

CS1632, LECTURE 12: SYSTEMS TESTING THE WEB WITH KATALON

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
- Automating testing for text-based programs is easy!
 - Just create an “input script” file and redirect to stdin
 - Redirect stdout to file and compare to expected output
- Turns out not everybody uses a text-based interface
 - GUIs, web pages, mobile applications, etc.
 - How do we deal with these?

Testing Concepts are Similar

- We will use different tools to test more complicated interfaces
- But the basic concepts remain the same
 - Testing is about comparing **observed behavior vs. expected behavior**
 - Preconditions
 - Execution steps
 - Postconditions

Testing the Web

- Just an example – other GUIs work similarly
- Keep in mind that we are going to *expect* certain things to occur or be seen, and then *observe* whether or not they occur or are seen.

Web = text

- Specially formatted and displayed text, but text!
- Text in the form of HTML

Theoretically, we could test web pages like so

```
// Any downsides to this?
@Test
public void testWeb() {
    String expectedHtml =
"<html><head></head><body>Hello world</body></html>";
    String pageText = getPage("http://example.com");
    assertEquals(expectedHtml, pageText);
}
```

Downsides

1. Fragile tests
 - Change the page, change the entire test.
2. Unreadable
 - Unclear which element in the HTML page tester is trying to check.
3. What about JavaScript?
 - Just check JS code letter by letter?
 - Remember JS is also a program. We should check functionality not code.
4. Simplistic and low-level
 - No semantic understanding (e.g. of links, textboxes)
 - No sense of HTML hierarchy
 - Tries to deal with a flat text string – almost like assembly programming

Web Testing Frameworks

- Sure, you could program everything in assembly
 - You could yourself parse HTML into a tree data structure
 - You could yourself interpret and run JS code on HTML elements
- Why not have a library that does all of that for you?
 - That creates the HTML tree for you and provides APIs to search it
 - That handles various events like clicking on a button or typing
 - That exercises JS code for you to modify the HTML page
- That library is a *web testing framework*
 - Framework often also includes an IDE to auto-generate test code, by simply interacting with the website on a web browser.

What is Selenium?

- An open-source web testing framework.
- Battle-hardened.
- Works with Windows, OS X, Linux, other OSes.
- Works with Java, Ruby, Python, other languages.
- Works with most modern web browsers.
- Has its own IDE.
- Can also be used for quick scripting.

What is Selenium?

- An open-source web testing framework.
- ~~• Battle-hardened.~~
- ~~• Works with Windows, OS X, Linux, other OSes.~~
- ~~• Works with Java, Ruby, Python, other languages.~~
- ~~• Works with most modern web browsers.~~
- ~~• Has its own IDE.~~
- ~~• Can also be used for quick scripting.~~

Ref: <https://seleniumhq.wordpress.com/2017/08/09/firefox-55-and-selenium-ide/>

What is Katalon?

