STP SOFTWARE TEST PROFESSIONALS
CONFERENCE SPRING 2018

Selenium for Beginners

Peter Kim
Joshua Peeling

Intro

Joshua Peeling

Peter Kim

Objectives

- Install and setup environment
  - Ruby with Selenium/WebDriver dependencies
- What is Selenium, JSON Wire Protocol, WebDriver?
- Basic Programming Concepts
- Test Automation with Selenium/WebDriver
- Next Steps ..

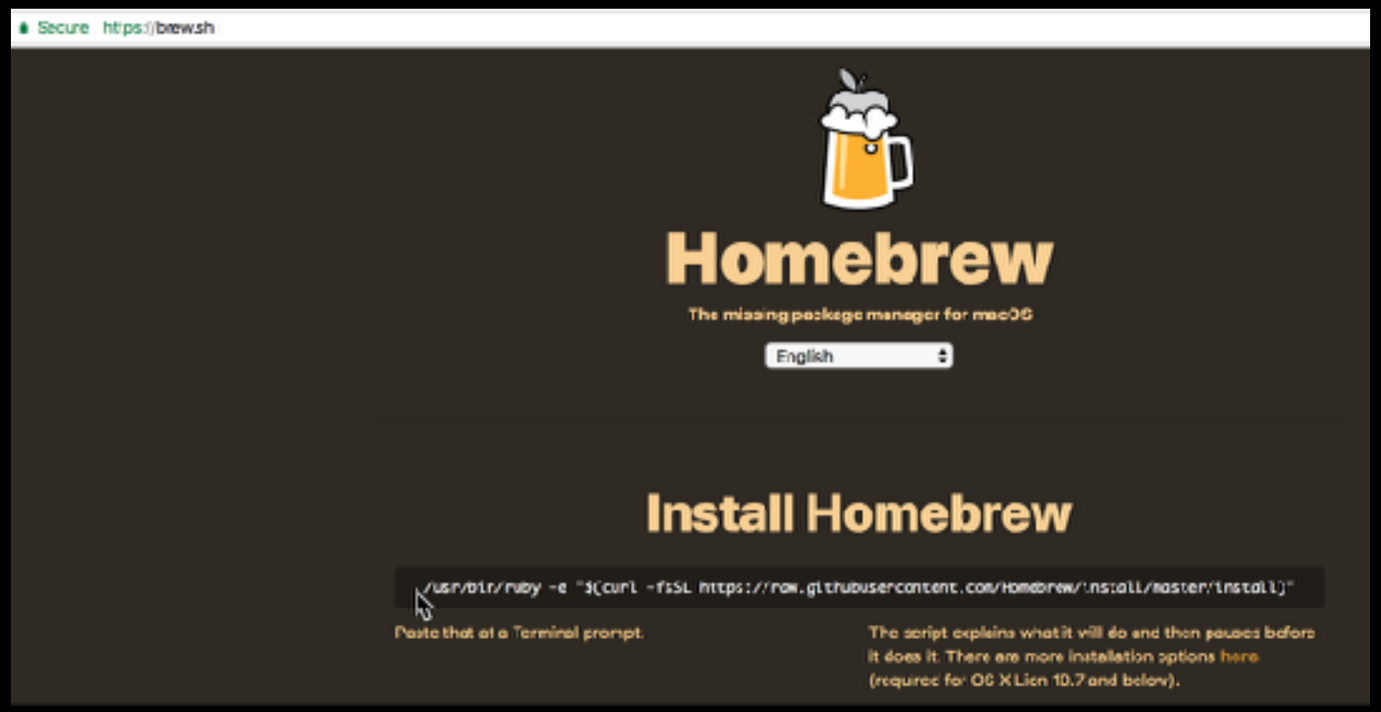
Installing Ruby (MacOS)

