикселях по горизонтали;
 - `Y` - смещение в пикселях по вертикали.
- Для того чтобы **получить фокус элемента**, можно использовать `.execute_script("return arguments[0].scrollIntoView(true);", element)` где `element` это объект `webdriver'a`
 - `.execute_script("alert('Ура Selenium')")` - вызывает модальное окно `Alert`;
 - `.execute_script("return document.title;")` - вернёт `title` открытого документа;
 - `.execute_script("return document.documentURI;")` - возвращает URL документа;
 - `.execute_script("return document.readyState;")` - возвращает состояние загрузки страницы, вернёт **complete** если страница загрузилась;
 - `.execute_script("return document.anchors;")` - возвращает список всех [якорей](#);
 - `[x.tag_name for x in browser.execute_script("return document.anchors;")]` - такой код даст возможность получить список всех тегов с якорями. **Очень полезная инструкция, используется если при скроллинге мы** не можем найти элемент для того чтобы "зацепиться" за элемент;
 - `.execute_script("return document.cookie;")` - возвращает список файлов **cookie**, разделенных точкой с запятой;
 - `.execute_script("return document.domain;")` - возвращает домен текущего документа;
 - `.execute_script("return document.forms;")` - вернёт список форм;
 - `window.scrollTo(x-coord, y-coord)` - прокрутка документа до указанных координат;

- `x-coord` пиксель по горизонтальной оси документа, будет отображён вверху слева;
- `y-coord` пиксель по вертикальной оси документа, будет отображён вверху слева.
- `.execute_script("return document.getElementsByClassName('container');")` - возвращает список всех элементов с заданным классом `class="container"`;
- `.execute_script("return getElementsByTagName('container');")` - возвращает список всех элементов с заданным именем `name="container"`.

5.6 Задачи по материалу

5.6.1 Задача

1. Откройте [сайт](#) с помощью Selenium;
2. При обновлении сайта, в `id="result"` появится число;
3. Обновить страницу возможно придется много раз, т.к. число появляется не часто;
4. Вставьте полученный результат в поле для ответа:

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as webdriver:
    webdriver.get('https://parsinger.ru/methods/1/index.html')
    while True:
        res = webdriver.find_element(By.ID, 'result').text
        webdriver.refresh()
        if res.isdigit():
            print(res)
            break
```

5.6.2

1. Откройте [сайт](#) с помощью Selenium;
2. На сайте есть определённое количество секретных cookie;
3. Ваша задача получить все значения и суммировать их;
4. Полученный результат вставить в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver

with webdriver.Chrome() as webdriver:
    res = []
    webdriver.get('https://parsinger.ru/methods/3/index.html')
    cookies = webdriver.get_cookies()
    for cookie in cookies:
        res.append(int(cookie['value']))
print(sum(res))
```

5.6.3

1. Откройте [сайт](#) с помощью Selenium;
2. Ваша задача получить все значения cookie с чётным или кратным 10 числа после "_" и суммировать их;
3. Полученный результат вставить в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver

with webdriver.Chrome() as webdriver:
    res = []
    webdriver.get('https://parsinger.ru/methods/3/index.html')
    cookies = webdriver.get_cookies()
    for cookie in cookies:
        digit = int(str(cookie['name']).split('_')[2])
        if digit % 2 == 0:
            res.append(int(cookie['value']))
print(sum(res))
```

5.6.4

1. Откройте [сайт](#) с помощью Selenium;
2. На сайте есть 42 ссылки, у каждого сайта по ссылке есть **cookie** с определёнными сроком жизни;
3. Цель: написать скрипт, который сможет найти среди всех ссылок страницу с самым длинным сроком жизни **cookie** и получить с этой страницы число;
4. Вставить число в поле для ответа.

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('https://parsinger.ru/methods/5/index.html')
    link = [x.find_element(By.TAG_NAME, 'a') for x in
browser.find_elements(By.CLASS_NAME, 'urls')]
    res = []
    while True:
        for x, br in zip(link, range(1, len(link) + 1)):
            browser.get(x.get_attribute('href'))
            code = browser.find_element(By.ID, 'result').text
            browser.back()
            res.append({
                'expiry': [x['expiry'] for x in browser.get_cookies()],
                'code': code,
            })
        if br == len(link):
            break
    print(int(sorted(res, key=lambda x: x['expiry'], reverse=True)[0]['code']))
```

5.6.5 Задача

Сопоставьте значения из двух списков

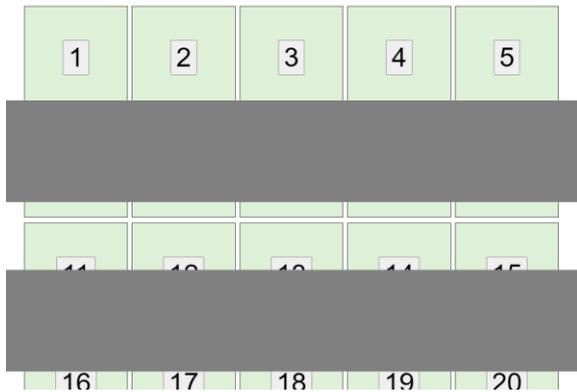
✔ Абсолютно точно.



<code>window.open('about:blank', 'tab_name');</code>	Создаёт новую вкладку
<code>return document.body.scrollHeight</code>	Вернёт значение высоты body
<code>return window.innerHeight</code>	Вернёт значение высоты окна браузера
<code>return window.innerWidth</code>	Вернёт значение ширины окна браузера
<code>window.scrollTo(X, Y)</code>	Прокручивает документ на заданное число
<code>alert('Учим Selenium')</code>	Вызывает модальное окно Alert
<code>return document.cookie;</code>	Возвращает список файлов cookie
<code>return document.domain;</code>	Возвращает домен текущего документа
<code>return document.forms;</code>	Вернёт список форм на странице
<code>window.scrollTo(x-coord, y-coord)</code>	Прокрутка документа до указанных координат.
<code>return document.getElementsByClassName('container');</code>	Возвращает список всех элементов с заданным классом
<code>return getElementsByTagName('container');</code>	Возвращает список всех элементов с заданным name

Следующий шаг Решить снова

5.6.6 Перекрытие элементов. Запуск JS



Когда вы хотите взаимодействовать с элементом который визуально перекрыт другим элементом, вы получите ошибку.

```
selenium.common.exceptions.ElementClickInterceptedException: Message:
element click intercepted: Element <button class="btn"
onclick="clicks()">...</button> is not clickable at point (135, 179). Other
element would receive the click: <div class="block2"></div>
```

Это ошибка нам сообщает, что другой элемент получит клик. Для того чтобы исключить такие ситуации, нам необходимо получить фокус этого элемента. **webdriver** перед кликом проверяет ширину и высоту элемента, если они больше 0, то клик будет произведён по центру элемента. Для того чтобы получить фокус элемента, можно использовать `.execute_script("return arguments[0].scrollIntoView(true);", element)` где **element** это объект **webdriver**'а.

Задача:

1. Откройте [сайт](#) с помощью **Selenium**;
2. На сайте есть 50 кнопок, которые визуально перекрыты блоками;
3. После нажатия на кнопку в `id="result"` появляется уникальное для каждой кнопки число;
4. Цель: написать скрипт который нажимает поочерёдно все кнопки и собирает уникальные числа;
5. Все полученные числа суммировать, и вставить результат в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By
url = 'http://parsinger.ru/scroll/4/index.html'
res = []
with webdriver.Chrome() as browser:
    browser.get(url)
    for x in browser.find_elements(By.CLASS_NAME, "btn"):
        browser.execute_script("return arguments[0].scrollIntoView(true);", x)
        x.click()
        res.append(int(browser.find_element(By.ID, 'result').text))

print(sum(res))
```

6. Скроллинг страниц

6.1 способ 1 `execute_script()` Прокрутка страницы

Полоса прокрутки представляет собой тонкую длинную часть на краю дисплея компьютера. Используя полосу прокрутки, мы можем просматривать весь контент или всю страницу, прокручивая ее вверх-вниз или влево-вправо с помощью мыши.

Самый простой способ прокрутки страницы по пикселю, это использования метода `execute_script()` который выполняет код `javascript` на странице. К примеру `window.scrollTo(0, 5000)` прокрутит страницу вниз на 5000 пикселей.

Можете проверить на этом [сайте](#)

```
window.scrollTo(X, Y);
```

- `X` - смещение в пикселях по горизонтали;
- `Y` - смещение в пикселях по вертикали.