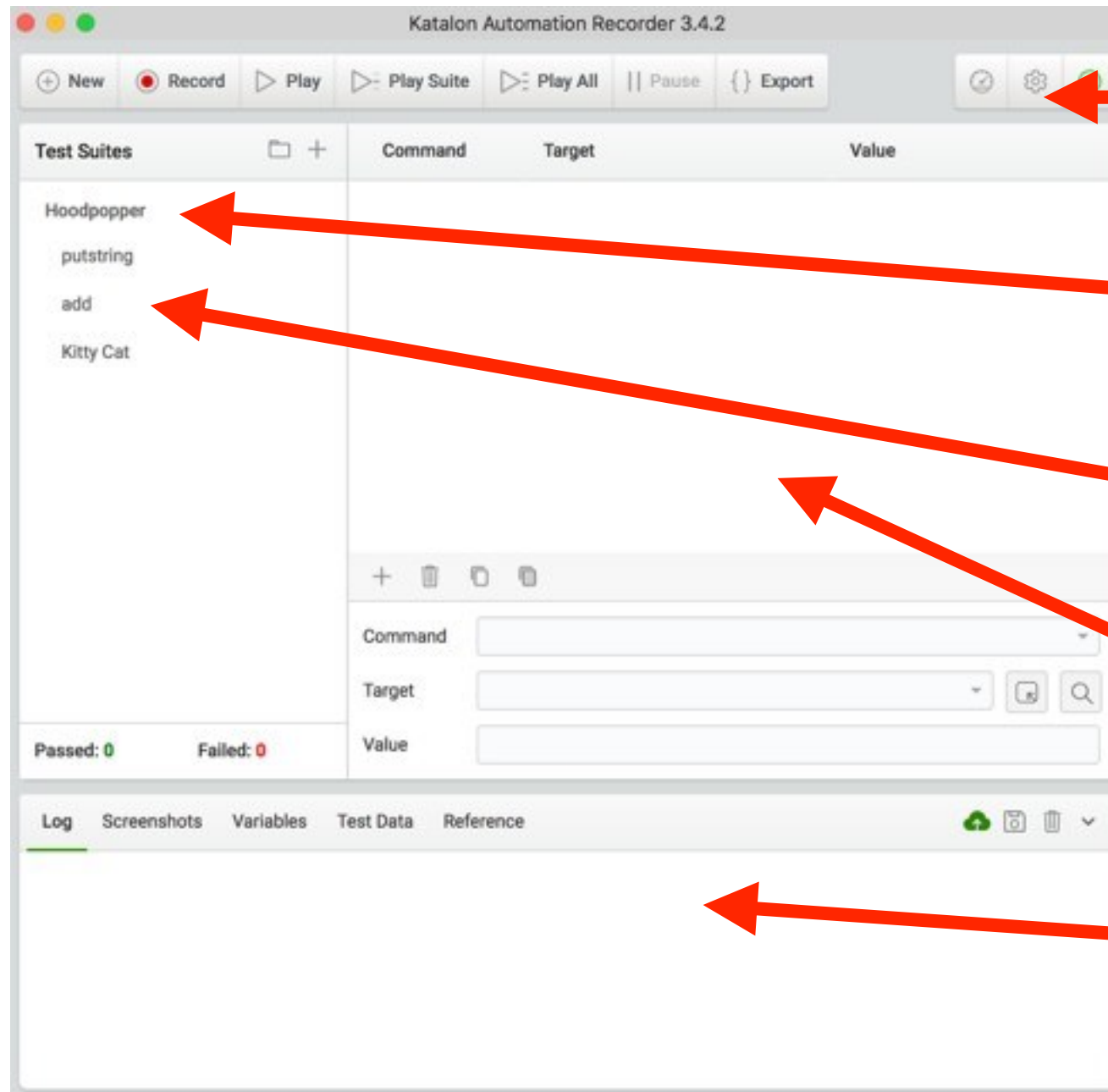
- The Selenium core library is fine, it's the IDE that's obsolete
- Lots of 3rd party IDEs using Selenium core have popped up
- One of them is Katalon
 - Provides all the features of Selenium IDE and more
 - Hides most of the technical complexities making it easier for testers
 - Is increasing its market share

Getting Started with Katalon

1. Download Chrome (if you have not)
2. Go to Chrome Web Store
3. Add extension Katalon Automation Recorder.
4. Click on the "K" icon in the upper right-hand corner

