



Pydoll is a library for automating chromium-based browsers without a WebDriver, offering realistic interactions.

[autoscraper-labs.github.io/pydoll/](#)

MIT license

5k stars

271 forks

34 watching

Branches

Activity

Custom properties

Tags

Public repository

thalissonvs

Merge pull request #209 from autoscraper-labs/fix/new-tab-loading

✓

41d8406 · yesterday

<div></div> .github	ci: add mkdocs-static-i18n to docume...	2 weeks ago
<div></div> docs	docs: configure chinese documentation	2 weeks ago
<div></div> examples	style: enhance Cloudflare bypass with ...	3 months ago
<div></div> pydoll	refactor: remove redundant asterisk fr...	yesterday
<div></div> tests	test: remove hardcoded URLs in new_...	yesterday
<div></div> .gitignore	fix: remove target directory from .giti...	2 months ago
<div></div> .python-version	feat: add .gitignore, .python-version, a...	9 months ago
<div></div> CHANGELOG.md	bump: version 2.3.0 → 2.3.1	2 weeks ago
<div></div> CONTRIBUTING.md	chore: add a contribution guide	4 months ago
<div></div> LICENSE	chore: create license file	5 months ago
<div></div> README.md	Update README.md	last week
<div></div> README_zh.md	docs: finish all the translation	3 weeks ago
<div></div> codecov.yml	ci: add Codecov configuration for cov...	5 months ago
<div></div> cz.yaml	bump: version 2.3.0 → 2.3.1	2 weeks ago
<div></div> mkdocs.yml	docs: enable directory URLs and clean...	2 weeks ago
<div></div> poetry.lock	Update poetry.lock	2 weeks ago
<div></div> pyproject.toml	build: add 'E701' to ignore list in ruff li...	yesterday





Pydoll: Automate the Web, Naturally



[Documentation](#) • [Getting Started](#) • [Advanced Features](#) • [Contributing](#) • [Support My Work](#)

Imagine the following scenario: you need to automate tasks in your browser. Maybe it's testing a web application, collecting data from a site, or even automating repetitive processes. Normally this involves using external drivers, complex configurations, and many compatibility issues.

Pydoll was born to solve these problems.

Built from scratch with a different philosophy, Pydoll connects directly to the Chrome DevTools Protocol (CDP), eliminating the need for external drivers. This clean implementation along with realistic ways of clicking, navigating and interacting with elements makes it practically indistinguishable from a real user.

We believe that powerful automation shouldn't require you to become an expert in configuration or constantly fight with bot protection systems. With Pydoll, you can focus on what really matters: your automation logic, not the underlying complexity or protection systems.

☀️ What makes Pydoll special?

- **Zero Webdrivers:** Say goodbye to webdriver compatibility issues
- **Human-like Interaction Engine:** Capable of passing behavioral CAPTCHAs like reCAPTCHA v3 or Turnstile, depending on IP reputation and interaction patterns
- **Asynchronous Performance:** For high-speed automation and multiple simultaneous tasks
- **Humanized Interactions:** Mimic real user behavior
- **Simplicity:** With Pydoll, you install and you're ready to automate.

⚖️ Ethical Use Disclaimer

Pydoll is designed to simulate realistic browser interactions in order to avoid being flagged as a bot during legitimate automation tasks. It does not break or circumvent CAPTCHA challenges in the traditional sense (e.g. by solving image puzzles or hacking the system).

Instead, it increases the likelihood of being treated as a real user by mimicking human behavior, which can result in passing behavioral-based protections like reCAPTCHA v3 or Cloudflare Turnstile.

We do not support or endorse using this tool to access websites against their terms of service, scrape personal data, or bypass explicit access restrictions.

Use it responsibly. 🚧

📦 Installation

```
pip install pydoll-python
```



And that's it! Just install and start automating.

Getting Started

Your first automation

Let's start with a real example: an automation that performs a Google search and clicks on the first result. With this example, we can see how the library works and how you can start automating your tasks.

```
import asyncio

from pydoll.browser import Chrome
from pydoll.constants import Key

async def google_search(query: str):
    async with Chrome() as browser:
        tab = await browser.start()
        await tab.go_to('https://www.google.com')
        search_box = await tab.find(tag_name='textarea', name='q')
        await search_box.insert_text(query)
        await search_box.press_keyboard_key(Key.ENTER)
        await (await tab.find(
            tag_name='h3',
            text='autoscraps-labs/pydoll',
            timeout=10,
        )).click()
        await tab.find(id='repository-container-header', timeout=10)

asyncio.run(google_search('pydoll python'))
```

