Consolidated Cheat Sheet: Selenium in Java, Python, and Cypress

# SELENIUM SYNTAX AND SNIPPETS

## 1. Installing Required Tools

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| Install Selenium | N/A (Handled via Maven/Gradle) | pip install selenium | npm install cypress | Installation command for respective tools. |
| Install Browser | Download ChromeDriver/  GeckoDriver | Download ChromeDriver/  GeckoDriver | Not needed; built into Cypress | Browser driver setup for Java and Python. |

## 2. Importing Libraries

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| Import Libraries | import org.openqa.selenium.\*; | from selenium import webdriver | Import not required;  auto-handled | Add necessary imports in Java and Python. |

## 3. Invoking Browsers

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Launch Chrome** | WebDriver driver = new ChromeDriver(); | from selenium import webdriver  driver = webdriver.Chrome() | cy.visit('http://example.com'); | Initialize Chrome browser for automation. |
| **Launch Firefox** | WebDriver driver = new FirefoxDriver(); | from selenium import webdriver  driver = webdriver.Firefox() | N/A | Initialize Firefox browser for automation. |
| **Maximize Window** | driver.manage().window().maximize(); | driver.maximize\_window() | Handled by default in Cypress | Ensure the browser window is maximized. |

## 4. Basic Browser Operations

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Topic | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| Navigate to URL | driver.get("http://example.com"); | driver.get("http://example.com") | cy.visit('http://example.com'); | Opens the specified URL. |
| Back Navigation | driver.navigate().back(); | driver.back() | cy.go('back'); | Navigates back in browser history. |
| Forward Navigation | driver.navigate().forward(); | driver.forward() | cy.go('forward'); | Navigates forward in browser history. |
| Refresh Page | driver.navigate().refresh(); | driver.refresh() | cy.reload(); | Refreshes the current browser page. |

## 5. Locating Elements

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **By ID** | driver.findElement(By.id("id")); | driver.find\_element (By.ID"id") | cy.get('#id'); | Locate element by its ID. |
| **By Name** | driver.findElement(By.name("name")); | driver.find\_element(By.NAME, "name") | cy.get('[name="name"]'); | Locate element by its name. |
| **By XPath** | driver.findElement(By.xpath ("//tag[@attr='value']")); | driver.find\_element(By.XPATH, "//tag[@attr='value']") | cy.xpath('//tag[@attr="value"]'); | Locate element using XPath expressions. |
| **By CSS Selector** | driver.findElement(By.cssSelector("button.btn-primary")); | driver.find\_element(By.CSS\_SELECTOR, "button.btn-primary") | cy.get('button.btn-primary'); | Locate element using CSS selector expressions |