Katalon

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



Settings

Test Suite

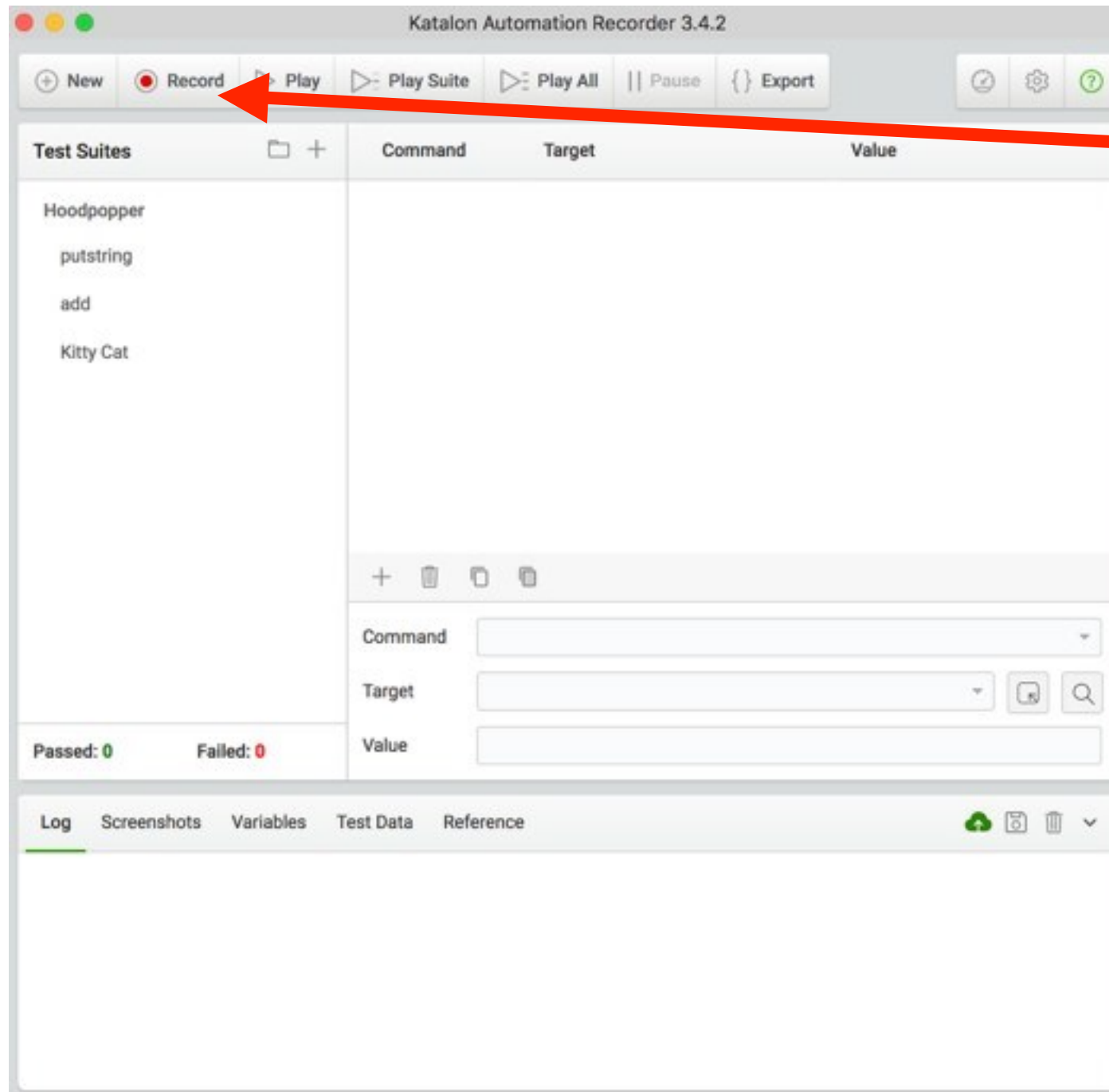
Test Cases

Test Steps

Logging output

Simple Scripting

1. Create a new test case
2. Record an operation (Press “Record”)
3. Do something
4. Stop recording
5. Run test case - it does what you just did













Record

USD ▾

Next 100 → View All

All ▾ Coins ▾ Tokens ▾

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Graph (7d)
1	 Bitcoin	\$156,374,192,171	\$9,243.23	\$6,012,490,000	16,917,700 BTC	0.04%	
2	 Ethereum	\$68,205,894,372	\$694.76	\$1,435,570,000	98,171,455 ETH	-0.85%	
3	 Ripple	\$30,948,560,140	\$0.791691	\$279,725,000	39,091,716,516 XRP *	-1.08%	
4	 Bitcoin Cash	\$18,167,322,569	\$1,067.64	\$459,296,000	17,016,338 BCH	0.74%	
5	 Litecoin	\$9,796,148,182	\$176.14	\$466,743,000	55,616,956 LTC	-1.65%	