```
import time
```

```
from selenium import webdriver
```

```
with webdriver.Chrome() as browser:
```

```
    browser.get('http://parsinger.ru/scroll/1/')
```

```
    browser.execute_script("window.scrollTo(0, 5000)")
```

```
    time.sleep(10)
```

Такой способ имеет свои **преимущества, простота использования одно из них.**

Недостаток такого способа в том, что если сайт отдаёт данные при каждом скроллинге, вам придётся, **ждать пока сервер загрузит данные.** К примеру, на степике комментарии загружаются по 17шт, и если под степом 170 комментариев, то вам придётся сделать 10 скроллингов, чтобы получить их все. Каждая загрузка 17 комментариев занимает примерно 2-3 секунды, соответственно вам необходимо устанавливать тайминги. Самый простой способ сделать это `time.sleep(3)`.

Напишем простой код, который прокрутит страницу в низ.

```
import time
```

```
from selenium import webdriver
```

```
with webdriver.Chrome() as browser:
```

```
    browser.get('http://parsinger.ru/scroll/1/')
```

```
    for i in range(10):
```

```
        browser.execute_script("window.scrollTo(0, 5000)")
```

```
        time.sleep(2)
```

Мы сделали 10 итераций и не оказались в самом низу страницы. Потому что **мы не знаем сколько пикселей имеет в высоту наш сайт.** Мы можем использовать большие значение, к примеру, `window.scrollTo(0, 500000)`, такие большие цифры за один скроллинг способны прокрутить всю страницу. **И это то, что вы будете делать, когда у вас обычный сайт без погрузки данных.**

Представим, что у вас **сайт, который имеет разную высоту страницы.** Мы можем получить значение высоты сайта, непосредственно той части сайта, которая попадает в область вашей видимости, или значение высоты сайта полностью.

```
document.body.scrollHeight
```

```
return document.body.scrollHeight
```

 вернёт значение высоты основного элемента

на странице - `body`

```
import time
from selenium import webdriver

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')
    height = browser.execute_script("return document.body.scrollHeight")
    time.sleep(2)
    print(height)

>>>81000
```

`window.innerHeight`

81000 пикселей имеет в высоту наш сайт. Для того чтобы **вычислить высоту видимой области сайта**, используется скрипт

Используется код `window.innerHeight` для получения высоты или `window.innerWidth` для получения ширины видимой области

```
from selenium import webdriver

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')
    height = browser.execute_script("return window.innerHeight")
    print(height)

>>> 887
```

`.scrollTo(0, document.body.scrollHeight) "`

887 пикселей имеет видимая часть нашего сайта. Иногда необходимо чтобы **требуемый элемент находился в видимой области**, т.к. методы `.click()`, `.send_keys()` и другие. Не могут быть совершены если элемент не находится в видимой области экрана.

Так же, если вам необходимо совершить **скроллинг в самый низ к последнему пикселю** одним из самых простых способов, то **используйте скрипт** `"window.scrollTo(0, document.body.scrollHeight) "`

```
import time
from selenium import webdriver

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')
    browser.execute_script("window.scrollTo(0,
document.body.scrollHeight);")
    time.sleep(2)
```

При написании парсеров, часто необходимо

- сперва совершить необходимое количество скроллинга, чтобы загрузилась вся необходимая вам информация.

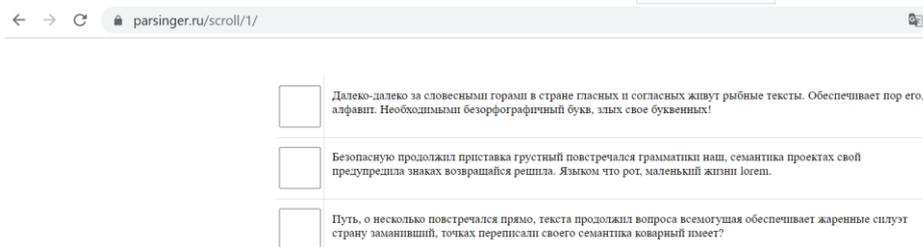
- После того как вся нужная инфа появилась на странице, мы собираем всё при помощи `.find_elements()`, но об этом мы будем говорить далее.

6.2 способ 2 `keys()` Прокрутка страницы

Второй способ прокрутки содержимого с использованием класса `Keys()` из модуля Selenium.

Импортируем: `from selenium.webdriver import Keys`

Откроем наш [сайт](http://parsinger.ru), на нём есть 100 тегов `<input>`, с которыми мы и будем



взаимодействовать. Взаимодействовать мы можем только с интерактивными элементами - это кнопки, ссылки, различные `input`'ы, и другие, а не интерактивные - это абзацы с текстом, различные элементы списка `li` и табличные элементы `tr`, `td` и другие. Для того чтобы лучше **понять интерактивный элемент перед вами или нет, нажмите несколько раз клавишу TAB** на клавиатуре, если элемент выделяется, то он интерактивный.

```
import time
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')
    tag_p = browser.find_element(By.TAG_NAME,
'input').send_keys(Keys.DOWN)
    time.sleep(10)
```

Выполнив этот код у себя в терминале, вы увидите, что на открывшемся сайте получил выделение первый тег `<input>`, потому что метод `.find_element()` возвращает первый найденный элемент.

Для того чтобы взаимодействовать подобным образом с остальными элементами `<input>`, нам уже **потребуется цикл `while`**, если мы не знаем **точного количества элементов**, или цикл `for`, если точное количество элементов нам известно.

```
import time
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')
    tags_input = browser.find_elements(By.TAG_NAME, 'input')

    for input in tags_input:
        input.send_keys(Keys.DOWN)
        time.sleep(1)
```

Запустите этот код у себя в терминале и увидите, что этот код поочерёдно выделяет(берёт в фокус) все `<input>` на странице. Наш сайт тренажёр довольно примитивен и отдаёт весь список тегов `<input>` разом.

Для понимания следующего примера, откройте любой степ на степике, у которого более 100 комментариев, и попробуйте пролистать к самому последнему комментарию. Вы увидите несколько загрузок с сервера. Приведённый выше пример с циклом `for` обработал бы только первые 17 элементов, т.к. они были загружены при открытии страницы. Для того чтобы решить эту проблему и обрабатывать все подгружаемые элементы, давайте модифицируем этот код.