## 6. Interacting with Elements

|  |  |  |  |
| --- | --- | --- | --- |
| Action | Java (Selenium) | Python (Selenium) | Cypress (JavaScript) |
| Open a URL | WebDriver driver = new ChromeDriver();  driver.get("https://example.com"); | from selenium import webdriver  driver = webdriver.Chrome()  driver.get("https://example.com") | cy.visit('https://example.com') |
| Find Element by ID | WebElement element = driver.findElement(By.id("elementID")); | element = driver.find\_element(By.ID, "elementID") | cy.get('#elementID') |
| Click an Element | WebElement button = driver.findElement(By.id("submitButton"));  button.click(); | button = driver.find\_element(By.ID, "submitButton")  button.click() | cy.get('#submitButton').click() |
| Send Text to Input Field | WebElement input = driver.findElement(By.name("username"));  input.sendKeys("myUsername"); | input = driver.find\_element(By.NAME, "username")  input.send\_keys("myUsername") | cy.get('input[name="username"]').type('myUsername') |
| Get Text of Element | WebElement textElement = driver.findElement(By.xpath("//h1"));  String text = textElement.getText(); | textElement = driver.find\_element(By.XPATH, "//h1")  text = textElement.text | cy.get('h1').invoke('text') |
| Check if Element is Visible | WebElement element = driver.findElement(By.id("elementID"));  boolean isVisible = element.isDisplayed(); | element = driver.find\_element(By.ID, "elementID")  isVisible = element.is\_displayed() | cy.get('#elementID').should('be.visible') |
| Select Dropdown Option | WebElement dropdown = driver.findElement(By.id("dropdown"));  Select select = new Select(dropdown);  select.selectByVisibleText("Option 1"); | from selenium.webdriver.support.ui  import Select  dropdown = driver.find\_element(By.ID, "dropdown")  select = Select(dropdown)  select.select\_by\_visible\_text("Option 1") | cy.get('#dropdown').select('Option 1') |
| Mouse Hover | Actions actions = new Actions(driver);  WebElement element = driver.findElement(By.id("hoverElement"));  actions.moveToElement(element).perform(); | from selenium.webdriver.common.action\_chains  import ActionChainsactions = ActionChains(driver)  element = driver.find\_element(By.ID, "hoverElement")  actions.move\_to\_element(element).perform() | cy.get('#hoverElement').trigger('mouseover') |
| Wait for Element to be Visible | WebDriverWait wait = new WebDriverWait(driver, 10);  WebElement element = wait.until(ExpectedConditions.visibilityOfElementLocated(By.id("elementID"))); | from selenium.webdriver.support.ui  import WebDriverWait from selenium.webdriver.support  import expected\_conditions as EC  wait = WebDriverWait(driver, 10)  element = wait.until(EC.visibility\_of\_element\_located((By.ID, "elementID"))) | cy.get('#elementID').should('be.visible') |
| Set Implicit Wait | driver.manage().timeouts().implicitlyWait(Duration.ofSeconds(10)); | driver.implicitly\_wait(10) | cy.wait(10000); |
| Get Element's Attribute Value | WebElement element = driver.findElement(By.id("inputField"));  String value = element.getAttribute("value"); | element = driver.find\_element(By.ID, "inputField")  value = element.get\_attribute("value") | cy.get('#inputField').invoke('val') |

# 7. Form Web Element Interactions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Form Element** | **Java (Selenium)** | **Python (Selenium)** | **Cypress (JavaScript)** | **Description** |
| Text Box (Input Field) | driver.findElement(By.id("username")).sendKeys("JohnDoe"); | driver.find\_element(By.ID, "username").send\_keys("JohnDoe") | cy.get('#username').type('JohnDoe'); | Enters text into an input field. |
| Password Field | driver.findElement(By.id("password")).sendKeys("securePass"); | driver.find\_element(By.ID, "password").send\_keys("securePass") | cy.get('#password').type('securePass'); | Enters a password in a secure input field. |
| Radio Button | driver.findElement(By.id("genderMale")).click(); | driver.find\_element(By.ID, "genderMale").click() | cy.get('#genderMale').check(); | Selects a radio button. |
| Checkbox | driver.findElement(By.id("acceptTerms")).click(); | driver.find\_element(By.ID, "acceptTerms").click() | cy.get('#acceptTerms').check(); | Checks a checkbox. |
| Uncheck Checkbox | driver.findElement(By.id("acceptTerms")).click(); | driver.find\_element(By.ID, "acceptTerms").click() | cy.get('#acceptTerms').uncheck(); | Unchecks a checkbox. |
| Dropdown (Select by Visible Text) | Select dropdown = new Select(driver.findElement(By.id("country"))); dropdown.selectByVisibleText("India"); | select = Select(driver.find\_element(By.ID, "country")) select.select\_by\_visible\_text("India") | cy.get('#country').select('India'); | Selects a dropdown option using visible text. |
| Dropdown (Select by Index) | dropdown.selectByIndex(2); | select.select\_by\_index(2) | cy.get('#country').select(2); | Selects a dropdown option by index. |
| Dropdown (Select by Value) | dropdown.selectByValue("IN"); | select.select\_by\_value("IN") | cy.get('#country').select('IN'); | Selects a dropdown option by value. |
| File Upload | driver.findElement(By.id("fileUpload")).sendKeys("C:\\path\\to\\file.txt"); | driver.find\_element(By.ID, "fileUpload").send\_keys("C:\\path\\to\\file.txt") | cy.get('#fileUpload').attachFile('file.txt'); | Uploads a file. |
| Text Area | driver.findElement(By.id("comments")).sendKeys("This is a comment."); | driver.find\_element(By.ID, "comments").send\_keys("This is a comment.") | cy.get('#comments').type('This is a comment.'); | Enters text into a text area. |
| Date Picker (Input Field) | driver.findElement(By.id("dob")).sendKeys("2025-02-09"); | driver.find\_element(By.ID, "dob").send\_keys("2025-02-09") | cy.get('#dob').type('2025-02-09'); | Selects a date in a date picker. |
| Slider (Move to Value) | WebElement slider = driver.findElement(By.id("volume")); Actions move = new Actions(driver); move.dragAndDropBy(slider, 50, 0).perform(); | slider = driver.find\_element(By.ID, "volume") ActionChains(driver).drag\_and\_drop\_by\_offset(slider, 50, 0).perform() | cy.get('#volume').invoke('val', '50').trigger('change'); | Adjusts a slider to a specific value. |
| Submit Form | driver.findElement(By.id("submitBtn")).click(); | driver.find\_element(By.ID, "submitBtn").click() | cy.get('#submitBtn').click(); | Clicks the submit button to submit a form. |