Do
stuff

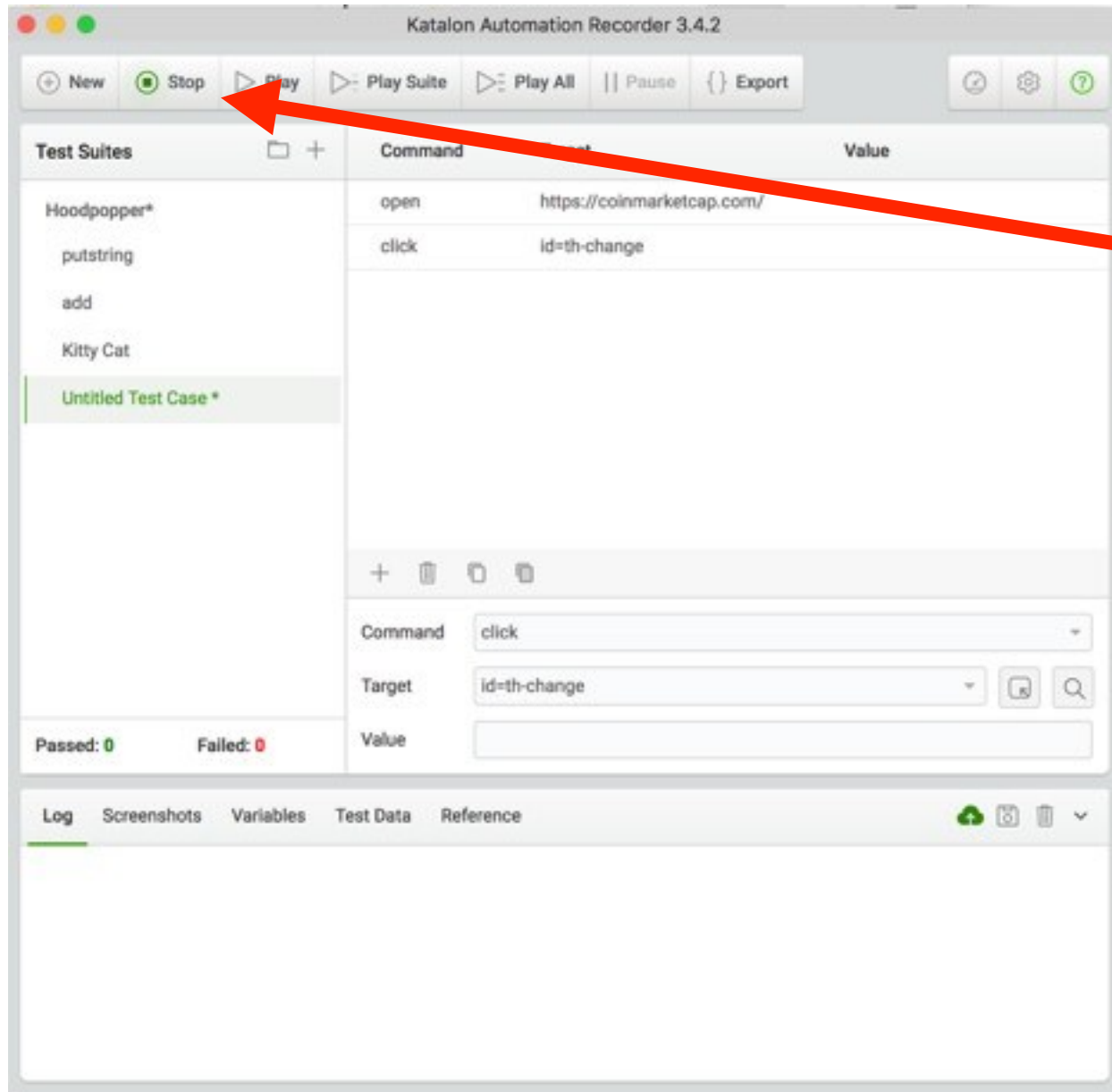
USD ▾

Next 100 → View

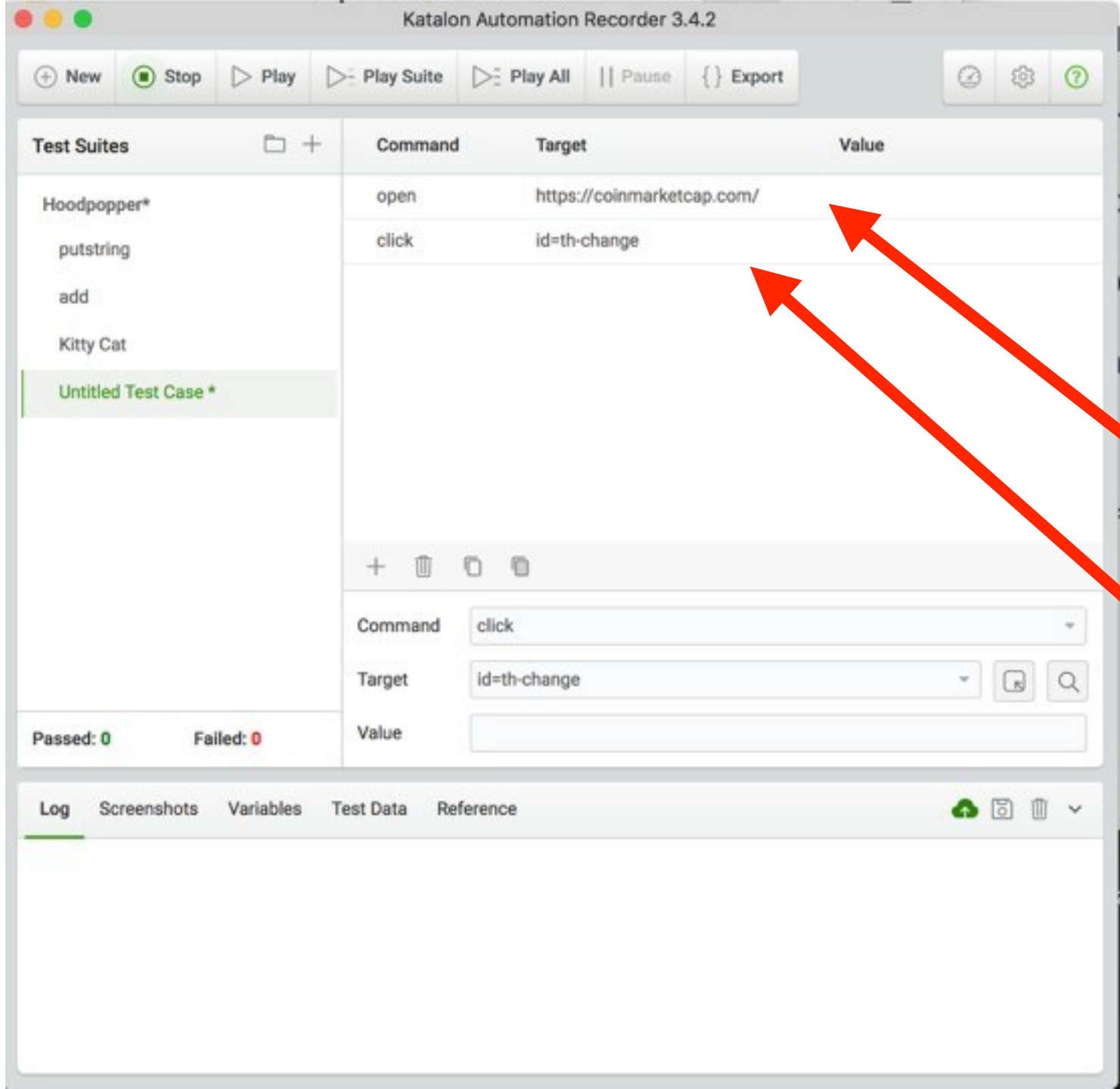
All ▾ Coins ▾ Tokens ▾

I pressed
this...

#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)
25	 Binance Coin	\$987,533,952	\$9.97	\$219,944,000	99,014,000 BNB *	23.56%
72	 Power Ledger	\$186,992,387	\$0.513810	\$10,523,500	363,932,947 POWR *	18.65%
46	 Electroneum	\$348,172,367	\$0.054266	\$2,736,370	6,416,055,176 ETN	17.46%
