Phase 2, Topic 1: What is Selenium?

Selenium is an open-source automation testing framework primarily used for automating web applications. It allows you to programmatically control web browsers and simulate user interactions like clicking buttons, filling forms, and navigating between pages.

Key Points about Selenium:

What Selenium Does:

- Automates web browser interactions
- Simulates user actions (clicks, typing, scrolling)
- Validates web page content and behavior
- Supports cross-browser testing

Use Cases:

- **Functional Testing**: Verify that web features work as expected
- **Regression Testing**: Ensure new code doesn't break existing functionality
- **Data Extraction**: Scrape information from websites
- **Repetitive Task Automation**: Automate boring, repetitive web tasks

Browsers Supported:

- Chrome (most popular)
- Firefox
- Edge
- Safari
- Opera

Selenium Components:

- **Selenium WebDriver**: The main tool we'll use - controls browsers directly
- **Selenium IDE**: Browser extension for record/playback (beginner-friendly)
- **Selenium Grid**: For running tests across multiple machines/browsers

Why Selenium with Python?

- Python's simple syntax makes automation scripts easy to read and maintain
- Large community and extensive documentation
- Great integration with testing frameworks like pytest
- Rich ecosystem of supporting libraries

Real-World Example:

Instead of manually testing a login form 50 times across different browsers, you can write a Selenium script that:

1. Opens the browser
2. Navigates to the login page

3. Enters username and password
4. Clicks login button
5. Verifies successful login
6. Repeats across Chrome, Firefox, Edge automatically

Phase 2, Topic 2: Setup

Selenium Setup Guide

Prerequisites

- Python 3.7+ installed on your system
- Basic command line knowledge

Step 1: Install Selenium

Open your terminal/command prompt and run:

```
pip install selenium
```

To verify installation:

```
pip show selenium
```

Step 2: Install WebDriver Manager (Recommended 2025 Approach)

Instead of manually downloading browser drivers, use webdriver-manager:

```
pip install webdriver-manager
```

This tool automatically downloads and manages browser drivers for you!

Step 3: Alternative - Manual Driver Installation

If you prefer manual setup:

For Chrome:

1. Check your Chrome version: `chrome://version/`
2. Download ChromeDriver from: https://chromedriver.chromium.org/
3. Extract and place in your PATH or project folder

For Firefox:

1. Download geckodriver from: https://github.com/mozilla/geckodriver/releases
2. Extract and place in your PATH or project folder

For Edge:

1. Download EdgeDriver from:
   https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/

Step 4: Verify Setup

Create a test file `test_setup.py`:

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

# Using webdriver-manager (recommended)
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

# Test: Open Google
driver.get("https://www.google.com")
print("Page title:", driver.title)

# Close browser
driver.quit()
print("Setup successful!")
```

Run the test:

```
python test_setup.py
```

Common Setup Issues & Solutions

Issue 1: "chromedriver not found"

Solution: Use webdriver-manager or add driver to PATH

Issue 2: Chrome version mismatch

Solution: Update Chrome browser or use webdriver-manager for auto-matching

Issue 3: Permission denied (Mac/Linux)

Solution: Make driver executable:

```
chmod +x chromedriver
```

Project Structure Setup

Create a clean project structure:

```
selenium_project/
├── drivers/          # Manual drivers (if not using webdriver-manager)
├── tests/           # Test files
├── pages/           # Page Object Model files (later)
├── data/            # Test data
├── reports/         # Test reports
└── requirements.txt # Dependencies
```

requirements.txt

```
selenium==4.15.0
webdriver-manager==4.0.1
pytest==7.4.3
pytest-html==4.1.1
```

Install all dependencies:

```
pip install -r requirements.txt
```

2025 Best Practices

1. **Always use webdriver-manager** - No more manual driver downloads!
2. **Use Service class** - Modern way to specify driver path
3. **Set browser options** - For headless mode, window size, etc.
4. **Use context managers** - For automatic cleanup (advanced topic)

Great! Now you have Selenium properly installed. Let me show you a quick practical example to test your setup:

first_selenium_test.py

Your first Selenium automation script

from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from webdriver_manager.chrome import ChromeDriverManager

import time

def test_google_search():

"""Simple test to verify Selenium setup works"""

Setup Chrome driver using webdriver-manager

print("Setting up Chrome driver...")

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

try:

Navigate to Google

print("Opening Google...")

driver.get("https://www.google.com")

```
        # Verify we're on the right page
        print(f"Page title: {driver.title}")
        assert "Google" in driver.title

        # Wait a moment to see the page
        time.sleep(2)

        print("✅ Setup successful! Browser opened and navigated to Google.")

    except Exception as e:
        print(f"❌ Error occurred: {e}")

    finally:
        # Always close the browser
        print("Closing browser...")
        driver.quit()

if __name__ == "__main__":
    test_google_search()
```

Key Setup Points:

1. **webdriver-manager** is the modern approach - it automatically handles driver downloads and version matching
2. **Service class** is the current best practice for specifying driver paths
3. Always use `driver.quit()` to properly close the browser and free up resources

Try This:

1. Copy the setup guide and install the required packages
2. Run the first Selenium test to verify everything works
3. You should see Chrome browser open, navigate to Google, and then close automatically

Any issues with the setup? Common problems are usually related to Python PATH, Chrome version mismatches, or permissions. The webdriver-manager approach eliminates most of these issues.

Ready for the next topic: **"Launching Browsers"**? We'll explore different ways to start browsers, set browser options, and handle multiple browser types!

Phase 2, Topic 3: Launching Browsers

Now that Selenium is set up, let's learn different ways to launch and configure browsers. This is where your automation journey really begins!

```
# browser_launching_examples.py

# Complete guide to launching different browsers with Selenium


from selenium import webdriver

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options as ChromeOptions

from selenium.webdriver.firefox.options import Options as FirefoxOptions

from selenium.webdriver.edge.options import Options as EdgeOptions

from webdriver_manager.chrome import ChromeDriverManager

from webdriver_manager.firefox import GeckoDriverManager

from webdriver_manager.microsoft import EdgeChromiumDriverManager

import time


def launch_basic_chrome():

    """Basic Chrome browser launch"""

    print("🚀 Launching Basic Chrome...")


    # Simple Chrome launch

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


    # Navigate to a website

    driver.get("https://www.example.com")

    print(f"Page title: {driver.title}")


    time.sleep(3)

    driver.quit()

    print("✅ Chrome closed\n")


def launch_chrome_with_options():
```

```
    """Chrome with custom options"""
    print("🚀 Launching Chrome with Custom Options...")

    # Create Chrome options
    chrome_options = ChromeOptions()

    # Add various options
    chrome_options.add_argument("--window-size=1920,1080")  # Set window size
    chrome_options.add_argument("--disable-notifications")  # Disable notifications
    chrome_options.add_argument("--disable-popup-blocking") # Disable popup blocking
    chrome_options.add_argument("--start-maximized")       # Start maximized

    # Optional: Run in headless mode (no GUI)
    # chrome_options.add_argument("--headless")

    # Launch with options
    driver = webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=chrome_options
    )

    driver.get("https://www.python.org")
    print(f"Page title: {driver.title}")
    print(f"Window size: {driver.get_window_size()}")

    time.sleep(3)
    driver.quit()
    print("✅ Chrome with options closed\n")

def launch_headless_chrome():
    """Chrome in headless mode (no visible window)"""
    print("🚀 Launching Headless Chrome...")
```

```
    chrome_options = ChromeOptions()
    chrome_options.add_argument("--headless")  # Run without GUI
    chrome_options.add_argument("--no-sandbox")
    chrome_options.add_argument("--disable-dev-shm-usage")

    driver = webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=chrome_options
    )

    driver.get("https://httpbin.org/ip")
    print(f"Headless page title: {driver.title}")
    print("✅ Headless Chrome test completed\n")

    driver.quit()

def launch_firefox():
    """Firefox browser launch"""
    print("🚀 Launching Firefox...")

    # Firefox options
    firefox_options = FirefoxOptions()
    # firefox_options.add_argument("--headless")  # Uncomment for headless

    driver = webdriver.Firefox(
        service=Service(GeckoDriverManager().install()),
        options=firefox_options
    )

    driver.get("https://www.mozilla.org")
    print(f"Firefox page title: {driver.title}")
```

```
    time.sleep(3)
    driver.quit()
    print("✅ Firefox closed\n")

def launch_edge():
    """Microsoft Edge browser launch"""
    print("🚀 Launching Microsoft Edge...")

    edge_options = EdgeOptions()
    edge_options.add_argument("--start-maximized")

    driver = webdriver.Edge(
        service=Service(EdgeChromiumDriverManager().install()),
        options=edge_options
    )

    driver.get("https://www.microsoft.com")
    print(f"Edge page title: {driver.title}")

    time.sleep(3)
    driver.quit()
    print("✅ Edge closed\n")

def browser_session_management():
    """Proper browser session management"""
    print("🚀 Demonstrating Browser Session Management...")

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

    try:
        # Navigate to website
```

```
        driver.get("https://www.github.com")
        print(f"Initial page: {driver.title}")

        # Get current URL
        print(f"Current URL: {driver.current_url}")

        # Navigate to another page
        driver.get("https://www.stackoverflow.com")
        print(f"New page: {driver.title}")

        # Browser navigation
        driver.back()  # Go back
        time.sleep(2)
        print(f"After back: {driver.title}")

        driver.forward()  # Go forward
        time.sleep(2)
        print(f"After forward: {driver.title}")

        driver.refresh()  # Refresh page
        time.sleep(2)
        print("Page refreshed")

    except Exception as e:
        print(f"Error: {e}")

    finally:
        # Always close browser properly
        driver.quit()
        print("✓ Browser session closed properly\n")

def multiple_browser_windows():
```

```
"""Working with multiple browser windows"""
print("🚀 Working with Multiple Windows...")

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

try:
    # Open first window
    driver.get("https://www.google.com")
    original_window = driver.current_window_handle
    print(f"Original window: {driver.title}")

    # Open new window/tab
    driver.execute_script("window.open('https://www.youtube.com', '_blank')")

    # Switch to new window
    for handle in driver.window_handles:
        if handle != original_window:
            driver.switch_to.window(handle)
            break

    print(f"New window: {driver.title}")
    time.sleep(2)

    # Switch back to original window
    driver.switch_to.window(original_window)
    print(f"Back to original: {driver.title}")

    # Close all windows
    for handle in driver.window_handles:
        driver.switch_to.window(handle)
        driver.close()
```

```
    except Exception as e:
        print(f"Error: {e}")

    finally:
        try:
            driver.quit()
        except:
            pass
        print("✓ All windows closed\n")

# Main execution
if __name__ == "__main__":
    print("=== Browser Launching Examples ===\n")

    # Try different browser launches
    launch_basic_chrome()
    launch_chrome_with_options()
    launch_headless_chrome()

    # Uncomment these if you have Firefox/Edge installed
    # launch_firefox()
    # launch_edge()

    browser_session_management()
    multiple_browser_windows()

    print(" All browser launching examples completed!")
```

Now let me create a practical reference guide for browser options:

Browser Options Reference Guide

Chrome Options (Most Common)

Window & Display Options

```
chrome_options = ChromeOptions()
chrome_options.add_argument("--start-maximized")           # Start maximized
chrome_options.add_argument("--window-size=1920,1080")     # Set specific size
chrome_options.add_argument("--headless")                  # Run without GUI
chrome_options.add_argument("--disable-gpu")               # Disable GPU (for
headless)
```

Security & Privacy Options

```
chrome_options.add_argument("--no-sandbox")                # Disable sandbox
(for Docker)
chrome_options.add_argument("--disable-dev-shm-usage")     # Overcome limited
resource problems
chrome_options.add_argument("--disable-notifications")     # Block notifications
chrome_options.add_argument("--disable-popup-blocking")    # Allow popups
chrome_options.add_argument("--incognito")                 # Start in incognito
mode
```

Performance Options

```
chrome_options.add_argument("--disable-extensions")        # Disable extensions
chrome_options.add_argument("--disable-plugins")           # Disable plugins
chrome_options.add_argument("--disable-images")            # Don't load images
(faster)
chrome_options.add_argument("--disable-javascript")        # Disable JavaScript
```

Automation-Friendly Options

```
chrome_options.add_argument("--disable-blink-features=AutomationControlled")  #
Hide automation
chrome_options.add_experimental_option("excludeSwitches", ["enable-automation"])
chrome_options.add_experimental_option('useAutomationExtension', False)
```

Firefox Options

```
firefox_options = FirefoxOptions()
firefox_options.add_argument("--headless")                 # Headless mode
firefox_options.add_argument("--width=1920")               # Set width
firefox_options.add_argument("--height=1080")              # Set height

# Set preferences
firefox_options.set_preference("dom.webnotifications.enabled", False)  # Disable
notifications
firefox_options.set_preference("media.volume_scale", "0.0")            # Mute
audio
```

Edge Options

```
edge_options = EdgeOptions()
edge_options.add_argument("--headless")                    # Headless mode
```

```
edge_options.add_argument("--start-maximized")             # Start maximized
edge_options.add_argument("--disable-notifications")       # Disable
notifications
```

Common Browser Patterns

1. Basic Browser Launch

```
def get_driver(browser="chrome", headless=False):
    if browser.lower() == "chrome":
        options = ChromeOptions()
        if headless:
            options.add_argument("--headless")
        return
webdriver.Chrome(service=Service(ChromeDriverManager().install()),
options=options)
    elif browser.lower() == "firefox":
        options = FirefoxOptions()
        if headless:
            options.add_argument("--headless")
        return
webdriver.Firefox(service=Service(GeckoDriverManager().install()),
options=options)
```

2. Production-Ready Browser Setup

```
def get_production_driver():
    options = ChromeOptions()
    options.add_argument("--headless")
    options.add_argument("--no-sandbox")
    options.add_argument("--disable-dev-shm-usage")
    options.add_argument("--disable-gpu")
    options.add_argument("--window-size=1920,1080")
    options.add_argument("--disable-notifications")

    return webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=options
    )
```

3. Context Manager Pattern (Advanced)

```
from contextlib import contextmanager

@contextmanager
def browser_session(browser="chrome", headless=False):
    driver = get_driver(browser, headless)
    try:
        yield driver
    finally:
        driver.quit()

# Usage:
# with browser_session("chrome", headless=True) as driver:
#     driver.get("https://example.com")
#     # Browser automatically closes after this block
```

Browser Comparison

Feature	Chrome	Firefox	Edge	Safari
Speed	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐
Stability	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐
Options	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐
Documentation	⭐⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐⭐	⭐⭐⭐

Recommendation: Start with Chrome for learning, as it has the best documentation and community support.

Best Practices

1. **Always use webdriver-manager** - Automatic driver management
2. **Set explicit window size** - Consistent behavior across environments
3. **Use headless mode for CI/CD** - Faster execution in production
4. **Implement proper cleanup** - Always call `driver.quit()`
5. **Handle exceptions** - Use try/finally blocks
6. **Consider browser options** - Disable unnecessary features for faster execution

Key Takeaways from Browser Launching:

Essential Concepts:

1. **webdriver-manager** automatically handles driver downloads and versions
2. **Browser options** let you customize browser behavior (headless, window size, etc.)
3. **Proper cleanup** with `driver.quit()` is crucial to prevent memory leaks
4. **Context managers** provide elegant resource management

Most Important Methods:

- `driver.get(url)` - Navigate to a webpage
- `driver.quit()` - Close browser and end session
- `driver.close()` - Close current window only
- `driver.back()`, `driver.forward()`, `driver.refresh()` - Navigation

Pro Tips:

- Use **headless mode** for faster execution in production/CI
- **Start maximized** for consistent element visibility
- **Disable notifications** to avoid interruptions during automation

Try This Exercise:

Run the browser launching examples and observe:

1. How different options change browser behavior
2. The difference between `close()` and `quit()`
3. How headless mode runs without visible windows

Ready for the next topic: **"Locating Elements"**? This is where we learn to find and identify specific parts of web pages - the foundation of all web automation!

Phase 2, Topic 4: Locating Elements

This is one of the most crucial skills in Selenium! Learning to find elements on web pages is the foundation of all automation. Let's explore all the different ways to locate elements.