## 8. Assertions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Assertion** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Assert Equals** | assertEquals(expected, actual); (JUnit) | assert expected == actual | expect(actual).to.equal(expected); | Checks if two values are equal. |
| **Assert Not Equals** | assertNotEquals(expected, actual); (JUnit) | assert expected != actual | expect(actual).not.to.equal(expected); | Checks if two values are not equal. |
| **Assert True** | assertTrue(condition); (JUnit) | assert condition | expect(condition).to.be.true; | Asserts that the condition is true. |
| **Assert False** | assertFalse(condition); (JUnit) | assert not condition | expect(condition).to.be.false; | Asserts that the condition is false. |
| **Assert Null** | assertNull(object); (JUnit) | assert object is None | expect(object).to.be.null; | Asserts that the object is null. |
| **Assert Not Null** | assertNotNull(object); (JUnit) | assert object is not None | expect(object).to.not.be.null; | Asserts that the object is not null. |
| **Assert Array Size** | assertEquals(expectedSize, array.length); | assert len(array) == expected\_size | expect(array).to.have.length(expected\_size); | Asserts that the array or collection has the expected size. |
| **Assert List Size** | assertEquals(expectedSize, list.size()); | assert len(list) == expected\_size | expect(list).to.have.length(expected\_size); | Asserts that the list has the expected size. |
| **Assert Contains** | assertTrue(list.contains(element)); | assert element in list | expect(list).to.include(element); | Asserts that an element is contained in the collection. |
| **Assert Not Contains** | assertFalse(list.contains(element)); | assert element not in list | expect(list).to.not.include(element); | Asserts that an element is not contained in the collection. |
| **Assert String Equals** | assertEquals(expectedString, actualString); | assert expected\_string == actual\_string | expect(actual\_string).to.equal(expected\_string); | Asserts that two strings are equal. |
| **Assert String Contains** | assertTrue(actualString.contains(expectedSubstring)); | assert expected\_substring in actual\_string | expect(actual\_string).to.include(expected\_substring); | Asserts that a string contains a specific substring. |
| **Assert Exception** | assertThrows(ExpectedException.class, () -> { /\* code \*/ }); | with pytest.raises(ExpectedException):# code | N/A | Asserts that a specific exception is thrown. |
| **Assert Greater Than** | assertTrue(actual > expected); | assert actual > expected | expect(actual).to.be.greaterThan(expected); | Asserts that the actual value is greater than the expected. |
| **Assert Less Than** | assertTrue(actual < expected); | assert actual < expected | expect(actual).to.be.lessThan(expected); | Asserts that the actual value is less than the expected. |
| **Assert Object Equality** | N/A | N/A | expect(actualObject).to.deep.equal(expectedObject); | Asserts that two JavaScript objects are deeply equal. |

