

# Тестирование Playwright

- [Начало](#)
- [Локаторы](#)
- [Actions](#)
- [События \(Events\)](#)
- [Аутентификация](#)
- [Pytest & Playwright](#)
- [Дополнительные возможности](#)
- [Ожидание](#)

# Начало

[Официальный сайт проекта](#)

## Установка

```
python -m pip install playwright
```

### Проверка установки

```
playwright --version
```

### Установка драйверов для браузеров

```
playwright install #Все браузеры  
playwright install name #Только name браузеры  
playwright install chromium #Chrome
```

### Установка pytest

```
python -m pip install pytest
```

### Установка плагина pytest-playwright

```
python -m pip install pytest-playwright
```

## Описание

Два режима работы: синхронный и асинхронный. Для синхронного:

```
from playwright.sync_api import sync_playwright
```

Для асинхронного режима:

```
import asyncio  
from playwright.async_api import async_playwright  
  
async def main():  
    async with async_playwright() as playwright:
```

```
browser = await playwright.chromium.launch()
page = await browser.new_page()
await page.goto("https://playwright.dev")
print(await page.title())
await browser.close()
```

```
asyncio.run(main())
```

Чаще используется синхронный режим.

Запуск и закрытие браузера:

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.goto("https://playwright.dev/python")

    docs_button = page.get_by_role('link', name="Docs")
    docs_button.click()

    browser.close()
```

headless=False обозначает визуальное открытие, slow\_mo задержка

### **Использование интерактивной консоли**

Иногда для удобства можно использовать консоль python для ручного тестирования покомандного ввода.

```
python
Python 3.13.1 (tags/v3.13.1:0671451, Dec 3 2024, 19:06:28) [MSC v.1942 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> from playwright.sync_api import sync_playwright
>>> playwright = sync_playwright().start()
>>> browser = playwright.chromium.launch(headless=False, slow_mo=100)
>>> page = browser.new_page()
>>> browser.close()
>>> playwright.stop()
```

Если создать `docs_button`, у нее будет метод `highlight()` для визуальной подсветки найденного элемента.

# Локаторы

Локаторы: способ поиска элементов на странице. Поэтому они являются методами page

В VSC Ctrl+Click по методу выводит код метода.

Локатор	Описание
page.get_by_role('link', name="Docs")	Поиск элемента по роли name - текст link <a> heading <h> radio, checkbox, button
page.get_by_label("Email address")	Для выделения элементов, у которых есть привязанная метка. Например <div><pre>&lt;div&gt;   &lt;label for="exampleInputEmail1" class="form-label mt-4"&gt;Email address&lt;/label&gt;   &lt;input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="Enter email"&gt;   &lt;small id="emailHelp" class="form-text text-muted"&gt;We'll never share your email with anyone else.&lt;/small&gt; &lt;/div&gt;</pre></div>
page.get_by_placeholder("Enter email")	Поиск элементов по placeholder
page.get_by_text("Something", exact=False)	Поиск по тексту. Exact=False ищет вхождение.
page.get_by_alt_text(text)	Поиск по атрибуту alt у изображений
page.get_by_title(text)	Атрибут title

Локатор	Описание
page.locator(text)	<p>Поиск по CSS. Можно использовать tagname, classname, id, attribute/value</p> <p>Примеры^</p> <p>css=h1</p> <p>footer</p> <p>&lt;tagName&gt;.&lt;classname&gt; button.btn-outline-sucess</p> <p>&lt;tagName&gt;#&lt;idname&gt; button#BtnGroupDrop1</p> <p>&lt;tagName&gt;[attribute] input[readonly]</p> <p>&lt;tagName&gt;[attribute=somevalue] input[value='correct value']</p>
	<p>Поиск по иерархии элементов.</p> <p>Если через пробелы-то вложенные элементы. Если через точку - то у элемента несколько классов. Но они не обязательно непосредственно вложенные.</p> <p>nav.bg-dark a.nav-link.active</p> <p>Для непосредственного вложения:</p> <p>nav.bg-dark &gt; a.nav-link.active</p>
	<p>Называются sudo классами,</p> <p>Класс и текст в теге. Для вхождения:</p> <p>h1:text('Navbars')</p> <p>Для полного соответствия:</p> <p>h1:text-is('Navbars')</p> <p>div.dropdown-menu:visible</p> <p>Для определения по номеру вхождения, когда их много</p> <p>:nth-match(button.btn-primary, 4)</p>
	<p>XPath</p> <p>Абсолютный путь: xpath=/html/head/title</p> <p>С любого начала: xpath=//h1/h2</p> <p>С указанием атрибута xpath=//h1[ @id='navbars' ]</p>
	<p>Функции XPath</p> <p>Для поиска по тексту, точно: //h1[text()='Headling1']</p> <p>Для поиска по тексту, содержит: //h1[contains(text(), 'Headling1')]</p> <p>Для поиска по тексту, содержит: //h1[contains(@class, 'btn')]</p>
Множественные условия	<p>Поиск родительского элемента</p> <p>page.get_by_label("Email address").locator("..")</p> <p>Фильтрация</p> <p>page.get_by_role("heading").filter(has_text="First")</p> <p>По дочернему элементу</p> <p>page.locator("div.form-group").filter(has=page.get_by_label("Password"))</p>

## Доступ к iframe