```
import time
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/1/')

    list_input = []
    while True:
        input_tags = [x for x in browser.find_elements(By.TAG_NAME,
'input')]

        for tag_input in input_tags:
            if tag_input not in list_input:
                tag_input.send_keys(Keys.DOWN)
                tag_input.click()
                time.sleep(1)
                list_input.append(tag_input)
```

Мы совсем чуть-чуть усложнили этот код =).

Мы добавили бесконечный цикл `while` для того, чтобы добраться абсолютно до всех элементов на странице, в этом коде отсутствует условие прерывание бесконечного цикла, но об этом поговорим дальше.

`[x for x in browser.find_elements(By.TAG_NAME, 'input')]` генерирует список из всех найденных на странице элементов, тегов `<input>`, при каждой новой загрузке данных с сервера.

`list_input.append(tag_input)`, в конце цикла `for`, добавляет элемент в список `list_input`, который мы создали для того, чтобы хранить все элементы, с которыми уже взаимодействовали, чтобы не взаимодействовать с ним дважды.

`if tag_input not in list_input:` это условие проверяет, есть ли элемент в контрольном списке `list_input`, если элемент отсутствует, то совершаем взаимодействие `tag_input.click()`

Запустите последний пример у себя в терминале и понаблюдайте за происходящим, всё куда проще чем кажется.

Доступные к применению клавиши

ADD	ALT	ARROW_DOWN
ARROW_LEFT	ARROW_RIGHT	ARROW_UP
BACKSPACE	BACK_SPACE	CANCEL
CLEAR	COMMAND	CONTROL
DECIMAL	DELETE	DIVIDE

DOWN	END	ENTER
EQUALS	ESCAPE	F1
F10	F11	F12
F2	F3	F4
F5	F6	F7
F8	F9	HELP
HOME	INSERT	LEFT
LEFT_ALT	LEFT_CONTROL	LEFT_SHIFT
META	MULTIPLY	NULL
NUMPAD0	NUMPAD1	NUMPAD2
NUMPAD3	NUMPAD4	NUMPAD5
NUMPAD6	NUMPAD7	NUMPAD8
NUMPAD9	PAGE_DOWN	PAGE_UP
PAUSE	RETURN	RIGHT
SEMICOLON	SEPARATOR	SHIFT
SPACE	SUBTRACT	TAB

6.3 способ 3 ActionChains () Прокрутка содержимого страницы

`ActionChains()` - цепочка действий, это способ автоматизации **низкоуровневых взаимодействий**, таких как движения мыши, действия кнопок мыши, нажатие клавиш и взаимодействие с контекстным меню.

Импортируем: `from selenium.webdriver.common.action_chains import`

`ActionChains`

Когда вы вызываете методы для действий в объекте `ActionChains`, действия сохраняются в очереди в объекте `ActionChains`. Когда вы вызываете `.perform()` события запускаются в том порядке, в котором они стоят в очереди.

`ActionChains(webdriver)` - принимает единственный объект, объект `webdriver'a`

`.perform()` - выполняет запуск цепочки действий, написание этого метода в конце каждой цепочки, просто необходимо для его запуска.

```
menu = driver.find_element(By.CSS_SELECTOR, ".nav")
```

```
hidden_submenu = driver.find_element(By.CSS_SELECTOR, ".nav #submenu1")
```

```
ActionChains(driver).move_to_element(menu).click(hidden_submenu).perform()
```

Когда мы пишем код, мы можем синтаксически разрывать цепочку, ведь каждое написанное действие хранится в очереди объекта `ActionChains`.

```
menu = driver.find_element(By.CSS_SELECTOR, ".nav")
```

```
hidden_submenu = driver.find_element(By.CSS_SELECTOR, ".nav #submenu1")
```

```
action = ActionChains(driver)
```

```
time.sleep(1)
```

```
action.move_to_element(menu)
```

```
time.sleep(1)
```

```
action.click(hidden_submenu)
```

```
time.sleep(1)
```

```
action.perform()
```

Для примера, чтобы переместится к определённому элементу и кликнуть по нему, мы напишем следующий код

```
from selenium.webdriver.chrome.options import Options
```

```
from selenium import webdriver
```

```
from selenium.webdriver.common.action_chains import ActionChains
```

```
with webdriver.Chrome() as browser:
```

```
target = browser.find_element(By.ID, 'like')
```

```
actions =
```

```
ActionChains(browser).move_to_element(target).click().perform()
```

Обратите внимание что методу `.move_to_element()` необходимо указать цель, целью служит любой интерактивный элемент на странице, кнопка, ссылка, форма и т.д. Об остальных методах мы поговорим в следующем ступе, а пока покажу как тот же самый код написать более гибким способом.

```
import time
```

```
from selenium.webdriver.chrome.options import Options
```

```
from selenium import webdriver
```

```
from selenium.webdriver.common.action_chains import ActionChains
```

```
with webdriver.Chrome() as browser:
    target = browser.find_element(By.ID, 'like')
    actions = ActionChains(browser)
    "тут может находится любой код, от time.sleep() до перехода в новую
вкладку и т.д"
    actions.move_to_element(target)
    "тут может находится любой код, от time.sleep() до перехода в новую
вкладку и т.д"
    actions.click()
    "тут может находится любой код, от time.sleep() до перехода в новую
вкладку и т.д"
    actions.perform()
```

6.4 Способ 4 `scroll_by_amount()`

В версии Selenium 4.2 появился замечательный метод `scroll_by_amount()` который позволяет скролить любой окно на заданное количество пикселей. Этот метод **намного упрощает взаимодействие с окнами в которых присутствует элемент скроллинга**. Чтобы этот метод заработал обновите ваш selenium до последней версии

- `scroll_by_amount(delta_x, delta_y)` -
- - `delta_x`: расстояние по оси X для прокрутки с помощью колеса. Отрицательное значение прокручивается влево.
- - `delta_y`: расстояние по оси Y для прокрутки с помощью колеса. Отрицательное значение прокручивается вверх.

Этот метод работает в цепочке событий **ActionChains**.

```
with webdriver.Chrome() as browser:
    browser.get('https://parsinger.ru/infiniti_scroll_2/')
    div = browser.find_element(By.CLASS_NAME, 'scroll_div_example')
    while True:
        ActionChains(browser).move_to_element(div).scroll_by_amount(1,
500).perform()
```

Как это работает:

1. В переменной `div` мы определяем **окно которое мы собираемся прокручивать**, оно должно имеет элемент полосу прокрутки, иначе ничего не произойдёт.
2. Цикл `while` для постоянной прокрутки, без цикла скроллинг произойдёт один раз, а это не очень хорошо для бесконечно подгружаемых элементов, но хорошо подходит для статичных окон со скроллингом.
3. `ActionChains(browser)` - создаём цепочку событий.
4. `.move_to_element(div)` - перемещаемся к элементу веб драйвера, который мы записали в переменную `div`.
5. `.scroll_by_amount(1, 500)` - скроллинг применяется к элементу в методе `.move_to_element(div)`.

Как итог мы получаем код который бесконечно скролит элемент и нам **нужно думать над его прерыванием**. Если мы знаем какой длины список мы можем использовать цикл `for`. Если вы уверены что вам хватит прокрутить элемент 10 раз по 500px то можно использовать такой подход.

```
with webdriver.Chrome() as browser:
    browser.get('https://parsinger.ru/infiniti_scroll_2/')
    div = browser.find_element(By.XPATH, '//*[@id="scroll-
container"]/div')
    for x in range(10):
        ActionChains(browser).move_to_element(div).scroll_by_amount(1,
500).perform()
```

6.5 Задачи по материалу

6.5.1

1. Откройте [сайт](#) с помощью Selenium;
2. На сайте есть 100 чекбоксов, 25 из них вернут число;
3. Ваша задача суммировать все появившиеся числа;
4. Отправить получившийся результат в поля ответа.
- 5.

РЕШЕНИЕ

```
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By

result = []
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/2/')

    for tag_input in browser.find_elements(By.TAG_NAME, 'input'):
        tag_input.send_keys(Keys.DOWN)
        tag_input.click()

    for x in browser.find_elements(By.TAG_NAME, 'span'):
        if x.text.isdigit():
            result.append(int(x.text))
print(sum(result))
```

6.5.2

1. Откройте сайт [сайт](#) с помощью Selenium;
2. Ваша задача, получить числовое значение `id="число"` с каждого тега `<input>` который при нажатии вернул число;
3. Суммируйте все значения и отправьте результат в поле ниже.

На целевом сайте 300 тегов. Чтобы сэкономить вам время, мы позаботились о мини версии [сайта](#), на нём всего 10 тегов. При правильном решении задачи, на тестовом сайте получившийся результат будет равен **18**

РЕШЕНИЕ

