UnCodemy

Playwright with Python PDF

Playwright for Python is installed with pip, uses the pytest-playwright plugin for ready-made page fixtures, and offers synchronous and asynchronous APIs. Run playwright install to download browsers, write tests as plain pytest functions that take a page argument, and use locators with expect for stable assertions.

Key takeaways

- Install the pytest-playwright package, then run playwright install for browsers.
- The page fixture gives each test a fresh browser context.
- Use the sync API with pytest for simple tests and async for async code bases.
- Locators and expect work like the JavaScript version, with Python naming.
- Run with flags such as --headed, --browser and --tracing for debugging.

What is Playwright with Python?

Playwright for Python is the Python binding of the Playwright browser automation library, usually combined with pytest through the pytest-playwright plugin to write and run end-to-end tests.

Why use Playwright with Python?

Teams that already write Python for scripts, data work or backend services can reuse their skills and tools. Playwright supplies fast, reliable browser control, and pytest supplies test discovery, fixtures and parameters. The feature set is the same as in other languages: auto-waiting, locators, tracing, network control and several browsers.

How do I install it?

```
pip install pytest-playwright
playwright install
```

The first command installs the library and the pytest plugin; the second downloads browser builds. Use a virtual environment so versions do not clash with other projects. In CI, run `playwright install --with-deps` where system packages are needed.

What does a first test look like?

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

def test_has_title(page: Page):
    page.goto("https://playwright.dev/")
    expect(page).to_have_title(re.compile("Playwright"))

def test_get_started_link(page: Page):
    page.goto("https://playwright.dev/")
    page.get_by_role("link", name="Get started").click()
    expect(page.get_by_role("heading", name="Installation")).to_be_visible()
```

The `page` fixture comes from the plugin and opens a new browser context for each test, so tests do not share cookies or storage.

How do locators look in Python?

Names use snake case: `get_by_role`, `get_by_label`, `get_by_text`, `get_by_test_id`, `filter` and `locator`. Assertions use `expect(locator).to_have_text(...)`, `to_be_visible()` and `to_have_count(...)`. They retry just as in other languages, so avoid `time.sleep`.

Which command line options help?

Option	Purpose
`--headed`	Show the browser window
`--browser firefox`	Choose a browser, repeat for several
`--slowmo 500`	Slow actions for watching
`--tracing on`	Record traces for failures
`--video on`	Record video
`--screenshot only-on-failure`	Save images when tests fail
Combine with pytest options like `-k` to select tests and `-n` with the xdist plugin for parallel runs.	

How do I share setup with fixtures?

Create a `conftest.py` with fixtures. For example, a fixture can log in once and return a page that is already authenticated, or supply a base URL through the `base_url` setting. Use `browser_context_args` to set viewport, locale or storage state for all tests. Keep fixtures small and name them for what they provide.

Should I use the sync or async API?

Use the sync API for pytest tests, since it reads simply and works with the plugin. The async API is for code that already uses `asyncio`, such as a scraper or service. Do not mix them in the same test module.

How do I structure a project?

A typical layout has `tests/`, `pages/` for page objects, `conftest.py`, `pytest.ini` and a requirements file. Page objects keep locators in one class per page. Store environment settings in variables or a config file, not in the tests.

How do I run in CI?

Install dependencies, run `playwright install --with-deps`, run `pytest` with tracing on failure, and upload the `test-results` folder as an artifact so failures can be opened with the trace viewer. Keep browsers cached between runs to save time.

How do I run tests in parallel with Python?

Install the pytest-xdist plugin and run `pytest -n auto` to spread tests across processes. Because each test gets a fresh browser context, parallel runs stay isolated, provided that test data does not collide. Use unique names for created records and avoid shared files. Check the application can handle the load, and lower the worker count if the test environment is small.

How do I debug failures quickly?

Run a single test with `--headed` and `--slowmo`, set the `PWDEBUG=1` environment variable to open the inspector, and enable tracing so the trace viewer shows each step, network call and snapshot. Use `page.pause()` to stop at a point and try locators interactively. These tools shorten the loop between a failing line and an understanding of why it failed.

What common errors do beginners meet?

A missing browser install gives an executable-not-found message, so run playwright install. Importing the async API in a sync test causes confusing errors. A fixture named page defined by you can shadow the plugin one. Using time.sleep hides timing problems. Keep versions of Playwright and the plugin aligned, and read the error text carefully before searching.

How do I practise Playwright with Python?

Automate one flow you use daily, such as logging in to a practice site and checking a dashboard. Add a second test with different data using pytest parametrize, then run both in two browsers. Reading the trace after a deliberate failure teaches you more than reading documentation alone.

Frequently asked questions

How do I install Playwright for Python?

Install pytest-playwright with pip, then run playwright install to download the browsers, preferably inside a virtual environment.

What does the page fixture do?

The page fixture from pytest-playwright gives each test a fresh browser context and page, so tests stay isolated from each other.

Should I use sync or async Playwright in Python?

Use the sync API with pytest for most tests, and choose async only when your code already runs on asyncio.

Can I run tests in several browsers?

Yes. Pass the browser option more than once, such as chromium and firefox, and pytest runs the tests for each browser.

Do I still need time.sleep?

No. Playwright auto-waits for actions and expect assertions retry, so fixed sleeps should be avoided in nearly all cases.

Sources and further reading

- Playwright for Python
- pytest documentation

About this PDF

These PDFs are study material for learning and revision. They are not a certificate, a degree or a promise of a job, and technology changes, so check official documentation for the latest details. Spot an error? Write to info@uncodemy.com and the trainer team will review it.

More free resources and courses: uncodemy.com | info@uncodemy.com | +91 98183 66550