Playwright Cheatsheet

A quick reference guide to commonly used Playwright commands for browser automation and testing.

Browser Start and Close

Search... JavaScript Python (Sync) Python (Async)

```
const { chromium } = require('playwright');

const browser = await chromium.launch();
```

Start a Chrome browser instance

```
const { chromium } = require('playwright');

const browser = await chromium.launch({ channel: 'chrome' });
```

Start a Microsoft Edge browser instance

```
const { chromium } = require('playwright');
```

Start a Firefox browser instance

```
const { firefox } = require('playwright');

const browser = await firefox.launch();
```

Start a WebKit browser instance

```
const { webkit } = require('playwright');
const browser = await webkit.launch();
```

Close the browser instance

```
await browser.close();
```

Context Management

Create a new browser context

```
const context = await browser.newContext();
```

Close the browser context

```
await context.close();
```

Page / Tab Management

Open a new page / tab

```
const page = await context.newPage();
```

Wait for a new page / tab to be opened (e.g. from clicking a link with target="_blank")

```
// add wait before the action that opens the new page
const newPagePromise = context.waitForEvent('page');

// perform the action that opens the new page,
```

```
// e.g. clicking a link
await page.locator('a[target="_blank"]').click();

// wait for the new page to be opened
const newPage = await newPagePromise;
```

List all pages

```
const pages = context.pages();
```

Make the page / tab the active one

```
await page.bringToFront();
```

Close the current page / tab

```
await page.close();
```

Check if the page / tab is closed

```
const isClosed = page.isClosed();
```

Page Information

Get the current page URL

```
const url = page.url();
```

Get the current page title

```
const title = await page.title();
```

Page Assertions

Requires `import { expect } from '@playwright/test'`

Assert the url of a page equals a specific value

```
await expect(page).toHaveURL('https://example.com/dashboard');
```

Assert the url of a page using a function

```
await expect(page).toHaveURL(url => {
  return url.pathname === '/results'
    && url.searchParams.get('page') === '1';
});
```

Assert the title of a page equals a specific value

```
await expect(page).toHaveTitle('Dashboard');
```

Navigation

Navigate to a URL

```
await page.goto('https://example.com');
```

Reload the current page

```
await page.reload();
```

Navigate back in history

```
await page.goBack();
```

Navigate forward in history

```
await page.goForward();
```

Element Selection

Select an element using a CSS selector

```
const element = page.locator('#element');
```

Select an element containing specific text

```
const element = page.getByText('Submit');
```

Select a form element by its associated label text

```
const element = page.getByLabel('Username');
```

Select an element by its ARIA role and name

```
const element = page.getByRole('button', { name: 'Submit' });
```

Select an input element by its placeholder text

```
const element = page.getByPlaceholder('Enter your email');
```

Select an image element by its alt text

```
const element = page.getByAltText('Company Logo');
```

Select an element by its title attribute

```
const element = page.getByTitle('Close');
```

Select an element by its `data-testid` attribute

```
const element = page.getByTestId('submit-button');
```

Waiting for Element States

Wait for an element to be present in the DOM

```
await page.locator('#menu').waitFor({ state: 'attached' });
```

Wait for an element to be visible on the page

```
await page.locator('#menu').waitFor({ state: 'visible' });
```

Wait for an element to be visible on the page with a custom timeout

```
await page.locator('#menu').waitFor({
    state: 'visible',
    timeout: 30 * 60 * 1000 // 30 minutes
  });
```

Wait for an element to be hidden or removed from the page

```
await page.locator('#menu').waitFor({ state: 'hidden' });
```

Wait for an element to be removed from the DOM

```
await page.locator('#menu').waitFor({ state: 'detached' });
```

Element State

Get the text content of an element

```
const text = await page.locator('#element').textContent();
```

Get the inner text of an element

```
const text = await page.locator('#element').innerText();
```

Get the inner HTML of an element

```
const html = await page.locator('#element').innerHTML();
```

Get the outer HTML of an element

```
const html = await page.locator('#element').outerHTML();
```

Get the value of a specific attribute of an element

```
const href = await page.locator('#element').getAttribute('href');
```

Get the value of an input element

```
const value = await page.locator('#input').inputValue();
```

Get the bounding box of an element

```
// box contains x, y, width, height
const box = await page.locator('#element').boundingBox();
```

Check if an element is visible on the page

```
const isVisible = await page.locator('#element').isVisible();
```

Check if an element is hidden on the page

```
const isHidden = await page.locator('#element').isHidden();
```

Check if an element is enabled

```
const isEnabled = await page.locator('#element').isEnabled();
```

Check if an element is disabled

```
const isDisabled = await page.locator('#element').isDisabled();
```