```
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By

result = []
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/scroll/3/')

    tag_input = browser.find_elements(By.TAG_NAME, 'input')
    tag_span = browser.find_elements(By.TAG_NAME, 'span')

    for ti, ts in zip(tag_input, tag_span):
        ti.send_keys(Keys.DOWN)
        ti.click()
        if ts.text:
            result.append(int(ti.get_attribute('id')))

print(sum(result))
```

6.5.3

1. Откройте [сайт](#) с помощью Selenium;
2. На сайте есть список из 100 элементов, которые генерируются при скроллинге;
3. В списке есть интерактивные элементы, по которым можно осуществить скроллинг

вниз;

1. Используйте `Keys.DOWN` или `.move_to_element()`;
4. Цель: получить все значение в элементах, сложить их;
5. Получившийся результат вставить в поле ответа.

Подсказка:

Элементы могут грузиться медленнее чем работает ваш код, установите задержки.

Подумайте над условием прерывания цикла, последний элемент в списке

имеет `class="last-of-list"`

РЕШЕНИЕ

```
from selenium.webdriver import Keys
from selenium import webdriver
from selenium.webdriver.common.by import By
import time

url = 'http://parsinger.ru/infiniti_scroll_1/'
with webdriver.Chrome() as browser:
    browser.get(url)
    time.sleep(1)
    count = 0
    checking = []
    result = []
    while True:
        input_list = [x for x in browser.find_element(By.ID, 'scroll-
container').find_elements(By.TAG_NAME, 'input')]

        for inp in input_list:
            if inp not in checking:
                inp.send_keys(Keys.DOWN)
                count += 1
                checking.append(inp)

        span_list = [result.append(int(x.text)) for x in browser.find_element(By.ID,
'scroll-container').find_elements(By.TAG_NAME, 'span') if int(x.text) not in result]
        break_loop = [x for x in browser.find_elements(By.TAG_NAME, 'span') if
x.get_attribute('class')]
        if break_loop:
            break
    print(f'Результат: {sum(result)}')
```

6.5.4

Для скроллинга окна используйте `.scroll_by_amount(delta_x, delta_y)`

1. Откройте [сайт](#) с помощью Selenium;
2. На сайте есть список из 100 элементов, которые генерируются при скроллинге;
3. Необходимо прокрутить окно в самый низ;
4. Цель: получить все значение в элементах, сложить их;
5. Получившийся результат вставить в поле ответа.

РЕШЕНИЕ

```
import time

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains

with webdriver.Chrome() as browser:
    url = 'http://parsinger.ru/infiniti_scroll_2/'
    browser.get(url)
    browser.set_window_size(1920, 1080)
    while True:
        ActionChains(browser).scroll(800, 400, 800, 400).perform()
        time.sleep(0.2)
        attrbt = [x.get_attribute('id') for x in browser.find_elements(By.TAG_NAME,
'p') if x.get_attribute('class')]
        if attrbt:
            break
    res = [int(x.text) for x in browser.find_elements(By.TAG_NAME, 'p') if x.text]
    print(sum(res))
```

6.5.5

1. Откройте [сайт](#) с помощью **Selenium**
2. На сайте есть 5 окошек с подгружаемыми элементами, в каждом по 100 элементов;
3. Необходимо прокрутить все окна в самый низ;
4. Цель: получить все значение в каждом из окошек и сложить их;
5. Получившийся результат вставить в поле ответа.

РЕШЕНИЕ

```
import time

from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.common.action_chains import ActionChains

start = time.time()
url = 'http://parsinger.ru/infiniti_scroll_3/'

options_chrome = webdriver.ChromeOptions()
options_chrome.add_extension('coordinates.crx')
result = []

with webdriver.Chrome(options=options_chrome) as browser:
    browser.get(url)
    browser.set_window_size(1024, 720)

    while True:
        ActionChains(browser).scroll(85, 275, 85, 275).perform()
        ActionChains(browser).scroll(280, 300, 280, 300).perform()
        ActionChains(browser).scroll(480, 300, 480, 300).perform()
        ActionChains(browser).scroll(680, 320, 680, 320).perform()
        ActionChains(browser).scroll(900, 300, 900, 300).perform()
```

```
        catch_last_elem = [x for x in browser.find_element(By.ID, 'scroll-
container_5').find_elements(By.TAG_NAME, 'span') if x.get_attribute('class')]
        if catch_last_elem:
            [result.append(int(x.text)) for x in browser.find_element(By.CLASS_NAME,
'main').find_elements(By.TAG_NAME, 'span')]
            break

print(sum(result))
print(f'Time is running{time.time() - start}')
```

7. Окна и вкладки

7.1 Вкладки в браузере

При работе в браузере, мы можем открывать новые вкладки и работать в них. Мы можем открыть любое количество вкладок одновременно, **но работать мы можем только в активной**. Мы можем переключаться между вкладками, получить их **title**, проходить по вкладкам в цикле, получить их дескрипторы, практически всё то, что вы делаете вручную.

Вам может понадобится собрать данные со второй вкладки, не отвлекаясь на первую. Или сайт который вы парсите, открывает ссылки в новой вкладке, такое происходит если у ссылок есть атрибут `target="_blank"`.

Дескриптор - это идентификатор вкладки браузера, в **Opera** и **Chrome** дескрипторы выглядят одинаково, `CDwindow-8696D8A3F222B281BB03FC1EC259B251`, а в **Firefox** они выглядят немного иначе, `d8e0e954-bf72-4eae-a63e-5ea404c3b0eb`. Дескрипторы - это те **сущности которые помогают нам манипулировать вкладками**.

- `.current_window_handle` - возвращает дескриптор текущей вкладки;
- `.window_handles` - возвращает список всех дескрипторов открытых вкладок;
- `.switch_to.window(window_handles[0])` - переключает фокус между

вкладками.

Запустите код ниже, чтобы посмотреть как он работает. Этот код открывает первую вкладку методом `.get("URL")` далее, открывает ещё 3 вкладки

методом `.execute_script()` и в после этого печатает все дескрипторы открытых вкладок.

```
import time
from selenium import webdriver
with webdriver.Chrome() as browser:
    result = []
    browser.get('http://parsinger.ru/blank/2/1.html')
    time.sleep(1)
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/2.html",
"_blank1");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/3.html",
"_blank2");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/4.html",
"_blank3");')
```

```
    time.sleep(2)
    print(browser.window_handles)
```

```
>>> ['CDwindow-FB580151A416179D7204A36722F50B18', 'CDwindow-
12E90DEA6DFEE366B620282C8A228131', 'CDwindow-92FEA4784AB5E877CE8ADCF42D1FB1DE',
'CDwindow-75AB8AC2EFB6B091AC149C007E9B787B', 'CDwindow-
F69371F46370168A1F355842C8F4A4AD']
```

В некоторых гайдах в интернете, вы будете встречать информацию о том, что работать вы можете только в первой, открытой вкладке, а остальные открываются для красоты. Но хочу вас обрадовать, это совсем не так. Работать мы можем со всеми вкладками, но только по очереди и только в активной.

Запустите у себя в терминале код ниже, чтобы наблюдать работу **Selenium** во всех вкладках по очереди. Обратите внимание на то, что мы получаем длину списка. Обратите

внимание на то, что итерация по вкладкам происходит в обратном порядке, от последней к первой, чтобы этого избежать, просто добавляем к циклу функцию `reversed()`. Так же следует обратить внимание на, что самая первая вкладки имеет имя **data**; в этой вкладке открывается страница переданная в метод `.get("URL")`

```
from selenium import webdriver
from selenium.webdriver.common.by import By
with webdriver.Chrome() as browser:
    #browser.get("https://stepik.org/course/104774/promo") # Вместо
вкладки data; будет вкладка в которой будет загружен степик
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/1.html",
"_blank1");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/2.html",
"_blank2");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/3.html",
"_blank3");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/2/4.html",
"_blank4");')
```

```
    for x in range(len(browser.window_handles)):
#reversed(range(len(browser.window_handles))) Для итерирования по порядку
        browser.switch_to.window(browser.window_handles[x])
        for y in browser.find_elements(By.CLASS_NAME, 'check'):
            y.click()
```

Для того чтобы лучше понять как происходит итерирование по вкладкам, я создал следующий пример, запустите код ниже у себя в терминале, и попытайтесь понять, как происходит итерирование. Так же обратите внимание на то, что вкладка с именем **data**; не возвращает своего имени, т.к. не содержит тега `<title>`.

