

# Fundamentos do Playwright

---

## 1 - Seletores no Playwright

---

### `page.get_by_role()` – Localizar por papel acessível

#### Objetivo:

Localizar elementos baseando-se no papel acessível (role) e no nome visível, tornando os testes mais robustos e próximos da interação real do usuário.

#### Exemplo:

```
def test_get_by_role(page):
    '''page.goto('https://automationexercise.com/')
    page.pause()
    page.get_by_role('link', name=' Signup / Login').click()
    page.get_by_role('button', name='Login').click()'''
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.locator('#navbarColor01').get_by_role('button', name='Dropdown').
    click()
```

#### Explicação:

O `get_by_role()` busca elementos com base no papel acessível, como `button`, `link`, `textbox`, `checkbox`, entre outros. Ele considera o texto visível para garantir que o elemento selecionado seja o esperado. Isso torna os testes mais confiáveis, legíveis e menos sensíveis a mudanças na estrutura HTML.

#### Referência oficial:

 <https://playwright.dev/python/docs/locators#locate-by-role>

---

### `page.get_by_text()`

#### Objetivo:

Localizar qualquer elemento que contenha um texto específico, útil para mensagens, labels e textos visíveis na tela.

### Exemplo:

```
def test_get_by_text(page):
    page.goto('https://automationexercise.com/')
    page.pause()
    page.get_by_text('Full-Fledged practice website for').first.click()
```

### Explicação:

O `get_by_text()` encontra elementos que contenham o texto informado. É uma forma rápida de localizar textos fixos, mas deve-se tomar cuidado pois pode haver múltiplos elementos com o mesmo texto, podendo selecionar o errado.

### Referência oficial:

<https://playwright.dev/python/docs/locators#locate-by-text>

## page.get\_by\_label()

### Objetivo:

Localizar campos de formulário usando o texto do label associado, garantindo uma boa prática de acessibilidade.

### Exemplo:

```
def test_get_by_label(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_label('Valid input', exact=True).fill('Teste')
    page.get_by_label("Recipient's username", exact=True).fill('Teste')
```

### Explicação:

O `get_by_label()` busca inputs ou textareas ligados a um `<label>`, tornando a seleção dos campos mais estável e alinhada com padrões de acessibilidade.

### Referência oficial:

<https://playwright.dev/python/docs/locators#locate-by-label>

## page.get\_by\_placeholder()

### Objetivo:

Localizar campos de entrada através do texto placeholder, útil quando os campos têm essa indicação.

### Exemplo:

```
def test_get_by_placeholder(page):
    page.goto('https://automationexercise.com/login')
    page.pause()
    page.get_by_placeholder('Name').fill('Teste')
```

### Explicação:

O seletor `get_by_placeholder()` localiza campos que usam o atributo `placeholder`, o texto cinza dentro do campo antes do usuário digitar.

### Referência oficial:

<https://playwright.dev/python/docs/locators#locate-by-placeholder>

## page.get\_by\_title()

### Objetivo:

Selecionar elementos pelo atributo `title`, que normalmente exibe uma dica (tooltip) ao passar o mouse.

### Exemplo:

```
def test_get_by_title(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_title('Source Title').nth(1).click()
```

### Explicação:

O `get_by_title()` localiza elementos que possuem o atributo `title` com o texto informado, útil para interagir com tooltips e dicas visuais.

### Referência oficial:

<https://playwright.dev/python/docs/locators#locate-by-title>

## page.locator() – Seletores tradicionais (CSS / XPath)

### Objetivo:

Entender como usar seletores genéricos quando os seletores acessíveis não forem suficientes.

---

## CSS

### Exemplo:

```
def test_locator_css(page):
    page.goto('https://automationexercise.com/')
    page.pause()
    page.locator('#accordion .panel-title').first.click()
```

## XPath

### Exemplo:

```
def test_locator_xpath(page):
    page.goto('https://automationexercise.com/login')
    page.pause()
    page.locator('//*[@id="form"]/div/div/div[1]/div/form/button').click()
```

### Explicação:

O método `page.locator()` permite usar seletores CSS e XPath para localizar elementos. O XPath é poderoso e permite selecionar elementos por hierarquia e atributos, mas pode ser frágil e difícil de manter, especialmente quando baseado em posições relativas. Use com cautela e prefira seletores acessíveis quando possível.

---

## Conclusão

- Priorize sempre IDs e seletores acessíveis ( `get_by_role()` , `get_by_label()` , etc.).
- Use `page.locator()` para casos em que os seletores acessíveis não atendem.
- Evite XPath complexos e longos para manter seus testes mais estáveis.

### Referência oficial:

- <https://playwright.dev/python/docs/locators#locate-by-css-or-xpath>
-

## 2 - Comandos Essenciais do Playwright com Python

---

### `page.goto()` – Acessar uma página

#### Objetivo:

Navegar o navegador para uma URL específica, controlando como e quando o carregamento é considerado completo.

#### O que faz:

Envia o navegador para uma URL, como se o usuário digitasse e desse *Enter*.

#### Como funciona:

- Por padrão espera o evento `"load"`.
- Pode-se mudar com `wait_until`:
  - `"load"` → espera o carregamento completo.
  - `"domcontentloaded"` → espera que o DOM esteja pronto.
  - `"networkidle"` → espera que a rede esteja inativa.

#### Exemplo:

```
page.goto("https://automationexercise.com/")
```

#### Exemplo com espera específica:

```
page.goto("https://automationexercise.com/", wait_until="networkidle")
```

#### Referência oficial:

-  <https://playwright.dev/python/docs/api/class-page#page-goto>

### `page.click()` – Clicar em um botão ou link

#### Objetivo:

Simular o clique do usuário em um elemento, respeitando as condições ideais de interação.

#### O que faz:

Clica em um elemento, com auto-wait para garantir que ele esteja pronto para receber o clique.

### Parâmetros importantes:

- `button` : "left" ou "right".
- `position` : coordenadas relativas dentro do elemento.
- `modifiers` : teclas como ["Shift"], ["Control"] .
- `timeout` : tempo máximo de espera.

### Exemplo:

```
page.get_by_role("link", name="Signup / Login").click()
```

### Exemplo completo:

```
def test_click(page):
    page.goto("https://automationexercise.com/")
    page.pause()
    page.get_by_role("link", name="Website for automation").click(button="right")
    page.get_by_role("link", name="Website for automation").click(position={
        "x": 10, "y": 10})
    page.get_by_role("link", name="Website for automation").click(position={
        "x": 0, "y": 0})
    page.get_by_role("link", name="(5) H&M").click(modifiers=["Ctrl"]) # (Alt|Control|ControlOrMeta|Meta|Shift)
```

### Referência oficial:

- <https://playwright.dev/python/docs/input#mouse-click>

## `page.fill()` – Preencher campos de texto

### Objetivo:

Inserir texto em campos (input/textarea) substituindo o conteúdo anterior de uma vez.

### O que faz:

Preenche o campo sem simular digitação letra a letra.

### Exemplo:

```
def test_fill(page):
    page.goto('https://automationexercise.com/login')
    page.pause()
    page.get_by_role("textbox", name="Name").fill('Jeferson', timeout=10000)
    page.locator("form").filter(has_text="Signup").get_by_placeholder("Email Address").fill('teste@teste.com')
    page.get_by_role("button", name="Signup").click()
```

### Referência oficial:

- <https://playwright.dev/python/docs/input#text-input>

## page.check() / uncheck() – Checkbox

### Objetivo:

Garantir que um checkbox esteja marcado ou desmarcado de forma determinística.

### Exemplo:

```
def test_check_uncheck(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_role("checkbox", name="Default checkbox").check()
    page.get_by_role("checkbox", name="Default checkbox").uncheck()
    page.pause()
```

### Boas práticas:

- Use `.check()` / `.uncheck()` em vez de `click()` para garantir o estado final esperado.

### Referência oficial:

- <https://playwright.dev/python/docs/input#checkboxes-and-radio-buttons>

## `page.select_option()` – Selecionar Opção

### Objetivo:

Selecionar uma ou mais opções em elementos `<select>` de forma explícita.

### Exemplo:

```
def test_select_option(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_label("Example select").select_option("2")
    page.get_by_label("Example multiple select").select_option(['3', '5'])
```

### Ou usando label diretamente:

```
page.select_option("select[name='country']", label="India")
```

### Referência oficial:

- 🔗 <https://playwright.dev/python/docs/input#select-options>

## `page.press()` – Pressionar teclas

### Objetivo:

Simular o pressionamento de teclas ou combinações (atalhos).

### Exemplo:

```
def test_press(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_role("textbox", name="Example textarea").fill('teste teste teste')
    page.get_by_role("textbox", name="Example textarea").press('Control+A')
    page.get_by_role("textbox", name="Example textarea").press('Control+C')
    page.get_by_placeholder("name@example.com").press('Control+V')
```

### Boas práticas:



- Use para navegação, atalhos e submit.
- Para digitação de texto visível, prefira `.type()`.

