Шпаргалка: Selenium

Selenium WebDriver — это драйвер браузера: через него можно передавать браузеру команды и имитировать действия пользователя.

Selenium может имитировать большинство действий пользователя в браузере:

- клик по элементу: кнопке, радиобаттону, чекбоксу, выпадающему списку;
- заполнение текстовых полей;
- навигация между страницами: переходы вперёд, назад или по URL;
- навигация внутри страницы: обновление, скроллинг.

Поиск элемента и назначение его локатором

Локатор — это объект, который помогает отыскать любой элемент на странице. Это запрос с определённым синтаксисом. Он обращается к элементам страницы по их характеристикам. Например, локатор может быть таким: `.//button[text()='Выйти']`.

В Selenium есть специальный класс `By`. Внутри хранятся методы, которые позволяют искать элементы по разным критериям.

Локатор помогает найти один или несколько элементов сразу. Под каждый элемент важно подбирать подходящий локатор.

Поиск по HTML

XPath — это язык, на котором можно написать запрос. Запрос описывает путь до элемента.

Поиск по элементам

Команда поиска по XPath — `$x("xpath")`. После неё нужно ввести путь к элементу. Например: `$x("html/body/div/div/header/img")`.

Поиск по элементу и атрибуту

Атрибуты сообщают дополнительные сведения об элементе.

Можно искать по любому атрибуту. Но лучше выбрать уникальный: так поиск будет точнее.

Чтобы обратиться к атрибуту, нужно:

- поместить атрибут в квадратные скобки `[]`;
- написать символ `@`.

Для логотипа запрос будет выглядеть так: `$x("html/body/div/div/header/img[@alt]")`.

Можно добавить в запрос значение атрибута, чтобы сделать поиск точнее. Значение атрибута `alt` — `Логотип проекта Mesto`. Тогда нужно:

- поставить знак равно;
- значение атрибута поместить в одинарные кавычки `''`.

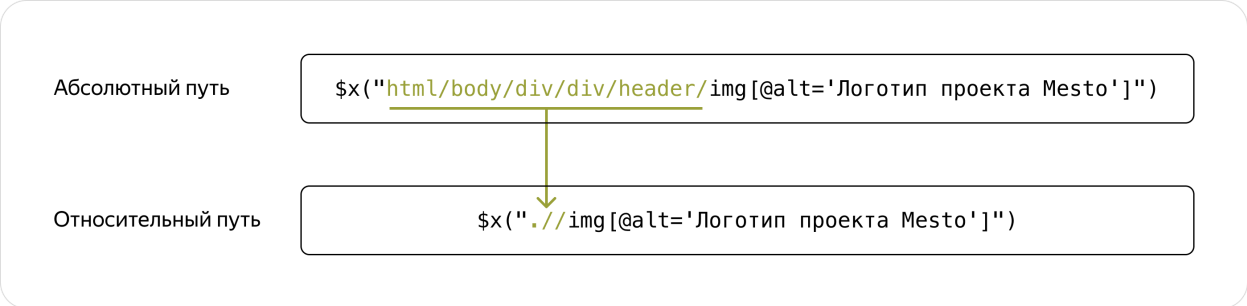
Получится `$x("html/body/div/div/header/img[@alt='Логотип проекта Mesto']")`.

Относительный и абсолютный поиск

Можно заменить весь путь до корневого элемента символами `.//`. Запрос получится таким: `$x(".//img[@alt='Логотип проекта Mesto']")`.

Путь с указанием всех элементов `$x("html/body/div/div/header/img[@alt='Логотип проекта Mesto']")` называется **абсолютным**.

Сокращённая форма `$x(".//img[@alt='Логотип проекта Mesto']")` — **относительный путь**.



Поиск по элементу и тексту

Чтобы искать по элементу и тексту, пользуются:

- путём до элемента;
- функцией `text()` — её помещают в квадратные скобки `[]`.

Запрос будет выглядеть так: `$x(".//button[text()='Выйти']")`.