- Should already be pre-installed

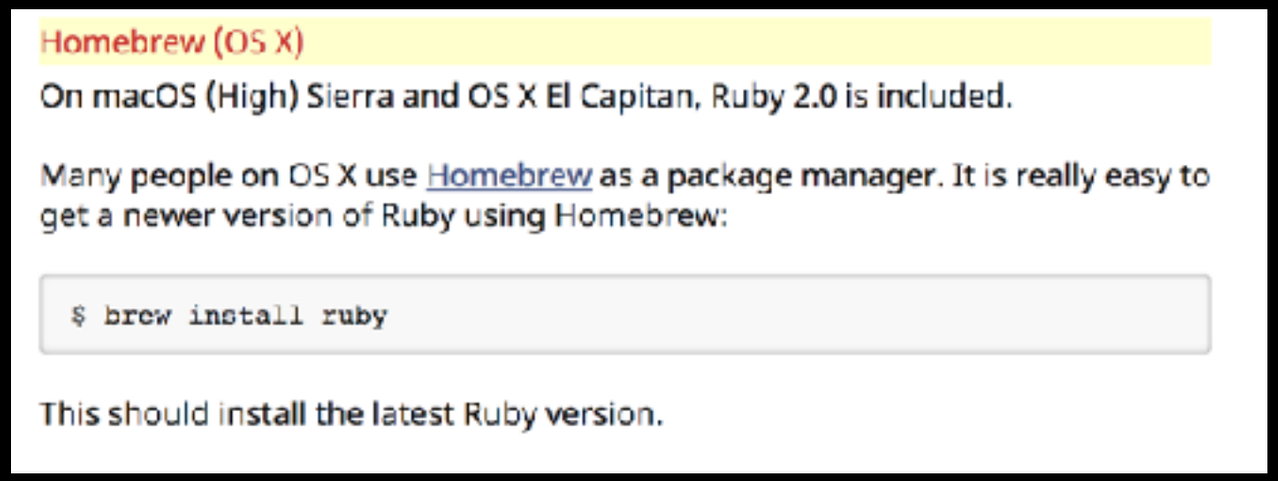
stpcon — bash — 100×44

```
/Users/pkim/stpcon => ruby --version
ruby 2.3.0p0 (2015-12-25 revision 53290) [x86_64-darwin14]
/Users/pkim/stpcon =>
```

Installing Ruby (MacOS)

- 

- Install Ruby



Installing Ruby - Windows

Chromedriver

- Download and install

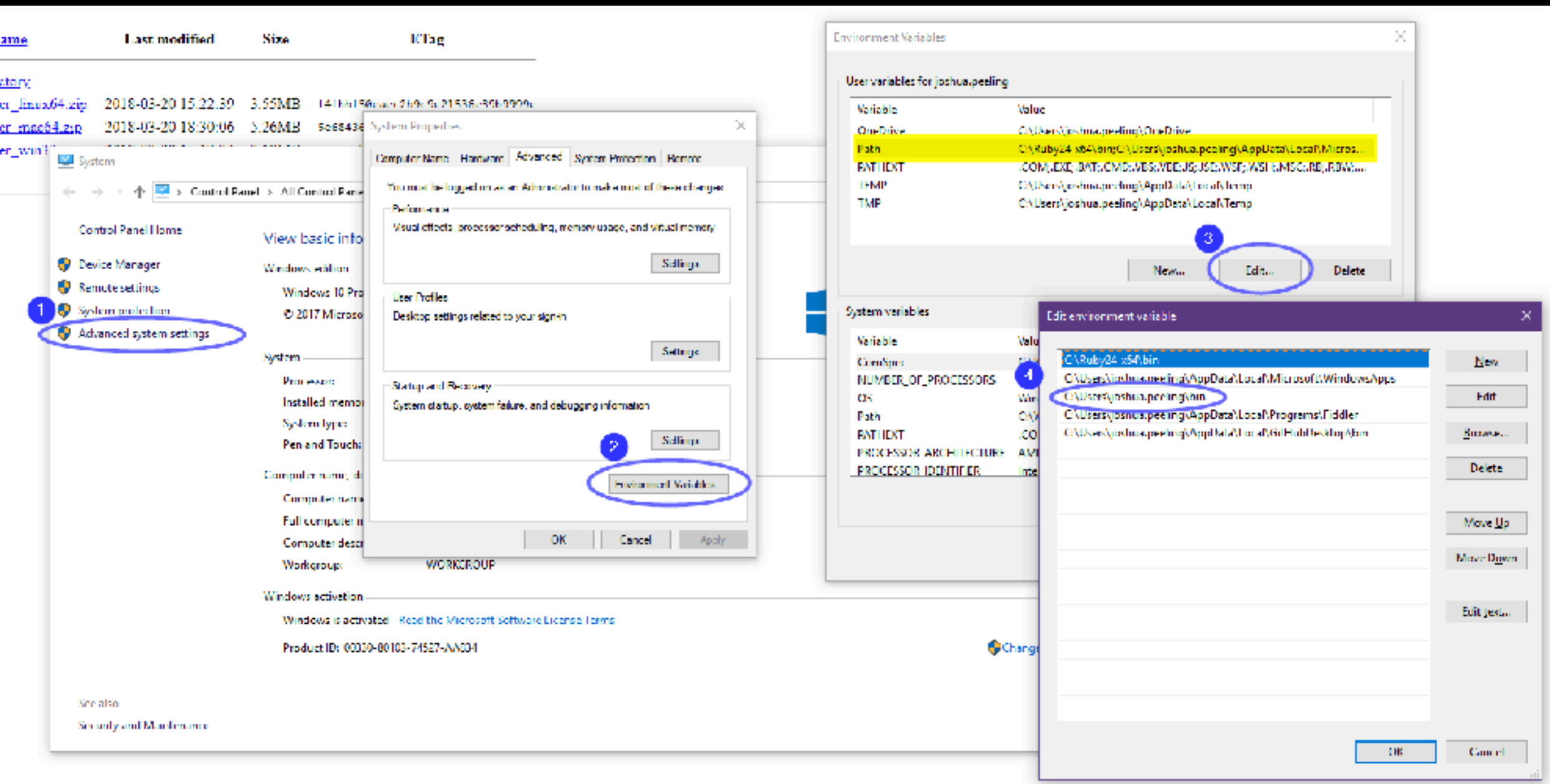
  https://sites.google.com/a/chromium.org/chromedriver/downloads