```
'''Test for auth module'''
```

```
import pytest
```

```
from playwright.sync_api import Browser, Page, expect

AUTH_URL = "https://wood.bobrobotirk.ru/auth"

@pytest.fixture
def page_and_auth(browser: Browser):
    context = browser.new_context(
        storage_state="playwright/.auth/vk.json"
    )
    page = context.new_page()
    yield page
    context.close()

def test_first(page_and_auth: Page):
    page_and_auth.goto(AUTH_URL)
    vkframe = page_and_auth.frame(url=lambda url: "id.vk.com" in url)

    if vkframe:
        authbutton = vkframe.get_by_role("button", name="Продолжить как")
        expect(authbutton, "Кнопка Продолжить как... отсутствует").to_be_visible(timeout=20000)
        authbutton.click()

        back_button = page_and_auth.get_by_role("button", name="Авторизация успешна")
        expect(back_button, "Сервис не произвел авторизацию").to_be_visible()
        back_button.click()
    else:
        assert False, "VK фрейм не найден."
```

# Actions

Действие	Описание
click()	Однократное нажатие. Опции: button="left" modifiers=["Shift", "Alt"] с зажатой кнопкой Shift timeout=2_000 Задержка перед ошибкой. Обычно 30 сек. force=True Ошибка сразу же если не найден.
dblclick()	Двойной щелчок. Опции как у click +: delay=100 - задержка в миллисекундах
hover()	Навести мышь на выбранный элемент
fill("my text")	Заполнить поле ввода текстом my text. Аналогично Ctrl-V
clear()	Очистить поле ввода
type("my text", delay=100)	Имитация побуквенного ввода
check() Еще: set_checked(True)	Выбор radiobutton, checkbox, switch is_checked() для checkbox проверяет, выбран ли checkbox
uncheck()	убрать выбор
select_option("text")	Выбор опции из раскрывающегося списка. Но если отсутствует - будет Timeout Error. Если передать список - будет множественный выбор.
	Для раскрытия Dropdown элемента: нажатие на него, выбор элемента и нажатие
set_input_files("")	Для элемента позволяющего загружать файлы, имя файла из директории, из которой запускается скрипт. Можно передать список.
	Если по кнопке открывается меню выбора файла, то <div><pre>with page.expect_file_chooser() as fc_info:     file_input.click() #до этого через локатор найден file_input     file_chooser = fc_info.value     file_chooser.set_files("first.txt")</pre></div>



Действие	Описание
	press("KeyW") press("Shift+KeyW") press("Control+ArrowLeft")

# События (Events)

## События в `page.goto`

В переменной `wait_until`.

- `load`: загрузка всего контента
- `domcontentloaded`: загрузка `dom`
- `commit`: при получении ответа от сервера
- `networkidle`: до завершения всех событий сети. Для динамического контента не меньше чем `load`.

Можно считать время загрузки.

```
from playwright.sync_api import sync_playwright
from time import perf_counter

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    print('Page loading...')
    start = perf_counter()
    page.goto("https://playwright.dev/python", wait_until='load')
    delta = perf_counter() - start
    print(f'Page loaded in {delta} s.')

    browser.close()
```

## События динамического контента (React, ...)

Находим динамический элемент, кликаем по нему и при помощи `wait_for()` ждем.

```
from playwright.sync_api import sync_playwright
from time import perf_counter

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.goto("https://www.scrapethissite.com/pages/ajax-javascript/", timeout=60_000)
```

```
mylink = page.get_by_role("link", name="2014")
mylink.click()
print('Loading movie...')
start = perf_counter()
loadedcont = page.locator("td.film-title").first
loadedcont.wait_for()
delta = perf_counter() - start
print(f'Movie loaded in {delta} s.')

browser.close()
```

## Ожидание события.

Указываем тип события и функцию, выполняемую при наступлении события.