Поиск по CSS

Поиск по CSS отличается от поиска по XPath тем, что он не ищет текст внутри элемента.

Чтобы искать по CSS, нужно набрать сочетание клавиш Cntrl+F для Windows и Cmd+F для Mac во вкладке Elements в DevTools. Появится строка поиска — здесь предстоит искать по CSS-локатору или -селектору.

Например, можно ввести CSS-селектор `[class='logo header__logo']`. Селектор передаёт информацию, что ищет элемент с атрибутом `class` и значением `logo header__logo'`.

Чтобы сократить длину CSS-селектора, можно заменить атрибут на `.`. Тогда селектор станет выглядеть так `.logo.header__logo`.

Можно искать элемент по одному значению класса: тогда это будет выглядеть так — `.logo` или `.header__logo`.

Поиск элементов через атрибуты

Чтобы найти изображение, можно применить поиск по атрибуту `alt` — `[alt]`; или поиск по атрибуту и его значению `[alt='Логотип проекта Mesto']`.

Если элемент нельзя определить ни по классу, ни по атрибуту, нужно объединить их в один селектор. Например: `.logo[alt='Логотип проекта Mesto']` или `.logo.header__logo[alt='Логотип проекта Mesto']`.

Поиск без атрибутов

CSS-селекторы помогают двигаться по элементам и без атрибутов. Знак `>` позволяет опуститься по иерархии элементов DOM вглубь. Например: `html>body>div>div>header>img`.

Короткая запись

У CSS-селекторов, как и в XPath, есть короткая запись — `html img`. Пробел показывает, что нужно искать элемент `img` во всех дочерних элементах `html`. Уточнить поиск можно через классы и/или атрибуты: `html img.logo` или `html img.logo[alt]`.

Открыть и закрыть страницу

Любой автотест на Selenuim начинается с двух шагов — создания драйвера и импорта необходимых пакетов.

Импорт пакетов

Чтобы пользоваться командами Selenium, нужно импортировать пакет Selenium WebDriver:

```
from selenium import webdriver
```

Создание драйвера

Драйвер делают под тот браузер, в котором будут сами автотесты. Например, так выглядит код, который помогает создать драйвер под Google Chrome, Firefox, Internet Explorer.

```
driver = webdriver.Chrome() # Google Chrome
driver = webdriver.Firefox() # Firefox
driver = webdriver.Ie() # Internet Explorer
```

Специальные настройки

Вместе с драйвером можно добавить специальные настройки. Они не обязательны, но делают работу удобнее.

```
chrome_options = webdriver.ChromeOptions() # создали объект для опций
chrome_options.add_argument('--headless') # добавили настройку
chrome_options.add_argument('--window-size=640,480') # добавили ещё настройку
driver = webdriver.Chrome(options=chrome_options) # создали драйвер и передали в него настройки
```

Для примера есть такие опции:

- `--headless` запускает браузер без визуального окна. Это полезно, если запускаешь браузер в системах без графической составляющей.
- `—-window-size` запускает браузер с заданным разрешением экрана. Полезно, когда нужно протестировать интерфейс в системе с определённым разрешением.

Открыть страницу

Чтобы открыть страницу в браузере, нужен метод `get()`. Страницу, которую нужно открыть, передают в аргументе:

```
# импортировали пакет Selenium WebDriver
from selenium import webdriver

# создали драйвер
driver = webdriver.Chrome()

# открыли страницу
driver.get('https://ya.ru/')
```

Закрыть страницу

После тестирования браузер нужно обязательно закрыть. Если этого не сделать, фоновые процессы и порты могут завершить работу некорректно. Это может привести к утечке данных или к ошибке доступа.

Закрыть браузер можно вот так:

```
driver.quit()
```

Получить адрес страницы

Иногда нужно проверить URL страницы, на которой ты сейчас находишься. Первый шаг — получить адрес страницы. Для этого используют такой метод:

```
current_url = driver.current_url
```

Адрес страницы сохранился в `current_url`.

