



CS1632, LECTURE 11: SYSTEMS TESTING THE WEB WITH KATALON

Bill Laboon



Background

- So far, all of our testing has been with specific, often text-based, input and output
- Turns out not everybody uses a text-based interface
- GUIs, web pages, mobile applications, etc.

Testing Techniques are Similar

We can actually use many of the same tools and techniques to test more complicated interfaces. Now that you understand the basics of automated testing, it's possible to take what you've learned and apply it to a more complex interface.

But this remains:

EXPECTED BEHAVIOR vs OBSERVED BEHAVIOR

Testing the Web

- Just an example – similar ideas for testing other graphical or non-textual / mathematical interfaces
- 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!
- If your computer can process it, it's just 1s and 0s, which can be represented as text

Theoretically, we could test web pages like so...

```
# Any downsides to this?
def test_the_web
    expected_html = "<html><head>" +
        "</head><body><strong>Hello, world!</strong>" +
        "</body></html>"
    page_text = getPage "http://example.com"
    assert_equal expected_html, page_text
}
```

Downsides

1. Change the page, change the entire test
 - * Fragile tests!
2. What about JavaScript?
 - * Just check that the right JS is on page?
3. Unreadable
4. Simplistic and low-level
 - * Kind of like programming in assembly
5. No semantic understanding (e.g. of links, textboxes)

Web Testing Frameworks

Think of these as a higher-level programming language for testing web pages.

Sure, you could program everything in assembly, but this is rarely ideal.

Another way to think of it – just extra libraries so you don't have to program everything yourself.

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 Katalon?

“a better solution than Selenium based open source frameworks”

(after battling with Capybara and Selenium, I am inclined to agree with them)

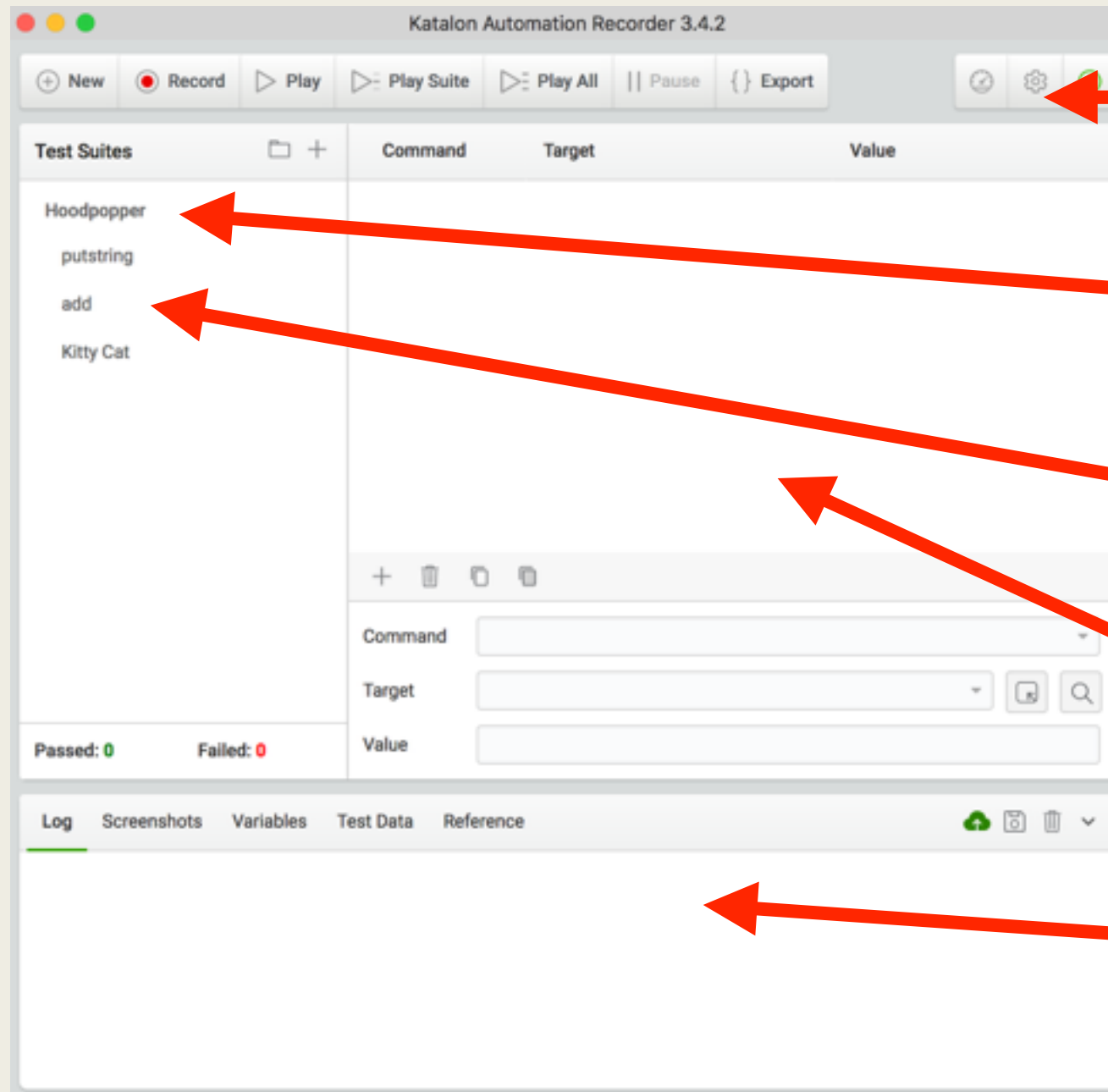
Almost the same interface, but with cool new features such as “works with browsers released after 2014”

