Selenium Testing & Debugging Cheatsheet

A quick reference for testing and debugging web applications using Selenium, covering common commands, debugging techniques, and best practices.

Core Selenium Commands

Basic Navigation

`driver.get(url)`	Loads a new web page.
`driver.current_url`	Returns the URL of the current page.
`driver.title`	Returns the title of the current page.
`driver.refresh()`	Refreshes the current page.
`driver.back()`	Navigates to the previous page in history.
`driver.forward()`	Navigates to the next page in history.

Element Interaction

`element.send_keys(value)`	Simulates typing into an element.
`element.click()`	Clicks on an element.
`element.clear()`	Clears the text of an input or textarea element.
`element.get_attribute(name)`	Gets the value of an element's attribute.
`element.text`	Gets the visible text of the element.
`element.is_displayed()`	Checks if the element is currently displayed.

Finding Elements

`driver.find_element(By.ID, id)`	Finds an element by its ID.
`driver.find_element(By.NAME, name)`	Finds an element by its name attribute.
`driver.find_element(By.CLASS_NAME, class_name)`	Finds an element by its class name.
`driver.find_element(By.TAG_NAME, tag_name)`	Finds an element by its tag name.
`driver.find_element(By.LINK_TEXT, link_text)`	Finds a link by its exact text.
`driver.find_element(By.PARTIAL_LINK_TEXT, partial_link_text)`	Finds a link by a partial match of its text.

Advanced Selenium Techniques

Explicit Waits

`WebDriverWait(driver, timeout).until(EC.presence_of_element_located((By.ID, 'element_id')))`	Waits until an element is present in the DOM.
`WebDriverWait(driver, timeout).until(EC.visibility_of_element_located((By.ID, 'element_id')))`	Waits until an element is visible.
`WebDriverWait(driver, timeout).until(EC.element_to_be_clickable((By.ID, 'element_id')))`	Waits until an element is clickable.
`WebDriverWait(driver, timeout).until(EC.text_to_be_present_in_element((By.ID, 'element_id'), text))`	Waits until specific text is present in the element.
`WebDriverWait(driver, timeout).until(EC.title_contains(title))`	Waits until the page title contains specific text.
`WebDriverWait(driver, timeout).until(EC.alert_is_present())`	Waits until an alert is present.

Handling Alerts and Popups

`alert = driver.switch_to.alert`	Switches the context to the currently active alert.
`alert.accept()`	Accepts the alert (clicks 'OK').
`alert.dismiss()`	Dismisses the alert (clicks 'Cancel').
`alert.send_keys(text)`	Sends text to the alert prompt.
`alert.text`	Gets the text of the alert.
`driver.switch_to.default_content()`	Switches back to the main document content.

Executing JavaScript

`driver.execute_script(script, *args)`	Executes JavaScript in the current browser context. `script`: The JavaScript code to execute. `*args`: Any arguments to pass to the script.
Example: `driver.execute_script("window.scrollTo(0, document.body.scrollHeight);")`	Scrolls to the bottom of the page.
Example: `driver.execute_script("arguments[0].click();", element)`	Clicks on a specific element using JavaScript.

Debugging Techniques

Common Exceptions

NoSuchElementException: Element not found.

- Verify the locator is correct.
- Ensure the element is present in the DOM.
- Use explicit waits to wait for the element to appear.

TimeoutException: Element not found within the specified time.

- Increase the timeout value.
- Verify the element is actually present.
- Check for dynamic content loading issues.

ElementNotInteractableException: Element is not clickable or visible.

- Ensure the element is visible and enabled.
- Check for overlapping elements.
- Scroll the element into view.

StaleElementReferenceException: Element is no longer attached to the DOM.

- Re-locate the element.
- Avoid storing element references for long periods.

Debugging Strategies

1. **Take Screenshots**: Capture the state of the browser at the point of failure.

~~~
driver.save_screenshot(“error.png”)
```
~~~

2. **Inspect the DOM**: Use browser developer tools to inspect the DOM structure and element attributes.

3. **Add Logging**: Log important events and variables to track the test flow.

~~~
import logging
logging.basicConfig(level=logging.INFO)
logging.info(“Clicking the button”)
element.click()
```
~~~

4. **Use Debugging Tools**: Utilize Python’s `pdb` or other debugging tools to step through the code.

~~~
import pdb; pdb.set_trace()
```
~~~

Selenium Grid

Selenium Grid allows running tests in parallel across different browsers and operating systems. It consists of a Hub and Nodes.

Hub: Central point that receives test requests and distributes them to available nodes.

Nodes: Registers with the Hub and provides the browsers and OS environments for running tests.

Best Practices

Code Maintainability

1. **Use Page Object Model (POM)**: Create classes representing web pages, encapsulating locators and actions. This promotes reusability and reduces code duplication.
2. **Use Data-Driven Testing**: Parameterize tests with data from external sources to improve coverage and maintainability.
3. **Avoid Hardcoded Waits**: Use explicit waits instead of hardcoded `time.sleep()` calls to improve test reliability.

Test Reliability

1. **Run Tests in Isolation**: Ensure tests do not depend on each other to avoid cascading failures.
2. **Use Test Fixtures**: Set up and tear down test environments to ensure consistent starting conditions.
3. **Handle Dynamic Content**: Use robust locators and explicit waits to handle dynamic content and AJAX requests.

Parallel Execution

1. **Use Selenium Grid**: Distribute tests across multiple machines and browsers to reduce test execution time.
2. **Parallel Test Runners**: Utilize test runners like `pytest-xdist` or `nose-parallel` to run tests in parallel within a single machine.