

Seleniumテスト完全ガイド - 新人エンジニア向け

対象読者: 新人エンジニア・テスト自動化初心者

目的: Seleniumを使ったWebテストの基礎から実践まで

所要時間: 約20分

目次

1. Seleniumとは
2. 環境構築
3. 基本的な要素取得方法
4. 実践的なテストコード
5. ページオブジェクトパターン
6. よくあるエラーと対処法

[](#)

1. Seleniumとは

定義

Seleniumは、Webブラウザを自動操作してUIテストを行うためのツールです。人間がブラウザで行う操作（クリック、入力、画面遷移など）を自動化できます。

目的

[手動テスト]

ブラウザを開く

↓

ログイン画面を開く

↓

IDとパスワードを入力

↓

ログインボタンをクリック

↓

結果を確認

↓

同じ操作を繰り返す... 🤖

[Seleniumテスト]

一度スクリプトを書けば



何度でも自動実行！ ✨

対応ブラウザ

- Chrome
- Firefox
- Edge
- Safari

✂ 2. 環境構築（Python版）

ステップ1: Seleniumをインストール

```
bash
pip install selenium
```

ステップ2: ChromeDriverを準備

方法A: 手動ダウンロード

1. ChromeDriver公式サイトにアクセス
2. 使用中のChromeバージョンに合ったドライバをダウンロード
3. PATHの通った場所に配置

方法B: 自動管理（推奨）

```
bash
pip install webdriver-manager
```

```
python
from selenium import webdriver
from webdriver_manager.chrome import ChromeDriverManager
from selenium.webdriver.chrome.service import Service

service = Service(ChromeDriverManager().install())
driver = webdriver.Chrome(service=service)
```

```
<a name="section3"></a>
```

3. 基本的な要素取得方法

HTML構造の理解

Webページは階層構造になっています：

```
html

<div id="main">
  <div class="section">
    <ul>
      <li><a href="/item1">商品1</a></li>
      <li><a href="/item2">商品2</a></li>
    </ul>
  </div>
</div>
```

要素の取得方法一覧

方法	使い方	説明
By.ID	driver.find_element(By.ID, "main")	ID属性で検索（最も確実）
By.NAME	driver.find_element(By.NAME, "username")	name属性で検索（フォーム要素）
By.CLASS_NAME	driver.find_element(By.CLASS_NAME, "section")	クラス名で検索
By.TAG_NAME	driver.find_element(By.TAG_NAME, "h1")	タグ名で検索
By.CSS_SELECTOR	driver.find_element(By.CSS_SELECTOR, "div#main")	CSSセレクタで検索
By.XPATH	driver.find_element(By.XPATH, "//*[@id='main']/div/ul/li[1]/a")	XPathで検索

XPathとCSSセレクタの違い

XPath（階層を明示的に指定）

```
python

# 親から順番にたどる
element = driver.find_element(By.XPATH, "//*[@id='main']/div/ul/li[1]/a")
```

階層のイメージ:

```
div[@id='main']    ← IDが'main'のdivを探す
├── div            ← その中のdiv
│   ├── ul        ← その中のul
│   │   ├── li[1] ← 最初のli
│   │   └── a      ← その中のリンク
```

CSSセレクタ（シンプルで読みやすい）

```
python
# スペースで階層を表現
element = driver.find_element(By.CSS_SELECTOR, "div#main ul li:first-child a")
```

記法の対応表:

目的	XPath	CSSセレクタ
ID指定	<code>//div[@id='main']</code>	<code>div#main</code>
クラス指定	<code>//div[@class='section']</code>	<code>div.section</code>
直下の子	<code>/ul/li</code>	<code>> ul > li</code>
最初の要素	<code>[1]</code>	<code>:first-child</code>

```
<a name="section4"> </a>
```

💡 4. 実践的なテストコード

基本のテストコード例

```
python
```

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

# ブラウザを起動
driver = webdriver.Chrome()

try:
    # ページにアクセス
    driver.get("https://example.com/login")

    # 要素が表示されるまで待機 (最大10秒)
    wait = WebDriverWait(driver, 10)

    # ユーザー名を入力
    username_field = wait.until(
        EC.presence_of_element_located((By.ID, "username"))
    )
    username_field.send_keys("test_user")

    # パスワードを入力
    password_field = driver.find_element(By.ID, "password")
    password_field.send_keys("test_password")

    # ログインボタンをクリック
    login_button = driver.find_element(By.ID, "login-button")
    login_button.click()

    # ログイン成功を確認
    success_message = wait.until(
        EC.presence_of_element_located((By.CLASS_NAME, "success"))
    )
    assert "ログインしました" in success_message.text

    print("✅ テスト成功")

finally:
    # ブラウザを閉じる
    driver.quit()
```

よく使う操作メソッド

python

```
# クリック
element.click()

# テキスト入力
element.send_keys("入力文字列")

# テキスト取得
text = element.text

# 属性値取得
value = element.get_attribute("value")

# 要素が有効かチェック
is_enabled = element.is_enabled()

# 要素が表示されているかチェック
is_displayed = element.is_displayed()

# スクリーンショット
driver.save_screenshot("screenshot.png")
```

iframeの扱い

```
python

# iframeに切り替え
iframe = driver.find_element(By.TAG_NAME, "iframe")
driver.switch_to.frame(iframe)

# iframe内の要素を操作
element_inside = driver.find_element(By.ID, "some-id")
element_inside.click()

# 元のページに戻る
driver.switch_to.default_content()
```


5. ページオブジェクトパターン

なぜページオブジェクトパターンを使うのか？

悪い例（ベタ書き）：

```
python
```

```
def test_login():
    driver.find_element(By.ID, "username").send_keys("user")
    driver.find_element(By.ID, "password").send_keys("pass")
    driver.find_element(By.ID, "login-button").click()

def test_another_login():
    driver.find_element(By.ID, "username").send_keys("admin")
    driver.find_element(By.ID, "password").send_keys("admin")
    driver.find_element(By.ID, "login-button").click()
```

問題点:

- 同じコードの繰り返し
- HTMLが変わったら全部修正が必要
- 読みにくい

ページオブジェクトパターンの実装

BasePage（基底クラス）