47	 Ardor	\$343,571,910	\$0.343916	\$5,369,710	998,999,495 ARDR *	17.05%



Stop
recording



Auto-generated
web testing
script:

open URL

Click on element
with id=th-change

You can add your own commands

- Modify a recorded script or create from scratch
- Click “+” button to add a test step
- Can then click to modify
- Note that it is NOT a textbox, so it is a little awkward to use

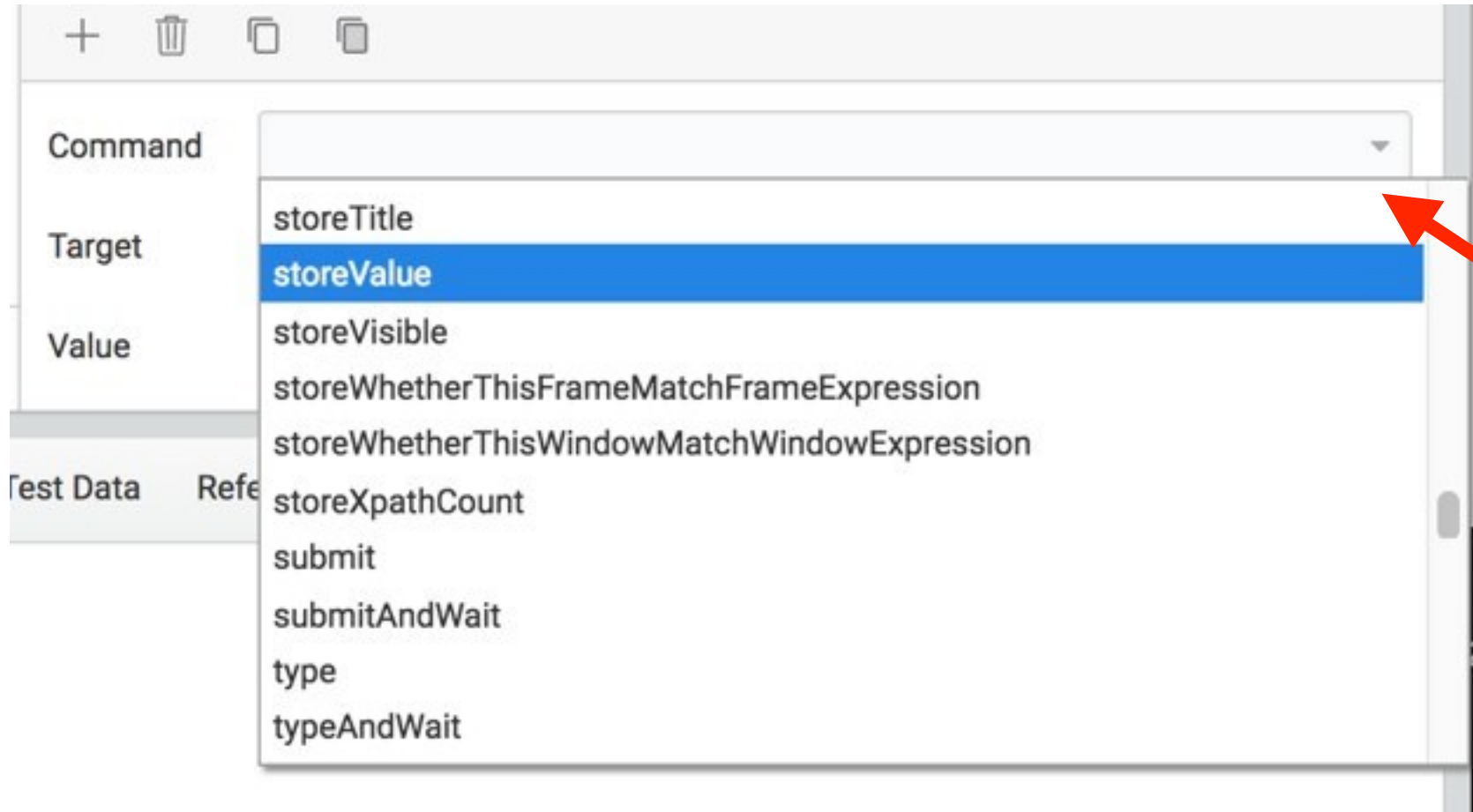
The test step is divided into three parts