## 9. Advanced Concepts

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Handling Alerts** | driver.switchTo().alert().accept(); | `alert = driver.switch\_to.alert  alert.accept()` | cy.on('window:alert', () => { ... }) | Handle JavaScript alerts. |
| **Handling Frames** | driver.switchTo().frame("frameName"); | driver.switch\_to.frame("frameName") | cy.frameLoaded('iframeSelector') | Switch to a specific frame. |
| **File Upload** | driver.findElement(By.id("upload")).sendKeys("path"); `element = driver.find\_element\_by\_id("upload") | element.send\_keys("path")` | cy.get('#upload').attachFile('file.jpg'); | Automate file uploads. |
|  |  |  |  |  |
| **Taking Screenshots** | `File screenshot = ((TakesScreenshot) driver).getScreenshotAs(OutputType.FILE);  FileUtils.copyFile(screenshot, new File("path/to/screenshot.png"));` | driver.save\_screenshot("screenshot.png") | cy.screenshot('screenshot') | Capture a screenshot of the browser. |
| **Scroll to Element** | ((JavascriptExecutor) driver).executeScript("arguments[0].scrollIntoView(true);", element); | driver.execute\_script("arguments[0].scrollIntoView(true);", element) | cy.get('selector').scrollIntoView(); | Scrolls the page to bring a specific element into view. |
| **Scroll by Pixels** | ((JavascriptExecutor) driver).executeScript("window.scrollBy(0,500);"); | driver.execute\_script("window.scrollBy(0,500);") | cy.scrollTo(0, 500); | Scrolls the page vertically by a specified pixel amount. |
| **Scroll to Bottom** | ((JavascriptExecutor)driver).executeScript("window.scrollTo(0, document.body.scrollHeight);"); | driver.execute\_script("window.scrollTo(0, document.body.scrollHeight);") | cy.scrollTo('bottom'); | Scrolls to the bottom of the page. Useful for lazy loading content. |
| **Scroll to Top** | ((JavascriptExecutor) driver).executeScript("window.scrollTo(0, 0);"); | driver.execute\_script("window.scrollTo(0, 0);") | cy.scrollTo('top'); | Scrolls to the top of the page. |
| **Scroll Horizontally** | ((JavascriptExecutor) driver).executeScript("window.scrollBy(500,0);"); | driver.execute\_script("window.scrollBy(500,0);") | cy.scrollTo('right'); | Scrolls horizontally by a specific amount. |
| **Scroll Using Actions Class** | new Actions(driver).scrollByAmount(0, 500).perform(); | ActionChains(driver).scroll\_by\_amount(0, 500).perform() | *Not applicable; Cypress uses cy.scrollTo() instead* | Uses the **Actions class** to scroll, useful for interacting with scrollable elements. |
| **Infinite Scroll Handling** | While(true){((JavascriptExecutor)driver)  .executeScript("  window.scrollTo(0, document.body.scrollHeight);  ");  Thread.sleep(1000); } | while True:  driver.execute\_script("  window.scrollTo(0, document.body.scrollHeight);  ")  time.sleep(1) | cy.scrollTo('bottom');  cy.wait(1000); (repeat as needed) | Simulates infinite scrolling by continuously scrolling to the bottom and waiting for new content to load. |

## 10. Mouse Actions

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Hover Over Element** | Actions actions = new Actions(driver);  actions.moveToElement(element).perform(); | from selenium.webdriver.common.action\_chains  import ActionChainsactions = ActionChains(driver)  actions.move\_to\_element(element).perform() | cy.get('#id').trigger('mouseover') | Simulate hovering over an element. |
| **Double Click** | Actions actions = new Actions(driver);  actions.doubleClick(element).perform(); | actions = ActionChains(driver)  actions.double\_click(element).perform() | cy.get('#id').dblclick() | Perform a double-click operation. |
| **Drag and Drop** | Actions actions = new Actions(driver);actions.dragAndDrop(source, target).perform(); | actions = ActionChains(driver)  actions.drag\_and\_drop(source, target).perform() | N/A | Simulate drag-and-drop actions. |
| **File Upload via Mouse** | N/A | actions.click\_and\_hold(element).perform() | N/A | Simulate file upload by dragging files. |