Чтобы проверить, что это именно тот адрес, который нужен, понадобится `assert`. Например, вот так ты проверишь, что сейчас находишься на странице `ya.ru`:

```
assert current_url == 'https://ya.ru/'
```

Если страницу нужно проверить только один раз в программе, можно не сохранять её адрес в переменную:

```
assert driver.current_url == 'https://ya.ru/'
```

Адрес можно проверить не целиком, а частично. Например, что в url есть `ya.ru`. Понадобится оператор `in`:

```
assert 'ya.ru' in driver.current_url
```

Найти элемент

Искать элементы в Selenium можно двумя способами — через метод `findElement()` и `findElements()`:

- `findElement()` нужен, чтобы искать один элемент. Если в результате найдётся несколько элементов, которые попадают под критерий, метод вернёт только первый. Метод `findElement()` возвращает объект `WebElement`.
- `findElements()` нужен, чтобы искать и возвращать несколько элементов. Элементы возвращаются в виде списка `List<WebElement>`.

Для обоих этих методов понадобятся аргументы: в скобках тебе нужно указать ориентир, по которому Selenium будет искать элемент. Например, id кнопки или другие атрибуты, XPath или HTML-тег. Поможет класс `By`.

Класс By

Он помогает задавать критерии поиска элементов:

```
By.CLASS_NAME # поиск по наименованию атрибута class
By.CSS_SELECTOR # поиск по CSS-селектору
By.ID # поиск по атрибуту id
By.LINK_TEXT # поиск по тексту ссылки (имеется в виду не сама ссылка вида https://..., а текст вну
три объекта ссылки, в следующем уроке будет пример
By.PARTIAL_LINK_TEXT # поиск по части текста ссылки, то же условие, что и для By.linkText(text)
By.NAME # поиск по атрибуту name
By.TAG_NAME # поиск по HTML-тегу
By.XPATH # поиск по XPath
```

Чтобы применять методы класса `By`, сначала нужно его импортировать:

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

В итоге целиком получится так:

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

driver = webdriver.Chrome()
driver.get("http://www.example.com")

# Для поиска одного элемента
driver.find_element(By.XPATH, ".//img")

# Для поиска группы элементов
driver.find_elements(By.XPATH, ".//button")
```

Кликнуть по элементу

В Selenium за действие «клик» отвечает метод `click()`. Чтобы кликнуть по элементу, сперва его нужно найти.

В коде поиск и клик по элементу выглядят так:

```
driver.find_element(By.LINK_TEXT, "Войти").click()
```

Это действие можно разбить на два этапа: сперва найти элемент и затем кликнуть по нему:

```
element = driver.find_element(By.LINK_TEXT, "Войти")
element.click()
```

Задать ожидание

В Selenium есть **ожидания**. Это код, который останавливает исполнение теста на указанный период. Ожидания делят на явные и неявные.

Явные ожидания останавливают тест на точный период времени. Их задают через класс `WebDriverWait`. Его нужно импортировать:

```
from selenium.webdriver.support.wait import WebDriverWait
```

Потом — просто написать имя класса, а в скобках указать драйвер и время, на которое нужно остановить:

```
from selenium.webdriver.support.wait import WebDriverWait
# Остановить тест на три секунды
WebDriverWait(driver, 3)
```

Чтобы было удобнее работать с ожиданием, в коде можно задать специальное условие. Это можно сделать через класс `expected_conditions`. Нужно указать его, а потом через точку — само условие. Частые условия:

- `element_to_be_clickable` — элемент кликабелен.
- `presence_of_element_located` — ожидает наличие элемента на странице.
- `visibility_of_element_located` — проверяет, что элемент есть на странице и его видно.

Например, «пока элемент будет кликабелен» — `expected_conditions.element_to_be_clickable`.

Ожидание с условием задают через метод `until()`:

```
# Ожидание, что кнопка станет кликабельной, не больше трёх секунд
WebDriverWait(driver, 3).until(expected_conditions.element_to_be_clickable((By.TAG_NAME, "button")))
```

Явное ожидание лучше применять, если ты ищешь определённые элементы на странице и нужно дождаться их загрузки.

У Selenium есть и **неявные ожидания**.

Явное ожидание ты указываешь каждый раз, когда нужно остановить тест. Неявные нужно указать только один раз — с помощью метода `implicitly_wait()`. Теперь все команды, которые идут за ним, будут сопровождаться паузой:

```
# Неявное ожидание в три секунды
driver.implicitly_wait(3)
driver.find_element(By.TAG_NAME, "button")
```

Selenium будет искать элемент не дольше трёх секунд — `implicitly_wait(3)`. Ожидание прописывается один раз, а дальше применяется автоматически. Если написать дальше ещё один метод, он тоже выполнится с ожиданием.

Если элемент так и не найдётся через три секунды, программа выдаст ошибку — `NoSuchElementException: Message: no such element: Unable to locate element`.

Лучше применять явное ожидание — им легче управлять. Неявное можно написать, когда ждёшь именно того, что элемент появится в DOM.

> ❗ Не стоит смешивать явные и неявные ожидания в коде. Это может привести к тому, что условия будут конфликтовать друг с другом.

Заполнить поле ввода

Чтобы заполнить поле, применяют метод `sendKeys()`. В скобках пишут текст, который нужно ввести, — `send_keys("Купить хомячка")`.

```
driver.find_element(By.TAG_NAME, "input").send_keys("Яндекс")
```

Прежде чем заполнить поле, иногда его нужно очистить. Чтобы это сделать, применяют метод `clear()`. Нужно найти элемент, а потом написать `.clear()`.

```
driver.find_element(By.TAG_NAME, "input").send_keys("Яндекс")
driver.find_element(By.TAG_NAME, "input").clear()
driver.find_element(By.TAG_NAME, "input").send_keys("Практикум")
```

Получить текст элемента

Чтобы получить текст элемента, применяют метод `text()`.

```
driver.find_element(By.CLASS_NAME, "mg-story-not-found__title").text
```

Перейти к элементу

Когда Selenium выполняет действия с элементами, он автоматически настраивает прокрутку страницы так, чтобы элемент попал в видимое окно браузера.

Если автоматической настройки недостаточно, помогает метод, который прокручивает элемент в зону видимости — прокрутка, или скролл. Пригодится специальный метод `execute_script()`.

```
element = driver.find_element(By.ID, "root")
driver.execute_script("arguments[0].scrollIntoView();", element)
```

Что здесь происходит:

- Ищешь элемент, до которого нужно прокрутить, — `find_element(By.ID, "root")`. Сохраняешь его в переменную `element`.
- Применяешь `execute_script()` к объекту `driver`.
- В скобках пишешь **скрипт** `"arguments[0].scrollIntoView();”` — он производит прокрутку.
- После скрипта в скобках указываешь элемент, до которого будешь скроллить, — `element`.

Работа с куками из тестов

Иногда в автотесте нужно обратиться к определённой куке: посмотреть её значение или поменять. Так ты сможешь протестировать функциональность с разными данными или включить/выключить её.

Со страницы можно **получить все куки** с помощью метода `getCookies()`:

```
driver.get_cookies()
```

Чтобы **получить конкретную куку** по имени, понадобится метод `getCookieNamed()`:

```
cookie_name = "_yasc"
cookie = driver.get_cookie(cookie_name)
```

Изменить поля объекта куки нельзя. Возможно только удалить куку и добавить новую — с тем же именем, но уже другим наполнением.

Удалить куки поможет метод `delete_cookie()`:

```
driver.delete_cookie("new_cookie")
```

Можно удалить все куки сразу:

```
driver.delete_all_cookies()
```

Добавить новую куку. Нужно создать объект, а потом вызвать метод `add_cookie()`. Как аргументы передай имя и значение:

```
new_cookie = {"name": "new_cookie", "value": "new_value"}
driver.add_cookie(new_cookie)
```