- 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 - How?
 - E.g. Type what?

Common Commands

- open - open a URL
- click - click on a web element
- type - type something in a web element
- assert - assert that something is true
 - The *sine qua non* of testing!

Many, many, many others



Pull down to
peruse

Hello, assertions, my old friend...
I've come to assert you again..

- We are going to use assertions to specify expected behavior
- Same as traditional Junit assertions, just at different level of abstraction

A subset of Katalon (Selenium) assertions...

- **assertText** - Assert that text for element equals a regex
- **assertTextPresent** - Assert that regex exists somewhere on the page
- **assertElementPresent** - Assert that element exists somewhere on page
- **assertCookie** - Assert that a cookie exists
- **assertAlert** - Assert that an alert took place
- **assertEditable** - Assert that an element is editable
- **assertEval** - Evaluate some JavaScript and assert the result

Hood Popped - Compile Operation

```
== disasm: <RubyVM::InstructionSequence:<compiled>@<compiled>>=====
0000 trace 1 ( 1)
0002 putobject_OP_INT2FIX_O_1_C_
0003 putobject_OP_INT2FIX_O_1_C_
0004 opt_plus <callinfo!mid:+, argc:1, ARGS_SKIP>
0006 leave
```

[Back](#)

I want to assert something about this particular text section...
but how?

Select can help specify a target



- Lots of ways to specify an element on a webpage
 - CSS
 - xpath
 - id
 - Other tag
- Select can help you find one that works
 - It will find a value which uniquely identifies that element

+

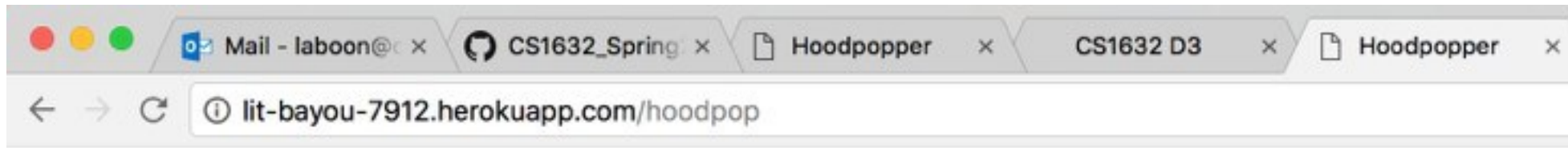
🗑

📄

📄

Command	assertText ▾		
Target	//p ▾		
Value	*putstring*		

Select



Hood Popped - Compile Operation

```
== disasm: <RubyVM::InstructionSequence:<compiled>@<compiled>>=====
0000 trace 1 ( 1)
0002 putobject_OP_INT2FIX_O_1_C_
0003 putobject_OP_INT2FIX_O_1_C_
0004 opt_plus <callinfo!mid:+, argc:1, ARGS_SKIP>
0006 leave
```

[Back](#)