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”, Selenium and Katalon call “test suites”
- Test suites contain test cases
- Test cases contain test steps



Settings

Test Suite

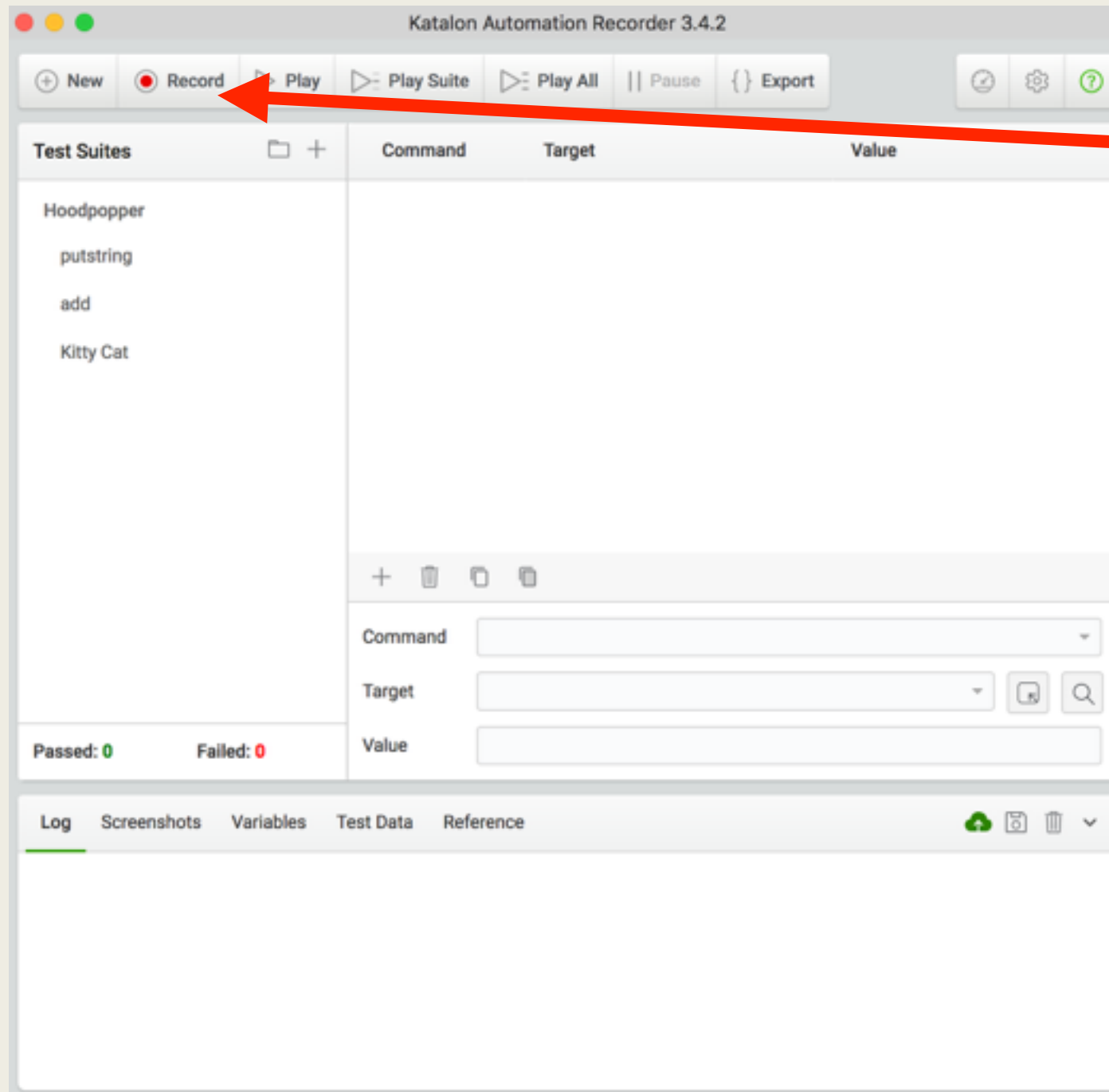
Test Cases

Commands

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

Secure | <https://coinmarketcap.com>





USD ▾

Next 100 → View All

All ▾ Coins ▾ Tokens ▾

