Getting started - Library

Installation

Pip

pypi package 1.49.0

```
pip install --upgrade pip
pip install playwright
playwright install
```

Conda

conda | microsoft v1.49.0

```
conda config --add channels conda-forge
conda config --add channels microsoft
conda install playwright
playwright install
```

These commands download the Playwright package and install browser binaries for Chromium, Firefox and WebKit. To modify this behavior see installation parameters.

Usage

Once installed, you can `import` Playwright in a Python script, and launch any of the 3 browsers (`chromium`, `firefox` and `webkit`).

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
    page = browser.new_page()
    page.goto("http://playwright.dev")
```

```
    print(page.title())
    browser.close()
```

Playwright supports two variations of the API: synchronous and asynchronous. If your modern project uses asyncio, you should use async API:

```
import asyncio
from playwright.async_api import async_playwright

async def main():
    async with async_playwright() as p:
        browser = await p.chromium.launch()
        page = await browser.new_page()
        await page.goto("http://playwright.dev")
        print(await page.title())
        await browser.close()

asyncio.run(main())
```

First script

In our first script, we will navigate to `https://playwright.dev/` and take a screenshot in WebKit.

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.webkit.launch()
    page = browser.new_page()
    page.goto("https://playwright.dev/")
    page.screenshot(path="example.png")
    browser.close()
```

By default, Playwright runs the browsers in headless mode. To see the browser UI, set headless option to `False`. You can also use slow_mo to slow down execution. Learn more in the debugging tools section.

```
firefox.launch(headless=False, slow_mo=50)
```

Interactive mode (REPL)

You can launch the interactive python REPL:

```
python
```

and then launch Playwright within it for quick experimentation:

```
from playwright.sync_api import sync_playwright
playwright = sync_playwright().start()
# Use playwright.chromium, playwright.firefox or playwright.webkit
# Pass headless=False to launch() to see the browser UI
browser = playwright.chromium.launch()
page = browser.new_page()
page.goto("https://playwright.dev/")
page.screenshot(path="example.png")
browser.close()
playwright.stop()
```

Async REPL such as `asyncio` REPL:

```
python -m asyncio
```

```
from playwright.async_api import async_playwright
playwright = await async_playwright().start()
browser = await playwright.chromium.launch()
page = await browser.new_page()
await page.goto("https://playwright.dev/")
await page.screenshot(path="example.png")
await browser.close()
await playwright.stop()
```

Pyinstaller

You can use Playwright with Pyinstaller to create standalone executables.

main.py

```
from playwright.sync_api import sync_playwright

with sync_playwright() as p:
    browser = p.chromium.launch()
```

```
    page = browser.new_page()
    page.goto("https://playwright.dev/")
    page.screenshot(path="example.png")
    browser.close()
```

If you want to bundle browsers with the executables:

Bash | PowerShell | Batch

```
PLAYWRIGHT_BROWSERS_PATH=0 playwright install chromium
pyinstaller -F main.py
```

> **NOTE**
>
> Bundling the browsers with the executables will generate bigger binaries. It is recommended to only bundle the browsers you use.

Known issues

`time.sleep()` leads to outdated state

Most likely you don't need to wait manually, since Playwright has auto-waiting. If you still rely on it, you should use `page.wait_for_timeout(5000)` instead of `time.sleep(5)` and it is better to not wait for a timeout at all, but sometimes it is useful for debugging. In these cases, use our wait (`wait_for_timeout`) method instead of the `time` module. This is because we internally rely on asynchronous operations and when using `time.sleep(5)` they can't get processed correctly.

incompatible with `SelectorEventLoop` of `asyncio` on Windows

Playwright runs the driver in a subprocess, so it requires `ProactorEventLoop` of `asyncio` on Windows because `SelectorEventLoop` does not supports async subprocesses.

On Windows Python 3.7, Playwright sets the default event loop to `ProactorEventLoop` as it is default on Python 3.8+.

Threading

Playwright's API is not thread-safe. If you are using Playwright in a multi-threaded environment, you should create a playwright instance per thread. See threading issue for more details.