```
from playwright.sync_api import sync_playwright

def onload(page):
    print("Page loaded", page)

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.on("load", onload)
    page.goto("https://bootswatch.com/default")
    browser.close()
```

## Пример просмотра событий запросов

```
from playwright.sync_api import sync_playwright

def onrequest(request):
    print("Request send: ", request)

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.on("request", onrequest)
    page.goto("https://bootswatch.com/default")
    browser.close()
```

## Вывод:

```
Request send: <Request url='https://bootswatch.com/default' method='GET'>
Request send: <Request url='http://bootswatch.com/default/' method='GET'>
Request send: <Request url='https://bootswatch.com/default/' method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/bootstrap/dist/css/bootstrap.css' method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/bootstrap-icons/font/bootstrap-icons.min.css'
method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/prismjs/themes/prism-okaidia.css'
method='GET'>
Request send: <Request url='https://bootswatch.com/_assets/css/custom.min.css' method='GET'>
Request send: <Request url='https://www.googletagmanager.com/gtag/js?id=G-KGDJB EFF3W' method='GET'>
Request send: <Request
url='https://cdn.carbonads.com/carbon.js?serve=CKYIE23N&placement=bootswatchcom' method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/bootstrap/dist/js/bootstrap.bundle.min.js'
method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/prismjs/prism.js' method='GET'>
Request send: <Request url='https://bootswatch.com/_assets/js/custom.js' method='GET'>
Request send: <Request url='https://bootswatch.com/_vendor/bootstrap-icons/font/fonts/bootstrap-
icons.woff2?1fa40e8900654d2863d011707b9fb6f2' method='GET'>
Request send: <Request url='https://www.google-analytics.com/g/collect?v=2&tid=G-
KGDJB EFF3W&gtm=45je54g0v9135688085za200&_p=1744912703288&gcd=13131313111&npa=0&dma=0&tag_
exp=102509682~102803279~102813109~102887800~102926062~103027016~103051953~103055465~10
3077950~103106314~103106316~103130495~103130497&cid=128797986.1744912704&ul=ru-
ru&sr=1280x720&uaa=x86&uab=64&uafvl=Not%253AA-
Brand%3B24.0.0.0%7CChromium%3B134.0.6998.35&uamb=0&uam=&uap=Windows&uapv=10.0.0&uaw=0&ar
e=1&frm=0&pscdl=noapi&_s=1&sid=1744912703&sct=1&seg=0&dl=https%3A%2F%2Fbootswatch.com%2Fde
fault%2F&dt=Bootswatch%3A%20Default&en=page_view&_fv=1&_nsi=1&_ss=1&_ee=1&tfd=2018'
method='POST'>
Request send: <Request
url='https://srv.carbonads.net/ads/CKYIE23N.json?segment=placement:bootswatchcom&v=true'
method='GET'>
```

Есть событие при выборе файла.

Удаление прослушивания события: `page.remove_listener("name_event", func)`

## События всплывающих окон (`alert`, `confirm`, `prompt`)

Событие `page.on("dialog", func)`, устанавливаем ожидание до возможного появления окна.

Функция:

```
def on_dialog(dialog)
    dialog.accept()
    dialog.dismiss()
```

Если диалог prompt, то для заполнения поля ввода в функции ассерт нужно добавить строковую переменную.

```
def on_dialog(dialog)
    dialog.accept("Text for enter")
```

## Событие скачивания файла

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.goto("https://bootswatch.com/default")
    btn = page.get_by_role("link", name="Download me")
    with page.expect_download() as download_info:
        btn.click()
    download = download_info.value
    download.save_as(fname)
    browser.close()
```

## Второй вариант: добавить listener

```
from playwright.sync_api import sync_playwright

def saving_func(download):
    fname = "first.jpg"
    download.save_as(fname)

with sync_playwright() as playwright:
    browser = playwright.chromium.launch(headless=False, slow_mo=500)
    page = browser.new_page()
    page.goto("https://bootswatch.com/default")
    page.once("download", saving_func)
    btn = page.get_by_role("link", name="Download me")
    with page.expect_download() as download_info:
        btn.click()
```

```
browser.close()
```

# Аутентификация

При 2FA аутентификации возникают проблемы при повторном исполнении скрипта. Для обхода этого используют контекст браузера.