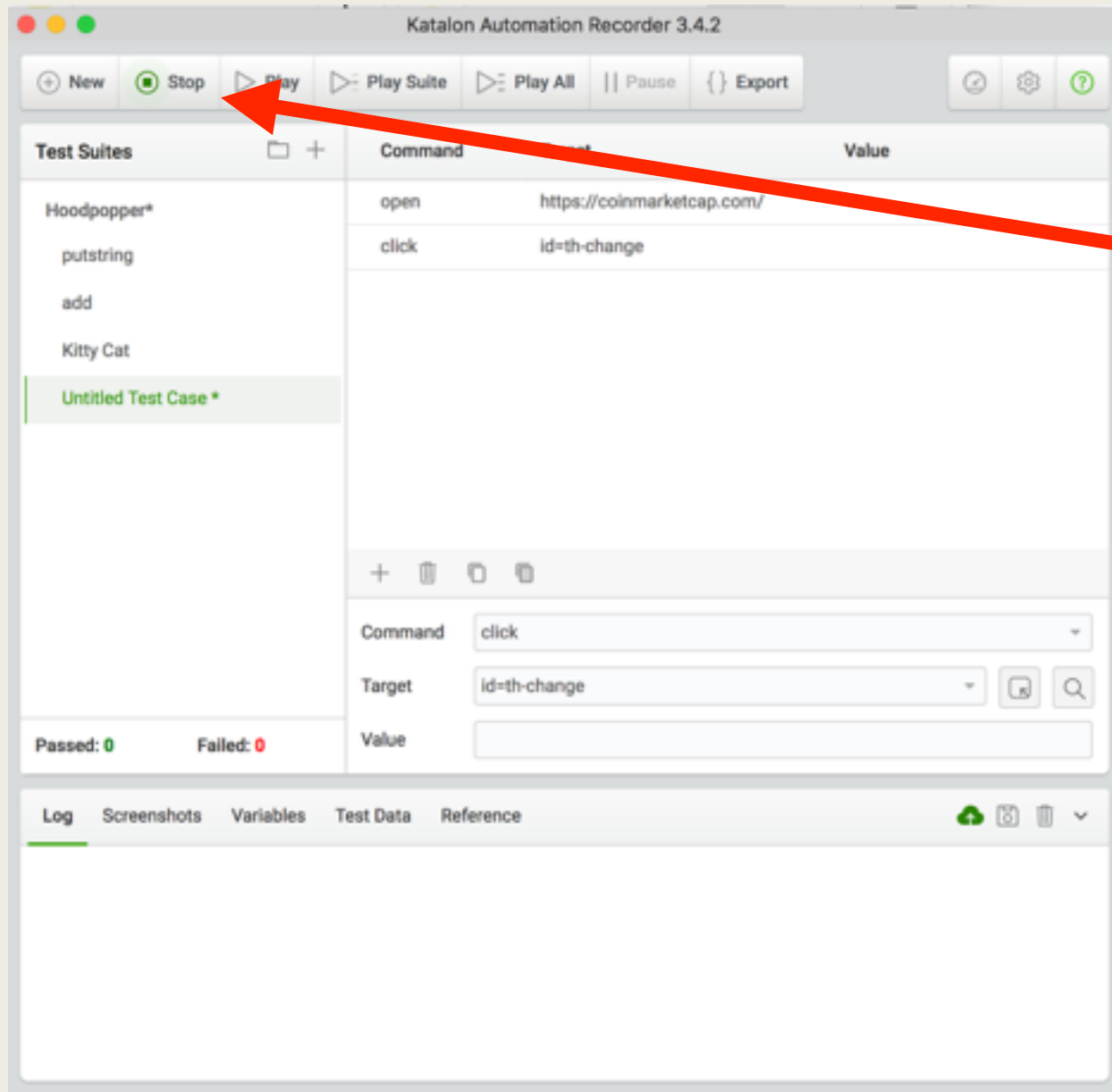
#	Name	Market Cap	Price	Volume (24h)	Circulating Supply	Change (24h)	Graph (7d)
1	 Bitcoin	\$156,374,192,171	\$9,243.23	\$6,012,490,000	16,917,700 BTC	0.04%	
2	 Ethereum	\$68,205,894,372	\$694.76	\$1,435,570,000	98,171,455 ETH	-0.85%	
3	 Ripple	\$30,948,560,140	\$0.791691	\$279,725,000	39,091,716,516 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