#### Referência oficial:

- <https://playwright.dev/python/docs/input#keys-and-shortcuts>

## `page.type()` – Digitar como um usuário real

#### Objetivo:

Digitar texto caracter por caracter, com controle de delay para simular digitação natural.

#### Exemplo:

```
def test_type(page):
    page.goto('https://bootswatch.com/default/')
    page.pause()
    page.get_by_role("textbox", name="Example textarea").type(
        'Lorem Ipsum is simply dummy text of the printing and typesetting ind
        ustry.', delay=150)
```

#### Quando usar:

- Quando o sistema reage a cada tecla digitada (ex: buscas em tempo real).

#### Referência oficial:

- <https://playwright.dev/python/docs/input#type-characters>

## `page.hover()` – Passar o mouse por cima

#### Objetivo:

Acionar elementos que aparecem ao passar o mouse (menus, overlays, etc.).

#### Exemplo:

```
def test_hover(page):
    page.goto('https://automationexercise.com')
    page.pause()
    page.locator('.single-products:visible').filter(has_text = 'Madame Top For
    Women').hover()
```

```
page.locator(
    "div:nth-child(9) > .product-image-wrapper > .single-products > .product-overlay > .overlay-content > .btn").click()
```

#### Referência oficial:

- <https://playwright.dev/python/docs/api/class-locator#locator-hover>

## page.dblclick() – Duplo clique

#### Objetivo:

Executar um duplo clique em um elemento.

#### Exemplo:

```
def test_dblclick(page):
    page.goto('https://automationexercise.com/login')
    page.pause()
    page.locator('.login-form h2').dblclick()
```

#### Referência oficial:

- <https://playwright.dev/python/docs/api/class-mouse#mouse-dblclick>

## expect() – Validações

#### Objetivo:

Fazer assertivas nos elementos da página para validar resultados esperados nos testes de forma automática e confiável.

#### Exemplo:

```
from playwright.sync_api import expect

def test_expect(page):
    page.goto('https://automationexercise.com')
    page.pause()
    page.locator('.single-products:visible').filter(has_text = 'Madame Top For Women').hover()
    page.locator(
```

```
"div:nth-child(9) > .product-image-wrapper > .single-products > .product-overlay > .overlay-content > .btn").click()
expect(page.locator('#cartModal')).to_contain_text('Your product has been added to cart.', timeout=10000)
expect(page.get_by_role('button', name='Continue Shopping')).to_be_visible()
expect(page.get_by_role('button', name='Continue Shopping')).to_be_enabled()
page.get_by_role('button', name='Continue Shopping').click()
expect(page.locator('#cartModal')).not_to_be_visible()
```

### Explicação:

O comando `expect()` é usado para validar estados e propriedades de elementos na página, como visibilidade, texto, habilitação, entre outros. Ele aguarda automaticamente até que a condição esperada seja verdadeira ou até o tempo de timeout acabar, evitando falhas por tempo insuficiente de espera. É fundamental para garantir que o fluxo do teste gerou o resultado esperado.

### Referência oficial

- <https://playwright.dev/python/docs/test-assertions#list-of-assertions>

### Resumo

- `goto` : acessa uma URL
- `click` : clica em qualquer elemento
- `fill` : preenche campos
- `check` / `unchecked` : checkbox
- `select_option` : selecionar valores em combo
- `press` : simular teclas
- `type` : simular digitação real
- `hover` : mouse sobre elemento
- `dblclick` : duplo clique
- `expect` : fazer validações

## 3 - Auto-wait no Playwright

### Objetivo:

Compreender o funcionamento do sistema de espera automática (auto-wait) do Playwright que torna os testes mais estáveis sem a necessidade de comandos manuais de espera.


### Explicação:

O Playwright **espera automaticamente** antes de executar ações (como `click()`, `fill()`, `hover()`) até que o elemento esteja em condições ideais para interação:

- Visível na tela
- Habilitado para interação
- Estável (não está mudando de posição ou sendo removido do DOM)
- Livre de animações que impeçam interação

Esse recurso elimina a necessidade de usar comandos como `time.sleep()` ou `wait_for_selector()`, tornando os testes mais confiáveis e rápidos para escrever.

### Referência oficial

-  [Documentação oficial – Actionability](#)