## Шаг 1. Сохранение контекста.

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:
    browser = playwright.firefox.launch(headless=False, slow_mo=500)
    context = browser.new_context()
    page = context.new_page()
    page.goto("https://vk.ru")
    page.pause() # Откроется доп. окно. Проходим авторизацию, в доп.окне play
    context.storage_state(path="playwright/.auth/vk.json")
    context.close()
```

## Шаг 2. Использование контекста

```
from playwright.sync_api import sync_playwright

with sync_playwright() as playwright:
    browser = playwright.firefox.launch(headless=False, slow_mo=500)
    context = browser.new_context(storage_state="playwright/.auth/vk.json")
    page = context.new_page()
    page.goto("https://vk.ru")
    page.pause() #в реальных скриптах это убирается)
    context.close()
```

# Pytest & Playwright

## Pytest

Имена файлов тестов должны иметь префикс `test_` или постфикс `_test`. Имена тестов должны иметь префикс `test_`

В модуле `utils` функция `root`, отнимающая 1 от входного параметра. Пример теста:

```
import utils

def test_first():
    num24 = utils.root(25)
    assert num24 == 24
```

### Запуск теста:

```
pytest second_test.py
```

Ключ `-v` отображает расширенную информацию.

Ключ `-s` разрешает вывод данных из тестируемых функций.

Запуск без указания имени файла исполняет все тесты.

Желательно определение типов в функциях.

### Виды проверок

Проверка	Описание
<code>assert mynum == 32</code>	Проверка на значение
<code>assert type(dt) == dict</code>	Проверка типа
<code>assert "timestamp" in dt_list</code>	Проверка присутствия в списке

Для вывода доп. информации:

```
assert mynum == 32, "Число mynum должно быть равно 32"
```

### Фикстуры:



Для упаковки повторяющихся действий.

```
import json
import report
import pytest

@pytest.fixture
def report_json():
    report.generate_report()
    with open("report.json") as file:
        return json.load(file)

def test_report_json(report_json):
    assert type(report_json) == dict
```

Фикстура выполняется каждый раз при вызове. Для однократного исполнения фикстуры нужно добавить `scope="session"`

```
import json
import report
import pytest

@pytest.fixture(scope="session")
def report_json():
    report.generate_report()
    with open("report.json") as file:
        return json.load(file)

def test_report_json(report_json):
    assert type(report_json) == dict
```

Виды scope

session	Один раз в пределах сессии
function	Каждый раз
module	Если несколько тестовых файлов, то один раз в пределе запуска.

**Pytest-playwright**

Данный плагин упрощает работу с pytest. Встроенные фикстуры. Задачу создания и удаления playwright, browser берет на себя.

```
from playwright.sync_api import Page

def test_first(page: Page):
    page.goto("https://playwright.dev/python")
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    assert page.url == "https://playwright.dev/python/docs/intro"
```

## Настройка pytest

Через консоль:

- --headed Отображать окно
- --slowmo=500 Замедление работы
- --browser=firefox Настройка браузера

Через конфигурационный файл pytest.ini

```
[pytest]
addopts = --headed --slowmo=500 --browser=firefox
```

## Фикстуры

Добавить фикстуру

```
import pytest
from playwright.sync_api import Page

@pytest.fixture
def load_testpage(page: Page):
    page.goto("https://playwright.dev/python")
    return page

def test_first(load_testpage: Page):
    link = load_testpage.get_by_role("link", name="GET STARTED")
    link.click()
    assert load_testpage.url == "https://playwright.dev/python/docs/intro"
```

Автоматическое исполнение фикстуры перед каждой функцией:

```

import pytest
from playwright.sync_api import Page

@pytest.fixture(autouse=True)
def load_testpage(page: Page):
    page.goto("https://playwright.dev/python")
    return page

def test_first(page: Page):
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    assert page.url == "https://playwright.dev/python/docs/intro"

```

Пред и пост исполнение кода:

```

import pytest
from playwright.sync_api import Page

@pytest.fixture(autouse=True)
def load_testpage(page: Page):
    page.goto("https://playwright.dev/python")
    yield page
    page.goto("https://yandex.ru")

def test_first(page: Page):
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    assert page.url == "https://playwright.dev/python/docs/intro"

```

### Пример проверки авторизации ВК

```

"""Test for auth module"""
import pytest
from playwright.sync_api import Browser, Page, expect

AUTH_URL = "https://wood.bobrobotirk.ru/auth"

@pytest.fixture
def page_and_auth(browser: Browser):
    context = browser.new_context(

```

