??????????? Playwright

??????

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: 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>`
	`  <label for="exampleInputEmail1" class="form-label mt-4">Email address</label>`
	`  <input type="email" class="form-control" id="exampleInputEmail1" aria-describedby="emailHelp" placeholder="Enter email">`
	`  <small id="emailHelp" class="form-text text-muted">We'll never share your email with anyone else.</small>`
	`</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)	Поиск по CSS. Можно использовать tagname, classname, id, attribute/value Примеры^ css=h1 footer <tagname>.<classname> button.btn-outline-sucess <tagname>#<idname> button#BtnGroupDrop1 <tagname>[attribute] input[readonly] <tagname>[attribute=somevalue] input[value='correct value']
	Поиск по иерархии элементов. Если через пробелы-то вложенные элементы. Если через точку - то у элемента несколько классов. Но они не обязательно непосредственно вложенные. nav.bg-dark a.nav-link.active Для непосредственного вложения: nav.bg-dark > a.nav-link.active
	Называются sudo классами, Класс и текст в теге. Для вхождения: h1:text('Navbars') Для полного соответствия: h1:text-is('Navbars') div.dropdown-menu:visible Для определения по номеру вхождения, когда их много :nth-match(button.btn-primary, 4)
	XPath Абсолютный путь: xpath=/html/head/title С любого начала: xpath=//h1/h2 С указанием атрибута xpath=//h1[@id='navbars']
	Функции XPath Для поиска по тексту, точно: //h1[text()='Headling1'] Для поиска по тексту, содержит: //h1[contains(text(), 'Headling1')] Для поиска по тексту, содержит: //h1[contains(@class, 'btn')]
Множественные условия	Поиск родительского элемента page.get_by_label("Email address").locator("..") Фильтрация page.get_by_role("heading").filter(has_text="First") По дочернему элементу page.locator("div.form-group").filter(has=page.get_by_label("Password"))

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
	`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")`

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

??????? (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-KGDJBEFF3W'
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-
KGDJBEFF3W&gtm=45je54g0v9135688085za200&_p=1744912703288&gcd=13l3l3l3l1l1&npa=0&dma=0&tag_exp=
102509682~102803279~102813109~102887800~102926062~103027016~103051953~103055465~103077950~1031
06314~103106316~103130495~103130497&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&are=1&fr
m=0&pscdl=noapi&_s=1&sid=1744912703&sct=1&seg=0&dl=https%3A%2F%2Fbootswatch.com%2Fdefault%2F&d
t=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 accept 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()
```

????????????

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
assert mynum == 32	Проверка на значение
assert type(dt) == dict	Проверка типа
assert "timestamp" in dt_list	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.fixtures
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 исполняется каждый раз при вызове. Для однократного исполнения фикстуры нужно добавить scope="session"

```
import json
import report
import pytest

@pytest.fixtures(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."
```

?????????????? ???????????

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.

????????

```
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()	Наличие класса у элемента Несколько классов: "class1 class2"
	Должно быть полное соответствие. Но можно использовать регулярки.
	`expect(mylink).to_have_class(`
	`re.compile(r"navbar__link")`
	`)`

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()	
expect(mymenu).to_have_value()	Элемент в меню выбора

expect(mymultimenu).to_have_values([])	Несколько выбранных

not_ - префикс отрицания

Pytest

Теория

Виды тестирования

- Модульное: небольшой элемент/модуль
- Компонентное: проверка подсистем по отдельности
- Альфа/бета: в реальных условиях на настоящих данных в прод версии
- Комплексное: проверка связей интерфейсов между парами и группами компонентов
- Системное: поведение как в целом, так и в частности
- ПСИ: проверка удовлетворения системы требованиям
- Пилотное: опытная эксплуатация под тщательным контролем

Планирование тестирования

1. Планирование тестирования - описывает все процессы. Понимание места тестирования
    1. операционный и организационный контекст
    2. Риски качества системы + ранжирование, понимание каждого возможностей тестирования для смягчения рисков
    3. Потребности временнЫх ресурсах
    4. План мероприятий по тестированию. Задачи, состав участников.

    ```
    Важно Ожидание качества (численные критерии, предьявляемые перед началом) и
    Опыт качества (численное значение критериев после выполнения работ).
    Жизненный цикл системы = Жизненный цикл разработки + эксплуатации
    ```

2. Подготовка к тестированию
    1. Обучение тестировщиков до нужного уровня
    2. Спроектировать систему тестирования (окружение, процедура, распределение задач,)
3. Проведение тестирования
    1. Получение версии для тестирования
    2. Проведение тестов

    ```
    Предпочтительнее алгоритм Дано-Ожидаемо-Проверка
    Лучше пофункциональное тестирование. Интегральные тесты хороши но не дают нужной детализации.
    Каждый тест должен возвращать состояние в начальное, они должны быть последовательно-независимыми
    ```

4. Совершенствование
    1. Документирование тестирования
    2. Информирование о результатах
    3. При изменении контекста изменение процесса

Построение успешного процесса

- Что делает группа тестирования, когда применяет успешный процесс, и какие преимущества она получает.
- Нереалистичные ожидания со стороны руководства
- Получить согласие на изменение процесса сложнее самого изменения
- Процесс усовершенствования требует наличия плана