Secure		https://coinmarketcap.com						
USD ▾							Next 100 →	View
All ▾	Coins ▾	Tokens ▾						
#	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%		

I pressed
this...





Stop
recording

Katalon Automation Recorder 3.4.2

⊕ New ⊖ Stop ▶ Play ▶ Play Suite ▶ Play All || Pause {} Export ⌚ ⚙️ ?

Test Suites	Command	Target	Value
Hoodpopper*	open	https://coinmarketcap.com/	
putstring	click	id=th-change	
add			
Kitty Cat			
Untitled Test Case *			

Passed: 0 Failed: 0

Log Screenshots Variables Test Data Reference

Command: click
Target: id=th-change
Value:

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

Test step est omnis divisa in partes tres (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?)

Hello, assertions, my old friend...

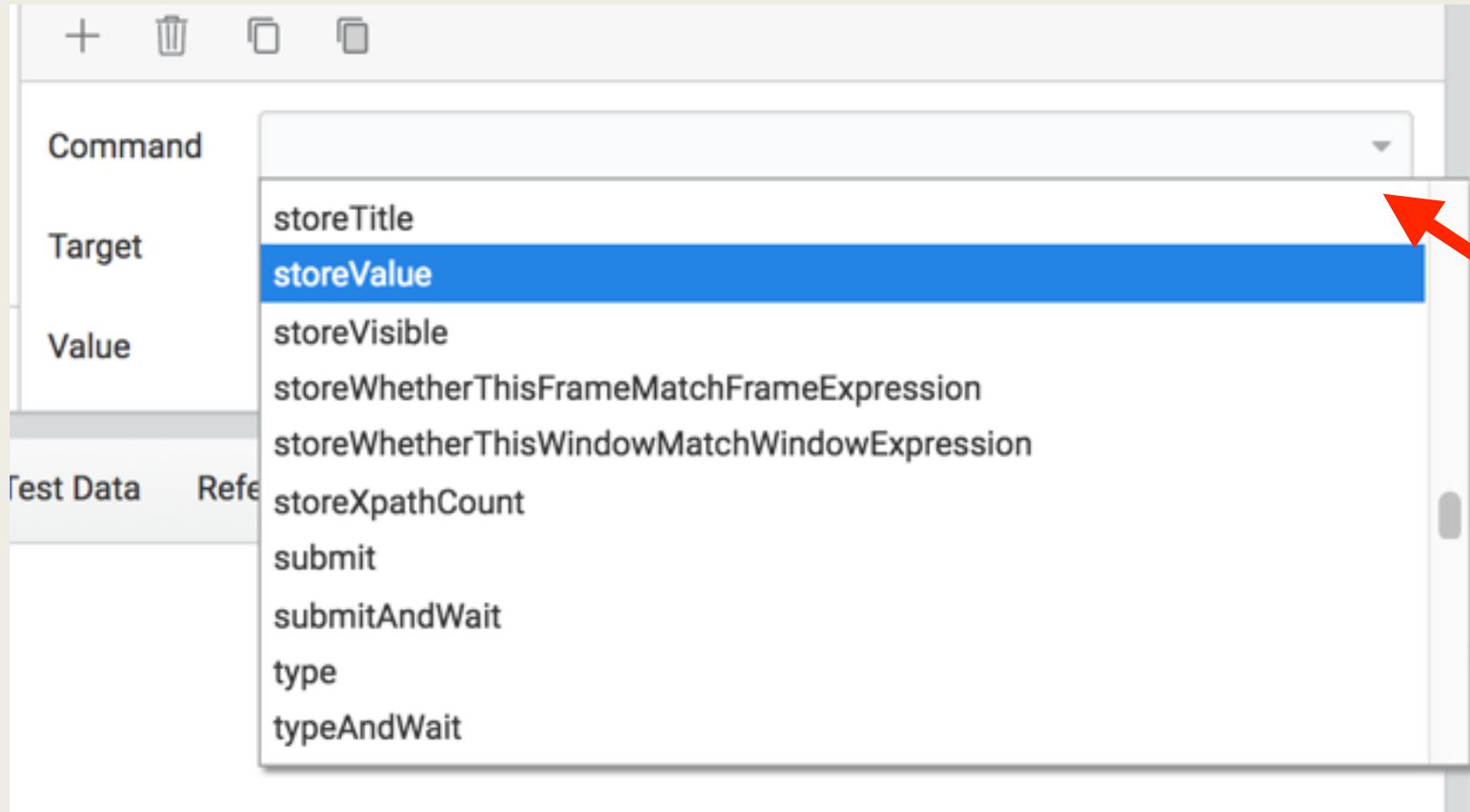
I've come to assert you again..

