pip install selenium==4.\*

pip list

from selenium import webdriver

from selenium.webdriver.common.by import By

import time

import math

textarea - <https://htmlbook.ru/html/textarea>

* Id
  + **#bullet**
  + **[id="bullet"]**вместо **#bullet**
* tag
* значение атрибута
  + Например, давайте найдем элемент h1 по значению его атрибута value: **[value="Cat memes"]**
* Name
  + Этот вариант поиска является разновидностью поиска по значению атрибута и записывается так же: **[name="bullet-cat"]**
* Class
  + **.jumbotron-heading**
  + Поиск по классу можно записать в виде **[class="jumbotron-heading"]**

!Внимание. Символ пробела " " является значащим в CSS-селекторах. Это важный символ, который разделяет описание предка и потомка. Если бы мы записали селектор #post2.title без пробела, то в данном примере не было найдено ни одного элемента. Такая запись означала бы, что мы хотим найти элемент, который одновременно содержит id "post2" и класс "title".

Списки

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

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

<option selected>--</option>

<option value="1">Python</option>

<option value="2">Java</option>

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

</select>

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

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

execute\_scripte()

С помощью метода execute\_script можно выполнить программу, написанную на языке JavaScript, как часть сценария автотеста в запущенном браузере.

rom selenium import webdriver

browser = webdriver.Chrome()

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

button = browser.find\_element(By.TAG\_NAME, "button") browser.execute\_script("return arguments[0].scrollIntoView(true);", button) button.click()

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

# Element.scrollIntoView() - <https://developer.mozilla.org/ru/docs/Web/API/Element/scrollIntoView>

Метод **Element.scrollIntoView()** интерфейса [Element](https://developer.mozilla.org/ru/docs/Web/API/Element) прокручивает контейнер родителя элемента так, чтобы элемент, на котором был вызван scrollIntoView(), стал виден пользователю.

element.scrollIntoView();

element.scrollIntoView(alignToTop); // аргумент типа Boolean

element.scrollIntoView(scrollIntoViewOptions); // аргумент типа Object

**Загрузка файлов**

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

import os

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

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

element.send\_keys(file\_path)

**Модальные окна**

**Alert**

Для этого нужно сначала переключиться на окно с alert, а затем принять его с помощью команды **accept()**:

alert = browser.switch\_to.alert

alert.accept()

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

alert = browser.switch\_to.alert

alert\_text = alert.text

Confirm

confirm = browser.switch\_to.alert

confirm.accept()

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

confirm.dismiss()

Prompt

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

prompt = browser.switch\_to.alert

prompt.send\_keys("My answer")

prompt.accept()

Переход на новую вкладку в браузере

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

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

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

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

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

current\_window = browser.current\_window\_handle

Организация задержки

**implicitly wait**

browser = webdriver.Chrome() # говорим WebDriver искать каждый элемент в течение 5 секунд

browser.implicitly\_wait(5)

## **Про Exceptions**

* Если элемент не был найден за отведенное время, то мы получим **NoSuchElementException.**
* Если элемент был найден в момент поиска, но при последующем обращении к элементу DOM изменился, то получим **StaleElementReferenceException**. Например, мы нашли элемент **Кнопка** и через какое-то время решили выполнить с ним уже известный нам метод click. Если кнопка за это время была скрыта скриптом, то метод применять уже бесполезно — элемент "устарел" (stale) и мы увидим исключение.
* Если элемент был найден в момент поиска, но сам элемент невидим (например, имеет нулевые размеры), и реальный пользователь не смог бы с ним взаимодействовать, то получим **ElementNotVisibleException**.

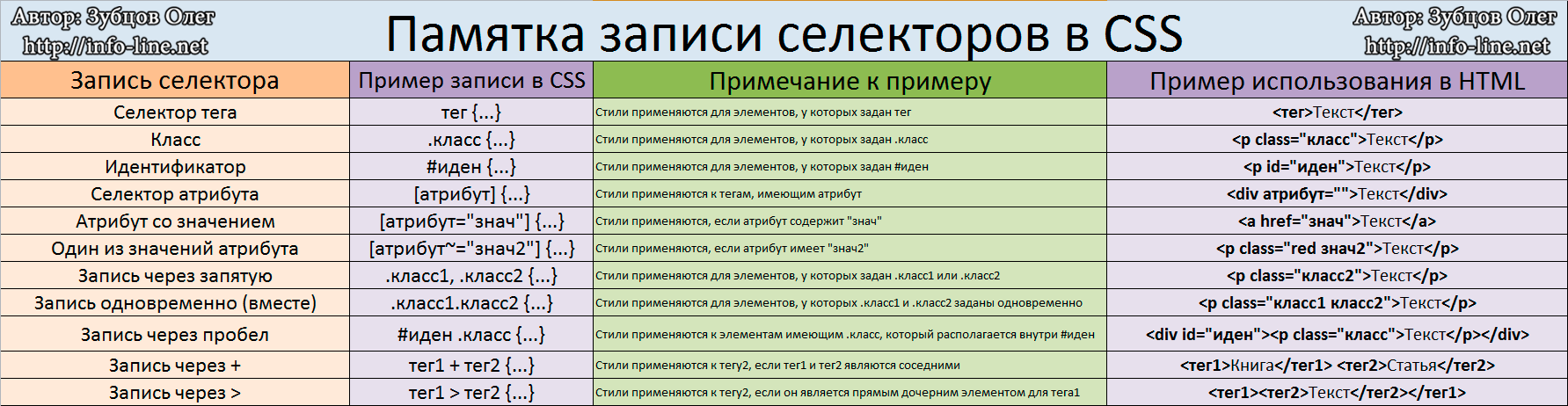
## **Explicit Waits (WebDriverWait и expected\_conditions)**