```
python

class BasePage:
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)
```

LoginPage（ログインページ）

```
python
```

```
from selenium.webdriver.common.by import By
from selenium.webdriver.support import expected_conditions as EC
```

```
class LoginPage(BasePage):
    # ロケータを定義
    USERNAME_FIELD = (By.ID, "username")
    PASSWORD_FIELD = (By.ID, "password")
    LOGIN_BUTTON = (By.ID, "login-button")

    def enter_username(self, username):
        element = self.wait.until(
            EC.presence_of_element_located(self.USERNAME_FIELD)
        )
        element.send_keys(username)
        return self

    def enter_password(self, password):
        element = self.driver.find_element(*self.PASSWORD_FIELD)
        element.send_keys(password)
        return self

    def click_login(self):
        element = self.driver.find_element(*self.LOGIN_BUTTON)
        element.click()
        return HomePage(self.driver)
```

テストコード

```
python
```



```
def test_login():
    driver = webdriver.Chrome()

    try:
        login_page = LoginPage(driver)
        login_page.driver.get("https://example.com/login")

        # メソッドチェーン
        home_page = (login_page
                     .enter_username("test_user")
                     .enter_password("test_password")
                     .click_login())

        assert "ホーム" in home_page.driver.title

    finally:
        driver.quit()
```

ページコンポーネントパターン

複数の製品がある一覧ページの場合：

```
python
```

```
class BaseComponent:
    def __init__(self, root_element):
        self.root = root_element

class Product(BaseComponent):
    def get_name(self):
        return self.root.find_element(
            By.CLASS_NAME, "product-name"
        ).text

    def get_price(self):
        price_text = self.root.find_element(
            By.CLASS_NAME, "product-price"
        ).text
        return float(price_text.replace("$", ""))

    def add_to_cart(self):
        button = self.root.find_element(
            By.CLASS_NAME, "add-to-cart-button"
        )
        button.click()

class ProductsPage(BasePage):
    def get_products(self):
        product_elements = self.driver.find_elements(
            By.CLASS_NAME, "product-item"
        )
        return [Product(element) for element in product_elements]

    def get_product_by_name(self, name):
        products = self.get_products()
        return next(p for p in products if p.get_name() == name)

# テストコード
def test_product_prices():
    products_page = ProductsPage(driver)
    products_page.driver.get("https://example.com/products")

    backpack = products_page.get_product_by_name("Backpack")
    assert backpack.get_price() == 29.99
```

⚠ 6. よくあるエラーと対処法

エラー1: NoSuchElementException

原因: 要素が見つからない

対処法:

```
python

# 待機を追加
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

wait = WebDriverWait(driver, 10)
element = wait.until(
    EC.presence_of_element_located((By.ID, "target-id"))
)
```

エラー2: ElementNotInteractableException

原因: 要素をクリックできない（隠れている、無効など）

対処法:

```
python

# 要素が有効になるまで待機
element = wait.until(
    EC.element_to_be_clickable((By.ID, "button-id"))
)
element.click()
```

エラー3: StaleElementReferenceException

原因: ページ更新後に古い要素を参照している

対処法:

```
# 要素を再取得する
def safe_click(driver, locator):
    for _ in range(3): # 3回までリトライ
        try:
            element = driver.find_element(*locator)
            element.click()
            break
        except StaleElementReferenceException:
            continue
```

7. テスト観点のチェックリスト

画面・操作系

- ☐ 画面遷移が正しいか
- ☐ ボタンの活性/非活性制御が正しいか
- ☐ 入力項目の初期値が正しいか
- ☐ エラーメッセージが表示されるか

データ取得・表示系

- ☐ サーバリクエストが1回だけか
- ☐ 取得結果が設計通りか
- ☐ 表示項目の編集（日付、金額など）が正しいか

DB更新系

- ☐ 登録・更新・削除が正しく行われるか
- ☐ トランザクションが正しく動作するか

8. まとめ

新人エンジニアが押さえるべきポイント

1. 基本の要素取得をマスターする
 - ID、NAME、CLASS_NAMEから始める
 - 慣れたらXPathやCSSセレクタに挑戦
2. 待機処理を忘れずに
 - `WebDriverWait`を使って安定したテストを書く
3. ページオブジェクトパターンを活用
 - コードの再利用性が高まる

- メンテナンスが楽になる

4. エラーハンドリングを丁寧に

- try-finallyで必ずブラウザを閉じる
- リトライ処理を適切に使う

学習の進め方

Step 1: 基本操作を習得

↓

Step 2: 実際のページで練習

↓

Step 3: ページオブジェクトパターンに挑戦

↓

Step 4: CI/CDに組み込む


9. 参考資料


公式ドキュメント

- [Selenium公式サイト](#)
- [Selenium with Python](#)

おすすめツール

- pytest: Pythonのテストフレームワーク
- Allure: テストレポート生成ツール
- BrowserStack: クラウドでのブラウザテスト

 **作成日:** 2025年9月30日

 **対象:** 新人エンジニア

 **タグ:** Selenium, テスト自動化, Python, WebDriver, ページオブジェクト