Check if a checkbox or radio button is checked

```
const isChecked = await page.locator('#checkbox').isChecked();
```

Check if an element is editable

```
const isEditable = await page.locator('#input').isEditable();
```

Element Assertions

Requires `import { expect } from '@playwright/test'`

Assert that an element is attached to the DOM

```
await expect(page.locator('#element')).toBeAttached();
```

Assert that an element is visible on the page

```
await expect(page.locator('#element')).toBeVisible();
```

Assert that an element is hidden on the page

```
await expect(page.locator('#element')).toBeHidden();
```

Assert that an element contains specific text (case insensitive)

```
await expect(page.locator('#element')).toContainText('Welcome Master Bruce');
```

Assert that an element contains specific text (case insensitive)

```
await expect(page.locator('#element')).toContainText(
    'wElComE mAster bRuCe',
    { ignoreCase: true }
  );
```

Assert that an element does not contain specific text

```
await expect(page.locator('#element')).not.toContainText('Error');
```

Assert that an input element has a specific value

```
await expect(page.locator('#input')).toHaveValue('Hello World');
```

Assert that a multi-select element has specific selected values

```
await expect(page.locator('#multi-select')).toHaveValues([
    'red', 'green'
  ]);
```

Assert that an element contains a specific CSS class

```
await expect(page.locator('#element')).toHaveClass("error");
```

Assert that an element does not contain a specific CSS class

```
await expect(page.locator('#element')).not.toHaveClass("error");
```

Assert that an element has a specific CSS style

```
await expect(page.locator('#element')).toHaveCSS(
    'display', 'block'
  );
```

Assert that an element has a specific attribute with a specific value

```
await expect(page.locator('#element')).toHaveAttribute('alt-text');
```

Assert that an element has a specific attribute with a specific value

```
await expect(page.locator('#element')).toHaveAttribute(
    'alt-text', 'Company Logo'
  );
```

Assert that a checkbox or radio button is checked

```
await expect(page.locator('#checkbox')).toBeChecked();
```

Assert that a checkbox or radio button is not checked

```
await expect(page.locator('#checkbox')).not.toBeChecked();
```

Assert that an element is enabled

```
await expect(page.locator('#element')).toBeEnabled();
```

Assert that an element is disabled

```
await expect(page.locator('#element')).toBeDisabled();
```

Assert that an element is focused

```
await expect(page.locator('#element')).toBeFocused();
```

Element Click / Hover / Drag and Drop

Click on an element

```
await page.locator('#button').click();
```

Right click on an element

```
await page.locator('#button').click({ button: 'right' });
```

Double click on an element

```
await page.locator('#button').dblclick();
```

Click on an element with keyboard modifiers (e.g. Ctrl, Shift)

```
await page.locator('#button').click({ modifiers: ['Control'] });
```

Click on an element at specific coordinates

```
await page.locator('#button').click({ position: { x: 10, y: 5 } });
```

Hover over an element

```
await page.locator('#button').hover();
```

Hover over an element at specific coordinates relative to top-left of the element

```
await page.locator('#button').hover({ position: { x: 10, y: 5 } });
```

Drag an element and drop it onto another element

```
await page.locator('#source').dragTo(page.locator('#target'));
```

Drag an element and drop it onto another element with an offset

```
await page.locator('#source').dragTo(page.locator('#target'), {
    // relative to the top-left of the source element
    sourcePosition: { x: 10, y: 5 }
    // relative to the top-left of the target element
    targetPosition: { x: 10, y: 5 }
  });
```

Mouse

Move the mouse to specific coordinates relative to viewport

```
await page.mouse.move(100, 200);
```

Click the mouse at the current position

```
await page.mouse.down();
await page.mouse.up();
```

Click the mouse at specific coordinates relative to viewport

```
await page.mouse.click(100, 200);
```

Click the mouse with a specific button (e.g. right button) at specific coordinates relative to viewport

```
await page.mouse.click(100, 200, { button: 'right' });
```

Double-click the mouse at specific coordinates relative to viewport

```
await page.mouse.dblclick(100, 200);
```

Press the mouse button down

```
await page.mouse.down();
```

Press the mouse button down with options

```
await page.mouse.down({ button: 'right'});
```

Release the mouse button

```
await page.mouse.up();
```

Release the mouse button with options

```
await page.mouse.up({ button: 'right'});
```

Scroll the mouse wheel by deltaX and deltaY

```
await page.mouse.wheel(0, 100);
```

Form Input Element Interactions

Fill a text input

```
await page.locator('#input').fill('Hello World');
```

Use `press` to type text input

```
await page.locator('#input').press('Enter');
```

Use `press` to send a key chord (e.g. Control+A to select all text)

```
await page.locator('#input').press('Control+A');
```