## 11. Chrome options

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Option** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Disable Notifications** | ChromeOptions options = new ChromeOptions();options.addArguments("--disable-notifications");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Optionsoptions = Options()  options.add\_argument("--disable-notifications")  driver = webdriver.Chrome(options=options) | N/A | Disable browser notifications. |
| **Headless Mode** | ChromeOptions options = new ChromeOptions();  options.addArguments("--headless");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Optionsoptions = Options()  options.add\_argument("--headless")  driver = webdriver.Chrome(options=options) | npx cypress run --headless | Run Chrome without a UI. |
| **Set Proxy** | ChromeOptions options = new ChromeOptions();  options.addArguments("--proxy-server=http://proxy.example.com:8080");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Optionsoptions = Options()  options.add\_argument("--proxy-server=http://proxy.example .com:8080")  driver = webdriver.Chrome(options=options) | --proxy=http://proxy.example.com:8080 | Set the browser's proxy server. |
| **Disable GPU Hardware Acceleration** | ChromeOptions options = new ChromeOptions();  options.addArguments("--disable-gpu");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options  options = Options()  options.add\_argument("--disable-gpu")  driver = webdriver.Chrome(options=options) | N/A | Disable GPU hardware acceleration. |
| **Disable Extensions** | ChromeOptions options = new ChromeOptions();  options.addArguments("--disable-extensions");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options  options = Options()  options.add\_argument("--disable-extensions")  driver = webdriver.Chrome(options=options) | N/A | Disable all browser extensions. |
| **Set Window Size** | ChromeOptions options = new ChromeOptions();  options.addArguments("window-size=1200x600");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options  options = Options()  options.add\_argument("window-size=1200x600")  driver = webdriver.Chrome(options=options) | "viewportWidth": 1200, "viewportHeight": 600 | Set the initial window size of the browser. |
| **Incorporate User Data** | ChromeOptions options = new ChromeOptions();  options.addArguments("user-data-dir=/path/to/your/chrome/profile");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options  options = Options()  options.add\_argument("user-data-dir=<path>")  driver = webdriver.Chrome(options=options) | N/A | Use a specific Chrome profile or user data. |
| **Incognito Mode** | ChromeOptions options = new ChromeOptions();  options.addArguments("--incognito");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options  options = Options()  options.add\_argument("--incognito")  driver = webdriver.Chrome(options=options) | N/A | Open the browser in incognito (private browsing) mode. |
| **Remote Debugging** | ChromeOptions options = new ChromeOptions();  options.addArguments("--remote-debugging-port=9222");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Optionsoptions = Options()  options.add\_argument("--remote-debugging-port=9222")  driver = webdriver.Chrome(options=options) | N/A | Enable remote debugging of the browser. |
| **Disable Sandbox** | ChromeOptions options = new ChromeOptions();  options.addArguments("--no-sandbox");  WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Optionsoptions = Options()  options.add\_argument("--no-sandbox")  driver = webdriver.Chrome(options=options) | N/A | Disable the sandboxing feature for Chrome. |

## 12. File Downloads

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Automate Downloads** | HashMap<String, Object> prefs = new HashMap<>(); prefs.put("download.default\_directory", "path/to/download"); ChromeOptions options = new ChromeOptions(); options.setExperimentalOption("prefs", prefs); WebDriver driver = new ChromeDriver(options); | from selenium.webdriver.chrome.options  import Options options = Options() prefs = {"download.default\_directory": "path/to/download"} options.add\_experimental\_option("prefs", prefs) driver = webdriver.Chrome(options=options) | N/A | Configures browser settings for file downloads. |
| **Verify File Exists** | Java File I/O APIs to check file existence at download location | import os assert os.path.exists("path/to/downloaded/file") | Use Node.js fs module for checking files | Validates that the file is downloaded successfully. |

## 13. Using Service() Method in Python

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Service Initialization** | N/A | from selenium.webdriver.chrome.service import Service from selenium import webdriver service = Service("path/to/chromedriver") driver = webdriver.Chrome(service=service) | N/A | Demonstrates how to use Service() for initializing WebDriver. |

## 14. Additional Chrome Options

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Topic** | **Java Code** | **Python Code** | **Cypress Command** | **Description** |
| **Disable Notifications** | ChromeOptions options = new ChromeOptions(); options.addArguments("--disable-notifications"); WebDriver driver = new ChromeDriver(options); | options = Options() options.add\_argument("--disable-notifications") driver = webdriver.Chrome(options=options) | N/A | Suppresses browser notifications. |
| **Start in Incognito** | options.addArguments("--incognito"); WebDriver driver = new ChromeDriver(options); | options.add\_argument("--incognito") driver = webdriver.Chrome(options=options) | N/A | Opens the browser in incognito mode. |