Click on what you want to select

+

🗑

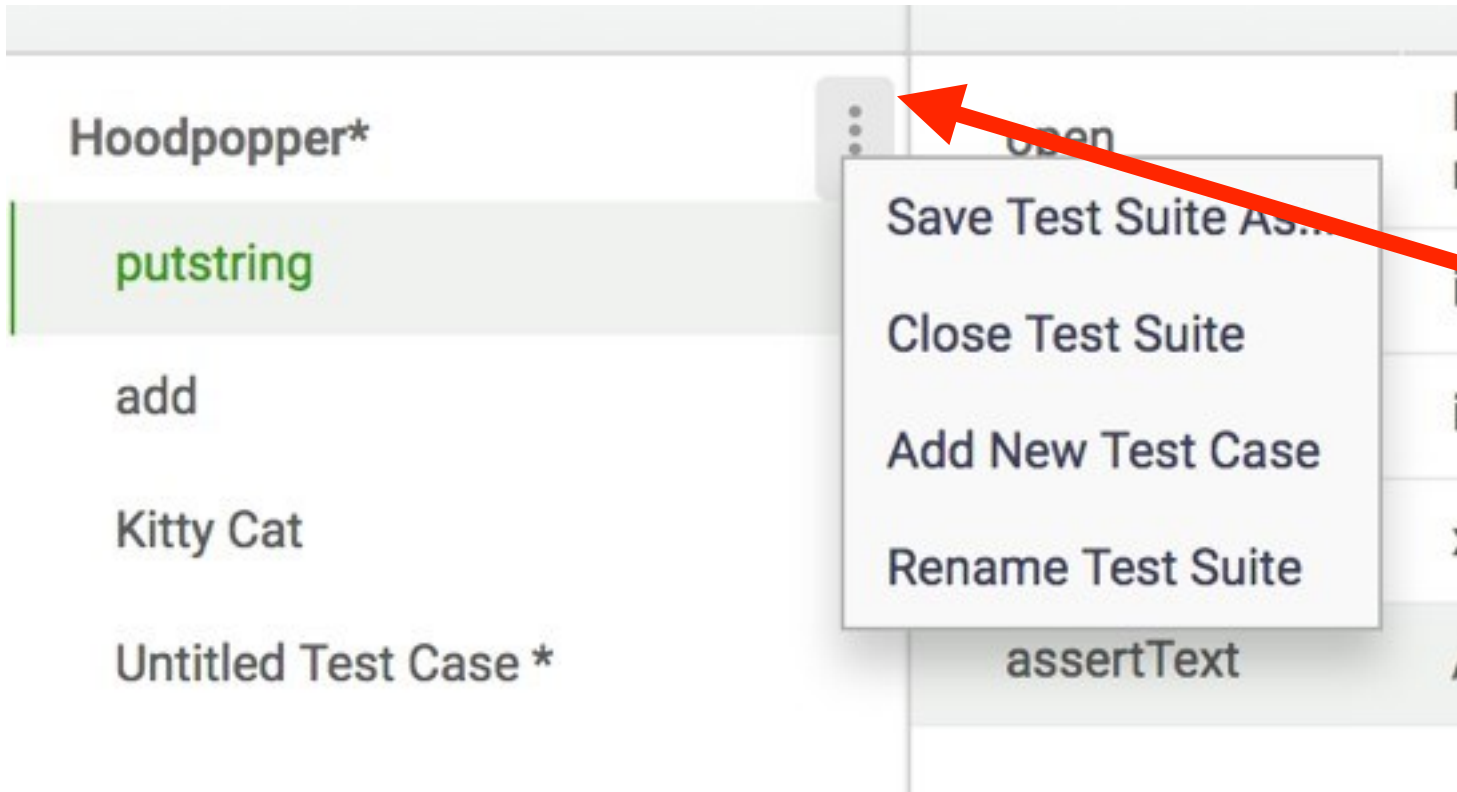
📄

📄

Command	assertText		
Target	//p	<div>📄</div>	<div>🔍</div>
Value	*putstring*		

Finds a way to uniquely specify
that element!