- Place the downloaded file “chromedriver*.exe” into a designated folder (create the folder “bin”)

- Update your environment variable, PATH, with a new entry:

  /Users/<YOUR USERID>/bin

Updating PATH to chrome driver (Window)



Installing Selenium

- Open Terminal (on Windows - “Ruby Terminal”)
- gem install selenium-webdriver

  Selenium-WebDriver libraries
- gem install rspec

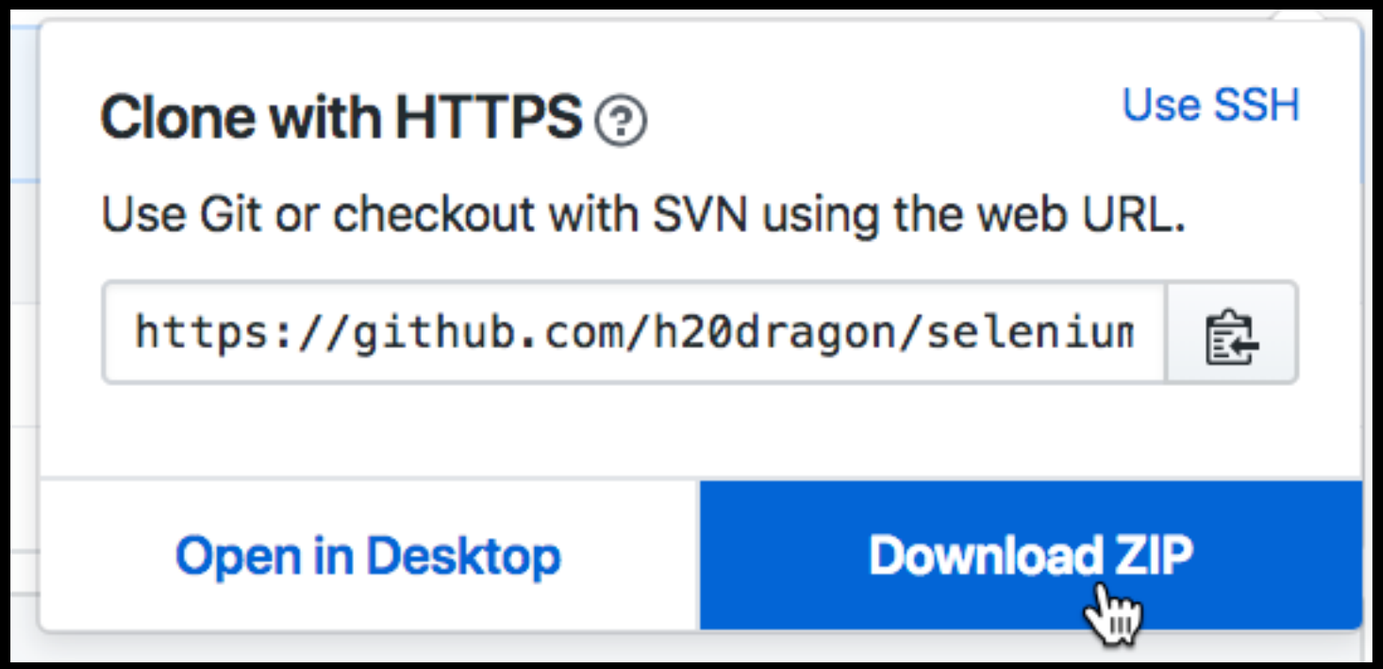
  Test Framework / Assertions
- gem install rake