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

Common Commands

- open - open a URL
- click - click on a web element
- type - type something in a web element

Many, many, many others



Pull down to
peruse

Select can be helpful to find a way to 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

```
== 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?

Command

assertText

Target

//p

Value

putstring

Select

Mail - laboon@c x CS1632_Spring x Hoodpopper x CS1632 D3 x Hoodpopper x

lit-bayou-7912.herokuapp.com/hoodpop

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

+

🗑️

📄

📄

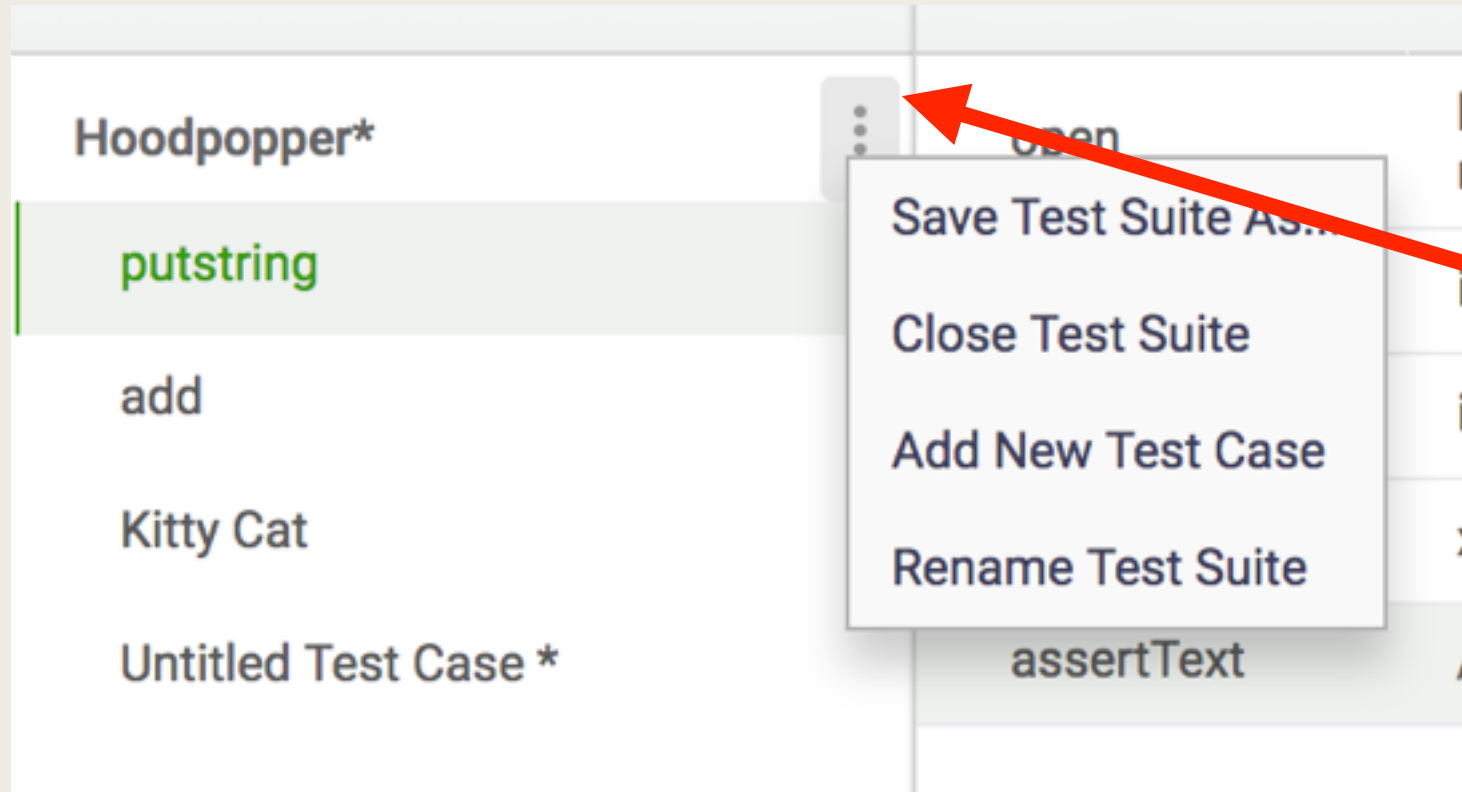
Command	assertText		
Target	//p	<div>📄</div>	<div>🔍</div>
Value	*putstring*		

Finds a way to uniquely specify that element!

Lots of fun assertions...

- **assertText / assertTextPresent** - Assert that text exists (on an element (former) or entire page (latter)). Note that this is a regex!
- **assertCookie** - Assert that a cookie exists.
- **assertElementPresent** - Assert that an element exists somewhere on the page.
- **assertAlert** - Assert that an alert took place.
- **assertEditable** - Assert that an element is editable.