Without configurations, just a simple script, we can do a complete Google search! Okay, now let's see how we can extract data from a page, using the same previous example. Let's consider in the code below that we're already on the Pydoll page. We want to extract the following information:

- Project description
- Number of stars

 README  MIT license



- Number of pull requests

Let's get started! To get the project description, we'll use xpath queries. You can check the documentation on how to build your own queries.

```
description = await (await tab.query(
    '//h2[contains(text(), "About")]/following-sibling::p',
    timeout=10,
)).text
```

And that's it! Let's understand what this query does:

1. `//h2[contains(text(), "About")]` - Selects the first `<h2>` that contains "About"
2. `/following-sibling::p` - Selects the first `<p>` that comes after the `<h2>`

Now let's get the rest of the data:

```
number_of_stars = await (await tab.find(
    id='repo-stars-counter-star'
)).text
```

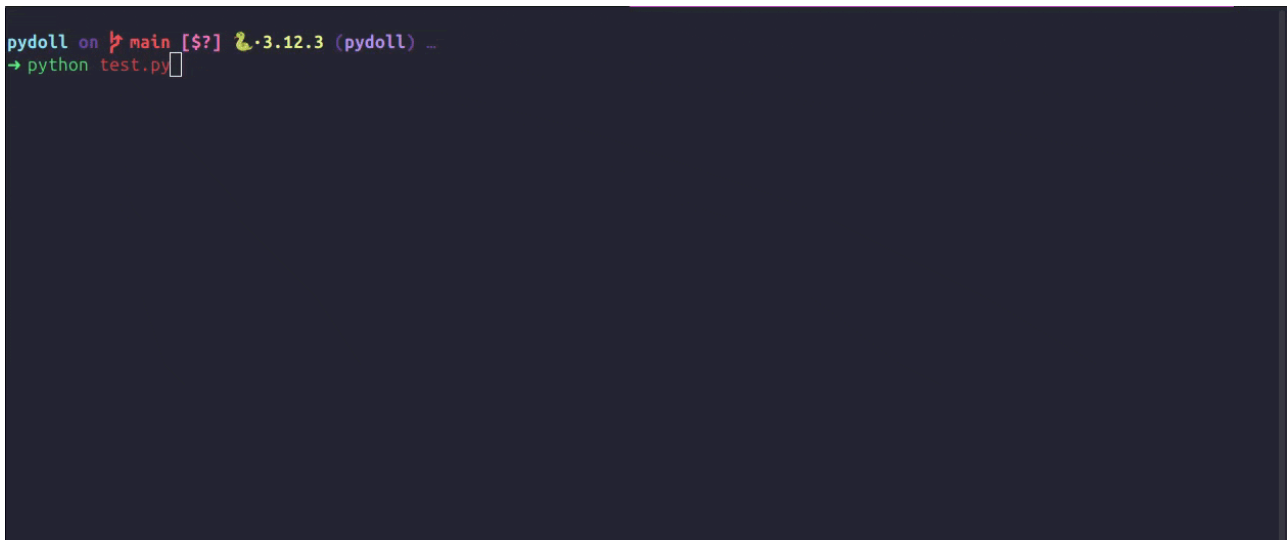
```

number_of_forks = await (await tab.find(
    id='repo-network-counter'
)).text
number_of_issues = await (await tab.find(
    id='issues-repo-tab-count',
)).text
number_of_pull_requests = await (await tab.find(
    id='pull-requests-repo-tab-count',
)).text

data = {
    'description': description,
    'number_of_stars': number_of_stars,
    'number_of_forks': number_of_forks,
    'number_of_issues': number_of_issues,
    'number_of_pull_requests': number_of_pull_requests,
}
print(data)

```

In the image below we can see the execution speed and the result of the automation. For demonstration purposes, the browser is not displayed.



In just 5 seconds, we managed to extract all the necessary data! This is the speed you can expect from automation with Pydoll.