  Ruby task/job manager

Example Code

git@github.com:h20dragon/selenium101.git

https://github.com/h20dragon/selenium101.git



Basic Programming Concepts

- “require” external libraries

  selenium-webdriver

- “require_relative” libraries by relative path
- Classes (Objects)
  - Attributes
  - Methods (functions)

require and require_relative

- require ‘selenium-webdriver’
- require_relative ‘../common/utils’

Example: require/ require_relative

```
[exercises]$ cat common/mylib.rb | nl


     1  def echo(s)
     2     puts "[ECHO]: " + s
     3  end

[exercises]$
[exercises]$
[exercises]$ irb
2.4.1 :001 > echo("STPCon")
NoMethodError: undefined method `echo' for main:Object
        from (irb):1
        from /Users/peterkim/.rvm/rubies/ruby-2.4.1/bin/irb:11:in `<main>'
2.4.1 :002 >
2.4.1 :003 >
2.4.1 :004 >
2.4.1 :005 >
2.4.1 :006 >   require_relative './common/mylib'
 => true
2.4.1 :007 > echo("STPCon")
[ECHO]: STPCon
 => nil
2.4.1 :008 >
```

Class

- “Class” is a blue-print - used to create an object.
  - Define data needed by the class
  - Define services that the class provides
- Reference: Object Oriented Design
  * Encapsulation
  * Abstraction
  * Polymorphism
  * Encapsulation

Example: Class

```
ruby_classes]$ cat vehicle.rb  |  nl
     1  class Vehicle

     2     attr_accessor :engineType
     3     attr_accessor :wheels

     4     def initialize(des = 'TBD')
     5       puts __FILE__ + (__LINE__).to_s +  "  [Vehicle]: initialize"

     6       # Place any initialization code here
     7       @description = des
     8       @wheels = 0
     9       @totalPassengers = 0
    10     end

    11     def dump()
    12       puts __FILE__ + (__LINE__).to_s + "  [Vehicle]:dump"
    13       puts "o Description: " + @description
    14       puts "o Wheels    : " +  @wheels.to_s
    15       puts "o Passengers : " + @totalPassengers.to_s
    16    end

    17  end
ruby_classes]$
ruby_classes]$
ruby_classes]$ irb
.4.1  001 > require_relative 'vehicle'
=> true
.4.1 :002 > m3 = Vehicle.new('My M3')
Users/peterkim/stp/stpcon_2017_Fall/demystifying_se/demystifying_se/exercises/basics/ruby_classes/vehicle.rb7  [Vehicle]: initialize
=> #<Vehicle:0x007fd1ad1e9610 @description="My M3", @wheels=0, @totalPassengers=0>
.4.1 :003 >
.4.1 :004 >  m3.dump
Users/peterkim/stp/stpcon_2017_Fall/demystifying_se/demystifying_se/exercises/basics/ruby_classes/vehicle.rb16  [Vehicle]:dump
o Description: My M3
o Wheels    : 0
o Passengers : 0
=> nil
```

Selenium-WebDriver

require ‘selenium-webdriver’

Selenium/WebDriver What is it?

- RESTFul web service, leveraging JSON, over HTTP

- Selenium RC (Server/Core) - JS Injection (JS App running inside the browser)

- WebDriver
  * “.. should do what a user can do”
  * Tiny API (More readable code)
  * No JS Injection

- W3C Standard for browser automation

  https://github.com/SeleniumHQ/selenium/wiki/JsonWireProtocol

Selenium/WebDriver References

- https://w3c.github.io/webdriver/webdriver-spec.html
- http://www.rubydoc.info/gems/selenium-webdriver/Selenium/WebDriver
- https://github.com/SeleniumHQ/selenium/wiki/Ruby-Bindings