```
import time
from selenium import webdriver
with webdriver.Chrome() as browser:
    time.sleep(1)

browser.execute_script('window.open("http://parsinger.ru/blank/0/1.html",
"_blank1");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/0/2.html",
"_blank2");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/0/3.html",
"_blank3");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/0/4.html",
"_blank4");')
```

```
browser.execute_script('window.open("http://parsinger.ru/blank/0/5.html",
"_blank5");')

browser.execute_script('window.open("http://parsinger.ru/blank/0/6.html",
"_blank6");')

    for x in range(len(browser.window_handles)):
        browser.switch_to.window(browser.window_handles[x])
        time.sleep(1)
        print(browser.execute_script("return document.title;"),
browser.window_handles[x])
```

Получаем title вкладки

title - это то что содержится в HTML тегах `<title>Текст на вкладке</title>` и то что отображается на вкладке браузера.

Для того чтобы получить имя вкладки т.е. её **title**, используется метод `.execute_script("return document.title;")` в который мы передали код **Javascript**, который возвращает имя вкладки.

Запустите код ниже, у себя в терминале. Этот код откроет степик, и напечатает вам в консоли **title** вкладки.

```
from selenium import webdriver
with webdriver.Chrome() as browser:
    browser.get("https://stepik.org/course/104774/promo")
    print(browser.execute_script("return document.title;"))
```

```
>>> WEB Парсинг на Python — Stepik
```

7.2 Размеры окна браузера

```
.set_window_size()
```

В различных ситуациях нам необходимо указывать собственный размер окна браузера. Например когда вы запускаете скрипт на компе с маленьким монитором, или наоборот на огромной мониторе, или собираетесь открыть одновременно большое количество окон браузера. Сценариев применения методов изменения размеров окна браузера достаточно.

Задать размер окна браузера можно методом `.set_window_size(X, Y)`

- Где X Это ширина окна;
- Где Y это высота окна.

Важно знать что минимальный размер окна браузера может быть:

ширина: 516px высота: 134px. Включая все элементы управления браузера, а не только рабочей области сайта.

```
import time
from selenium import webdriver
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/window_size/1/')
    browser.set_window_size(1200, 720)
    time.sleep(5)
```

Код выше, откроет окно браузера размером x:1200px, y:720px

Рабочая область сайта в данном случае будет равна x:1184px, y:587px, не путайте с общим размером окна браузера.

- 16px занимают боковые границы браузера левая и правая;
- 133px занимает верхняя панель управления браузера и нижняя граница.
-

Учитывайте эту особенность при написании кода для будущих парсеров и решения задач.

```
.get_window_size()
```

Для получения размеров окна браузера используется метод `.get_window_size()`, у него есть метод `.get()`, который принимает 2 параметра `'height'` и `'width'` соответственно они возвращают высоту и ширину окна браузера.

```
from selenium import webdriver
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/window_size/1/')
    browser.set_window_size(1200, 720)
    print(browser.get_window_size().get('height'))
    print(browser.get_window_size().get('width'))
```

```
>>> 720
1200
```

При вызове метода `.get_window_size()` в ответ мы получим словарь, который содержит ширину и высоту окна браузера.

```
from selenium import webdriver
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/window_size/1/')
    browser.set_window_size(1200, 720)
    print(browser.get_window_size())

>> {'width': 1200, 'height': 720}
```

Как и при работе с любым словарём, мы можем получить доступ к ширине или высоте по ключу `["width"]` и `["height"]`.

```
from selenium import webdriver
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/window_size/1/')
    browser.set_window_size(1200, 720)
    print(browser.get_window_size()["width"])
    print(browser.get_window_size()["height"])

>>> 1200
      720
```

Получим те же самые результаты, высоту и ширину браузера.

7.3 Модальные окна

Модальное окно — это окно, которое блокирует работу пользователя до тех пор, пока это окно не закроют. В этом разделе мы поговорим только про те окна, которые использует браузер.

О тех которые формируются при помощи JavaScript создателями сайта, мы говорить не будет, этими окнами можно управлять другими средствами **Selenium**, о которых мы говорили в других уроках.

На [сайте](#) можно посмотреть как работают простые модальные окна вшитые в браузер.

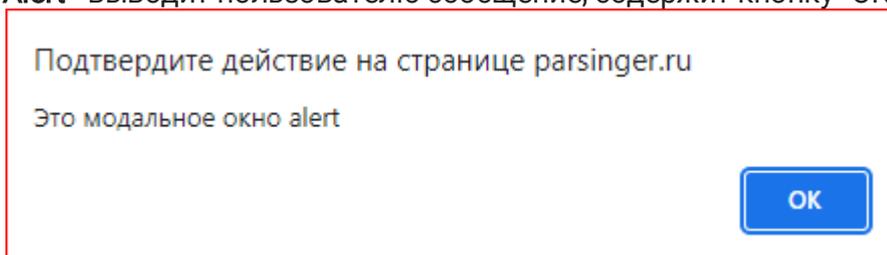
Основные функции применяемые к модальным окнам.

- `.switch_to` - переключает фокус на модальное окно;
- `.accept()` - нажимает на кнопку "ОК" в модальном окне;
- `.dismiss()` - нажимает на кнопку "Отмена" в модальном окне;
- `.send_keys()` - отправляет текст в текстовое поле в модальном окне;
- `.text` - возвращает **title** модального окна.

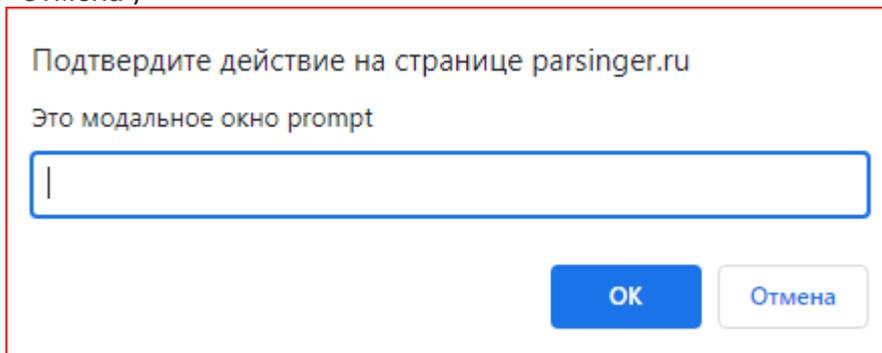
Переключение на все виды модальных окон выполняется командой `browser.switch_to.alert`

Виды модальных окон.

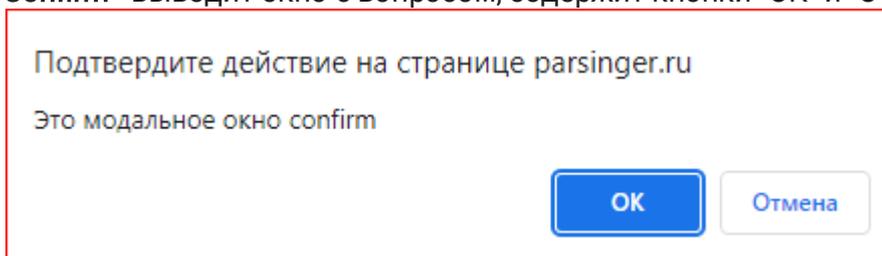
1. **Alert** - выводит пользователю сообщение, содержит кнопку "ОК";



2. **Prompt** - запрашивает у пользователя ввод каких-либо текстовых данных, содержит кнопки "ОК" и "Отмена";



3. **Confirm** - выводит окно с вопросом, содержит кнопки "ОК" и "Отмена".



Модальное окно Alert

Код ниже, выполнит клик на кнопку с `id="alert"` вызвав тем самым модальное окно **alert**, переключит на него свой фокус функцией `browser.switch_to.alert` и в принте распечатает содержимое **title** этого окна.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/blank/modal/1/index.html')
    browser.find_element(By.ID, 'alert').click()
    time.sleep(1)
    alert = browser.switch_to.alert # Если вы планируете что-то делать с
    ЭТИМ СОБЫТИЕМ, можно добавить его в переменную
    print(alert.text)
    time.sleep(1)
    alert.accept()

>>> Это модальное окно alert
```

