5-best-practices-and-optimization

July 28, 2025

0.1 S7.5. Best Practices And Optimization

0.1.1 Intro

- This section provides **strategies** for writing **maintainable, efficient,** and **robust** Selenium scripts.
- By following these best practices, you will:
  - Improve the **performance** of your automation or web scraping projects.
  - Make the codebase **easier to debug, extend, and maintain**.
  - Build more **scalable and reliable** Selenium workflows for both development and production use.

0.1.2 1. Write Maintainable Code:

Page Object Model (POM):

- Use the **Page Object Model** design pattern to separate element locators and page interactions.
- Each **web page is represented by a class**.
- Web elements are defined as variables and actions (e.g., click, input) as methods.
- Benefits:
  - Better **code reusability**
  - Cleaner **code readability**
  - Easier **maintenance**

```
# Example (POM structure)
class LoginPage:
  def __init__(self, driver):
      self.driver = driver
      self.username_field =(By.ID, "username")

  def login(self, username, password):
      self.username_field.send_keys(username)
      # ... other interactions
```

- **presentations for POM:** | Concept | Represented As | | ————-| ———— | | **Page** | `Class` | | **Elements** | `Attributes` | | **Interactions** | `Methods` |
  - **In summary:**

* Each **webpage = Python class**
* Each **element (input, button, etc.) = attribute** (like `self.login_button`)
* Each **action (click, type, submit) = method** (like `def login()`)

Descriptive Variable Names

- Avoid overly generic names like `element1`, `button1`
- Use descriptive names link `submit_button`, `search_input`, `login_link` to define web elements

[3]:
```
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException


edge_driver_path = r"D:\WEB SCRAPING\S7.Selenium\edgedriver_win64\msedgedriver.
 ↪exe"
service = Service(edge_driver_path)
driver = webdriver.Edge(service=service)
url = "https://github.com/login"
driver.get(url)
print("  URL Successfully loaded!")
driver.maximize_window()

class LoginPage():
    def __init__(self, driver):
        self.driver = driver
        self.username = (By.ID, "login_field")
        self.password = (By.ID, "password")
        self.login_button = (By.NAME, "commit")

    def login(self, username, password):
        # For ex. self.username is one tuple, not two values.
        # To unpack that tuple into two separate arguments, you use the *␣
 ↪operator:
        self.driver.find_element(*self.username).send_keys(username)
        self.driver.find_element(*self.password).send_keys(password)
        self.driver.find_element(*self.login_button).click()

# constructor stores driver inside the LoginPage object
# Pass driver to Page class
login_page = LoginPage(driver)
login_page.login('Akashpagi', "9211@hfhf")

driver.quit()
```

```
URL Successfully loaded!
```

0.1.3 3. Enhance Performance in Selenium

3.1. Use Appropriate Waits:

- **1. Avoid `time.sleep()` Overuse**
  - `time.sleep()` introduces unnecessary delays.
  - It always waits the full duration — even if the element loads earlier.
- **2. Prefer Explicit and Implicit Waits**
  - **Explicit Waits:** (recommended): Waits for a specific condition.
  - **Ex:** "'python WebDriverWait(driver, 10).until(EC.presence_of_element_located((By.ID, "username")))
  - **Implicit Waits:** Set once and applies globally.
  - **Ex:** "'python driver.implicitly_wait(10)
- **3. Explicit Waits = Better Control**
  - Use when dealing with dynamic content like alerts, AJAX, or modals.

0.1.4 3.2. Optimize Locator Strategies

1. Use Faster Locators First - `ID` → Fastest and most reliable - `NAME` → Also fast and readable - `XPath` → Slower and more brittle, avoid unless necessary

2. Avoid Absolute XPath - Fragile to DOM changes; prefer relative XPath or CSS Selectors when needed

0.1.5 3.3. Reuse Browser Sessions

1. Avoid Launching New Browser for Every Script - Reuse browser sessions for multiple actions or test cases

2. Use Headless Mode for Speed - Ideal for scraping or automation without UI - **Ex:** "'python from selenium.webdriver.edge.options import Options options = Options() options.add_argument('–headless') driver = webdriver.Edge(service=service, options=options)

0.1.6 4. Robustness and Error Handling

4.1. Try-Except Blocks * Wrap code for critical interactions within `try-except` blocks to handle unexpected failures * Catch **specific exceptions** (like `TimeoutException`, `NoSuchElementException`) when possible * Helps avoid script crashes and provides controlled error messages

```
from selenium.common.exceptions import TimeoutException, NoSuchElementException
try:
    element = driver.find_element(By.ID, "example")
    element.click()
except NoSuchElementException:
    print(" Element not found.")
except TimeoutException:
    print(" Operation timed out.")
```

```
except Exception as e:
    print(f"  An unexpected error occurred: {e}")
```

4.2. Release Resources - Always close the browser session using:

```
driver.quit()
```

- Frees up memory and system resources
- Can also be placed inside a `finally` block to ensure it runs even after exceptions

0.1.7 5. Logging and Debugging

5.1. Logging Instead of Print Statements * Avoid using `print()` for debugging * Use the `logging` module for better control, level separation, and log file storage

```
import logging

logging.basicConfig(level=logging.INFO, filename="test.log",
                    format="%(asctime)s - %(levelname)s - %(message)s")

logging.info("Script started successfully.")
logging.error("Something went wrong.")
```

4.2. Capture Screenshots on Failure * Use `save_screenshot()` to capture the browser view when an error occurs

```
try:
    # test action
    driver.find_element(By.ID, "wrong_id").click()
except Exception as e:
    driver.save_screenshot("error_screenshot.png")
    print(f"  Error captured: {e}")
```

4.3. Use Developer Tools in Browser * Press `F12` or right-click → **Inspect** * Helps identify: * Element locators (ID, class, XPath, etc.) * Dynamic content or AJAX elements * JavaScript-driven UI changes * Essential for writing reliable locators

0.1.8 6. Security Considerations

6.1. Secure Sensitive Data: * Avoid exposing credentials directly in scripts * Use encrypted files or environment variables to store usernames, passwords, tokens

6.2. Respect Website Policies: * Be ethical in your automation and scraping practices * Check the website's `robots.txt` and Terms of Service before scraping * Avoid actions that could overload or harm the website

[ ]:
```
from selenium import webdriver
from selenium.webdriver.edge.service import Service
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
from selenium.common.exceptions import TimeoutException
```

```
edge_driver_path = r"D:\WEB SCRAPING\S7.Selenium\edgedriver_win64\msedgedriver.
 ↪exe"
service = Service(edge_driver_path)
driver = webdriver.Edge(service=service)
url = "https://github.com/login"
driver.get(url)
print(" URL Successfully loaded!")
driver.maximize_window()

class LoginPage():
    def __init__(self, driver):
        self.driver = driver
        self.wait = WebDriverWait(driver, 10)
        self.username = (By.ID, "login_field")
        self.password = (By.ID, "password")
        self.login_button = (By.NAME, "commit")

    def login(self, username, password):
        # For ex. self.username is one tuple, not two values.
        # To unpack that tuple into two separate arguments, you use the *␣
 ↪operator:
        self.driver.find_element(*self.username).send_keys(username)
        self.driver.find_element(*self.password).send_keys(password)
        self.driver.find_element(*self.login_button).click()

# constructor stores driver inside the LoginPage object
# Pass driver to Page class
login_page = LoginPage(driver)
login_page.login('Akashpagi', "9211@hfhf")

driver.quit()
```

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]:

[ ]: