Selenium WebDriver with Ruby Cheatsheet

A concise guide to using Selenium WebDriver with Ruby for automated web testing. Covers setup, basic commands, element interaction, waits, and best practices.

Setup and Basic Commands

Installation & Setup

Install Selenium WebDriver:

```
gem install selenium-webdriver
```

Basic Project Setup:

1. Create a new Ruby file (e.g., `example.rb`).
2. Require the Selenium WebDriver library.

Require Statement:

```
require 'selenium-webdriver'
```

Driver Configuration (ChromeDriver):

Ensure ChromeDriver is in your system's PATH.

Initialize Driver:

```
driver = Selenium::WebDriver.for :chrome
```

Alternative Driver Initialization (with options):

```
options = Selenium::WebDriver::Chrome::Options.new
options.add_argument('--headless') # Run in headless mode
driver = Selenium::WebDriver.for :chrome, options: options
```

Browser Navigation

Launch and Navigate:	`driver.get 'https://www.example.com'`
Get Current URL:	`current_url = driver.current_url` `puts current_url`
Get Page Title:	`title = driver.title` `puts title`
Refresh Page:	`driver.navigate.refresh`
Navigate Back/Forward:	`driver.navigate.back` `driver.navigate.forward`
Quit/Close Browser:	`driver.quit # Closes all browser windows and ends the WebDriver session` `driver.close # Closes current window`

Finding and Interacting with Elements

Finding Elements

Finding a Single Element:

Use `find_element` to locate the first matching element.

By ID:

```
element = driver.find_element(id: 'element_id')
```

By Name:

```
element = driver.find_element(name: 'element_name')
```

By Class Name:

```
element = driver.find_element(class: 'element_class')
```

By CSS Selector:

```
element = driver.find_element(css: '#element_id > .element_class')
```

By XPath:

```
element = driver.find_element(xpath: '//div[@id="element_id"]')
```

Finding Multiple Elements:

Use `find_elements` to locate all matching elements (returns an array).

Example:

```
elements = driver.find_elements(class: 'element_class')
elements.each { |el| puts el.text }
```

Interacting with Elements

Clicking:	`element.click`
Sending Keys (Typing):	`element.send_keys 'text to type'`
Clearing Text Field:	`element.clear`
Getting Text:	`text = element.text` `puts text`
Getting Attribute Value:	`attribute_value = element.attribute('value')` `puts attribute_value`
Is Element Displayed?:	`element.displayed?`
Is Element Enabled?:	`element.enabled?`
Is Element Selected?:	`element.selected?`

Waits, Alerts and Frames

Waits and Synchronization

Implicit Wait:
Sets a default wait time for all `find_element` calls. Not recommended for large projects.

Example:

```
driver.manage.timeouts.implicit_wait = 10 # seconds
```

Explicit Wait:
Waits for a specific condition to be true before proceeding.

Example:

```
wait = Selenium::WebDriver::Wait.new(timeout: 10)
element = wait.until { driver.find_element(id: 'element_id') }
```

Expected Conditions (with explicit wait):

- `element_to_be_clickable(element)`
- `presence_of_element_located(locator)`
- `visibility_of_element_located(locator)`
- `title_contains(title)`

Sleep (Avoid if possible):
Only use as a last resort. Introduce delays.

Example:

```
sleep 2 # seconds
```

Alerts and Frames

Switching to Alert:	`alert = driver.switch_to.alert`
Accepting Alert:	`alert.accept`
Dismissing Alert:	`alert.dismiss`
Getting Alert Text:	`alert_text = alert.text` `puts alert_text`
Switching to IFrame (by index):	`driver.switch_to.frame(0)`
Switching to IFrame (by name or ID):	`driver.switch_to.frame('frame_name')`
Switching back to Default Content:	`driver.switch_to.default_content`

Advanced Interactions and Best Practices

Advanced Interactions

Action Chains (Mouse Actions):
Performs complex user interactions like hover, drag and drop, right-click, etc.

Example (Hover):

```
element = driver.find_element(id: 'hover_element')
driver.action.move_to(element).perform
```

Example (Drag and Drop):

```
source = driver.find_element(id: 'draggable')
target = driver.find_element(id: 'droppable')
driver.action.drag_and_drop(source, target).perform
```

Example (Right Click):

```
element = driver.find_element(id: 'right_click_element')
driver.action.context_click(element).perform
```

Screenshots and Debugging

Taking Screenshot:	`driver.save_screenshot('screenshot.png')`
Saving HTML Source:	`File.write('page_source.html', driver.page_source)`
Debugging Tips:	• Use `puts` or `p` to print variable values. • Inspect element in browser's developer tools. • Use explicit waits to avoid timing issues.

Best Practices - Page Object Model (POM)

Page Object Model (POM):
Create separate classes for each page, encapsulating locators and actions.

Example:

```
class LoginPage
  def initialize(driver)
    @driver = driver
    @username_field = { id: 'username' }
    @password_field = { id: 'password' }
    @login_button = { id: 'login' }
  end

  def enter_username(username)

@driver.find_element(@username_field).se
nd_keys(username)
  end

  def enter_password(password)

@driver.find_element(@password_field).se
nd_keys(password)
  end

  def click_login

@driver.find_element(@login_button).clic
k
  end

  def login(username, password)
    enter_username(username)
    enter_password(password)
    click_login
  end
end
```