A more complex example

Let's now move to a case that you've probably encountered many times: a captcha like Cloudflare's. Pydoll has a method to try to handle this, although, as mentioned earlier, effectiveness depends on various factors. In the code below, we have a complete example of how to handle a Cloudflare captcha.

```

import asyncio

from pydoll.browser import Chrome
from pydoll.constants import By

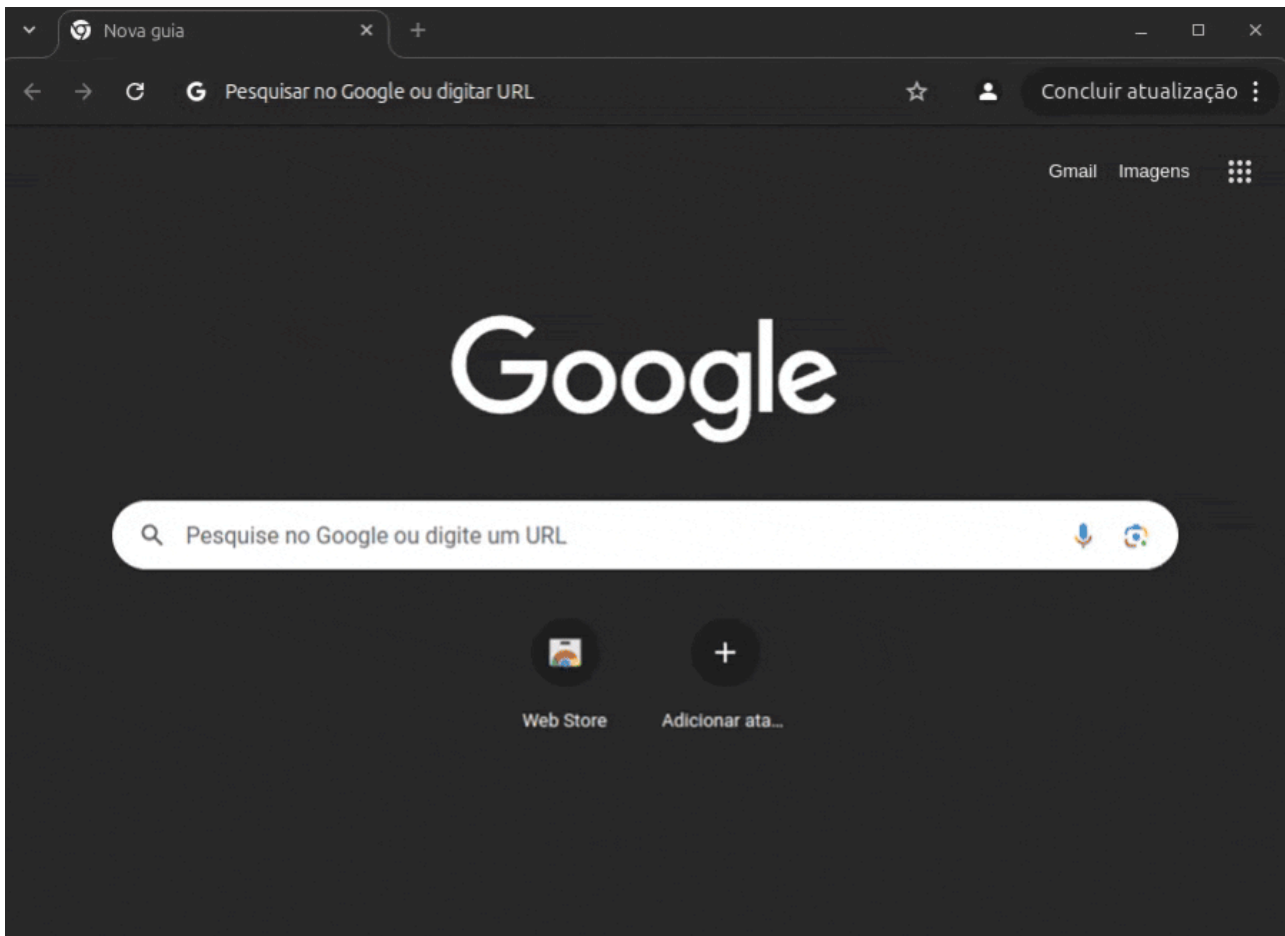
async def cloudflare_example():
    async with Chrome() as browser:
        tab = await browser.start()
        async with tab.expect_and_bypass_cloudflare_captcha():
            await tab.go_to('https://2captcha.com/demo/cloudflare-turnstile')
            print('Captcha handled, continuing...')
            await asyncio.sleep(5) # just to see the result :)

```



```
asyncio.run(cloudflare_example())
```

Below we have the result of the execution:



With just a few lines of code, we managed to handle one of the most difficult captchas to deal with. This is just one of the many functionalities that Pydoll offers. But it doesn't stop there!