# EVERGREEN JAVASCRIPT SNIPPETS FOR AUTOMATION TESTING

Here are 20 evergreen **JavaScript snippets** that you can use in the execute\_script() method in Selenium, along with their **usage**:

## 1. Click on an Element

**JavaScript:**

document.getElementById('element\_id').click();

**Usage:** Clicks on an element with a specific ID (e.g., a button or a link).

## 2. Scroll to an Element

**JavaScript:**

document.getElementById('element\_id').scrollIntoView();

**Usage:** Scrolls the page to bring the specified element into view.

## 3. Get the Page Title

**JavaScript:**

return document.title;

**Usage:** Retrieves the title of the current webpage.

## 4. Change the Value of an Input Field

**JavaScript:**

document.getElementById('input\_id').value = 'new\_value';

**Usage:** Changes the value of an input field (e.g., text box).

## 5. Get the Value of an Input Field

**JavaScript:**

return document.getElementById('input\_id').value;

**Usage:** Retrieves the current value from an input field.

## 6. Submit a Form

**JavaScript:**

document.getElementById('form\_id').submit();

**Usage:** Submits a form programmatically.

## 7. Get the Current URL

**JavaScript:**

return window.location.href;

**Usage:** Retrieves the current URL of the page.

## 8. Set a Cookie

**JavaScript:**

document.cookie = "cookie\_name=cookie\_value; path=/";

**Usage:** Sets a cookie in the browser.

## 9. Get All Cookies

**JavaScript:**

return document.cookie;

**Usage:** Retrieves all the cookies stored in the browser.

## 10. Disable JavaScript Alerts (Popups)

**JavaScript:**

window.alert = function() {}; // Disables alert popups

**Usage:** Disables JavaScript alert() popups on the page.

## 11. Highlight an Element (for Debugging)

**JavaScript:**

var element = document.getElementById('element\_id');

element.style.border = '3px solid red';

**Usage:** Adds a red border around an element for debugging purposes.

## 12. Get All Links on a Page

**JavaScript:**

var links = document.getElementsByTagName('a');

var linkHrefs = [];

for (var i = 0; i < links.length; i++) {

linkHrefs.push(links[i].href);

}

return linkHrefs;

**Usage:** Retrieves all the URLs of the links (<a> tags) on a page.

## 13. Remove an Element from the DOM

**JavaScript:**

var element = document.getElementById('element\_id');

element.parentNode.removeChild(element);

**Usage:** Removes an element from the DOM.

## 14. Set the Window Size

**JavaScript:**

window.resizeTo(1024, 768);

**Usage:** Resizes the browser window to the specified dimensions (e.g., 1024x768).

## 15. Get the Window's Height and Width

**JavaScript:**

return [window.innerWidth, window.innerHeight];

**Usage:** Retrieves the width and height of the current window.

## 16. Change an Element's CSS Style

**JavaScript:**

document.getElementById('element\_id').style.backgroundColor = 'blue';

**Usage:** Changes the background color of an element.

## 17. Scroll the Page to the Top

**JavaScript:**

window.scrollTo(0, 0);

**Usage:** Scrolls the page to the top.

## 18. Scroll the Page to the Bottom

**JavaScript:**

window.scrollTo(0, document.body.scrollHeight);

**Usage:** Scrolls the page to the bottom.

## 19. Trigger Mouse Hover on an Element

**JavaScript:**

var event = new MouseEvent('mouseover', { bubbles: true, cancelable: true });

document.getElementById('element\_id').dispatchEvent(event);

**Usage:** Triggers a mouse hover event over a specified element.

## 20. Wait for an Element to be Visible

**JavaScript:**

return document.querySelector('#element\_id').offsetParent !== null;

**Usage:** Checks if an element is visible on the page.

These JavaScript snippets can be executed in the browser's context using execute\_script() and are widely used for interacting with web pages. They can be used for tasks such as clicking elements, manipulating styles, scrolling, retrieving information, and disabling popups, among others.