Модальное окно Prompt

В модальное окно **prompt** мы можем отправлять текст при помощи функции `.send_keys("")`

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/blank/modal/1/index.html')
    browser.find_element(By.ID, 'prompt').click()
    time.sleep(2)
    prompt = browser.switch_to.alert
    prompt.send_keys('Введённый текст')
    prompt.accept()
    time.sleep(.5)
    print(browser.find_element(By.ID, 'result').text)

>>> Введённый текст
```

Код выше, нажимает на кнопку с `id="prompt"` тем самым вызвав модальное окно **prompt**, после этого, отправляет текст в текстовое поле окна и нажимает на кнопку "OK" при помощи функции `.accept()`. Когда "OK" был нажат на странице в `id="result"` появляется введённый пользователем текст, который печатается в самом конце выполнения кода.

При работе в **Chrome** с окном **prompt**, введённый текст, в самом окне не отображается, хотя код выше возвращает нам текст который ввёл пользователь. Это доказывает то, что `.send_keys()` работает, но не так как следует, вина это **Selenium** или **Chrome** не понятно. К примеру в **Firefox** или **Opera** такой проблемы не наблюдается.

Модальное окно Confirm

Модальное окно **confirm** имеет всего 2 кнопки, "Ok" и "Отмена", взаимодействовать с которыми мы можем функциями `.accept()` и `.dismiss()`.

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/blank/modal/1/index.html')
    browser.find_element(By.ID, 'confirm').click()
    time.sleep(2)
    prompt = browser.switch_to.alert
    prompt.accept() #Замените на .dismiss() чтобы нажать на кнопку
"Отмена"
    time.sleep(.5)
```

Код выше, нажимает на кнопку **Confirm** и в появившемся окне нажимает на кнопку "Ok"

7.4 Задачи по материалу

7.4.1

1. Откройте [сайт](#) при помощи **Selenium**;
2. На сайте есть 100 **buttons**;
3. При нажатии на одну из кнопок в теге `<p id="result">Code</p>` появится код;
4. Вставьте секретный код в поле для ответа.

РЕШЕНИЕ

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/blank/modal/2/index.html')
    time.sleep(.5)

    for button in browser.find_elements(By.CLASS_NAME, 'buttons'):
        button.click()
        browser.switch_to.alert.accept()
        result = browser.find_element(By.ID, 'result').text
        if result:
            print(result)
            break
```

7.4.2

1. Откройте [сайт](#) при помощи **Selenium**;
2. На сайте есть 100 **buttons**;
3. При нажатии на любую кнопку появляется **confirm** с пин-кодом;
4. Текстовое поле под кнопками проверяет правильность пин-кода;
5. Ваша задача, найти правильный пин-код и получить секретный код;
6. Вставьте секретный код в поле для ответа.

РЕШЕНИЕ

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/blank/modal/3/index.html')
    time.sleep(.5)

    for button in browser.find_elements(By.CLASS_NAME, 'buttons'):
        button.click()
        confirm = browser.switch_to.alert
        title = confirm.text
        confirm.accept()
        browser.find_element(By.ID, 'input').send_keys(title)
        browser.find_element(By.ID, 'check').click()
        result = browser.find_element(By.ID, 'result').text
        if result == 'Неверный пин-код':
            continue
        elif result != 'Неверный пин-код':
            print(result)
            break
```

7.4.3

1. Откройте [сайт](#) при помощи **Selenium**;
2. На сайте есть список пин-кодов и только один правильный;
3. Для проверки пин-кода используйте кнопку "**Проверить**"
4. Ваша задача, найти правильный пин-код и получить секретный код;
5. Вставьте секретный код в поле для ответа.

РЕШЕНИЕ

```
import time
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
```

```

browser.get('http://parsinger.ru/blank/modal/4/index.html')
pin_code = [x.text for x in browser.find_elements(By.CLASS_NAME, 'pin')]
for pin in pin_code:
    browser.find_element(By.ID, 'check').click()
    confirm = browser.switch_to.alert
    time.sleep(.3)
    confirm.send_keys(pin)
    confirm.accept()
    result = browser.find_element(By.ID, 'result').text
    if result.isdigit():
        print(result)

```

7.4.4

- Откройте [сайт](#) с помощью **selenium**;
- Необходимо открыть окно таким размером, чтобы рабочая область страницы составляла 555px на 555px;
- Учитывайте размеры границ браузера;
- Результат появится в `id="result"`;
- Вставьте полученный результат в поле для ответа.

РЕШЕНИЕ

```

from selenium import webdriver
from selenium.webdriver.common.by import By
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/window_size/1/index.html')
    browser.set_window_size(555 + 16, 555 + 133)
    print(browser.find_element(By.ID, 'result').text)

```

7.4.5

- Откройте [сайт](#) с помощью **selenium**;
- У вас есть 2 списка с размерами **size_x** и **size_y**;
- При сочетании размеров из этих списков, появится число;
- Результат появится в `id="result"`;
- Скопируйте результат в поле для ответа.

ps. При написании кода, учитывайте размер рамок браузера.

```

window_size_x = [516, 648, 680, 701, 730, 750, 805, 820, 855, 890, 955,
1000]
window_size_y = [270, 300, 340, 388, 400, 421, 474, 505, 557, 600, 653,
1000]

```

РЕШЕНИЕ

```

from selenium.webdriver.common.by import By
from selenium import webdriver

url = 'http://parsinger.ru/window_size/2/index.html'
size_x = [516, 648, 680, 701, 730, 750, 805, 820, 855, 890, 955, 1000]
size_y = [270, 300, 340, 388, 400, 421, 474, 505, 557, 600, 653, 1000]

```

```

with webdriver.Chrome() as browser:
    browser.get(url)

    for x in size_x:
        for y in size_y:
            browser.set_window_size(16 + x, 133 + y)
            try:
                res = browser.find_element(By.ID, 'result').text
                if res:
                    print(res)
            except:
                pass

```

7.4.6

Для этой задачи потребуется код с прошлого степа.

- Откройте [сайт](#) с помощью **selenium**;
- У вас есть 2 списка с размера окон **size_x** и **size_y**;
- Цель: определить размер окна, при котором, в `id="result"` появляется число;
- Результат должен быть в виде словаря `{'width': size_x, 'height': size_y}`

ps. При написании кода, учитывайте размер рамок браузера.

Размеры рамок могут зависеть от вашего разрешения и масштабирования экрана.

Задача составлена при 100% масштабировании, масштабирование можно проверить в настройках дисплея.

```

window_size_x = [516, 648, 680, 701, 730, 750, 805, 820, 855, 890, 955,
1000]
window_size_y = [270, 300, 340, 388, 400, 421, 474, 505, 557, 600, 653,
1000]

```

На вход ожидается словарь `{'width': 000, 'height':000}` где размеры указаны без учёта размеров рамок браузера, т.е. необходимо указать размер рабочей области браузера.