```

        storage_state="playwright/.auth/vk.json"
    )
    #page = context.new_page()
    yield context
    context.close()

def get_cookie(all_cookies, cname):
    val = next((cookie['value'] for cookie in all_cookies if cookie.get('name') == cname), None)
    return val

def test_first(page_and_auth: Browser):
    page = page_and_auth.new_page()
    page.goto(AUTH_URL)
    vkframe = page.frame(url=lambda url: "id.vk.com" in url)

    if vkframe:
        authbutton = vkframe.get_by_role("button", name="Продолжить как")
        expect(authbutton, "Кнопка Продолжить как... отсутствует").to_be_visible(timeout=20000)
        authbutton.click()
        back_button = page.get_by_role("button", name="Авторизация успешна")
        expect(back_button, "Сервис не произвел авторизацию").to_be_visible()
        back_button.click()
        all_cookies = page_and_auth.cookies()
        session_id_value = get_cookie(all_cookies, 'session_id')
        assert session_id_value, "Cookie session_id не установлен."
    else:
        assert False, "VK фрейм не найден."

```

# Дополнительные ВОЗМОЖНОСТИ

## Скриншоты

Скрин страницы

```
page.screenshot(path="", full_page=True)
```

Скрин элемента тоже работает.

```
link.screenshot(path="")
```

## Запись видео

```
from playwright.sync_api import Browser

def test_first(browser: Browser):
    context = browser.new_context(
        storage_state="playwright/.auth/vk.json",
        record_video_dir="video/"
    )
    page = context.new_page()
    page.goto("https://playwright.dev/python")
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    assert page.url == "https://playwright.dev/python/docs/intro"
    page.goto("https://vk.ru")
```

Вариант с текстурой:

```
import pytest
from playwright.sync_api import Browser, Page

@pytest.fixture
def recordable(browser: Browser):
    context = browser.new_context(
```

```

    storage_state="playwright/.auth/vk.json",
    record_video_dir="video/"
)
page = context.new_page()
yield page
context.close()

def test_first(recordable: Page):
    recordable.goto("https://playwright.dev/python")
    link = recordable.get_by_role("link", name="GET STARTED")
    link.click()
    assert recordable.url == "https://playwright.dev/python/docs/intro"
    recordable.goto("https://vk.ru")

```

## Трассировка данных

Создание трассировки:

```

import pytest
from playwright.sync_api import Page, BrowserContext

@pytest.fixture(autouse=True)
def trace_test(context: BrowserContext):
    context.tracing.start(
        name="playwrite",
        screenshots=True,
        snapshots=True,
        sources=True
    )
    yield
    context.tracing.stop(path="trace.zip")

def test_first(page: Page):
    page.goto("https://playwright.dev/python")
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    assert page.url == "https://playwright.dev/python/docs/intro"

```

Просмотр трассировки

```
playwright show-trace trace.zip
```

## Запись действий

Старт записи

```
playwright codegen playwright.dev
```

В окне кода есть поле Target, позволяет переключать тип библиотек (playwright sync/async, pytest). Потом сохраняем полученную последовательность и устанавливаем нужные условия для проверки.

# Ожидание

```
from playwright.sync_api import Page, expect

DOCS_URL = "https://playwright.dev/python/docs/intro"

def test_first(page: Page):
    page.goto("https://playwright.dev/python")
    link = page.get_by_role("link", name="GET STARTED")
    link.click()
    #assert page.url == DOCS_URL
    expect(page).to_have_url(DOCS_URL)
```

expect(page).to_have_url	наличие url
expect(page).to_have_title	наличие title
link = page.get_by_role("link", name="GET STARTEDer") expect(link).to_be_visible()	Видимость элемента в переменной link
expect(link).to_be_enabled()	Доступный элемент
expect(heading).to_contain_text()	Присутствие текста (часть)
expect(heading).to_have_text()	Присутствие текста (полное совпадение)
expect(mylink).to_have_class()	<div>Наличие класса у элемента Несколько классов: "class1 class2" Должно быть полное соответствие. Но можно использовать регулярки.</div> <div><pre>expect(mylink).to_have_class(     re.compile(r"navbar__link") )</pre></div>
expect(mylink).to_have_id()	Наличие id
expect(mylink).to_have_attribute(attr_name, attr_value)	Наличие атрибута. При необходимости можно указать значение атрибута.
expect(mylink).to_be_editable()	
expect(mylink).to_be_empty()	
expect(mycheckbox).to_be_checked()	



<code>expect(mymenu).to_have_value()</code>	Элемент в меню выбора
<code>expect(mymultimenu).to_have_values([])</code>	Несколько выбранных

`not_` - префикс отрицания