irb - Interactive Ruby

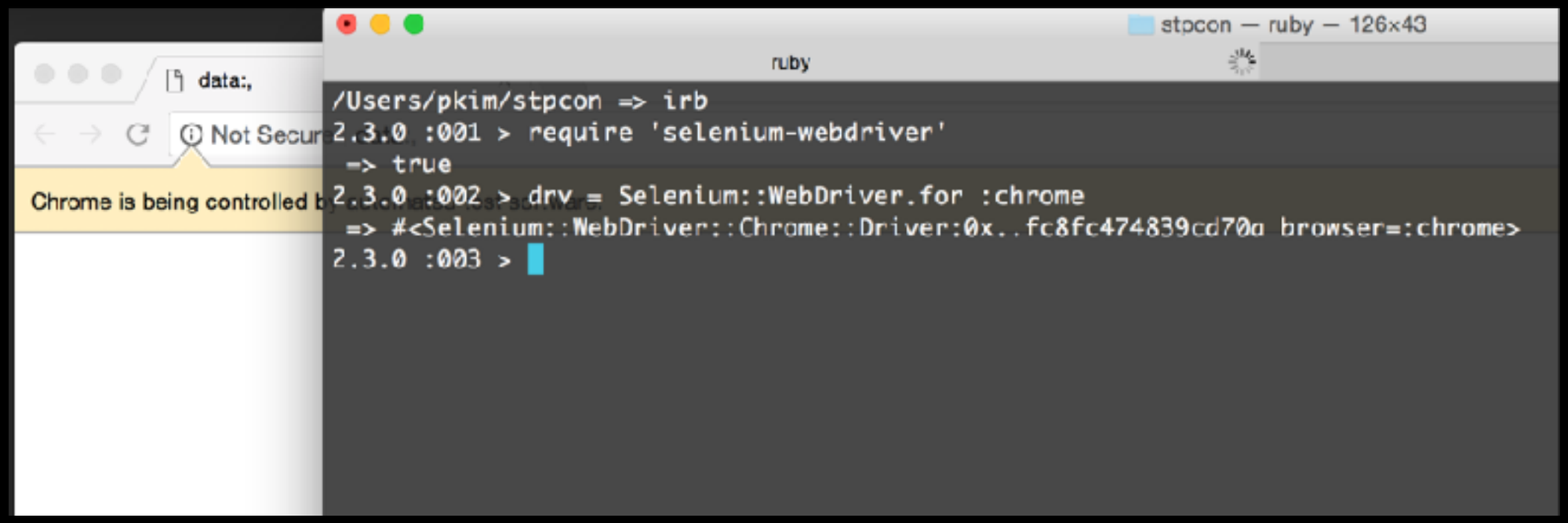
Start “irb” session.

Interactively enter statements.

e.g.

```
require ‘selenium-webdriver’
x = 100
puts “Hello World”
puts  “My value is “ + x
myHash = { :name => ‘Elvis’,  :car => ‘Cadillac’ }
puts “My hash #{myHash}”
```

Selenium-WebDriver Library



Start “irb”

```
/Users/pkim/stpcon => irb
2.3.0 :001 > require 'selenium-webdriver'
 => true
2.3.0 :002 >
```

Methods available Selenium-WebDriver

drv.methods.sort.each do |m| puts m end

```
2.3.0 :010 > drv.methods.sort.each do |m| puts m end
!
!.com
!=
!~
<=>
==
===
=~
[]
__id__
__send__
action
all
browser
capabilities
class
clone
close
current_url
define_singleton_method
display
dup
enum_for
eql?
equal?
execute_async_script
execute_script
extend
find_element
find_elements
first
freeze
frozen?
get
hash
inspect
instance_eval
instance_exec
instance_of?
instance_variable_defined?
instance_variable_get
instance_variable_set
instance_variables
```

```
itself
keyboard
kind_of?
local_storage
manage
method
methods
mouse
navigate
nil?
object_id
page_source
private_methods
protected_methods
public_method
public_methods
public_send
quit
ref
remove_instance_variable
respond_to?
save_screenshot
screenshot_as
script
send
session_storage
singleton_class
singleton_method
singleton_methods
switch_to
taint
tainted?
top
title
to_enum
to_json
to_s
trust
untaint
untrust
untrusted?
window_handle
window_handles
```

Basic Commands

- navigate.to <URL> - Load a URL
- get - Load a URL
- title - Get the currently loaded page's title
- current_url - Get the currently loaded page's URL
- close - Close browser
- quit - Close and quit browser

Exercise 1. Create Browser

/selenium101/exercises/1/exercise1.rb

- Objective: Open a chrome browser and **navigate** to a target website, while obtaining the **title** and **current URL**.

```
require 'selenium-webdriver'


drv = Selenium::WebDriver.for :chrome

puts "WebDriver object is " + drv.class.to_s


drv.navigate.to( url 'http://www.stpcon.com')

title = drv.title
url = drv.current_url


puts "Current Title is " + title
puts "Current URL is " + url


drv.quit
```

Exercise 2. Navigation

`/selenium101/exercises/2/exercise2.rb`

- back
- forward
- refresh
- navigate.to(url)

http://www.rubydoc.info/gems/selenium-webdriver/Selenium/WebDriver/Navigation

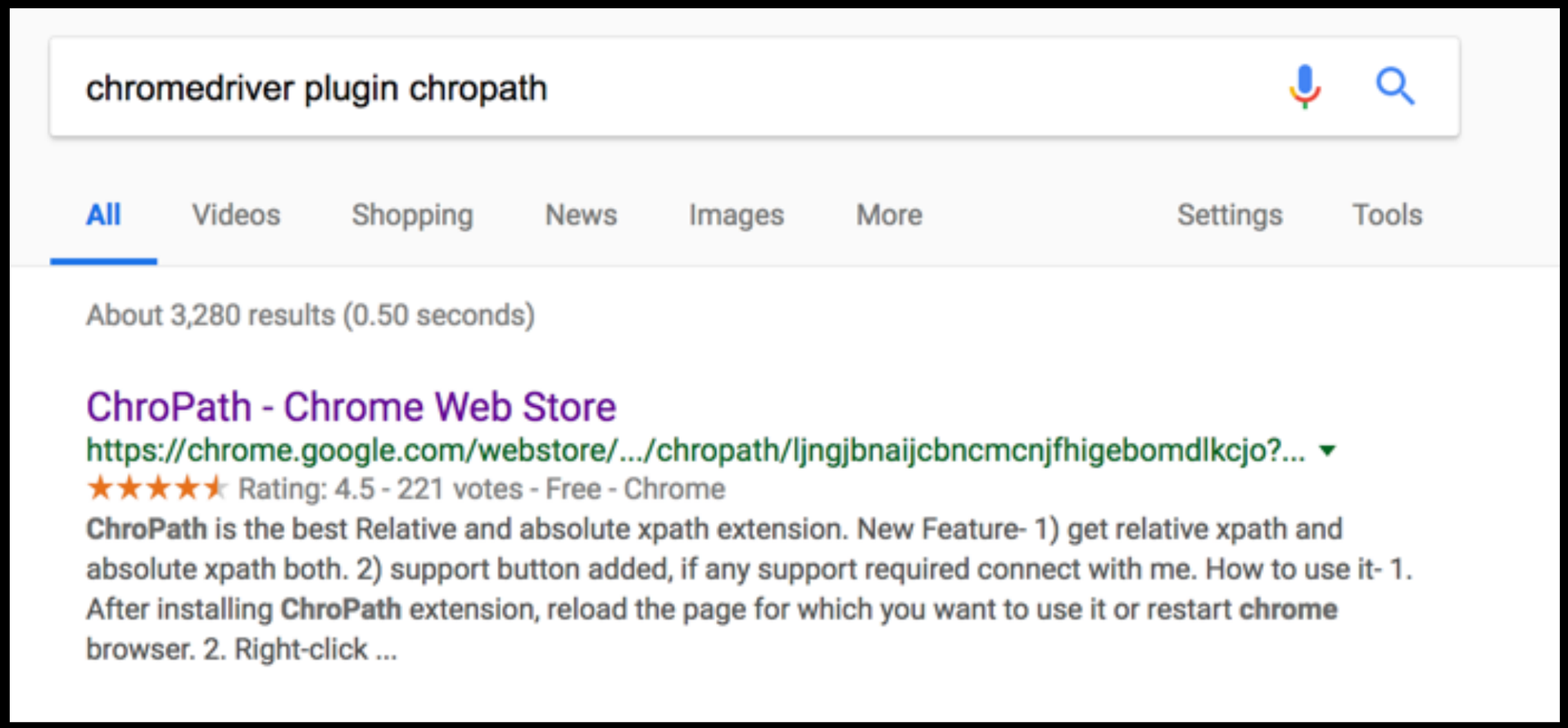
Exercise 2. - Navigation

/selenium101/exercises/2/exercise2.rb

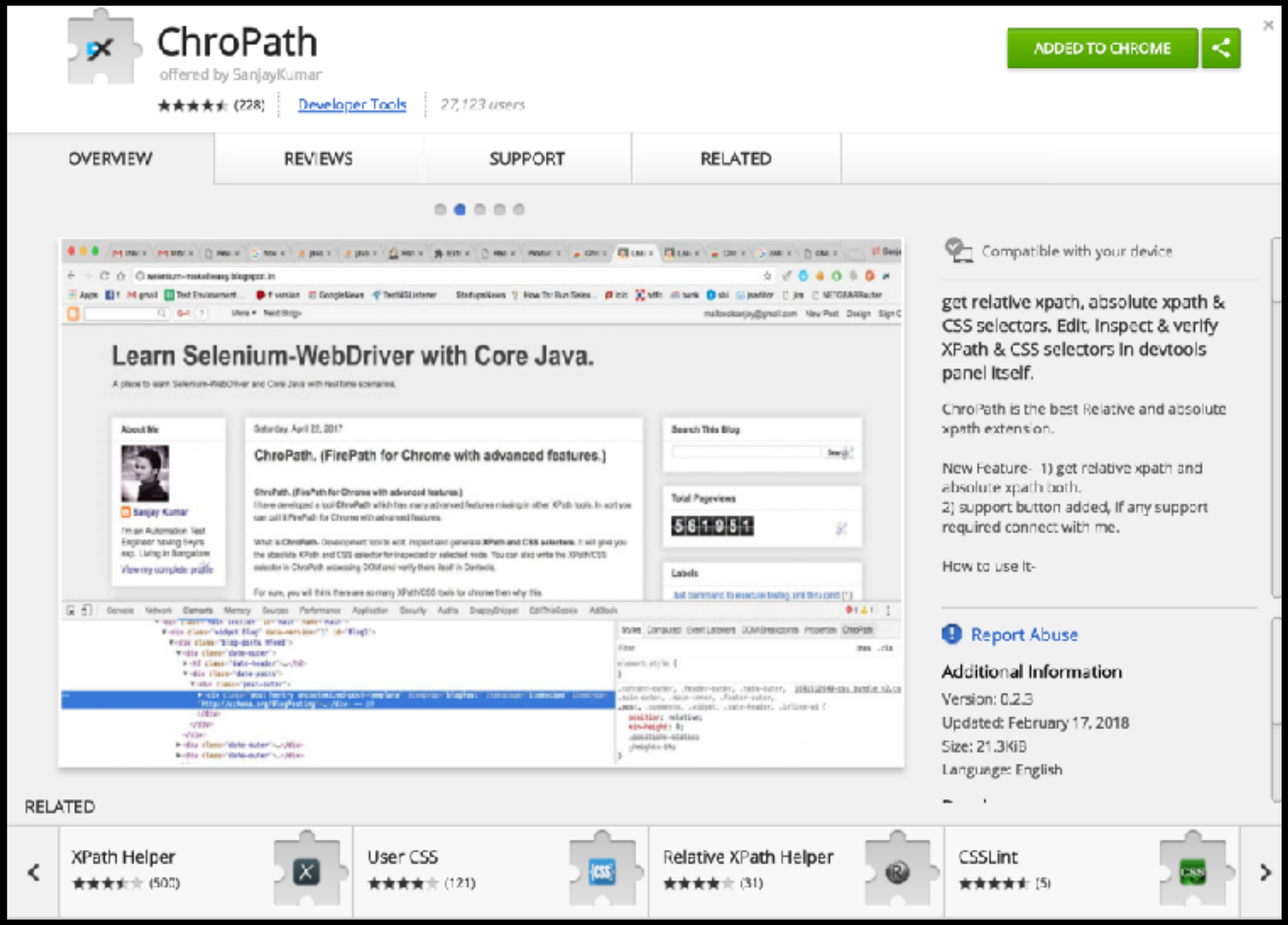
- Objective: Open a chrome browser and **navigate** to a target website, while obtaining the **title**, **current URL**, and **refresh** the page.

```
require 'selenium-webdriver'

host = 'http://www.stpcon.com'


drv = Selenium::WebDriver.for :chrome

drv.navigate.to( url: host)

puts "Title of #{host} is #{drv.title}"



drv.get('http://www.carmax.com')

puts "Title of Carmax site is " + drv.title

drv.navigate.back()

puts "After navigationg back, title is #{drv.title}"

drv.navigate.refresh()

drv.quit()
```

Selenium Locators Chrome Plugin: ChroPath



Selenium Locators Chrome Plugin: ChroPath



Selenium Locators

- A DSL/Language that gives you the ability to find web elements.

Selenium Locators

- A DSL/Language that gives you the ability to find web elements.
- This could consume significant QE resources to your end-to-end automated tests
  - UI changes
  - UI challenges in finding appropriate locator
    - Frames (within frames)
    - Alerts
    - Timing conditions and constraints
    - Refactoring of automation code

Selenium Locators

- FINDERS

```
class: 'class name',
class_name: 'class name',
css: 'css selector',
id: 'id',
link: 'link text',
link_text: 'link text',
name: 'name',
partial_link_text: 'partial link text',
tag_name: 'tag name',
xpath: 'xpath'
```

- find_element
- find_elements

Instance Method Details

`#find_element(*args) ⇒ WebDriver::Element`

Find the first element matching the given arguments.