Custom Configurations

Sometimes we need more control over the browser. Pydoll offers a flexible way to do this. Let's see the example below:

```
from pydoll.browser import Chrome
from pydoll.browser.options import ChromiumOptions as Options

async def custom_automation():
    # Configure browser options
    options = Options()
    options.add_argument('--proxy-server=username:password@ip:port')
    options.add_argument('--window-size=1920,1080')
    options.binary_location = '/path/to/your/browser'
    options.start_timeout = 20

    async with Chrome(options=options) as browser:
        tab = await browser.start()
        # Your automation code here
        await tab.go_to('https://example.com')
        # The browser is now using your custom settings

asyncio.run(custom_automation())
```

In this example, we're configuring the browser to use a proxy and a 1920x1080 window, in addition to a custom path for the Chrome binary, in case your installation location is different from the common defaults.

⚡ Advanced Features

Pydoll offers a series of advanced features to please even the most demanding users.

Advanced Element Search

We have several ways to find elements on the page. No matter how you prefer, we have a way that makes sense for you:

```
import asyncio
from pydoll.browser import Chrome

async def element_finding_examples():
    async with Chrome() as browser:
        tab = await browser.start()
        await tab.go_to('https://example.com')

        # Find by attributes (most intuitive)
        submit_btn = await tab.find(
            tag_name='button',
            class_name='btn-primary',
            text='Submit'
        )
        # Find by ID
        username_field = await tab.find(id='username')
        # Find multiple elements
        all_links = await tab.find(tag_name='a', find_all=True)
        # CSS selectors and XPath
        nav_menu = await tab.query('nav.main-menu')
        specific_item = await tab.query('//div[@data-testid="item-123"]')
        # With timeout and error handling
        delayed_element = await tab.find(
            class_name='dynamic-content',
            timeout=10,
            raise_exc=False # Returns None if not found
        )
        # Advanced: Custom attributes
        custom_element = await tab.find(
            data_testid='submit-button',
            aria_label='Submit form'
        )

asyncio.run(element_finding_examples())
```

The `find` method is more user-friendly. We can search by common attributes like `id`, `tag_name`, `class_name`, etc., up to custom attributes (e.g. `data-testid`).

If that's not enough, we can use the `query` method to search for elements using CSS selectors, XPath queries, etc. Pydoll automatically takes care of identifying what type of query we're using.

Concurrent Automation

One of the great advantages of Pydoll is the ability to process multiple tasks simultaneously thanks to its asynchronous implementation. We can automate multiple tabs at the same time! Let's see an example:

```
import asyncio
from pydoll.browser import Chrome
```

```

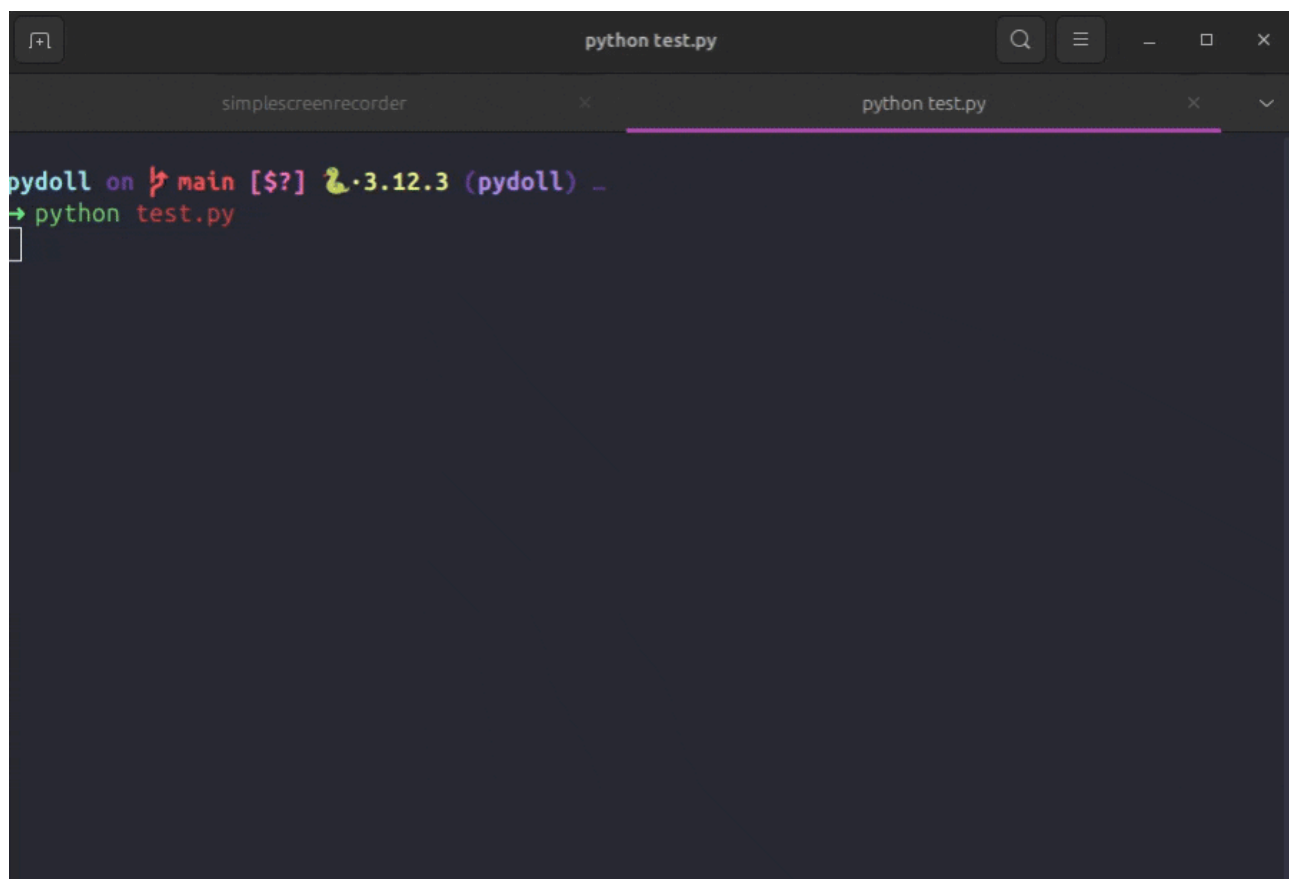
async def scrape_page(url, tab):
    await tab.go_to(url)
    title = await tab.execute_script('return document.title')
    links = await tab.find(tag_name='a', find_all=True)
    return {
        'url': url,
        'title': title,
        'link_count': len(links)
    }

async def concurrent_scraping():
    browser = Chrome()
    tab_google = await browser.start()
    tab_duckduckgo = await browser.new_tab()
    tasks = [
        scrape_page('https://google.com/', tab_google),
        scrape_page('https://duckduckgo.com/', tab_duckduckgo)
    ]
    results = await asyncio.gather(*tasks)
    print(results)
    await browser.stop()

asyncio.run(concurrent_scraping())