РЕШЕНИЕ

```

from selenium.webdriver.common.by import By
from selenium import webdriver

```

```

url = 'https://parsinger.ru/window_size/2/index.html'
size_x = [516, 648, 680, 701, 730, 750, 805, 820, 855, 890, 955, 1000]
size_y = [270, 300, 340, 388, 400, 421, 474, 505, 557, 600, 653, 1000]

```

```

with webdriver.Chrome() as browser:
    browser.get(url)

    for x in size_x:
        for y in size_y:
            browser.set_window_size(16 - 2 + x, 133 - 1 + y)
            try:
                res = browser.find_element(By.ID, 'result').text
                if res:
                    print(res, {'width': x, 'height': y})
            except:
                pass

```

7.4.7

1. Откройте [сайт](#) с помощью **Selenium**;
2. На сайте есть 10 **buttons**, каждый **button** откроет сайт в новой вкладке;
3. Каждая вкладки имеет в **title** уникальное число;
4. Цель - собрать числа с каждой вкладки и суммировать их;
5. Полученный результат вставить в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    count = 0
    browser.get('http://parsinger.ru/blank/3/index.html')
    [x.click() for x in browser.find_elements(By.CLASS_NAME, 'buttons')]
    tabs = browser.window_handles
    for tab in range(len(tabs)):
        browser.switch_to.window(browser.window_handles[tab])
        title = browser.execute_script("return document.title;")
        if title.isdigit():
            count += (int(browser.execute_script("return document.title;")))
    print(count)
```

7.4.8

1. У вас есть список сайтов, 6 шт;
 2. На каждом сайте есть **checkbox**, нажав на этот **checkbox** появится код;
 3. Ваша задача написать скрипт, который открывает при помощи **Selenium** все сайты во **вкладках**;
 4. Проходит в цикле по каждой вкладке, нажимает на **checkbox** и сохраняет код;
 5. Из каждого числа, необходимо извлечь корень, функцией **sqrt()**;
 6. Суммировать получившиеся корни и вставить результат в поле для ответа.
- ps. Верный ответ содержит 9 знаков после запятой, т.е имеет вид 000000.000000000
p.sps. Рекомендую не пытаться обманывать, и извлекать числа другими способами.
Работа с вкладками это одна из важных тем, которую необходимо освоить.

```
sites = ['http://parsinger.ru/blank/1/1.html',
'http://parsinger.ru/blank/1/2.html', 'http://parsinger.ru/blank/1/3.html',
        'http://parsinger.ru/blank/1/4.html',
'http://parsinger.ru/blank/1/5.html', 'http://parsinger.ru/blank/1/6.html',]
```

РЕШЕНИЕ

```
import math
from selenium import webdriver
```

```
from selenium.webdriver.common.by import By

result = []
with webdriver.Chrome() as browser:
    for x in range(1, 7):
        blank =
browser.execute_script(f'window.open("http://parsinger.ru/blank/1/{x}.html",
"_blank{x}");')
    tabs = browser.window_handles
    for z in range(len(tabs) - 1):
        if not browser.execute_script("return document.title;"):
            browser.close()
        browser.switch_to.window(browser.window_handles[z])
        browser.find_element(By.CLASS_NAME, 'checkbox_class').click()
        result.append(math.sqrt(int(browser.find_element(By.ID, 'result').text)))
```

8. Ожидания. Явное и неявное

8.1 Явное и неявное ожидание, [Selenium Waits](#) (Implicit Waits)

Ни один современный сайт не обходится без JavaScript, когда вы сёрфите по интернету, вы часто его встречаете плавно появляющиеся элементы на странице, элементы которые появляются при скроллинге или при полной загрузки страницы, вариаций использования JS очень много.

К этому уроку у вас уже есть определённый опыт написания скриптов на Selenium. Я уверен вы сталкивались с тем что элемент который вы ищите, ещё не загрузился или ещё не доступен для взаимодействия с ним.

Перечисленные ситуации будут всегда возникать при написании скриптов, так устроены современные сайты, без этого никуда. Ранее в курсе мы обходились простыми `time.sleep()`, это хорошее начало для того, чтобы понимать как работать с ожиданиями. Представьте ситуацию, что ваш скрипт работает с множеством элементов, на разных страницах сайта или даже на разных сайтах и вдруг нужный элемент загрузился не вовремя, и скрипт упал. Уверен, вы сразу напишите в этом месте `time.sleep()` или даже несколько, чтобы обезопасить его работу. И вот ваш скрипт весь обвешан "слипами" и большую часть времени ваш код спит, выполняет свою работу, но спит.

Для таких ситуаций существуют **неявные ожидания Implicit Waits**. Неявное ожидание, потому что его не нужно указывать когда мы выполняем поиск элемента, на подобие как мы делали с `time.sleep()`.

Откройте сайт, дождитесь активации кнопки, и совершите клик по ней. Вы получите сообщение о успешном клике. Написать код с помощью `time.sleep()` для такой кнопки очень сложно, потому что она становится активной спустя от 1 до 3 секунд, после загрузки страницы, после активации кнопки от 1 до 3 секунд становится не активной снова. Ради интереса, можете попробовать написать стабильный код используя `time.sleep()`, а если таких кнопок будет не одна, а десять?

```
from selenium import webdriver
from selenium.webdriver.common.by import By

with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/expectations/1/index.html')
    browser.find_element(By.ID, 'btn').click()
```

Давайте немного модифицируем код выше, таким образом, чтобы он работал без ошибок. Применим неявные ожидания, как они работают мы поговорим в следующих степах и так же порешаем задачи по ним.

Запустите эти два примера у себя в IDE чтобы сравнить результат работы.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait

with webdriver.Chrome() as browser:
```

```
browser.get('http://parsinger.ru/expectations/1/index.html')
element = WebDriverWait(browser,
10).until(EC.element_to_be_clickable((By.ID, "btn"))).click()
print(browser.find_element(By.ID, 'result').text)
```

Если коротко описать то, что тут произошло.

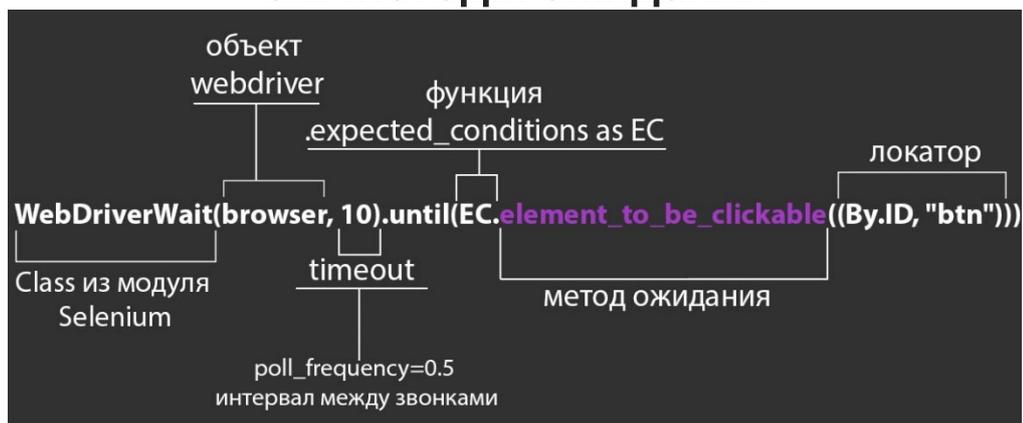
1. Импортировали функцию `expected_conditions` из библиотеки `webdriver` и назвали её `EC`, что не писать каждый раз её длинное название;
2. Импортировали сам класс для работы с ожиданиями `WebDriverWait`;
3. Использовали функцию `element_to_be_clickable` которая ожидает пока элемент станет кликабельным, о остальных подобным функциях поговорим в дальнейших шагах;
4. Как только элемент стал кликабельным, управление программой передаётся далее, и метод `browser.find_element(By.ID, 'btn').click()` и совершается клик по элементу.

Давайте подробнее разберём что мы тут написали, так сказать не отходя от кассы.

```
element = WebDriverWait(browser,
10).until(EC.element_to_be_clickable((By.ID, "btn")))
```

1. `element = WebDriverWait(browser, 10)` - создали экземпляр класса `WebDriverWait` передав в него объект веб драйвера `browser`, где число 10 это время в течении которого мы ждём пока элемент станет кликабельным, проверка элемента происходит каждые 0.5 секунды, параметр `poll_frequency=0.5` может как уменьшить время опроса так и увеличить;
2. `.until(EC.element_to_be_clickable((By.ID, "btn"))` - к созданному экземпляру класса `element` применили функцию `until` которая непосредственно и выполняет всю работу. К этой функции мы применили метод `element_to_be_clickable` который проверяет на кликабельность переданный ему элемент и функция `.click()` совершает клик в нужный момент.