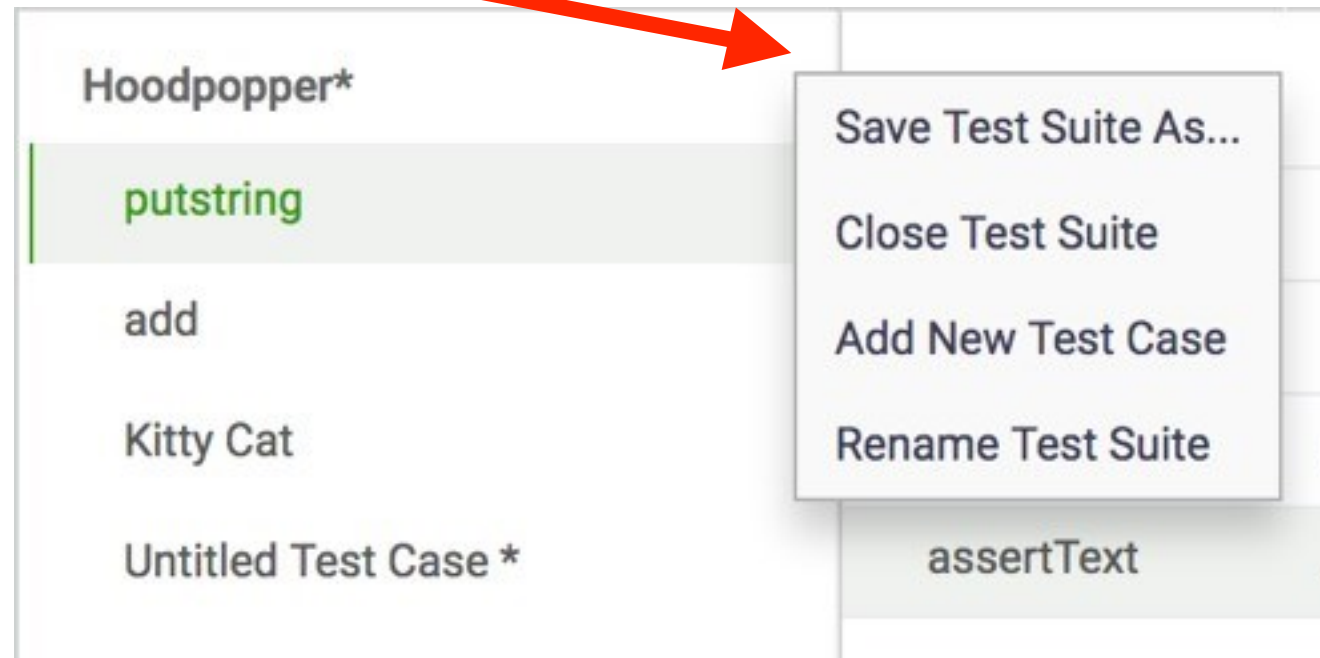
Adding a test case to a test suite



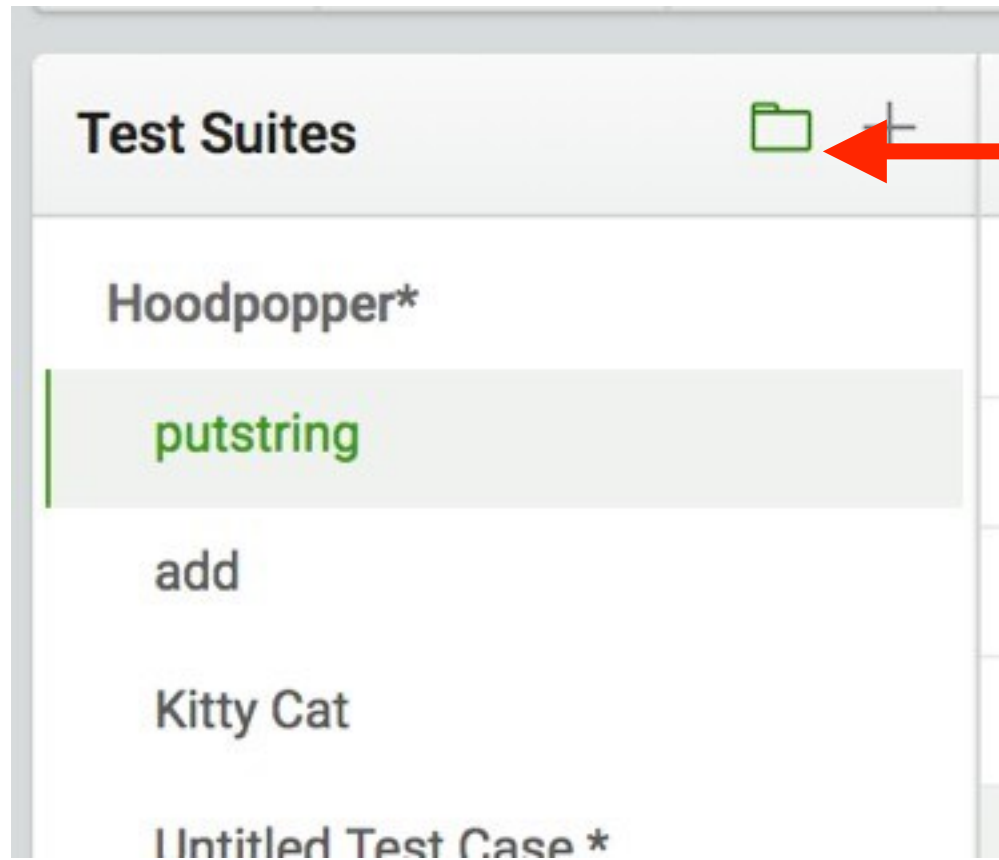
Click on the three dots next to the test suite and "Add New Test Case"

Saving A Test Suite

- Save test suite by pressing button with three dots next to it and “Save Test Suite As...”
- Note: the button only appears when you hover over it. It shows up around here



Opening a Test Suite



Click the “file” button to open a saved test suite

Let's Walk Through A Test Case

No Textbook Reading for This Chapter!

- Yay.
- Instead, please skim over:
<https://docs.katalon.com/katalon-recorder/docs/selenese-selenium-ide-commands-reference.html>
- It shows all the assertions you can do and more!