```

Below we see the incredible execution speed:



```

python test.py

```

We managed to extract data from two pages at the same time! Tell me if that's not incredible?

And there's much, much more! Event system for reactive automations, request interception and modification, and so on. Take a look at the documentation, you won't regret it!

Quick Troubleshooting

Browser not found?

```
from pydoll.browser import Chrome
from pydoll.browser.options import ChromiumOptions

options = ChromiumOptions()
options.binary_location = '/path/to/your/chrome'
browser = Chrome(options=options)
```

Browser starts after a FailedToStartBrowser error?

```
from pydoll.browser import Chrome
from pydoll.browser.options import ChromiumOptions

options = ChromiumOptions()
options.start_timeout = 20 # default is 10 seconds

browser = Chrome(options=options)
```

Need a proxy?

```
options.add_argument('--proxy-server=your-proxy:port')
```

Running in Docker?

```
options.add_argument('--no-sandbox')
options.add_argument('--disable-dev-shm-usage')
```

Documentation

For complete documentation, detailed examples and deep dives into all Pydoll functionalities, visit our [official documentation](#).

The documentation includes:

- **Getting Started Guide** - Step-by-step tutorials
- **API Reference** - Complete method documentation
- **Advanced Techniques** - Network interception, event handling, performance optimization

The chinese version of this README is [here](#).

Contributing

We would love your help to make Pydoll even better! Check out our [contribution guidelines](#) to get started. Whether it's fixing bugs, adding features or improving documentation - all contributions are welcome!

Please make sure to:

- Write tests for new features or bug fixes
- Follow code style and conventions
- Use conventional commits for pull requests
- Run lint checks and tests before submitting

💖 Support My Work

If you find Pydoll useful, consider [supporting me on GitHub](#).

You'll get access to exclusive benefits like priority support, custom features and much more!

Can't sponsor right now? No problem — you can still help a lot by:

- Starring the repository
- Sharing on social media
- Writing posts or tutorials
- Giving feedback or reporting issues

Every bit of support makes a difference — thank you!

Releases 38

📦 2.3.1 Latest
2 weeks ago

[+ 37 releases](#)

Sponsor this project

 **thalissonvs** Thalison Fernandes

💖 Sponsor

[Learn more about GitHub Sponsors](#)

Contributors 17



[+ 3 contributors](#)

Deployments 33

✅ **github-pages** yesterday

[+ 32 deployments](#)

Languages

● Python 100.0%