- **assertEval** - Evaluate some JavaScript and assert the result.

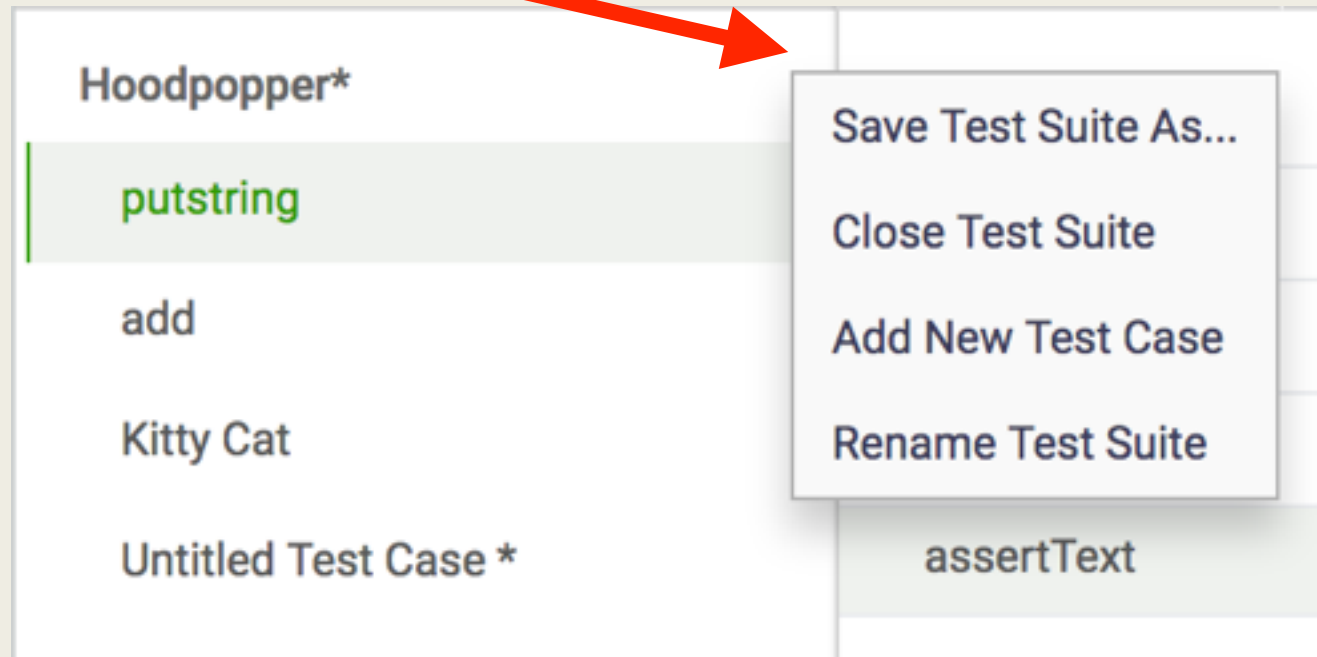
Adding a test case to a test suite



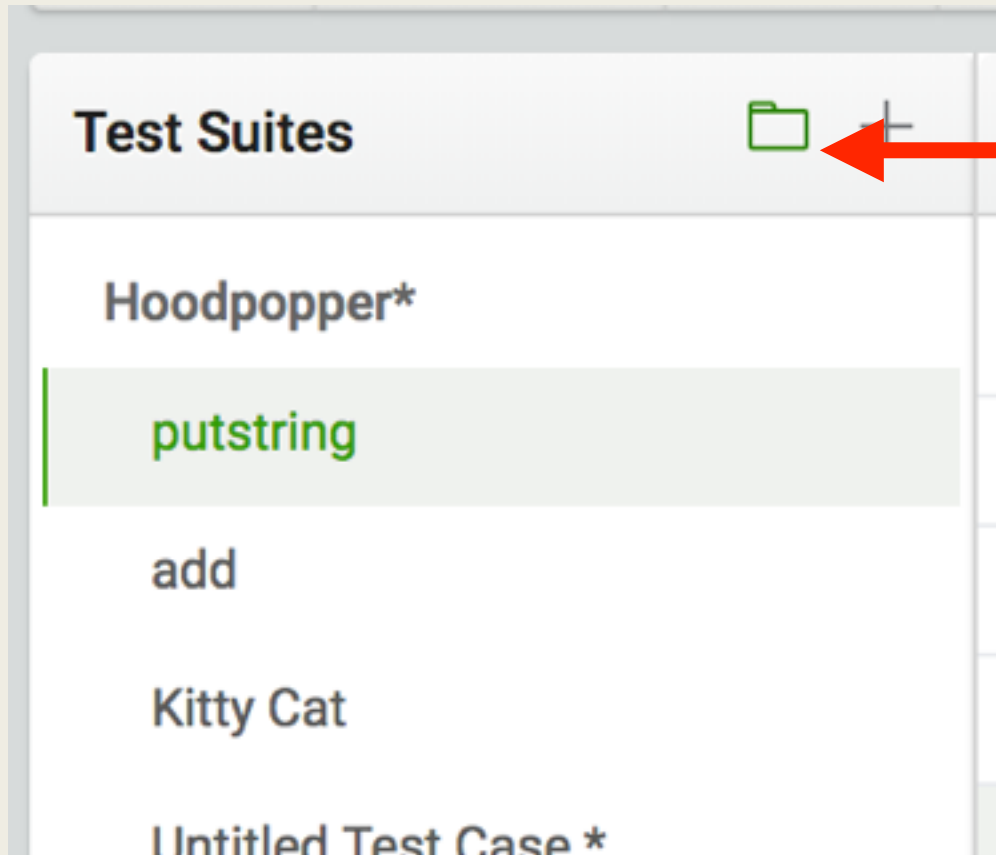
Click on the three dots next to the test suite and “Add New Test Case”

Saving A Test Suite

- Save the test suite (which will save all of the test cases) by pressing the button with three dots next to it and “Save Test Suite As...”
- Note: the button only appears when you hover over it. It should show up around here



Opening a Test Suite



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

Let's Walk Through A Test Case