Use `pressSequentially` to type text input with a delay between each key

```
await page.locator('#input').pressSequentially(
    'Hello',
    { delay: 100 }
  );
```

Clear a text input

```
await page.locator('#input').clear();
```

Check a checkbox

```
await page.locator('#checkbox').check();
```

Uncheck a checkbox

```
await page.locator('#checkbox').uncheck();
```

Select an option in a dropdown by label or value

```
await page.locator('.select-color').selectOption('Red');
```

Select multiple options in a multi-select dropdown

```
await page.locator('.select-color').selectOption([
    'Red', 'Green'
  ]);
```

Keyboard

Press a key

```
await page.keyboard.press('Enter');
```

Press a key chord (e.g. Control+A to select all text)

```
await page.keyboard.press('Control+A');
```

Type text with delay between each key

```
await page.keyboard.type('Hello World');
```

Type text with delay between each key

```
await page.keyboard.type('Hello World', { delay: 100 });
```

Hold a key down (without releasing it)

```
await page.keyboard.down('Shift');
```

Release a key that is being held down

```
await page.keyboard.up('Shift');
```

Element Interactions

Focus an element

```
await page.locator('#input').focus();
```

Remove focus from an element

```
await page.locator('#input').blur();
```

Scroll an element into view if it is not already visible

```
await page.locator('#element').scrollIntoViewIfNeeded();
```

Focuses on an element and selects all its text content

```
await page.locator('#element').selectText();
```

File Upload and Download

Upload a single file to a file input element

```
let el = page.locator('input[type="file"]');
await el.setInputFiles('photos/mountain.png');
```

Upload multiple files to a file input element

```
let el = page.locator('input[type="file"]');
await el.setInputFiles([
    'photos/mountain.png',
    'photos/river.png'
  ]);
```

Upload multiple files from memory to a file input element

```
let el = page.locator('input[type="file"]');
await el.setInputFiles([
    {
      name: 'file1.txt',
      mimeType: 'text/plain',
      buffer: Buffer.from('Hello World')
    }
  ]);
```

Clear a file input element

```
let el = page.locator('input[type="file"]');
await el.setInputFiles([]);
```

Download a file and save it to disk

```
// set up a download listener before clicking
const downloadEvent = await page.waitForEvent('download');

// trigger download
await page.locator('#download-link').click();

// wait for the download to complete
const download = await downloadEvent;
await download.saveAs('report.pdf');
```

Evaluate javascript

Evaluate JavaScript in the page context as a string

```
const sum = await page.evaluate('1 + 2');
```

Evaluate JavaScript in the page context as a function

```
await page.evaluate(() => alert('Hello World'));
```

Evaluate JavaScript in the page context with arguments

```
const sum = await page.evaluate(([a, b]) => {
  return a + b;
}, [1, 2]);
```

Alert / Prompt / Confirmation Dialogs

Listen for an alert dialog and accept it

```
page.on('dialog', async dialog => {
  await dialog.accept();
});
```

Listen for a prompt dialog and enter text before accepting it

```
page.on('dialog', async dialog => {
  await dialog.accept('Hello World');
});
```

Listen for a confirmation dialog and dismiss it

```
page.on('dialog', async dialog => {
  await dialog.dismiss();
});
```

Listen for an alert dialog and get its message

```
page.on('dialog', async dialog => {
  if(dialog.message() === 'Are you sure?') {
    await dialog.accept();
  } else {
    await dialog.dismiss();
  }
});
```

Cookies

Get all cookies for the current context

```
const cookies = await context.cookies();
```

Add cookies to the current context

```
await context.addCookies([{
   name: 'session-id',
   value: 'abc123',
   domain: 'example.com',
   path: '/',
   expires: 24 * 60 * 60, // 1 day from now
   httpOnly: true,
   secure: true,
   sameSite: 'Lax'
 }]);
```

Clear all cookies in the current context

```
await context.clearCookies();
```

Clear a cookie with a specific name

```
await context.clearCookies({ name: 'session-id' });
```

Clear cookies for a specific domain

```
await context.clearCookies({ domain: 'example.com' });
```

Viewport / Window Size

Get the current viewport size

```
// returns { width: number, height: number }
const viewportSize = page.viewportSize();
```

Set the viewport size

```
await page.setViewportSize({ width: 1280, height: 720 });
```

Screenshots

Take a screenshot of a specific element

```
const element = page.locator('#element');
await element.screenshot({ path: 'element.png' });
```

Take a screenshot of the current viewport

```
await page.screenshot({ path: 'viewport.png' });
```

Take a screenshot of the entire page

```
await page.screenshot({ path: 'fullpage.png', fullPage: true });
```