CS1632:
Systems Testing the Web with Selenium

Wonsun Ahn

Background

- Systems testing: testing the entire system as a whole
  - We would like to automate systems testing, just like we did for unit testing
- So far, all of our testing has been text-based java programs
  - A.k.a. Command Line Interface (CLI) programs
- Automating testing for CLI programs is easy!
  - Just create an “input script” and redirect to stdin
  - Redirect stdout to file and compare to expected output

Automated Systems Test for CLI Programs Using `Bash`

```
#!/bin/bash


# Test case 1: Rent out cat #2 and check cat list
echo -e "2\n2\n1\n5\n" | java -jar rentacat.jar > testcase1.observed.out
diff testcase1.observed.out testcase1.expected.out


…
```

Automated Systems Test for GUI Programs

- Turns out that not every program is a CLI program
  - Web pages, mobile applications, windows applications, etc.
  - A.k.a. Graphical User Interface (GUI) programs
  - How do we deal with these?

- The theory behind testing remain the same
  - Compare **observed behavior vs. expected behavior**
  - Preconditions
  - Execution steps
  - Postconditions

- But we need different tools to automate testing GUI programs

Insight: GUI Apps ≈ Text-based Apps

- GUI apps also have a text representation for the output
  - It's just that the text is rendered into a graphical representation for end-user

- Example: Web applications with HTML (HyperText Markup Language)
  - HTML text is fetched from web server when a URL is requested
  - HTML text is rendered by web browser into graphical elements
- Example: Mobile applications with XML (Extensible Markup Language)
  - XML text is fetched from mobile app server or generated by app
  - XML text is rendered by Android / iOS into graphical elements

- So, in theory, GUI apps could be tested just like text-based apps
  - Using a simple string comparison of expected and observed output text
  - Assuming rendering is bug-free (safe to assume for modern browsers, Android / iOS)

Testing a Web App like a Text-based App

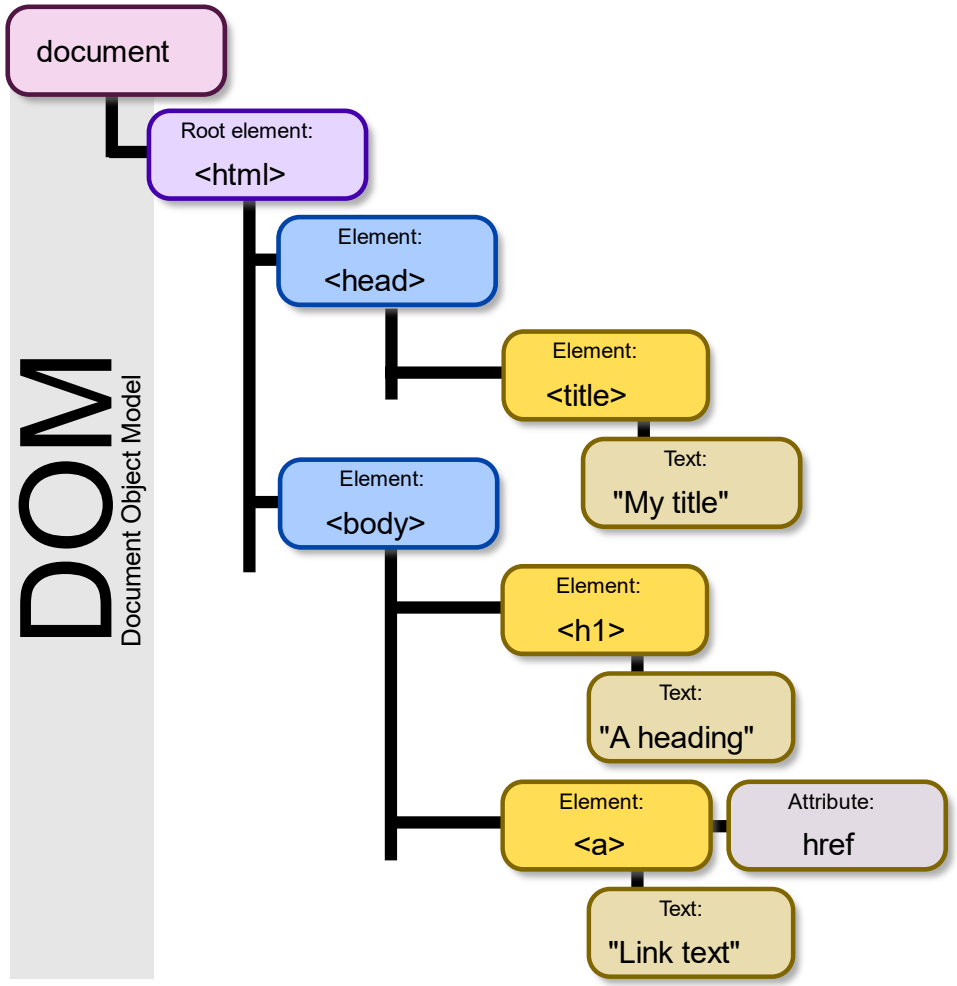
```
#!/bin/bash

# Test case 1: Fetch index.html and compare
wget example.com/index.html
diff index.html index.expected.html
…
```

1. HTML page is fetched from web server using `wget` utility
2. Fetched HTML page is compared against the expected HTML page

- Do you see any problems with this testing methodology?

HTML is organized into a hierarchy of elements

- Element: tag + content + attributes
- Tag: specifies type of element
  - <title>, <table>, <button>, <img>, …
  - <script>: JavaScript code that runs on browser
- Content: text to be displayed
- Attribute:
  - Specifies layout of element
  - Specifies action performed on interaction
- This model of web page is called DOM
  - DOM: Document Object Model



Problems with Naïve HTML Comparison

1. Tests are fragile
   - Trivial changes in HTML that don't impact final display can break test

2. Tests are untargeted
   - Changing HTML elements unrelated to test target will break test

3. JavaScript code is not functionally tested
   - JS code is compared verbatim to expected JS code, instead of executed
   - Changes in JS code that don't change functionality (e.g. commenting) breaks test

→ All of these lead to false positive defects while testing!

1. Tests are Fragile

- Are the two really different when displayed? (Hint: No)

[Expected HTML]

```
<html>
<head>
    <title>Example</title>
</head>
<body>
    …
</body>
</html>
```

[Observed HTML]

```
<HTML>
<HEAD>
    <TITLE>Example</TITLE>
</HEAD>
<BODY>
    …
</BODY>
</HTML>
```

2. Tests are Untargeted

- Which HTML element are we testing?

```
<html>
<head>
    <title>Example Domain</title>
    <style type="text/css">
    a:link, a:visited {
        color: #38488f;
    }
    </style>
</head>
<body>
    <a href="https://www.iana.org/domains/example">More info</a>
</body>
</html>
```

Title text?

CSS URL link color?

URL link?

Body text?

3. JavaScript code is not functionally tested

- Are the two really different when displayed? (Hint: No)

[Expected HTML]

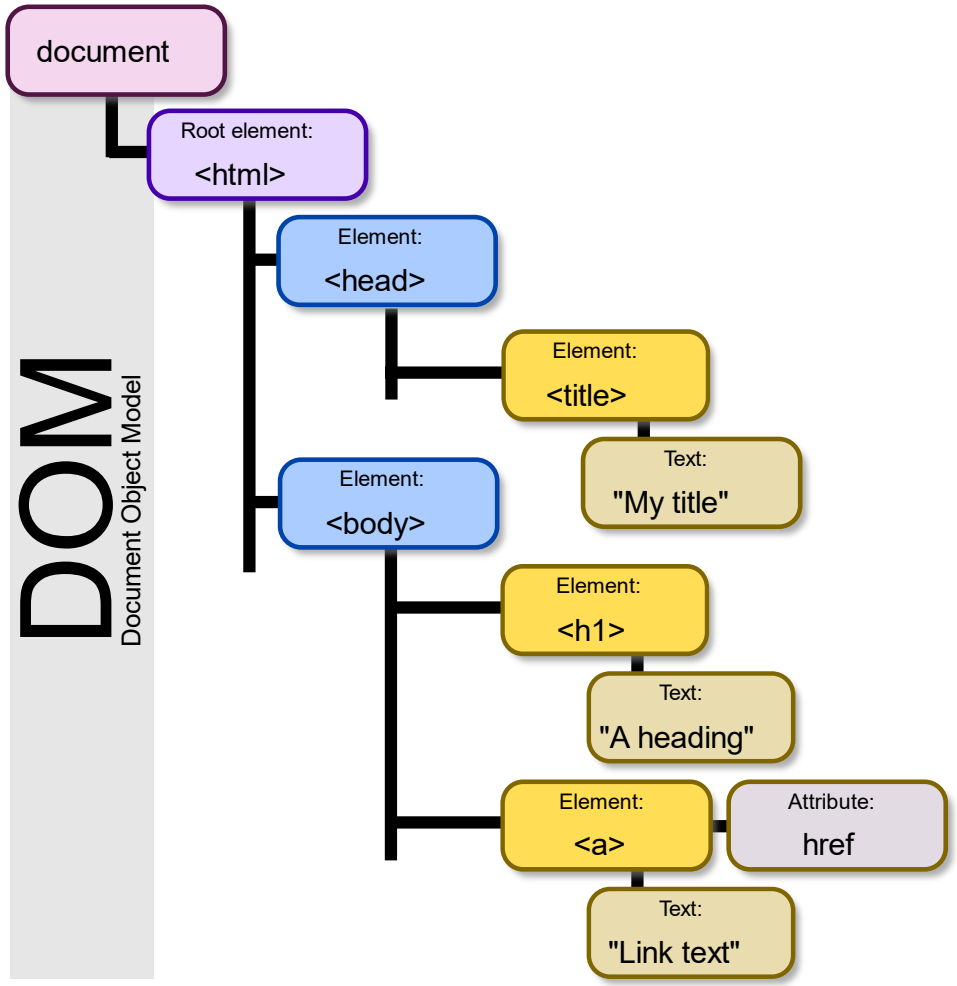
```
<html>
    <head>
        <script>
        function doAlert() {
            var msg = 'hello';
            alert(msg);
        }
        </script>
    </head>
    <body onload="doAlert();"></body>
</html>
```

[Observed HTML]

```
<html>
    <head>
        <script>
        function doAlert() {
            var message = 'hello';
            alert(message);
        }
        </script>
    </head>
    <body onload="doAlert();"></body>
</html>
```

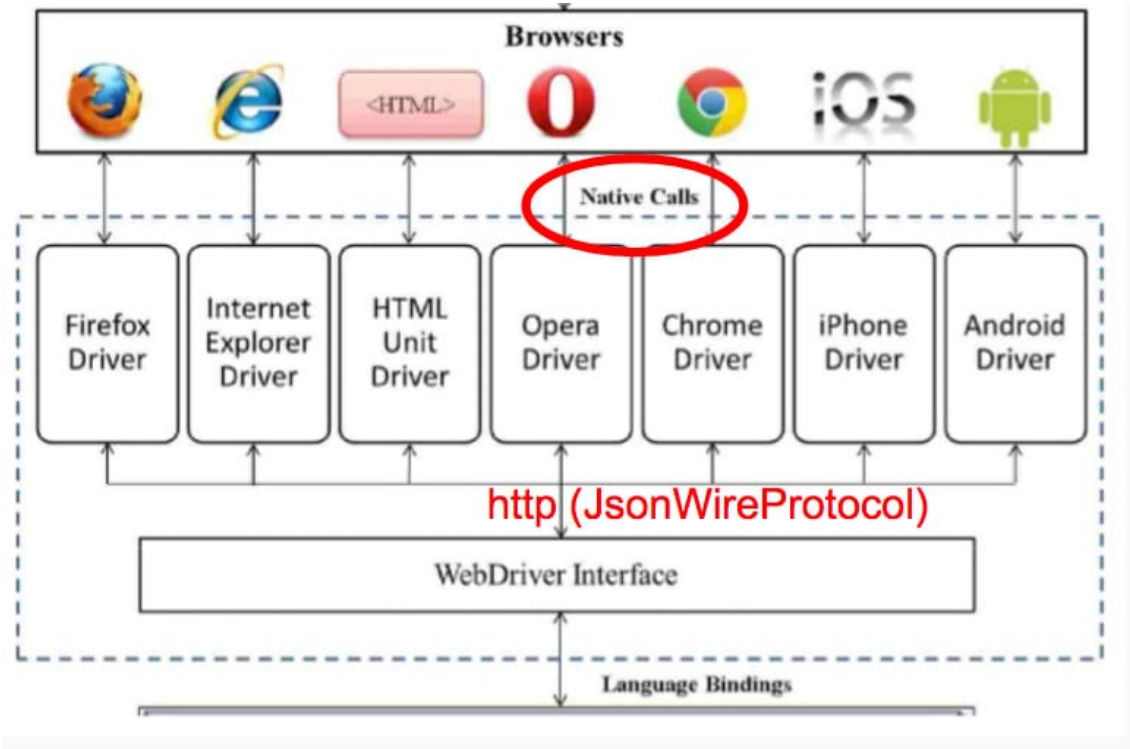
Solution: Web Testing Frameworks

- **Web testing framework**: Framework for testing web apps at a semantic level
  - **Robust**: Works at DOM level after parsing HTML into a DOM tree
  - **Targeted**: Provides APIs to target individual HTML elements
  - **Functionally tested**:
    - Provides APIs to call JavaScript code
    - Can emulate events like clicking or typing, which in turn invokes JavaScript code



Web drivers allows control of web browsers

- **Web driver**: An *interface* to control web browsers
  - Primarily intended to allow writing of automated tests
  - Also used to write scripts to automate repeated tasks (e.g. ordering groceries delivery)
- Web browser writers implement driver for their own web browser
  - Matching web driver is used to test web app on a particular browser platform



Selenium: A Web Testing Framework

- *Selenium*: An open-source web testing framework
  - Licensed under Apache License 2.0
  - Works with Windows, OS X, Linux, other OSes
  - Works with Java, Ruby, Python, other languages
- *Selenium = WebDriver + Grid + IDE*
  - *Selenium WebDriver*: Drivers available for Chrome and Firefox
  - *Selenium Grid*: Grid computing server to run Selenium tests in parallel https://www.selenium.dev/documentation/en/grid/
  - *Selenium IDE*: Browser extension for automated test script generation https://www.selenium.dev/selenium-ide/

Getting Started with Selenium IDE

1. Go to https://www.selenium.dev/selenium-ide/

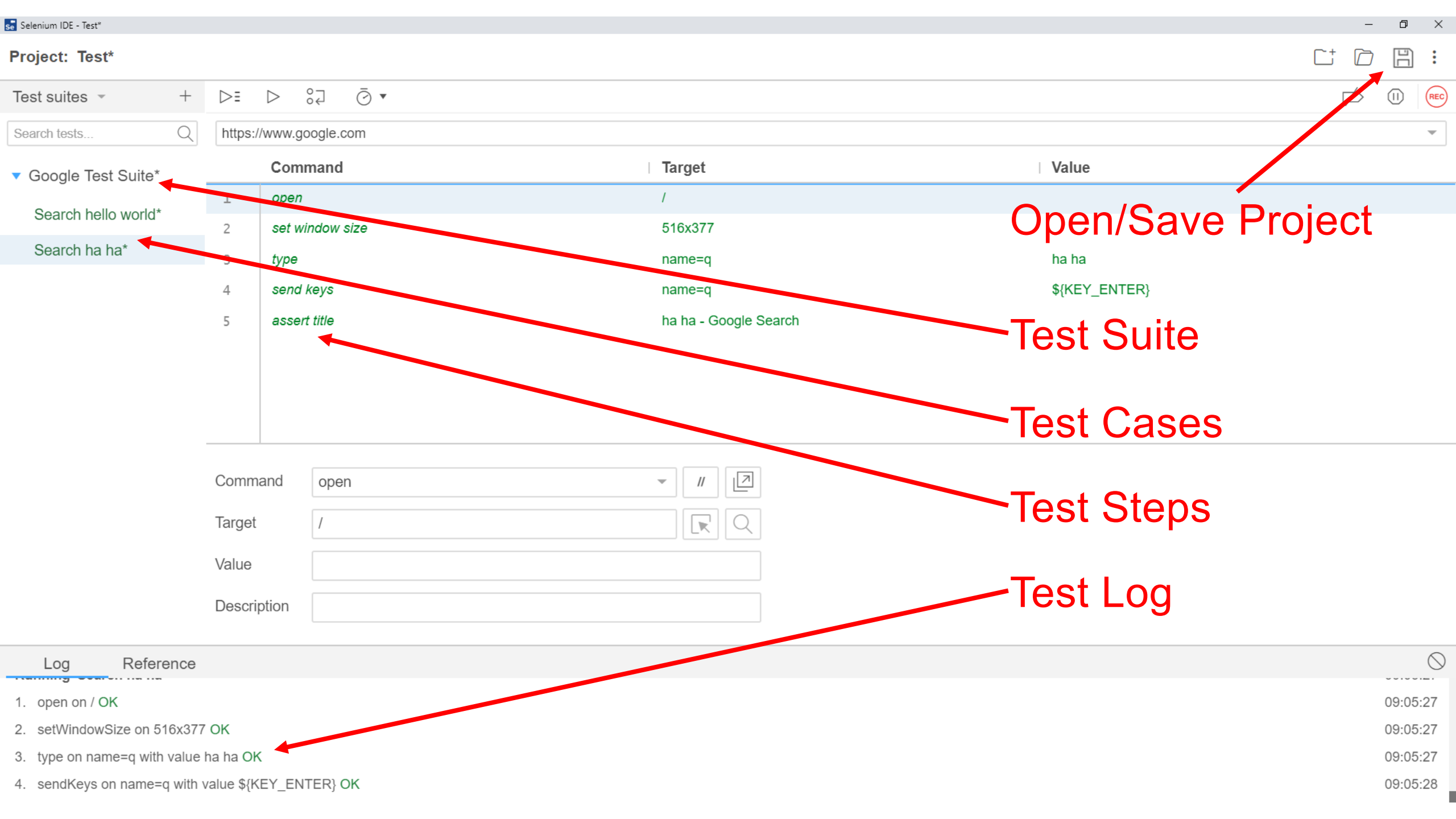
2. Add the appropriate extension for your web browser
   (By clicking on Chrome Download or Firefox download)

3. Install extension when taken to the webstore page

4. Click on the "Se" icon in the upper right-hand corner

Selenium IDE

- What we would call a “test plan”, Selenium calls a “test suite”
- Test suites contain test cases
- Test cases contain test steps


Selenium IDE - Test*
Project: Test*
Test suites
Search tests...
https://www.google.com
Google Test Suite*
Search hello world*
Search ha ha*
Command
Target
Value
1 open /
2 set window size 516x377
3 type name=q ha ha
4 send keys name=q ${KEY_ENTER}
5 assert title ha ha - Google Search
Command open
Target /
Value
Description
Log
Reference
1. open on / OK 09:05:27
2. setWindowSize on 516x377 OK 09:05:27
3. type on name=q with value ha ha OK 09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK 09:05:28
Open/Save Project
Test Suite
Test Cases
Test Steps
Test Log

Creating a Simple Test Case

1. Create a new test case
   - Select “Tests” in upper left dropdown box
   - Click on the “+” button
2. Record an operation (Press “REC” button)
3. Do something
4. Stop recording
5. Run test case - it does what you just did

Selenium IDE - Test

Project: Test

Tests

Search tests...

Search ha ha

Search hello world

https://www.google.com

1
2
3
4
5

Add new test case

TEST CASE NAME

Search yolo

ADD CANCEL

Command open

Target /

Value

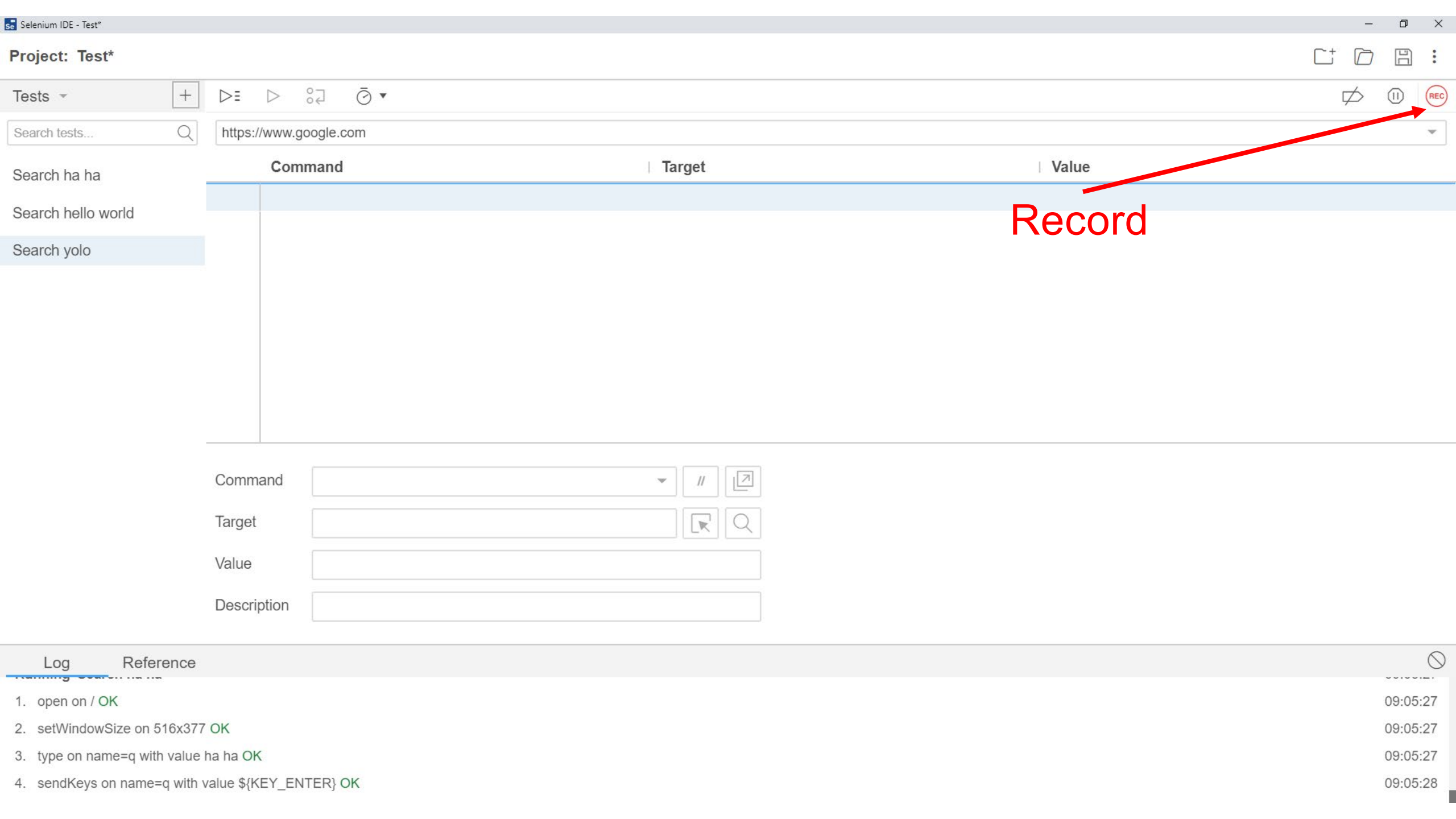
Description

Log Reference

1. open on / OK 09:05:27
2. setWindowSize on 516x377 OK 09:05:27
3. type on name=q with value ha ha OK 09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK 09:05:28

Type here to search

9:23 AM
2/24/2020


Selenium IDE - Test*
Project: Test*
Tests
Search tests...
Search ha ha
Search hello world
Search yolo
https://www.google.com
Command
Target
Value
Record
Command
Target
Value
Description
Log
Reference
1. open on / OK
09:05:27
2. setWindowSize on 516x377 OK
09:05:27
3. type on name=q with value ha ha OK
09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK
09:05:28

Selenium IDE - Test*

Project: Test*

Tests

Search tests...

- Search ha ha
- Search hello world
- Search yolo*

https://www.google.com

	Command	Target	Value
1	*open*	/	
2	*set window size*	516x377	
3	*type*	name=q	yolo
4	*send keys*	name=q	${KEY_ENTER}

Stop Recording

Command

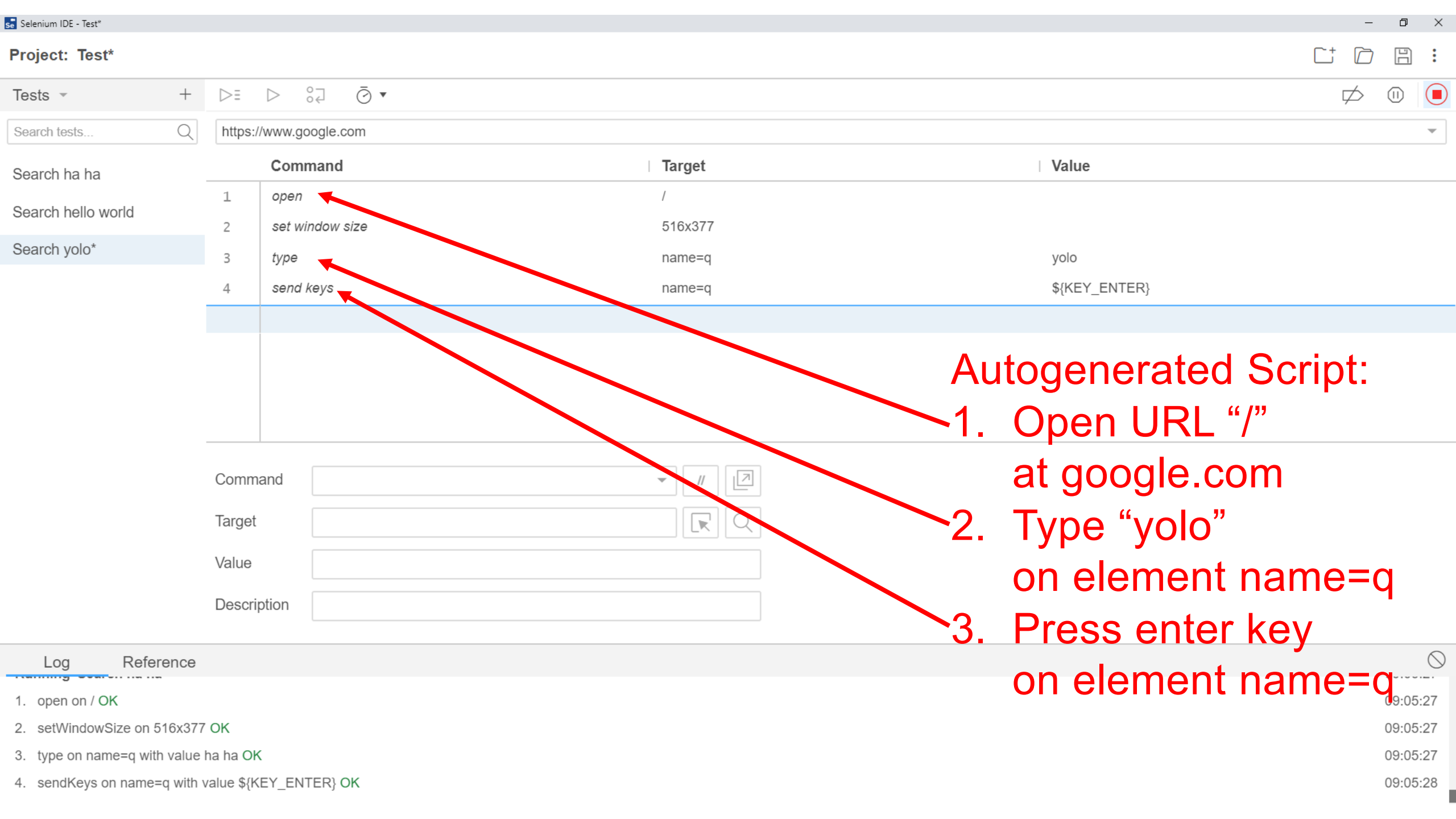
Target

Value

Description

Log Reference

1. open on / OK 09:05:27
2. setWindowSize on 516x377 OK 09:05:27
3. type on name=q with value ha ha OK 09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK 09:05:28


Selenium IDE - Test*
Project: Test*
Tests
Search tests...
Search ha ha
Search hello world
Search yolo*
https://www.google.com
Command
Target
Value
1 open / 516x377
2 set window size 516x377
3 type name=q yolo
4 send keys name=q ${KEY_ENTER}
Command
Target
Value
Description
Log
Reference
1. open on / OK 09:05:27
2. setWindowSize on 516x377 OK 09:05:27
3. type on name=q with value ha ha OK 09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK 09:05:28
Autogenerated Script:
1. Open URL "/" at google.com
2. Type "yolo" on element name=q
3. Press enter key on element name=q

Components of a Test Step

- Command – What to do
  - E.g. open a page, click on something, type
- Target – To what?
  - E.g. A URL or an element on the page
- Value – Using what value?
  - E.g. Type what?

Editing a Script After Recording

- You can modify a recorded script or create one from scratch
- Deleting a test step:
  1. Click on the test step you want to delete, highlighting it
  2. Press the “delete” key
- Modifying a test step:
  1. Click on the test step you want to modify, highlighting it
  2. Modify or choose Command / Target / Value / Description
- Adding a test step:
  1. Click on the row below the last command to add a test step, highlighting it
  2. Fill in or choose Command / Target / Value / Description
- Reordering test steps: just click and drag

Selenium IDE - Test*

Project: Test*

Tests

Search tests...

- Search ha ha
- Search hello world
- Search yolo*

https://www.google.com

	Command	Target	Value
1	*open*	/	
2	*set window size*	516x377	
3	*type*	name=q	yolo
4	*send keys*	name=q	${KEY_ENTER}
5	//		

Command

Target

Value

Description

- add selection
- answer on next prompt
- assert
- assert alert
- assert checked
- assert confirmation
- assert editable

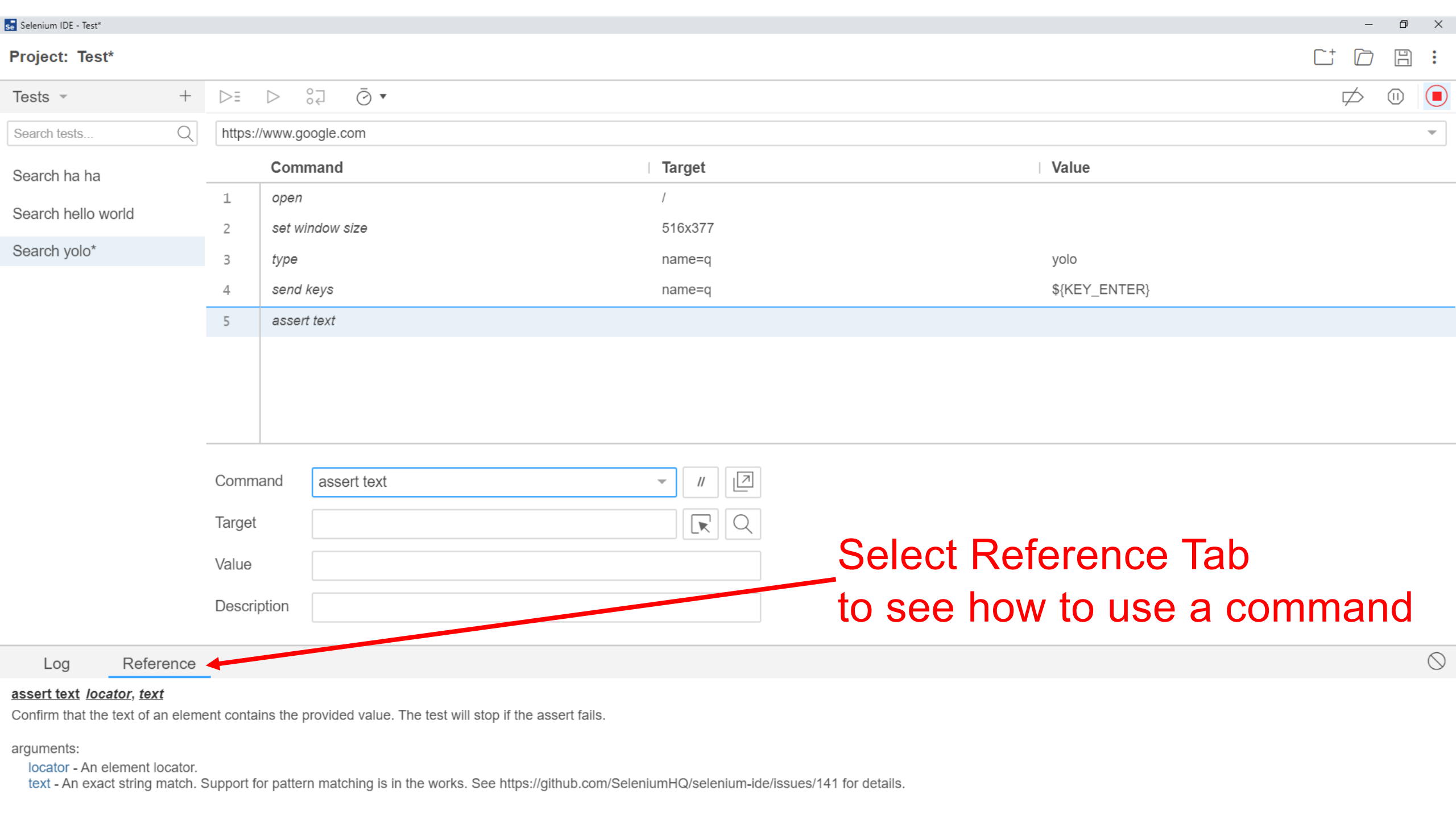
Pulldown to peruse;
Type to search command

Log Reference

1. open on / OK 09:05:27
2. setWindowSize on 516x377 OK 09:05:27
3. type on name=q with value ha ha OK 09:05:27
4. sendKeys on name=q with value ${KEY_ENTER} OK 09:05:28

Common Commands

- open url : Opens a URL and waits for the page to load before proceeding
  - Arguments: url - The URL to open (may be relative or absolute).
- click locator: Clicks on a target element (e.g. a link, button, or textbox)
  - Arguments: locator - An element locator
- type locator, value : Sets the value of an input field, as though you typed it
  - Arguments: locator - An element locator, value - The value to input
- assert text locator, text: Check that text of element contains provided value
  - Arguments: locator - An element locator, text - The provided string to match
- Note: 1st argument always goes to Target, 2nd argument to Value
  - If unsure of command, always use the “Reference” tab at the bottom


Selenium IDE - Test*
Project: Test*
Tests
Search tests...
Search ha ha
Search hello world
Search yolo*
https://www.google.com
Command
Target
Value
1 open /
2 set window size 516x377
3 type name=q yolo
4 send keys name=q ${KEY_ENTER}
5 assert text
Command assert text
Target
Value
Description
Select Reference Tab
to see how to use a command
Log
Reference
assert text locator, text
Confirm that the text of an element contains the provided value. The test will stop if the assert fails.
arguments:
locator - An element locator.
text - An exact string match. Support for pattern matching is in the works. See https://github.com/SeleniumHQ/selenium-ide/issues/141 for details.

Don’t Forget the Assertion in the End

- Purpose of a Selenium test case is to check a postcondition
  - Just like any other test case we have seen so far

- Like JUnit, assertions are used to check postconditions in Selenium

A Subset of Selenium Assertions…

- assert title text: Check that title of current page contains provided text
  - Arguments: text - The provided string to match
- assert element present locator: Check that element is present in page
  - Arguments: locator - An element locator
- assert text locator, text: Check that text of element contains provided text
  - Arguments: locator - An element locator, text - The provided string to match
- assert variable name, expected value: Check that variable is expected value
  - Arguments: variable name - The name of a variable
    expected value - The result you expect a variable to contain

Filling in Target Locator String

How do I generate the target locator string for this text element?

Different Types of Locator Strings

- *Locator*: A string that can uniquely identify an HTML element

- A *locator* can be expressed using:
  - Id: the **ID attribute** of the target element (guaranteed to be unique)
  - XPath: an **XML path** through the DOM hierarchy (very similar to file path)
  - CSS: a **CSS selector** for that HTML element
    - Leverages Cascading Style Sheets browser module for applying styles to elements

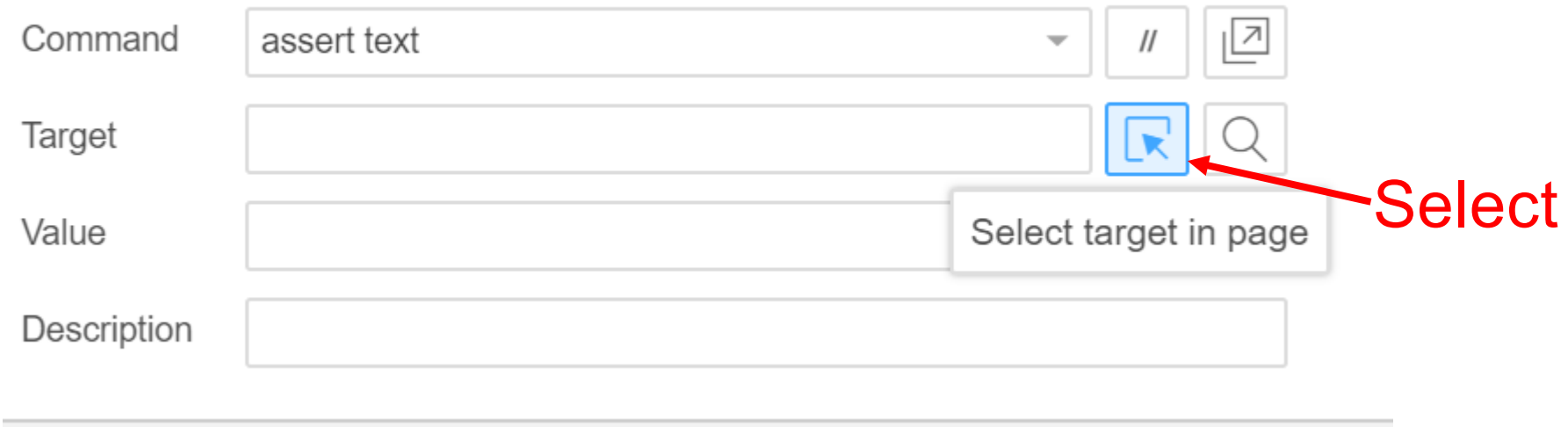
- There can be multiple XPaths and CSS selectors for same element
  - Much like multiple file paths (using absolute / relative, using wildcards, …)

Use Locator Appropriate for Test Scenario

- Choose locator that accurately implements execution step
  - Or you would not be performing the test intended

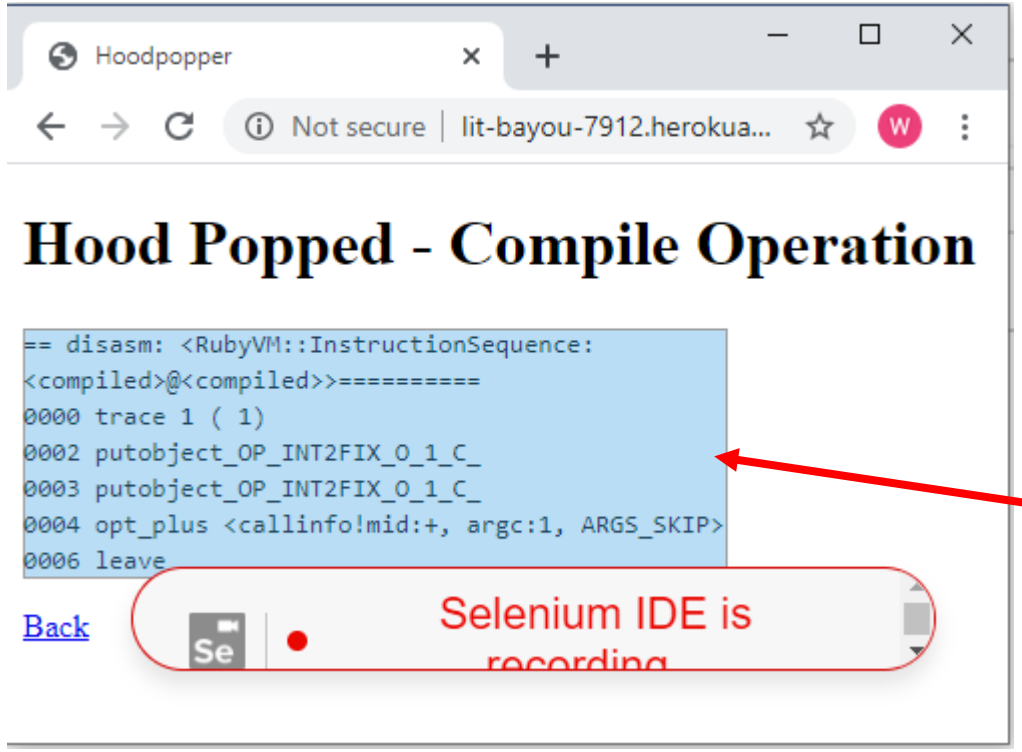
- Example execution steps and appropriate locators:
  - Click on the 3rd list item in the quick links menu:
    `xpath=//div[@id='menu-blockquick-links-3']/ul/li[3]/a`
  - Click on link text “Computing & Information”:
    `xpath=//a[contains(text(),'Computing & Information’)]`
  - Click on link to URL https://www.sci.pitt.edu:
    `xpath=//a[contains(@href, 'https://www.sci.pitt.edu')]`

Using Selector to Generate Locator String



- Selector button automatically generates locator strings for you!

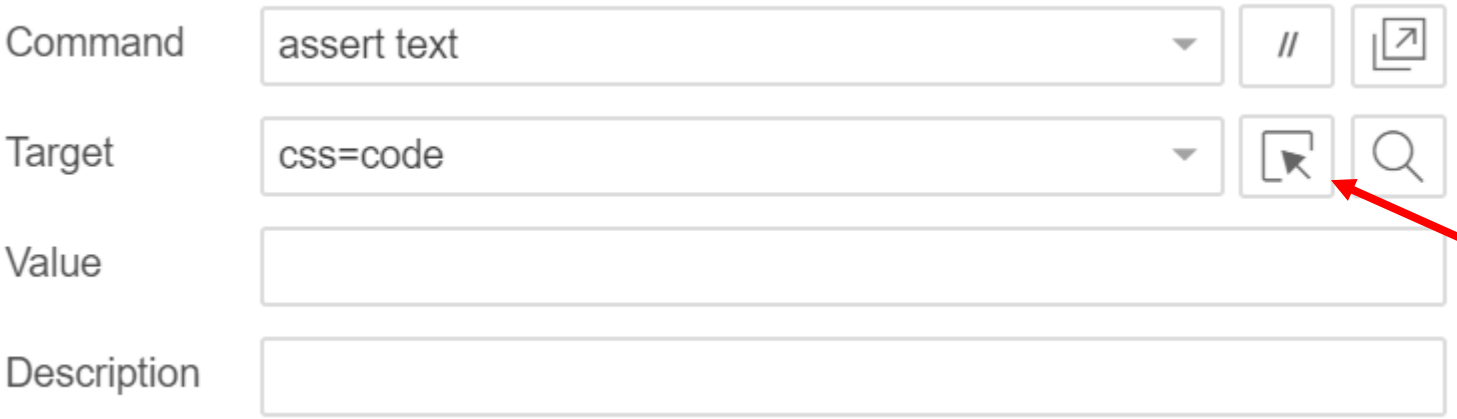
Using Selector to Generate Locator String



Click on HTML element you want to target

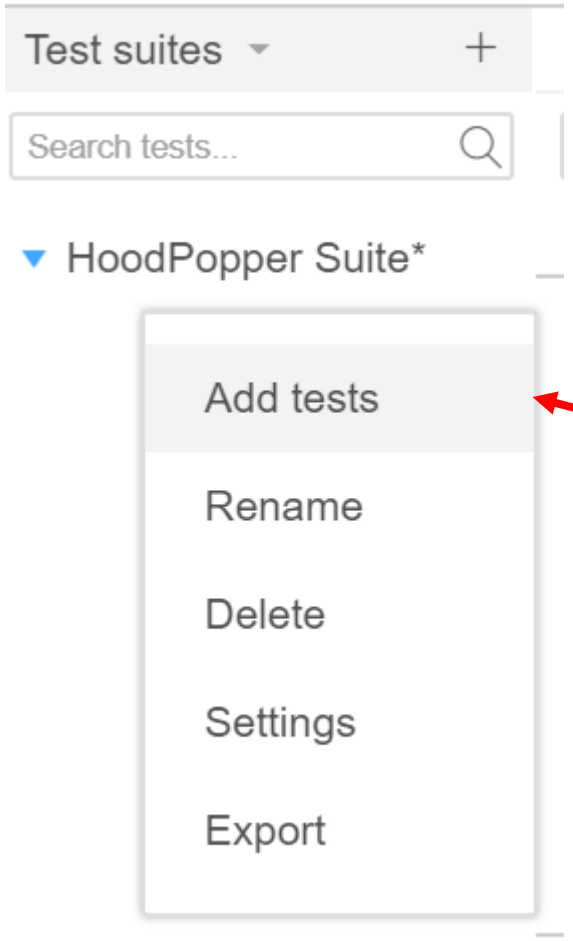
- Selected HTML element will highlight to blue when you hover over it

Using Selector to Generate Locator String

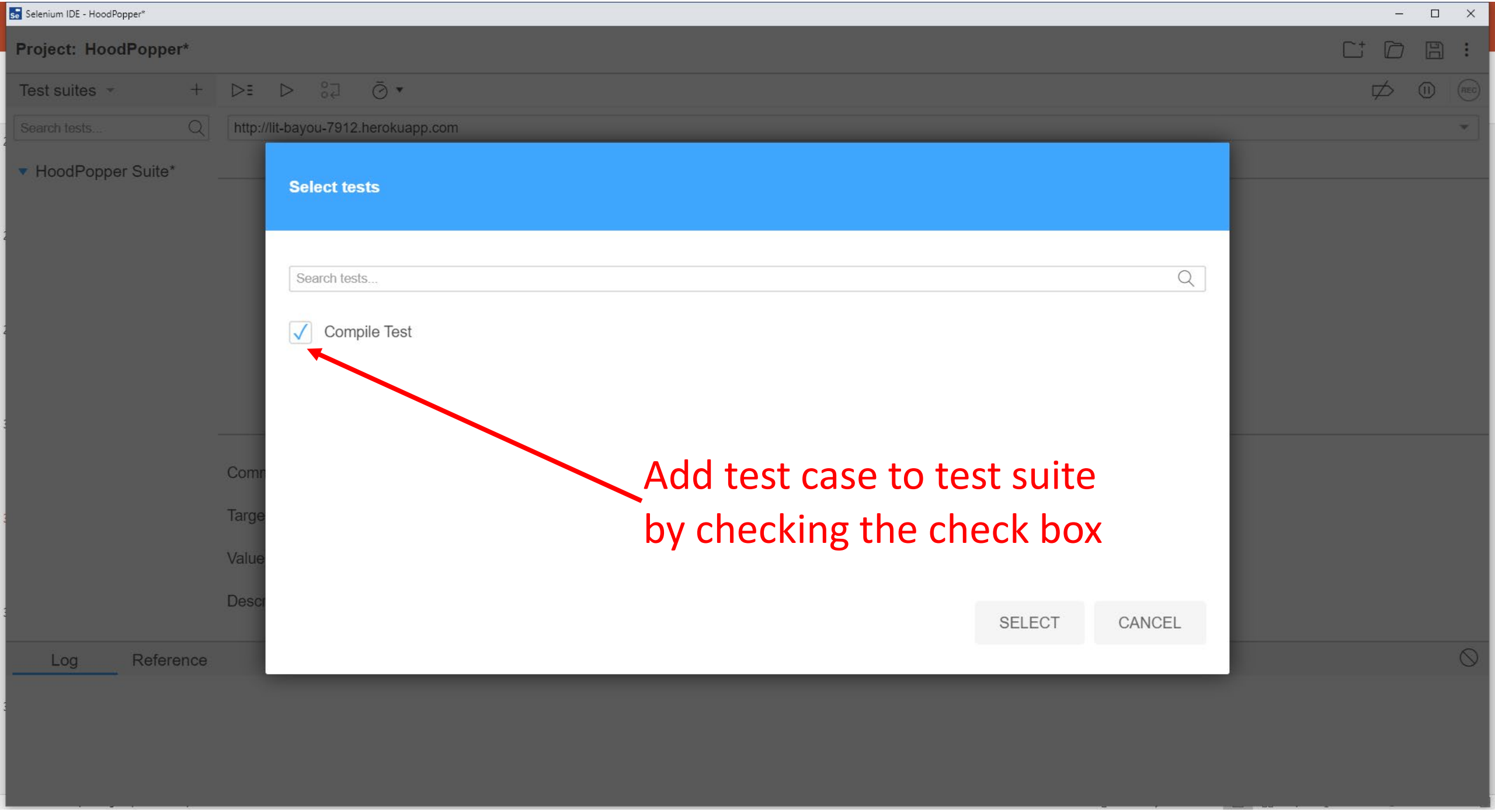


- Clicking on drop down button displays a list of options to choose from
- Choose the best option you think will be the most resilient to change
- You can even type in your own locator string (once you get the hang of it)

Adding a test case to a test suite



Select “Add tests” after right-clicking on Test Suite


Selenium IDE - HoodPopper*
Project: HoodPopper*
Test suites
Search tests...
http://lit-bayou-7912.herokuapp.com
HoodPopper Suite*
Select tests
Search tests...
Compile Test
Add test case to test suite
by checking the check box
SELECT
CANCEL
Log
Reference

No Textbook Reading for This Chapter

- Instead, please skim over: https://www.selenium.dev/selenium-ide/docs/en/api/commands
  - It shows all the assertions you can do and more!

- If you are interested, Appium is Selenium for Mobile Apps: http://appium.io/