```
# element_locating_examples.py

# Complete guide to locating elements in Selenium


from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.support.ui import WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.common.exceptions import NoSuchElementException, TimeoutException

from webdriver_manager.chrome import ChromeDriverManager

import time


def setup_driver():

    """Setup Chrome driver for examples"""

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

    driver.maximize_window()

    return driver


def locate_by_id_examples():

    """Locating elements by ID - Most reliable method"""

    print("🔍 Locating Elements by ID")


    driver = setup_driver()


    try:

        # Navigate to a form page

        driver.get("https://the-internet.herokuapp.com/login")


        # Find element by ID (most reliable)
```

```
        username_field = driver.find_element(By.ID, "username")
        password_field = driver.find_element(By.ID, "password")

        print(f"✓ Found username field: {username_field.tag_name}")
        print(f"✓ Found password field: {password_field.tag_name}")

        # You can interact with found elements
        username_field.send_keys("testuser")
        password_field.send_keys("testpass")

        time.sleep(2)

    except NoSuchElementException as e:
        print(f"✗ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def locate_by_name_examples():
    """Locating elements by Name attribute"""
    print("🔍 Locating Elements by Name")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Find by name attribute
        username = driver.find_element(By.NAME, "username")
        password = driver.find_element(By.NAME, "password")
```

```
        print(f"✓ Found by name - Username: {username.get_attribute('name')}")
        print(f"✓ Found by name - Password: {password.get_attribute('name')}")

    except NoSuchElementException as e:
        print(f"✗ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def locate_by_class_name_examples():
    """Locating elements by Class Name"""
    print("🔍 Locating Elements by Class Name")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Find by class name
        login_button = driver.find_element(By.CLASS_NAME, "radius")
        print(f"✓ Found button by class: {login_button.text}")

        # Find multiple elements with same class
        elements_with_radius = driver.find_elements(By.CLASS_NAME, "radius")
        print(f"✓ Found {len(elements_with_radius)} elements with 'radius' class")

    except NoSuchElementException as e:
        print(f"✗ Element not found: {e}")

    finally:
        driver.quit()
```

```
        print("Browser closed\n")

def locate_by_tag_name_examples():
    """Locating elements by HTML Tag Name"""
    print("🔍 Locating Elements by Tag Name")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com")

        # Find by tag name
        all_links = driver.find_elements(By.TAG_NAME, "a")
        print(f"✅ Found {len(all_links)} links on the page")

        # Print first 5 link texts
        for i, link in enumerate(all_links[:5]):
            print(f"   Link {i+1}: {link.text}")

        # Find all input fields
        input_fields = driver.find_elements(By.TAG_NAME, "input")
        print(f"✅ Found {len(input_fields)} input fields")

    except NoSuchElementException as e:
        print(f"❌ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def locate_by_xpath_examples():
    """Locating elements by XPath - Very powerful but complex"""
```

```
    print("🔍 Locating Elements by XPath")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Absolute XPath (not recommended - fragile)
        # username_abs = driver.find_element(By.XPATH,
"/html/body/div[2]/div/div/form/div[1]/div/input")

        # Relative XPath (recommended)
        username_rel = driver.find_element(By.XPATH, "//input[@id='username']")
        password_rel = driver.find_element(By.XPATH, "//input[@id='password']")

        print("✅ Found elements using relative XPath")

        # XPath with text
        login_button = driver.find_element(By.XPATH, "//button[text()=' Login']")
        print(f"✅ Found button by text: {login_button.text.strip()}")

        # XPath with contains
        flash_message = driver.find_element(By.XPATH, "//div[contains(@class, 'flash')]")
        print(f"✅ Found flash message div")

        # XPath with multiple conditions
        username_multi = driver.find_element(By.XPATH, "//input[@type='text' and
@name='username']")
        print("✅ Found username with multiple conditions")

        # XPath parent/child navigation
        form_element = driver.find_element(By.XPATH,
"//input[@id='username']/parent::div/parent::form")
```

```
        print(f"✓ Found parent form element: {form_element.tag_name}")

    except NoSuchElementException as e:
        print(f"✗ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def locate_by_css_selector_examples():
    """Locating elements by CSS Selector - Fast and flexible"""
    print("🔍 Locating Elements by CSS Selector")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # CSS by ID
        username_css = driver.find_element(By.CSS_SELECTOR, "#username")
        password_css = driver.find_element(By.CSS_SELECTOR, "#password")

        print("✓ Found elements using CSS selector with ID")

        # CSS by class
        button_css = driver.find_element(By.CSS_SELECTOR, ".radius")
        print(f"✓ Found button by class: {button_css.text}")

        # CSS by attribute
        username_attr = driver.find_element(By.CSS_SELECTOR, "input[name='username']")
        print("✓ Found username by attribute")
```

```
        # CSS with multiple classes
        # elements = driver.find_elements(By.CSS_SELECTOR, ".class1.class2")

        # CSS parent > child
        form_input = driver.find_element(By.CSS_SELECTOR, "form > div > div > input")
        print("✅ Found form input using parent-child CSS")

        # CSS with pseudo-selectors
        first_input = driver.find_element(By.CSS_SELECTOR, "input:first-of-type")
        print(f"✅ Found first input: {first_input.get_attribute('name')}")

    except NoSuchElementException as e:
        print(f"❌ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def locate_by_link_text_examples():
    """Locating links by their text content"""
    print("🔍 Locating Elements by Link Text")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com")

        # Find by exact link text
        ab_testing_link = driver.find_element(By.LINK_TEXT, "A/B Testing")
        print(f"✅ Found link by exact text: {ab_testing_link.text}")

        # Find by partial link text
```

```
        checkboxes_link = driver.find_element(By.PARTIAL_LINK_TEXT, "Checkboxes")
        print(f"✅ Found link by partial text: {checkboxes_link.text}")

        # Click the link
        ab_testing_link.click()
        time.sleep(2)

        print(f"✅ Navigated to: {driver.current_url}")

    except NoSuchElementException as e:
        print(f"❌ Element not found: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def advanced_element_finding():
    """Advanced element finding techniques"""
    print("🔍 Advanced Element Finding Techniques")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/checkboxes")

        # Find multiple elements
        checkboxes = driver.find_elements(By.CSS_SELECTOR, "input[type='checkbox']")
        print(f"✅ Found {len(checkboxes)} checkboxes")

        # Check status of each checkbox
        for i, checkbox in enumerate(checkboxes, 1):
            is_checked = checkbox.is_selected()
```

```
            print(f"   Checkbox {i}: {'Checked' if is_checked else 'Unchecked'}")

        # Wait for element to be present (we'll cover waits more later)
        wait = WebDriverWait(driver, 10)
        checkbox_element = wait.until(EC.presence_of_element_located((By.CSS_SELECTOR, "input[type='checkbox']")))
        print("✅ Found checkbox using explicit wait")

        # Check if element exists without throwing exception
        def element_exists(by, value):
            try:
                driver.find_element(by, value)
                return True
            except NoSuchElementException:
                return False

        exists = element_exists(By.ID, "nonexistent")
        print(f"✅ Non-existent element check: {exists}")

    except (NoSuchElementException, TimeoutException) as e:
        print(f"❌ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def element_locator_best_practices():
    """Demonstrate best practices for element location"""
    print("🔍 Element Locator Best Practices")

    driver = setup_driver()
```

```
try:
    driver.get("https://the-internet.herokuapp.com/login")

    # Best Practice 1: Use most stable locator
    # Priority: ID > Name > CSS > XPath

    # Best Practice 2: Create reusable locator functions
    def find_element_safely(by, value, timeout=10):
        try:
            wait = WebDriverWait(driver, timeout)
            return wait.until(EC.presence_of_element_located((by, value)))
        except TimeoutException:
            print(f"✗ Element not found: {by}='{value}'")
            return None

    username = find_element_safely(By.ID, "username")
    if username:
        print("✓ Found username safely")
        username.send_keys("admin")

    # Best Practice 3: Use data-testid attributes (if available)
    # <input data-testid="username-input">
    # username = driver.find_element(By.CSS_SELECTOR, "[data-testid='username-input']")

    # Best Practice 4: Combine multiple strategies
    def find_login_button():
        strategies = [
            (By.ID, "login-button"),
            (By.CSS_SELECTOR, "button[type='submit']"),
            (By.XPATH, "//button[contains(text(), 'Login')]"),
            (By.CSS_SELECTOR, ".radius")
        ]
```

```
            for by, value in strategies:
                try:
                    return driver.find_element(by, value)
                except NoSuchElementException:
                    continue
            return None

        login_btn = find_login_button()
        if login_btn:
            print(f"✓ Found login button: {login_btn.text}")

    except Exception as e:
        print(f"✗ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

# Main execution
if __name__ == "__main__":
    print("=== Element Locating Examples ===\n")

    locate_by_id_examples()
    locate_by_name_examples()
    locate_by_class_name_examples()
    locate_by_tag_name_examples()
    locate_by_xpath_examples()
    locate_by_css_selector_examples()
    locate_by_link_text_examples()
    advanced_element_finding()
    element_locator_best_practices()
```

```
print(" All element locating examples completed!")
```

Now let me create a practical reference guide for element locators:

Element Locator Strategies Reference

Locator Priority (Most Reliable to Least)

1. **ID** - Most reliable, unique identifier
2. **Name** - Usually reliable for form elements
3. **CSS Selector** - Fast and flexible
4. **XPath** - Most powerful but can be fragile
5. **Class Name** - Good for styling-based elements
6. **Tag Name** - Too generic, use sparingly
7. **Link Text** - Only for links, can break with text changes

Complete Locator Syntax Guide

1. By ID

```
# HTML: <input id="username" name="user">
element = driver.find_element(By.ID, "username")
```

When to use: Always prefer this when available. IDs should be unique.

2. By Name

```
# HTML: <input name="email" type="email">
element = driver.find_element(By.NAME, "email")
```

When to use: Form elements often have name attributes.

3. By Class Name

```
# HTML: <button class="btn-primary submit-btn">
element = driver.find_element(By.CLASS_NAME, "btn-primary")
```

Note: Use only single class name, not multiple classes.

4. By Tag Name

```
# HTML: <input>, <button>, <div>
elements = driver.find_elements(By.TAG_NAME, "input")  # Returns list
```

When to use: Finding all elements of a type, or when combined with other strategies.

5. By CSS Selector (Recommended)

Basic CSS Selectors

```
# By ID: #username
```

```
driver.find_element(By.CSS_SELECTOR, "#username")

# By class: .btn-primary
driver.find_element(By.CSS_SELECTOR, ".btn-primary")

# By tag: input
driver.find_element(By.CSS_SELECTOR, "input")

# By attribute: [name='email']
driver.find_element(By.CSS_SELECTOR, "[name='email']")

# Multiple classes: .btn.btn-primary
driver.find_element(By.CSS_SELECTOR, ".btn.btn-primary")
```

Advanced CSS Selectors

```
# Descendant: form input (input inside form)
driver.find_element(By.CSS_SELECTOR, "form input")

# Direct child: form > input (direct child only)
driver.find_element(By.CSS_SELECTOR, "form > input")

# Attribute contains: [class*='btn']
driver.find_element(By.CSS_SELECTOR, "[class*='btn']")

# Attribute starts with: [id^='user']
driver.find_element(By.CSS_SELECTOR, "[id^='user']")

# Attribute ends with: [id$='name']
driver.find_element(By.CSS_SELECTOR, "[id$='name']")

# Nth child: tr:nth-child(2) (second row)
driver.find_element(By.CSS_SELECTOR, "tr:nth-child(2)")

# First/Last: input:first-of-type, input:last-of-type
driver.find_element(By.CSS_SELECTOR, "input:first-of-type")
```

6. By XPath

Basic XPath

```
# Absolute (avoid): /html/body/div/form/input
# Relative (preferred): //input[@id='username']
driver.find_element(By.XPATH, "//input[@id='username']")

# By text: //button[text()='Submit']
driver.find_element(By.XPATH, "//button[text()='Submit']")

# Contains text: //button[contains(text(), 'Sub')]
driver.find_element(By.XPATH, "//button[contains(text(), 'Sub')]")
```

Advanced XPath

```
# Multiple attributes: //input[@type='text' and @name='username']
driver.find_element(By.XPATH, "//input[@type='text' and @name='username']")

# Contains attribute: //div[contains(@class, 'error')]
driver.find_element(By.XPATH, "//div[contains(@class, 'error')]")

# Parent: //input[@id='username']/parent::div
driver.find_element(By.XPATH, "//input[@id='username']/parent::div")
```

```
# Following sibling: //label[@for='username']/following-sibling::input
driver.find_element(By.XPATH, "//label[@for='username']/following-
sibling::input")

# Position: //tr[2]/td[3] (2nd row, 3rd column)
driver.find_element(By.XPATH, "//tr[2]/td[3]")
```

7. By Link Text

```
# Exact text match
driver.find_element(By.LINK_TEXT, "Click Here")

# Partial text match
driver.find_element(By.PARTIAL_LINK_TEXT, "Click")
```

Common HTML Patterns & Solutions

Forms

```
# Login form
username = driver.find_element(By.ID, "username")
password = driver.find_element(By.NAME, "password")
submit_btn = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
```

Tables

```
# All table rows
rows = driver.find_elements(By.CSS_SELECTOR, "table tr")

# Specific cell
cell = driver.find_element(By.XPATH, "//table/tbody/tr[2]/td[3]")

# Header by text
header = driver.find_element(By.XPATH, "//th[text()='Name']")
```

Dynamic Content

```
# Elements with dynamic IDs: id="user_12345"
user = driver.find_element(By.CSS_SELECTOR, "[id^='user_']")

# Or with XPath
user = driver.find_element(By.XPATH, "//div[starts-with(@id, 'user_')]")
```

Buttons

```
# Submit buttons
submit = driver.find_element(By.CSS_SELECTOR, "input[type='submit']")
button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")

# By text content
save_btn = driver.find_element(By.XPATH, "//button[text()='Save']")
```

Best Practices & Tips

1. Element Locator Hierarchy

```
def find_element_robust(driver, element_id):
    """Try multiple strategies to find element"""
    strategies = [
        (By.ID, element_id),
        (By.NAME, element_id),
        (By.CSS_SELECTOR, f"[data-testid='{element_id}']"),
        (By.CSS_SELECTOR, f"#{element_id}"),
        (By.XPATH, f"//input[@id='{element_id}']")
    ]

    for by, value in strategies:
        try:
            return driver.find_element(by, value)
        except NoSuchElementException:
            continue

    raise NoSuchElementException(f"Could not locate element: {element_id}")
```

2. Wait for Elements (Preview of next topic)

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

# Wait for element to be present
wait = WebDriverWait(driver, 10)
element = wait.until(EC.presence_of_element_located((By.ID, "username")))
```

3. Custom Data Attributes (Recommended)

```
<!-- HTML with test-specific attributes -->
<button data-testid="submit-button">Submit</button>
<input data-testid="username-input">

# Selenium code
submit_btn = driver.find_element(By.CSS_SELECTOR, "[data-testid='submit-
button']")
username = driver.find_element(By.CSS_SELECTOR, "[data-testid='username-
input']")
```

4. Element Validation

```
def is_element_present(driver, by, value):
    """Check if element exists without throwing exception"""
    try:
        driver.find_element(by, value)
        return True
    except NoSuchElementException:
        return False

# Usage
if is_element_present(driver, By.ID, "error-message"):
    print("Error message is displayed")
```

Troubleshooting Common Issues

Issue: Element not found

Solutions:

1. Check if element is in iframe
2. Wait for element to load (use WebDriverWait)
3. Verify locator syntax
4. Check if element is dynamically generated

Issue: Multiple elements found

Solutions:

1. Use more specific locator
2. Use `find_elements()` and select by index
3. Add additional attributes to locator

Issue: Stale element

Solutions:

1. Re-find element after page changes
2. Use fresh locators instead of storing elements

Quick Reference Card

Need	CSS Selector	XPath
ID	`#myid`	`//[@id='myid']`
Class	`.myclass`	`//[@class='myclass']`
Attribute	`[name='value']`	`//[@name='value']`
Text	N/A	`//[text()='text']`
Contains	`[attr*='value']`	`//[contains(@attr,'value')]`
Parent	N/A	`//element/parent::div`
Child	`parent > child`	`//parent/child`

Key Takeaways from Element Locating:

Locator Priority (Most to Least Reliable):

1. **ID** - Always use when available (most stable)
2. **Name** - Good for form elements
3. **CSS Selector** - Fast and flexible
4. **XPath** - Powerful but can be fragile
5. **Class Name** - Use cautiously (styling changes)
6. **Tag Name** - Too generic, rarely used alone

Essential Methods:

- `find_element(By.*, "value")` - Returns first matching element
- `find_elements(By.*, "value")` - Returns list of all matching elements
- `element.get_attribute("attribute_name")` - Get element attributes
- `element.text` - Get visible text content

Pro Tips:

- **Use data-testid attributes** for automation-friendly locators
- **Combine multiple strategies** for robust element finding
- **CSS Selectors are faster** than XPath for most cases
- **XPath is more powerful** for complex relationships

Practice Exercise:

Try the element locating examples and:

1. Identify elements using different locator strategies
2. Compare the reliability of different approaches
3. Practice both `find_element()` and `find_elements()`

Ready for the next topic: "**Interacting with Web Elements**"? This is where we learn to click buttons, fill forms, select dropdowns, and perform all the actions users do on web pages!

Phase 2, Topic 5: Interacting with Web Elements

Now that you can find elements, let's learn how to interact with them! This covers all the actions users perform on web pages - clicking, typing, selecting, checking boxes, and more.

web_element_interactions.py

Complete guide to interacting with web elements in Selenium

from selenium import webdriver

from selenium.webdriver.common.by import By

from selenium.webdriver.common.keys import Keys

from selenium.webdriver.support.ui import Select, WebDriverWait

from selenium.webdriver.support import expected_conditions as EC

from selenium.webdriver.chrome.service import Service

from selenium.webdriver.chrome.options import Options

from selenium.common.exceptions import NoSuchElementException, TimeoutException

from webdriver_manager.chrome import ChromeDriverManager

import time

def setup_driver():

```
    """Setup Chrome driver with options"""
    options = Options()
    options.add_argument("--start-maximized")
    driver = webdriver.Chrome(
        service=Service(ChromeDriverManager().install()),
        options=options
    )
    return driver

def basic_text_input_interactions():
    """Basic text input and form interactions"""
    print("📝 Basic Text Input Interactions")

    driver = setup_driver()

    try:
        # Navigate to a form page
        driver.get("https://the-internet.herokuapp.com/login")

        # Find input elements
        username_field = driver.find_element(By.ID, "username")
        password_field = driver.find_element(By.ID, "password")

        # Basic text input
        username_field.send_keys("tomsmith")
        password_field.send_keys("SuperSecretPassword!")

        print("✅ Text entered in form fields")

        # Clear field and re-enter
        username_field.clear()
        username_field.send_keys("admin")
```

```
        print("✓ Field cleared and updated")

        # Get current value
        current_username = username_field.get_attribute("value")
        print(f"✓ Current username value: {current_username}")

        # Special keys
        username_field.send_keys(Keys.CONTROL, "a")  # Select all
        username_field.send_keys("newuser")  # Replace selected text

        print("✓ Used special keys for text manipulation")

        time.sleep(2)

    except Exception as e:
        print(f"✗ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def click_interactions():
    """Different types of click interactions"""
    print("🖱 Click Interactions")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Fill form first
```

```
    driver.find_element(By.ID, "username").send_keys("tomsmith")
    driver.find_element(By.ID, "password").send_keys("SuperSecretPassword!")

    # Basic click
    login_button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
    login_button.click()

    print("✅ Basic click performed")

    # Wait for page to load and check result
    time.sleep(2)

    # Check if login was successful
    try:
        success_message = driver.find_element(By.CSS_SELECTOR, ".flash.success")
        print(f"✅ Login successful: {success_message.text}")

        # Click logout
        logout_link = driver.find_element(By.LINK_TEXT, "Logout")
        logout_link.click()
        print("✅ Logout clicked")

    except NoSuchElementException:
        print("❌ Login may have failed")

    time.sleep(2)

except Exception as e:
    print(f"❌ Error: {e}")

finally:
    driver.quit()
```

```
        print("Browser closed\n")

def checkbox_interactions():
    """Working with checkboxes"""
    print("☑ Checkbox Interactions")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/checkboxes")

        # Find all checkboxes
        checkboxes = driver.find_elements(By.CSS_SELECTOR, "input[type='checkbox']")

        print(f"✓ Found {len(checkboxes)} checkboxes")

        # Check status and interact with each
        for i, checkbox in enumerate(checkboxes, 1):
            is_checked = checkbox.is_selected()
            print(f"   Checkbox {i}: {'Checked' if is_checked else 'Unchecked'}")

            # Toggle checkbox state
            if not is_checked:
                checkbox.click()
                print(f"   ✓ Checked checkbox {i}")
            else:
                checkbox.click()
                print(f"   ✓ Unchecked checkbox {i}")

        time.sleep(2)

        # Verify final states
```

```
        print("\nFinal checkbox states:")
        for i, checkbox in enumerate(checkboxes, 1):
            is_checked = checkbox.is_selected()
            print(f"   Checkbox {i}: {'Checked' if is_checked else 'Unchecked'}")

    except Exception as e:
        print(f"❌ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def dropdown_interactions():
    """Working with dropdown/select elements"""
    print("📋 Dropdown/Select Interactions")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/dropdown")

        # Find dropdown element
        dropdown_element = driver.find_element(By.ID, "dropdown")

        # Create Select object
        dropdown = Select(dropdown_element)

        # Get all options
        all_options = dropdown.options
        print(f"✅ Dropdown has {len(all_options)} options:")
        for option in all_options:
            print(f"   - {option.text}")
```

```
    # Select by visible text
    dropdown.select_by_visible_text("Option 1")
    print("✓ Selected 'Option 1' by text")

    time.sleep(1)

    # Select by value
    dropdown.select_by_value("2")
    print("✓ Selected 'Option 2' by value")

    time.sleep(1)

    # Select by index (0-based)
    dropdown.select_by_index(1)  # Option 1
    print("✓ Selected 'Option 1' by index")

    # Get currently selected option
    selected_option = dropdown.first_selected_option
    print(f"✓ Currently selected: {selected_option.text}")

    # Check if it's a multi-select dropdown
    is_multiple = dropdown.is_multiple
    print(f"✓ Is multiple select: {is_multiple}")

except Exception as e:
    print(f"✗ Error: {e}")

finally:
    driver.quit()
    print("Browser closed\n")
```

```
def radio_button_interactions():
    """Working with radio buttons"""
    print("🔘 Radio Button Interactions")

    driver = setup_driver()

    try:
        # Create a simple HTML page with radio buttons
        html_content = """
        <html>
        <body>
            <h2>Radio Button Test</h2>
            <form>
                <input type="radio" id="option1" name="choice" value="1">
                <label for="option1">Option 1</label><br>

                <input type="radio" id="option2" name="choice" value="2">
                <label for="option2">Option 2</label><br>

                <input type="radio" id="option3" name="choice" value="3" checked>
                <label for="option3">Option 3 (Pre-selected)</label><br>
            </form>
        </body>
        </html>
        """

        # Save to temporary file and open
        with open("temp_radio.html", "w") as f:
            f.write(html_content)

        driver.get("file://" + os.path.abspath("temp_radio.html"))
```

```
    # Find all radio buttons
    radio_buttons = driver.find_elements(By.CSS_SELECTOR, "input[type='radio']")

    print(f"✓ Found {len(radio_buttons)} radio buttons")

    # Check which one is selected initially
    for i, radio in enumerate(radio_buttons, 1):
        is_selected = radio.is_selected()
        value = radio.get_attribute("value")
        print(f"   Radio {i} (value={value}): {'Selected' if is_selected else 'Not selected'}")

    # Select different radio button
    radio_buttons[0].click()  # Select first option
    print("✓ Selected first radio button")

    time.sleep(1)

    # Verify selection changed
    for i, radio in enumerate(radio_buttons, 1):
        is_selected = radio.is_selected()
        value = radio.get_attribute("value")
        print(f"   Radio {i} (value={value}): {'Selected' if is_selected else 'Not selected'}")

except Exception as e:
    print(f"✗ Error: {e}")

finally:
    try:
        import os
        os.remove("temp_radio.html")
    except:
        pass
```

```
        driver.quit()
        print("Browser closed\n")

def file_upload_interactions():
    """File upload interactions"""
    print("📁 File Upload Interactions")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/upload")

        # Find file input element
        file_input = driver.find_element(By.ID, "file-upload")

        # Create a temporary file to upload
        test_file_content = "This is a test file for Selenium upload demo."
        with open("test_upload.txt", "w") as f:
            f.write(test_file_content)

        # Get absolute path of the file
        import os
        file_path = os.path.abspath("test_upload.txt")

        # Upload file by sending file path to input element
        file_input.send_keys(file_path)
        print("✅ File path sent to upload input")

        # Click upload button
        upload_button = driver.find_element(By.ID, "file-submit")
        upload_button.click()
        print("✅ Upload button clicked")
```

```
            # Wait for upload to complete
            time.sleep(3)

            # Check if upload was successful
            try:
                uploaded_files = driver.find_element(By.ID, "uploaded-files")
                print(f"✅ Upload successful: {uploaded_files.text}")
            except NoSuchElementException:
                print("❌ Upload may have failed")

        except Exception as e:
            print(f"❌ Error: {e}")

        finally:
            try:
                import os
                os.remove("test_upload.txt")
            except:
                pass
            driver.quit()
            print("Browser closed\n")

    def advanced_text_interactions():
        """Advanced text input interactions"""
        print("⌨️ Advanced Text Interactions")

        driver = setup_driver()

        try:
            driver.get("https://the-internet.herokuapp.com/inputs")
```

```
# Find input field
number_input = driver.find_element(By.CSS_SELECTOR, "input[type='number']")

# Type numbers
number_input.send_keys("123")
print("✅ Entered number: 123")

time.sleep(1)

# Use arrow keys to increment/decrement
number_input.send_keys(Keys.ARROW_UP)
print("✅ Used UP arrow key")

time.sleep(1)

number_input.send_keys(Keys.ARROW_DOWN)
print("✅ Used DOWN arrow key")

# Clear and enter new value
number_input.clear()
number_input.send_keys("999")

# Use keyboard shortcuts
number_input.send_keys(Keys.CONTROL, "a")  # Select all
number_input.send_keys("555")  # Replace with new value

print("✅ Used keyboard shortcuts")

# Get final value
final_value = number_input.get_attribute("value")
print(f"✅ Final value: {final_value}")
```

```
        # Press Enter key
        number_input.send_keys(Keys.ENTER)
        print("✓ Pressed Enter key")

        time.sleep(2)

    except Exception as e:
        print(f"✗ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

def element_properties_and_states():
    """Getting element properties and states"""
    print("🔍 Element Properties and States")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Find elements
        username_field = driver.find_element(By.ID, "username")
        password_field = driver.find_element(By.ID, "password")
        login_button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")

        # Get element properties
        print("Element Properties:")
        print(f"✓ Username tag name: {username_field.tag_name}")
        print(f"✓ Username type: {username_field.get_attribute('type')}")
        print(f"✓ Username name: {username_field.get_attribute('name')}")
```

```
        print(f"✓ Username placeholder: {username_field.get_attribute('placeholder')}")

        # Get element states
        print("\nElement States:")
        print(f"✓ Username is displayed: {username_field.is_displayed()}")
        print(f"✓ Username is enabled: {username_field.is_enabled()}")
        print(f"✓ Button text: '{login_button.text}'")

        # Get CSS properties
        print("\nCSS Properties:")
        print(f"✓ Username font-size: {username_field.value_of_css_property('font-size')}")
        print(f"✓ Button background-color: {login_button.value_of_css_property('background-color')}")

        # Get element size and location
        print("\nElement Dimensions:")
        size = username_field.size
        location = username_field.location
        print(f"✓ Username size: {size['width']}x{size['height']}")
        print(f"✓ Username location: ({location['x']}, {location['y']})")

        # Fill form and check states
        username_field.send_keys("testuser")

        # Check if field has content
        has_content = len(username_field.get_attribute("value")) > 0
        print(f"✓ Username has content: {has_content}")

    except Exception as e:
        print(f"✗ Error: {e}")

    finally:
```

```
        driver.quit()
        print("Browser closed\n")

def form_submission_example():
    """Complete form submission example"""
    print("📋 Complete Form Submission Example")

    driver = setup_driver()

    try:
        driver.get("https://the-internet.herokuapp.com/login")

        # Fill the form
        username = driver.find_element(By.ID, "username")
        password = driver.find_element(By.ID, "password")

        username.clear()
        username.send_keys("tomsmith")

        password.clear()
        password.send_keys("SuperSecretPassword!")

        print("✅ Form filled")

        # Submit form (multiple ways)

        # Method 1: Click submit button
        # login_button = driver.find_element(By.CSS_SELECTOR, "button[type='submit']")
        # login_button.click()

        # Method 2: Press Enter in a field
        # password.send_keys(Keys.ENTER)
```

```
        # Method 3: Submit form directly
        form = driver.find_element(By.TAG_NAME, "form")
        form.submit()

        print("✓ Form submitted")

        # Wait for response
        time.sleep(3)

        # Check result
        try:
            success_msg = driver.find_element(By.CSS_SELECTOR, ".flash.success")
            print(f"✓ Success: {success_msg.text}")
        except NoSuchElementException:
            try:
                error_msg = driver.find_element(By.CSS_SELECTOR, ".flash.error")
                print(f"✗ Error: {error_msg.text}")
            except NoSuchElementException:
                print("✗ No feedback message found")

    except Exception as e:
        print(f"✗ Error: {e}")

    finally:
        driver.quit()
        print("Browser closed\n")

# Main execution
if __name__ == "__main__":
    print("=== Web Element Interactions Examples ===\n")
```

basic_text_input_interactions()

click_interactions()

checkbox_interactions()

dropdown_interactions()

radio_button_interactions()

file_upload_interactions()

advanced_text_interactions()

element_properties_and_states()

form_submission_example()

print(" All web element interaction examples completed!")

Now let me create a quick reference guide for element interactions:

Element Interactions Quick Reference

Basic Element Actions

Text Input Fields

```
# Basic text input
element.send_keys("text to type")

# Clear field
element.clear()

# Clear and enter new text
element.clear()
element.send_keys("new text")

# Get current value
current_value = element.get_attribute("value")

# Special key combinations
element.send_keys(Keys.CONTROL, "a")  # Select all (Ctrl+A)
element.send_keys(Keys.CONTROL, "c")  # Copy (Ctrl+C)
element.send_keys(Keys.CONTROL, "v")  # Paste (Ctrl+V)
```

Clicking Elements

```
# Basic click
element.click()

# Check if element is clickable
element.is_enabled()  # Returns True/False

# Submit forms
form.submit()  # Alternative to clicking submit button
field.send_keys(Keys.ENTER)  # Press Enter to submit
```

Checkboxes & Radio Buttons

```
# Check if selected
is_checked = checkbox.is_selected()

# Toggle checkbox
if not checkbox.is_selected():
    checkbox.click()  # Check it

# Radio buttons (clicking selects, can't unselect)
radio_button.click()
```

Dropdown/Select Elements

```
from selenium.webdriver.support.ui import Select

# Create Select object
dropdown = Select(driver.find_element(By.ID, "dropdown-id"))

# Select by visible text
dropdown.select_by_visible_text("Option Text")

# Select by value attribute
dropdown.select_by_value("option_value")

# Select by index (0-based)
dropdown.select_by_index(2)

# Get all options
all_options = dropdown.options
for option in all_options:
    print(option.text)

# Get currently selected option
selected = dropdown.first_selected_option
print(selected.text)

# For multi-select dropdowns
if dropdown.is_multiple:
    dropdown.select_by_visible_text("Option 1")
    dropdown.select_by_visible_text("Option 2")

    # Deselect
    dropdown.deselect_by_visible_text("Option 1")
    dropdown.deselect_all()
```

File Upload

```
# Find file input element
file_input = driver.find_element(By.CSS_SELECTOR, "input[type='file']")

# Send file path (must be absolute path)
import os
file_path = os.path.abspath("path/to/file.txt")
file_input.send_keys(file_path)

# Then click upload button if needed
upload_btn = driver.find_element(By.ID, "upload-button")
upload_btn.click()
```

Special Keys Reference

Common Keys

```
from selenium.webdriver.common.keys import Keys

# Navigation
Keys.ENTER
Keys.TAB
Keys.ESCAPE
Keys.SPACE
Keys.BACK_SPACE
Keys.DELETE

# Arrow keys
Keys.ARROW_UP
Keys.ARROW_DOWN
Keys.ARROW_LEFT
Keys.ARROW_RIGHT

# Page navigation
Keys.HOME
Keys.END
Keys.PAGE_UP
Keys.PAGE_DOWN

# Modifiers
Keys.CONTROL
Keys.ALT
Keys.SHIFT

# Function keys
Keys.F1, Keys.F2, ... Keys.F12
```

Key Combinations

```
# Ctrl combinations
element.send_keys(Keys.CONTROL, "a")  # Select all
element.send_keys(Keys.CONTROL, "c")  # Copy
element.send_keys(Keys.CONTROL, "v")  # Paste
element.send_keys(Keys.CONTROL, "z")  # Undo

# Multiple keys
element.send_keys(Keys.CONTROL, Keys.SHIFT, "a")  # Ctrl+Shift+A
element.send_keys(Keys.ALT, Keys.F4)  # Alt+F4
```

Element Properties & States

Getting Element Information

```
# Element properties
element.tag_name          # HTML tag (div, input, button, etc.)
element.text             # Visible text content
element.get_attribute("id")      # Specific attribute value
element.get_attribute("class")   # Class attribute
element.get_attribute("value")   # Input value

# Element states
element.is_displayed()    # Visible on page
element.is_enabled()      # Interactive/clickable
```

```
element.is_selected()     # Checked/selected (checkbox/radio)

# Element size and position
element.size              # {"width": 100, "height": 50}
element.location          # {"x": 10, "y": 20}
element.location_once_scrolled_into_view  # Location after scrolling

# CSS properties
element.value_of_css_property("color")
element.value_of_css_property("font-size")
element.value_of_css_property("background-color")
```

Element Screenshots

```
# Take screenshot of specific element
element.screenshot("element_screenshot.png")

# Get screenshot as binary data
screenshot_data = element.screenshot_as_png
```

Form Handling Patterns

Complete Form Example

```
def fill_login_form(driver, username, password):
    """Reusable form filling function"""

    # Find elements
    username_field = driver.find_element(By.ID, "username")
    password_field = driver.find_element(By.ID, "password")
    submit_button = driver.find_element(By.CSS_SELECTOR,
"button[type='submit']")

    # Clear and fill
    username_field.clear()
    username_field.send_keys(username)

    password_field.clear()
    password_field.send_keys(password)

    # Submit
    submit_button.click()

    # Or alternative submission methods:
    # password_field.send_keys(Keys.ENTER)
    # form = driver.find_element(By.TAG_NAME, "form")
    # form.submit()

# Usage
fill_login_form(driver, "testuser", "testpass")
```

Form Validation

```
def validate_form_submission(driver):
    """Check if form submission was successful"""
    try:
        # Look for success message
        success = driver.find_element(By.CSS_SELECTOR, ".success, .alert-
success")
        return True, success.text
```

```
    except NoSuchElementException:
        try:
            # Look for error message
            error = driver.find_element(By.CSS_SELECTOR, ".error, .alert-error")
            return False, error.text
        except NoSuchElementException:
            return None, "No feedback message found"

# Usage
success, message = validate_form_submission(driver)
if success:
    print(f"✅ Success: {message}")
elif success is False:
    print(f"❌ Error: {message}")
else:
    print(f"⚠ {message}")
```

Common Interaction Patterns

Wait Before Interaction

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

# Wait for element to be clickable
wait = WebDriverWait(driver, 10)
element = wait.until(EC.element_to_be_clickable((By.ID, "button-id")))
element.click()

# Wait for element to be visible
element = wait.until(EC.visibility_of_element_located((By.ID, "element-id")))
element.send_keys("text")
```

Safe Element Interaction

```
def safe_click(driver, by, value, timeout=10):
    """Safely click element with wait"""
    try:
        wait = WebDriverWait(driver, timeout)
        element = wait.until(EC.element_to_be_clickable((by, value)))
        element.click()
        return True
    except TimeoutException:
        print(f"Element not clickable: {by}={value}")
        return False

def safe_send_keys(driver, by, value, text, timeout=10):
    """Safely send keys with wait"""
    try:
        wait = WebDriverWait(driver, timeout)
        element = wait.until(EC.presence_of_element_located((by, value)))
        element.clear()
        element.send_keys(text)
        return True
    except TimeoutException:
        print(f"Element not found: {by}={value}")
        return False
```

Chain Actions

```
# Multiple actions in sequence
def complete_registration(driver, user_data):
    """Complete user registration form"""

    # Personal info
    driver.find_element(By.ID, "firstname").send_keys(user_data["first_name"])
    driver.find_element(By.ID, "lastname").send_keys(user_data["last_name"])
    driver.find_element(By.ID, "email").send_keys(user_data["email"])

    # Dropdown selection
    country_dropdown = Select(driver.find_element(By.ID, "country"))
    country_dropdown.select_by_visible_text(user_data["country"])

    # Checkbox agreement
    terms_checkbox = driver.find_element(By.ID, "terms")
    if not terms_checkbox.is_selected():
        terms_checkbox.click()

    # Submit
    driver.find_element(By.ID, "submit").click()
```

Troubleshooting Common Issues

Issue: Element not interactable

Solution: Wait for element or scroll into view

```
# Scroll element into view
driver.execute_script("arguments[0].scrollIntoView();", element)
element.click()
```

Issue: Stale element reference

Solution: Re-find element after page changes

```
# Don't store elements across page changes
# BAD:
# element = driver.find_element(By.ID, "button")
# driver.refresh()
# element.click()  # Will fail - stale reference

# GOOD:
driver.refresh()
element = driver.find_element(By.ID, "button")  # Re-find element
element.click()
```

Issue: Element covered by another element

Solution: Use JavaScript click or scroll

```
# JavaScript click (bypasses overlays)
driver.execute_script("arguments[0].click();", element)

# Or scroll element into view first
driver.execute_script("arguments[0].scrollIntoView(true);", element)
element.click()
```

Key Takeaways from Web Element Interactions:

Essential Methods:

- `element.send_keys("text")` - Type text into input fields
- `element.click()` - Click buttons, links, checkboxes
- `element.clear()` - Clear input fields
- `element.submit()` - Submit forms
- `Select(element)` - Handle dropdown/select elements

Element States:

- `element.is_displayed()` - Check if visible
- `element.is_enabled()` - Check if interactive
- `element.is_selected()` - Check if checked/selected
- `element.get_attribute("value")` - Get input values

Special Keys:

- Use `Keys.ENTER`, `Keys.TAB`, `Keys.CONTROL` for keyboard actions
- Combine keys: `element.send_keys(Keys.CONTROL, "a")` for Ctrl+A

Form Handling Best Practices:

1. **Always clear fields** before entering new text
2. **Use Select class** for dropdowns instead of clicking
3. **Check element states** before interacting
4. **Handle both success and error scenarios**

Practice Exercise:

Try the interaction examples and:

1. Fill out different types of forms
2. Practice with checkboxes, radio buttons, and dropdowns
3. Upload files and handle form submissions
4. Check element properties and states

Ready for the final topic in Phase 2: **"Browser Navigation"**? We'll learn to navigate between pages, handle multiple windows/tabs, and work with frames and iframes!

Phase 2.6: Browser Navigation in Selenium Python

Overview

Browser Navigation in Selenium refers to controlling browser actions like moving forward/backward through browser history, refreshing pages, managing multiple windows/tabs, and handling frames. This is essential for automating complex web application workflows.

1. Basic Browser Navigation Commands

Forward, Back, and Refresh Operations

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

# Setup WebDriver
driver = webdriver.Chrome(service=Service(ChromeDriverManager().install()))

try:
    # Navigate to first page
    driver.get("https://www.google.com")
    time.sleep(2)

    # Navigate to second page
    driver.get("https://www.github.com")
    time.sleep(2)

    # Go back to previous page
    driver.back()
    print(f"Current URL after back: {driver.current_url}")
    time.sleep(2)

    # Go forward to next page
    driver.forward()
    print(f"Current URL after forward: {driver.current_url}")
    time.sleep(2)

    # Refresh current page
    driver.refresh()
    print("Page refreshed")

finally:
    driver.quit()
```

Navigation Methods Summary

Method	Purpose	Example
`driver.get(url)`	Navigate to a specific URL	`driver.get("https://example.com")`
`driver.back()`	Go back one page in history	`driver.back()`
`driver.forward()`	Go forward one page in history	`driver.forward()`
`driver.refresh()`	Reload current page	`driver.refresh()`
`driver.current_url`	Get current page URL	`url = driver.current_url`
`driver.title`	Get current page title	`title = driver.title`

2. Window Handling

Managing Multiple Windows/Tabs

```
from selenium import webdriver
from selenium.webdriver.common.by import By
```

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

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

try:
    # Open main page
    driver.get("https://www.selenium.dev/selenium/web/window_switching.html")

    # Get current window handle
    main_window = driver.current_window_handle
    print(f"Main window handle: {main_window}")

    # Get all window handles
    all_windows = driver.window_handles
    print(f"Total windows: {len(all_windows)}")

    # Click link that opens new window
    driver.find_element(By.LINK_TEXT, "Open new window").click()
    time.sleep(2)

    # Get updated window handles
    all_windows = driver.window_handles
    print(f"Total windows after opening new: {len(all_windows)}")

    # Switch to new window
    for window in all_windows:
        if window != main_window:
            driver.switch_to.window(window)
            print(f"Switched to new window. Title: {driver.title}")
            break

    # Perform actions in new window
    time.sleep(2)

    # Close current window
    driver.close()

    # Switch back to main window
    driver.switch_to.window(main_window)
    print(f"Back to main window. Title: {driver.title}")

finally:
    driver.quit()
```

Practical Window Handling Function

```
def switch_to_new_window(driver, timeout=10):
    """
    Switch to newly opened window
    """
    original_windows = driver.window_handles

    # Wait for new window to open
    start_time = time.time()
    while len(driver.window_handles) <= len(original_windows):
        if time.time() - start_time > timeout:
            raise TimeoutError("New window did not open within timeout")
        time.sleep(0.5)

    # Switch to new window
    new_windows = driver.window_handles
```

```
    for window in new_windows:
        if window not in original_windows:
            driver.switch_to.window(window)
            return window

    return None

# Usage example
driver.find_element(By.LINK_TEXT, "Open in new tab").click()
new_window_handle = switch_to_new_window(driver)
print(f"Switched to new window: {driver.title}")
```

3. Frame and iFrame Handling

Switching Between Frames

```
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

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

try:
    driver.get("https://www.selenium.dev/selenium/web/iframes.html")

    # Method 1: Switch by frame index
    driver.switch_to.frame(0)  # Switch to first frame

    # Method 2: Switch by frame name or id
    driver.switch_to.default_content()  # Go back to main content
    driver.switch_to.frame("frame1")  # Switch by name/id

    # Method 3: Switch by WebElement
    driver.switch_to.default_content()
    frame_element = driver.find_element(By.TAG_NAME, "iframe")
    driver.switch_to.frame(frame_element)

    # Perform actions inside frame
    element_in_frame = driver.find_element(By.ID, "some-id")
    element_in_frame.click()

    # Always switch back to default content when done
    driver.switch_to.default_content()

finally:
    driver.quit()
```

Nested Frames Handling

```
def handle_nested_frames(driver):
    """
    Handle nested frames (frame within frame)
    """
    try:
        # Switch to parent frame
        driver.switch_to.frame("parentFrame")
        print("Switched to parent frame")

        # Switch to child frame within parent frame
        driver.switch_to.frame("childFrame")
```

```
        print("Switched to child frame")

        # Perform actions in child frame
        element = driver.find_element(By.ID, "child-element")
        element.click()

        # Go back to parent frame
        driver.switch_to.parent_frame()  # Goes to immediate parent
        print("Back to parent frame")

        # Go back to main content
        driver.switch_to.default_content()
        print("Back to main content")

    except Exception as e:
        print(f"Error handling frames: {e}")
        driver.switch_to.default_content()  # Always reset on error
```

4. Advanced Navigation Patterns

Page Navigation with Validation

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

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

    def navigate_and_verify(self, url, expected_title_contains=None):
        """
        Navigate to URL and verify successful navigation
        """
        try:
            self.driver.get(url)

            if expected_title_contains:
                self.wait.until(
                    EC.title_contains(expected_title_contains)
                )

            print(f"Successfully navigated to: {self.driver.current_url}")
            print(f"Page title: {self.driver.title}")
            return True

        except Exception as e:
            print(f"Navigation failed: {e}")
            return False

    def safe_back_navigation(self):
        """
        Safely navigate back with verification
        """
        current_url = self.driver.current_url
        self.driver.back()

        # Wait for navigation to complete
        self.wait.until(lambda driver: driver.current_url != current_url)
        print(f"Navigated back to: {self.driver.current_url}")

    def get_browser_info(self):
```

```
        """
        Get current browser and page information
        """
        return {
            'url': self.driver.current_url,
            'title': self.driver.title,
            'window_handles': len(self.driver.window_handles),
            'current_window': self.driver.current_window_handle
        }

# Usage
nav_helper = NavigationHelper(driver)
nav_helper.navigate_and_verify("https://example.com", "Example")
info = nav_helper.get_browser_info()
print(f"Browser info: {info}")
```

5. Common Navigation Scenarios

Scenario 1: Multi-tab Shopping Workflow

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

    try:
        # Open main shopping site
        driver.get("https://example-shop.com")
        main_window = driver.current_window_handle

        # Open multiple product pages in new tabs
        products = driver.find_elements(By.CLASS_NAME, "product-link")

        for i, product in enumerate(products[:3]):  # Open first 3 products
            # Right-click to open in new tab (using Actions class)
            from selenium.webdriver.common.action_chains import ActionChains
            from selenium.webdriver.common.keys import Keys


ActionChains(driver).key_down(Keys.CONTROL).click(product).key_up(Keys.CONTROL).
perform()
            time.sleep(1)

        # Switch between tabs and compare products
        all_windows = driver.window_handles

        for window in all_windows[1:]:  # Skip main window
            driver.switch_to.window(window)
            print(f"Product: {driver.title}")

            # Get product details
            try:
                price = driver.find_element(By.CLASS_NAME, "price").text
                print(f"Price: {price}")
            except:
                print("Price not found")

        # Return to main window
        driver.switch_to.window(main_window)

    finally:
        driver.quit()
```

Scenario 2: Frame-based Form Filling

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

    try:
        driver.get("https://example-with-frames.com")

        # Switch to form frame
        wait = WebDriverWait(driver, 10)
        frame = wait.until(EC.presence_of_element_located((By.ID, "form-
frame")))
        driver.switch_to.frame(frame)

        # Fill form inside frame
        wait.until(EC.presence_of_element_located((By.ID,
"username"))).send_keys("testuser")
        driver.find_element(By.ID, "password").send_keys("testpass")
        driver.find_element(By.ID, "submit").click()

        # Switch back to main content
        driver.switch_to.default_content()

        # Verify success message in main page
        success_msg = wait.until(EC.presence_of_element_located((By.CLASS_NAME,
"success")))
        print(f"Success: {success_msg.text}")

    finally:
        driver.quit()
```

6. Best Practices for Browser Navigation

1. Always Clean Up

```
# Always use try-finally or context managers
try:
    # Your navigation code
    pass
finally:
    driver.quit()  # Always close browser
```

2. Handle Window Switching Safely

```
def safe_window_switch(driver, target_window):
    """
    Safely switch windows with error handling
    """
    try:
        if target_window in driver.window_handles:
            driver.switch_to.window(target_window)
            return True
        else:
            print(f"Window handle {target_window} no longer exists")
            return False
    except Exception as e:
        print(f"Error switching windows: {e}")
        return False
```

3. Frame Switching with Reset

```
def safe_frame_operation(driver, frame_locator, operation_func):
    """
    Safely perform operations in frames with automatic reset
    """
    try:
        driver.switch_to.frame(frame_locator)
        operation_func(driver)
    except Exception as e:
        print(f"Frame operation failed: {e}")
    finally:
        driver.switch_to.default_content()  # Always reset
```

7. Common Issues and Solutions

Issue 1: Stale Window Handles

```
# Problem: Window handle becomes invalid
# Solution: Always get fresh window handles
def get_current_windows(driver):
    return driver.window_handles

# Use fresh handles each time
current_windows = get_current_windows(driver)
for window in current_windows:
    driver.switch_to.window(window)
    # Perform operations
```

Issue 2: Frame Detection

```
def is_frame_available(driver, frame_locator):
    """
    Check if frame is available before switching
    """
    try:
        WebDriverWait(driver, 5).until(
            EC.frame_to_be_available_and_switch_to_it(frame_locator)
        )
        driver.switch_to.default_content()  # Reset after check
        return True
    except:
        return False
```

Issue 3: Navigation Timing

```
def wait_for_page_load(driver, timeout=10):
    """
    Wait for page to fully load
    """
    try:
        WebDriverWait(driver, timeout).until(
            lambda driver: driver.execute_script("return document.readyState")
== "complete"
        )
        return True
    except:
        return False

# Usage
```

```
driver.get("https://example.com")
wait_for_page_load(driver)
```

Summary

Browser Navigation in Selenium covers:

- **Basic Navigation**: Forward, back, refresh operations
- **Window Management**: Handling multiple windows/tabs
- **Frame Handling**: Switching between frames and iframes
- **Advanced Patterns**: Complex navigation workflows
- **Best Practices**: Error handling and cleanup

Master these concepts to handle complex web application navigation scenarios effectively in your automation scripts.

I've completed the comprehensive guide for **Phase 2.6: Browser Navigation** from your Selenium roadmap. This covers all the essential navigation concepts including:

- Basic browser commands (forward, back, refresh)
- Window and tab handling with practical examples
- Frame and iframe management
- Advanced navigation patterns
- Best practices and common issues

The guide includes hands-on code examples that you can practice with, covering real-world scenarios like multi-tab workflows and frame-based form handling.

Are you ready to move on to the next topic? The next topic in your roadmap would be **Phase 3.1: Waits (Implicit vs Explicit Wait)**. Let me know when you'd like me to cover that topic!