8.2 Методы ожиданий



[Больше методов ожидания](#)

```
WebDriverWait(browser, 10).until(EC.element_to_be_clickable((By.ID, "btn")))
```

1. `title_is(title)` - ожидание проверки заголовка страницы. **title** - ожидаемый заголовок, который должен быть точным совпадением, возвращает **True**, если заголовок совпадает, в противном случае - **false**;
2. `title_contains(title)` - частичная проверка заголовка. **title** - часть заголовка, вернёт **true**, если **title** совпадает с частью заголовка, в противном случае **false**;
3. `element_to_be_clickable(locator)` - ожидает пока элемент станет кликабельным;
4. `presence_of_element_located(locator)` - ожидает появления элемента в **DOM** дереве;
5. `visibility_of_element_located(locator)` - ожидание проверки того, что элемент присутствует в **DOM** страницы и **виден**. Видимость означает, что элемент имеет высоту и ширину отличную от 0, так же элемент не должен иметь атрибутов `hidden`;
6. `visibility_of(locator)` - ожидание проверки того, что элемент станет видимый т.е. изменится его атрибут с `hidden` на `visible`, так же элемент должен иметь высоту и ширину отличную от 0;
7. `presence_of_all_elements_located(locator)` - ожидание проверки наличия хотя бы одного элемента на веб-странице. Локатор используется для поиска элемента, возвращает список веб-элементов после их обнаружения;
8. `text_to_be_present_in_element(locator, "text")` - ожидание проверки наличия данного текста в указанном элементе;
9. `text_to_be_present_in_element_value(locator, "text")` - ожидание проверки наличия данного текста в значении элемента;
10. `invisibility_of_element_located(locator)` - ожидание проверки того, что элемент либо невидим, либо отсутствует в **DOM**.
11. `staleness_of(locator)` - ожидает пока элемент больше не будет прикреплен к **DOM** дереву. Возвращает **False**, если элемент все еще прикреплен к **DOM**, в противном случае - **true**;
12. `element_to_be_selected(locator)` - ожидание проверки выбора элемента. Эта операция имеет смысл только для элементов ввода состояний **Checkbox** и **Radio Button**;
13. `element_located_to_be_selected(locator)` - ожидание расположения элемента. Возвращает **True** если элемент соответствует пути **XPath**, **By**. ;
14. `alert_is_present()` - ожидает появления модального окна **alert**, возвращает **true** если окно появилось, в противном случае возвращает **false**.

8.3 Ожидание заголовка

`.title_is(title)` И `.title_contains(title)`

Синтаксис: `WebDriverWait(browser, 5).until(EC.title_is('title changed'))`

`.title_is(title)`

`title_is(title)` - ожидание проверки заголовка страницы. `title` - ожидаемый заголовок, который должен быть точным совпадением, возвращает `True`, если заголовок совпадает, в противном случае - `false`.

Откройте [сайт](#), нажмите кнопку, обратите внимание на заголовок. Запустите код ниже у себя в терминале, чтобы посмотреть как он работает. Это код, ожидает 10 секунд пока заголовок не станет `"title changed"`, когда заголовок полностью совпадёт, код вернёт `true`.

```
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/expectations/2/index.html')
    element = WebDriverWait(browser, 10).until(EC.title_is('title
changed'))
    print(element)
```

`.title_contains(title)`

Синтаксис: `WebDriverWait(browser, 5).until(EC.title_contains('tle'))`

То же самое что и предыдущий код, только вернёт `true` если `title` совпадает частично. Запустите код и поймите разницу, измените часть заголовка чтобы лучше понять разницу.

```
from selenium import webdriver
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
with webdriver.Chrome() as browser:
    browser.get('http://parsinger.ru/expectations/2/index.html')
    element = WebDriverWait(browser, 10).until(EC.title_contains('tle'))
    print(element)
```

8.4 Задачи

8.4.1

1. Откройте [сайт](#) при помощи Selenium;
2. На сайте есть кнопка, которая становится активной после загрузки страницы с рандомной задержкой, от 1 до 3 сек;
3. После нажатия на кнопку, в **title** начнут появляться коды, с рандомным временем, от 0.1 до 0.6 сек;
4. Ваша задача успеть скопировать код из `id="result"`, когда **title** будет равен `"345FDG3245SFD"`;
5. Вставить появившийся код в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
url = 'http://parsinger.ru/expectations/3/index.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    WebDriverWait(browser, 10).until(EC.element_to_be_clickable((By.ID, "btn"))).click()
    if WebDriverWait(browser, 20).until(EC.title_is('345FDG3245SFD')):
        print(browser.find_element(By.ID, 'result').text)
```

8.4.2

1. Откройте [сайт](#) при помощи Selenium;
2. На сайте есть кнопка, которая становится активной после загрузки страницы с рандомной задержкой, от 1 до 3 сек;
3. После нажатия на кнопку, в **title** начнут появляться коды, с рандомным временем, от 0,1 до 0.6 сек;
4. В этот раз второй раз на кнопку кликать не нужно, а нужно получить **title** целиком, если часть **title** = "JK8HQ"
5. Используйте метод `title_contains(title)` с прошлого урока;
6. Вставьте полный текст заголовка который совпадает с частью заголовка из условия.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
url = 'http://parsinger.ru/expectations/4/index.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    WebDriverWait(browser, 10).until(EC.element_to_be_clickable((By.ID, "btn"))).click()
    if WebDriverWait(browser, 30).until(EC.title_contains('JK8HQ')):
        print(browser.execute_script("return document.title;"))
```

8.4.3

В этой задаче используйте метод `presence_of_element_located(locator)` который проверяет наличие элемента в DOM.

1. Откройте [сайт](#) при помощи Selenium;
2. На сайте есть кнопка, поведение которой вам знакомо;
3. После нажатие на кнопку, на странице начнётся создание элементов **class** с рандомными значениями;
4. Ваша задача применить метод, чтобы он вернул содержимое элемента с классом "ВМН21УУ", когда он появится на странице;
5. Полученное значение вставить в поле для ответа.

ps. Вы конечно можете и вручную найти элемент с этим классом, но кого вы обманите?

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
url = 'https://parsinger.ru/expectations/6/index.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    WebDriverWait(browser, 10).until(EC.element_to_be_clickable((By.ID, "btn"))).click()
    elem = WebDriverWait(browser, 30).until(EC.presence_of_element_located((By.CLASS_NAME,
'Y1DM2GR'))))
    print(elem.text)
```

8.4.4

А это задача для тех кто всё таки решил перехитрить прошлую задачу и решил её вручную. А для всех остальных, задача максимально проста, запустить тот же самый код из прошлой задачи, но на другом url и с другим class.

1. Откройте [сайт](#) при помощи Selenium;
2. На сайте есть кнопка, поведение которой вам знакомо;
3. После нажатие на кнопку, на странице начнётся создание элементов **class** с рандомными значениями;
4. Ваша задача применить метод, чтобы он вернул содержимое элемента с классом "Y1DM2GR", когда он появится на странице;
5. Полученное значение вставить в поле для ответа.

РЕШЕНИЕ

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
from selenium.webdriver.support.ui import WebDriverWait
url = 'http://parsinger.ru/expectations/6/index.html'
with webdriver.Chrome() as browser:
    browser.get(url)
    WebDriverWait(browser, 10).until(EC.element_to_be_clickable((By.ID, "btn"))).click()
    elem = WebDriverWait(browser, 30).until(EC.presence_of_element_located((By.CLASS_NAME,
'Y1DM2GR'))))
```

```
print(elem.text)
```