Элементы документации

- Точка тестирования:
- Итоговая цель тестирования:
- Краткосрочная цель и параметры тестирования:
- Документирование:
- Фиксирование версионности:
- Приоретизация тестирования:
- Параметры тестируемой подсистемы:
  - Блоки:
  - Функции в каждом блоке:
  - Переменные каждой функции:
  - Комбинированные ожидаемые результаты каждой функции от переменной:

Pytest

Соглашения об именованиях

Имя файла должна начинаться с test_
Функция тестирования должна начинаться с test_
Классы: Test<Something>

Запуск тестов

pytest
по-умолчанию без параметров - все файлы test_ в текущей папке и поддиректориях
pytest test_classes.py::TestEquality Запуск конкретного класса
pytest test_classes.py::TestEquality::test_equality Запуск конкретного метода в конкретном тестклассе
@pytest.mark.skip() или @pytest.mark.skipif() - декораторы для пропуска теста
test_file.py использование конкретного файла
dir конкретная директория
-v расширенный вывод
--tb=no включение/отключение traceback, по-умолчанию включено
-k Маркер
-k equality все классы/тесты с именем включающим слово equality
-k "equality and not equality_fail" все классы/тесты с именем включающим слово equality и без equality_fail
-k "(dict or ids) and not TestEquality"
--setup-show показывает последовательность применения fixture и самого теста
--fixtures -v расположение файла fixtures
--fixtures-per-test отдельно использовать fixtures для каждого теста

Возможные статусы:

PASSED (.), FAILED (F), SKIPPED (s),

XFAIL (x), Тест ожидался провальным (был обернут декоратором @pytest.mark.xfail()), и провалился
XPASS (X), Тест ожидался провальным (был обернут декоратором @pytest.mark.xfail()), но успешно
ERROR (E) При выполнении теста исключение

Пример полного файла (test_one.py):

```
def test_passing():
    assert (1, 2, 3) == (1, 2, 3)
```

запуск: pytest test_one.py

Примеры функций тестов
Изменение функции проверки

```
def assert_identical(c1: Card, c2: Card):
    __tracebackhide__ = True
    assert c1 == c2
    if c1.id != c2.id:
        pytest.fail(f'id\'s don\'t match. {c1.id} != {c2.id}')
```

Тестирование ожидаемого исключения

```
def test_no_path_raises():
    with pytest.raises(TypeError):
        cards.CardsDB()
```

Тестирование ожидаемого исключения с конкретным текстом через regex

```
def test_raises_with_info():
    match_regex = "missing 1 .* positional argument"
    with pytest.raises(TypeError, match=match_regex):
         cards.CardsDB()
```

Тестирование ожидаемого исключения с конкретным текстом через проверку наличия текста

```
def test_raises_with_info_alt():
    with pytest.raises(TypeError) as exc_info:
        cards.CardsDB()
    expected = "missing 1 required positional argument"
```

```
    assert expected in str(exc_info.value)
```

Fixtures
функции, запускающиеся до выполнения тестов (и/или после), для перевода системы в нужный контекст.
При ошибке в fixture генерится Error
@pytest.fixture() - запуск при каждом обращении
@pytest.fixture(scope="module") - один запуск на уровне модуля
scope='function' по умолчанию
scope='class' один запуск на уровне класса
scope='module'
scope='package' - в случае определения fixture в файле conftest.py
scope='session' - в случае определения fixture в файле conftest.py

Запуск до исполнения

```
import pytest


@pytest.fixture()
def some_data():
    """Return answer to ultimate question."""
    return 42


def test_some_data(some_data):
    """Use fixture return value in a test."""
    assert some_data == 42
```

Пример запуска с итераторами:

```
@pytest.fixture()
def cards_db():
    # setup part
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
        db = cards.CardsDB(db_path)
        # end setup part, return db object
        yield db
        # closing db after testing
        db.close()
```

```
def test_empty(cards_db):
    # in cards_db - db object, we can use it
    assert cards_db.count() == 0
```

Fixtures можно вынести в отдельный файл conftest.py В директории теста или в родительской директории

```
pytest --fixtures -v расположение файла
```

```
#ch3/a/conftest.py
from pathlib import Path
from tempfile import TemporaryDirectory
import cards
import pytest

@pytest.fixture(scope="session")
def cards_db():
    """CardsDB object connected to a temporary database"""
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
        db = cards.CardsDB(db_path)
        yield db
        db.close()

#ch3/a/test_count.py
import cards

def test_empty(cards_db):
    assert cards_db.count() == 0
def test_two(cards_db):
    cards_db.add_card(cards.Card("first"))
    cards_db.add_card(cards.Card("second"))
    assert cards_db.count() == 2
```

Взаимозапуск fixtures

```
    @pytest.fixture(scope="session")
def db():
    """CardsDB object connected to a temporary database"""
    with TemporaryDirectory() as db_dir:
        db_path = Path(db_dir)
```

```
    db_ = cards.CardsDB(db_path)
    yield db_
    db_.close()

@pytest.fixture(scope="function")
def cards_db(db):
    """CardsDB object that's empty"""
    db.delete_all()
    return db
```

Множественное использование fixtures

```
#ch3/c/conftest.py
@pytest.fixture(scope="function")
def non_empty_db(cards_db, some_cards):
...
```