# говорим Selenium проверять в течение 5 секунд, пока кнопка не станет кликабельной

button = WebDriverWait(browser, 5).until( EC.element\_to\_be\_clickable((By.ID, "verify"))

<https://selenium-python.readthedocs.io/api.html#module-selenium.webdriver.support.expected_conditions>

* title\_is
* title\_contains
* presence\_of\_element\_located
* visibility\_of\_element\_located
* visibility\_of
* presence\_of\_all\_elements\_located
* text\_to\_be\_present\_in\_element
* text\_to\_be\_present\_in\_element\_value
* frame\_to\_be\_available\_and\_switch\_to\_it
* invisibility\_of\_element\_located
* element\_to\_be\_clickable
* staleness\_of
* element\_to\_be\_selected
* element\_located\_to\_be\_selected
* element\_selection\_state\_to\_be
* element\_located\_selection\_state\_to\_be
* alert\_is\_present
* **find\_element(By.ID, value)** — поиск по уникальному атрибуту id элемента. Если ваши разработчики проставляют всем элементам в приложении уникальный id, то вам повезло, и вы чаще всего будет использовать этот метод, так как он наиболее стабильный;
* **find\_element(By.CSS\_SELECTOR, value)** — поиск элемента с помощью правил на основе CSS. Это универсальный метод поиска, так как большинство веб-приложений использует CSS для вёрстки и задания оформления страницам. Если find\_element\_by\_id вам не подходит из-за отсутствия id у элементов, то скорее всего вы будете использовать именно этот метод в ваших тестах;
* **find\_element(By.XPATH, value)** — поиск с помощью языка запросов XPath, позволяет выполнять очень гибкий поиск элементов;
* **find\_element(By.NAME, value)** — поиск по атрибуту name элемента;
* **find\_element(By.TAG\_NAME, value)** — поиск элемента по названию тега элемента;
* **find\_element(By.CLASS\_NAME, value)** — поиск по значению атрибута class;
* **find\_element(By.LINK\_TEXT, value)**— поиск ссылки на странице по полному совпадению;
* **find\_element(By.PARTIAL\_LINK\_TEXT, value)**— поиск ссылки на странице, если текст селектора совпадает с любой частью текста ссылки.



alert\_text.split(': ')[-1] – копирование текста

копирование тескта в буфер обмена:

alert = browser.switch\_to.alert

alert\_text = alert.text

addToClipBoard = alert\_text.split(': ')[-1]

pyperclip.copy(addToClipBoard)

**Git**

<https://learngitbranching.js.org/> — отличный интерактивный туториал

<https://git-scm.com/book/ru/v2/> — лучшая книга вообще

<https://hyperskill.org/learn/topic/257/>﻿

<https://stepik.org/course/4138/>﻿

<https://stepik.org/course/3145/>

<http://www-cs-students.stanford.edu/~blynn/gitmagic/intl/ru/index.html>

<https://habr.com/company/intel/blog/344962/>

<https://githowto.com/ru>



**Проверка результатов**

Как можно проверить ожидаемый результат? Для этого используется встроенная в Python инструкция **assert**, которая проверяет истинность утверждений. **assert True** не приводит к выводу дополнительных сообщений, а вот **assert False** вызовет исключение **AssertionError**

>>> assert abs(-42) == -42, "Should be absolute value of a number"

Traceback (most recent call last):

  File "<stdin>", line 1, in <module>

AssertionError: Should be absolute value of a number

Форматирование строк

<https://realpython.com/python-string-formatting/#2-new-style-string-formatting-strformat>

# Маркеровка

pytest.mark.**xfail**(*condition=False*, *\**, *reason=None*, *raises=None*, *run=True*, *strict=xfail\_strict*)

**Parameters:**

* **condition** (*Union[*[*bool*](https://docs.python.org/3/library/functions.html#bool)*,*[*str*](https://docs.python.org/3/library/stdtypes.html#str)*]*) – Условие для маркировки тестовой функции как xfail (True/False или строка условия). Если bool, вам также необходимо указать reason (видеть [стоку](https://docs.pytest.org/en/latest/historical-notes.html#string-conditions) условия).
* **reason** ([*str*](https://docs.python.org/3/library/stdtypes.html#str)) – Reason почему тестовая функция помечена как xfail.
* **raises** (*Type[*[*Exception*](https://docs.python.org/3/library/exceptions.html#Exception)*]*) – Класс исключения (или кортеж классов), который, как ожидается, будет вызван тестовой функцией; другие исключения не пройдут тест. Обратите внимание, что подклассы переданных классов также приведут к совпадению. (аналогично тому, как работает оператор исключения except).

**run** ([*bool*](https://docs.python.org/3/library/functions.html#bool)) – Должна ли на самом деле выполняться тестовая функция. Если False, функция всегда будет xfail и не будет выполнена (useful if a function is segfaulting).

* **strict** (*[bool](https://docs.python.org/3/library/functions.html" \l "bool" \o "(in Python v3.12))*) –
  + If False the function will be shown in the terminal output as xfailed if it fails and as xpass if it passes. In both cases this will not cause the test suite to fail as a whole. This is particularly useful to mark *flaky* tests (tests that fail at random) to be tackled later.
  + If True, the function will be shown in the terminal output as xfailed if it fails, but if it unexpectedly passes then it will **fail** the test suite. This is particularly useful to mark functions that are always failing and there should be a clear indication if they unexpectedly start to pass (for example a new release of a library fixes a known bug).

Defaults to **[xfail\_strict](https://docs.pytest.org/en/latest/reference/reference.html" \l "confval-xfail_strict)**, which is False by default.