Parameters:

- **how** (`:class`, `:class_name`, `:id`, `:link_text`, `:link`, `:partial_link_text`, `:name`, `:tag_name`, `:xpath`)
- **what** (`String`)

Returns:

- (`WebDriver::Element`)

Raises:

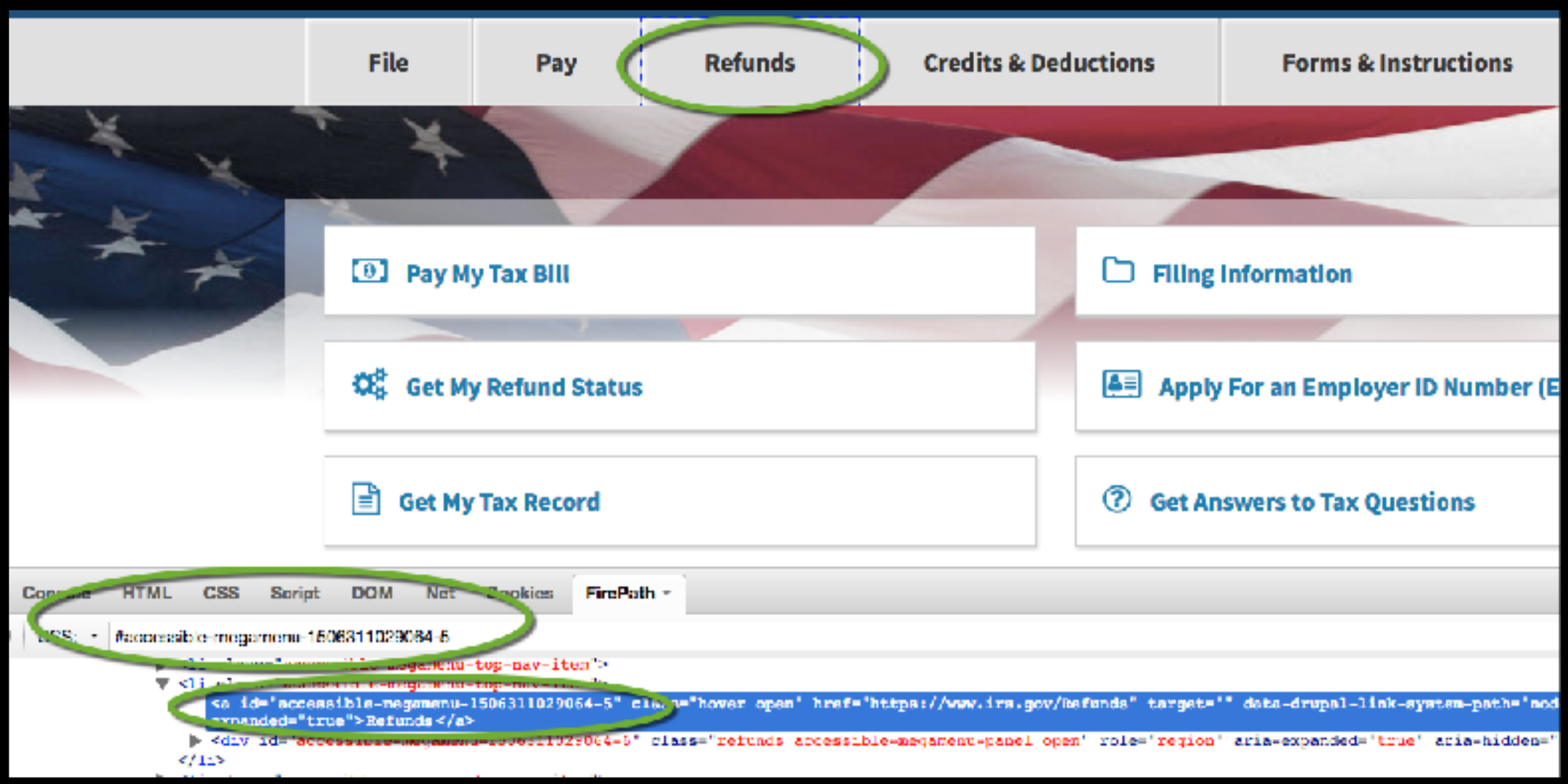
- (`NoSuchElementError`) — if the element doesn't exist

[View source]

http://www.rubydoc.info/gems/selenium-webdriver/0.0.28/Selenium/WebDriver/Find

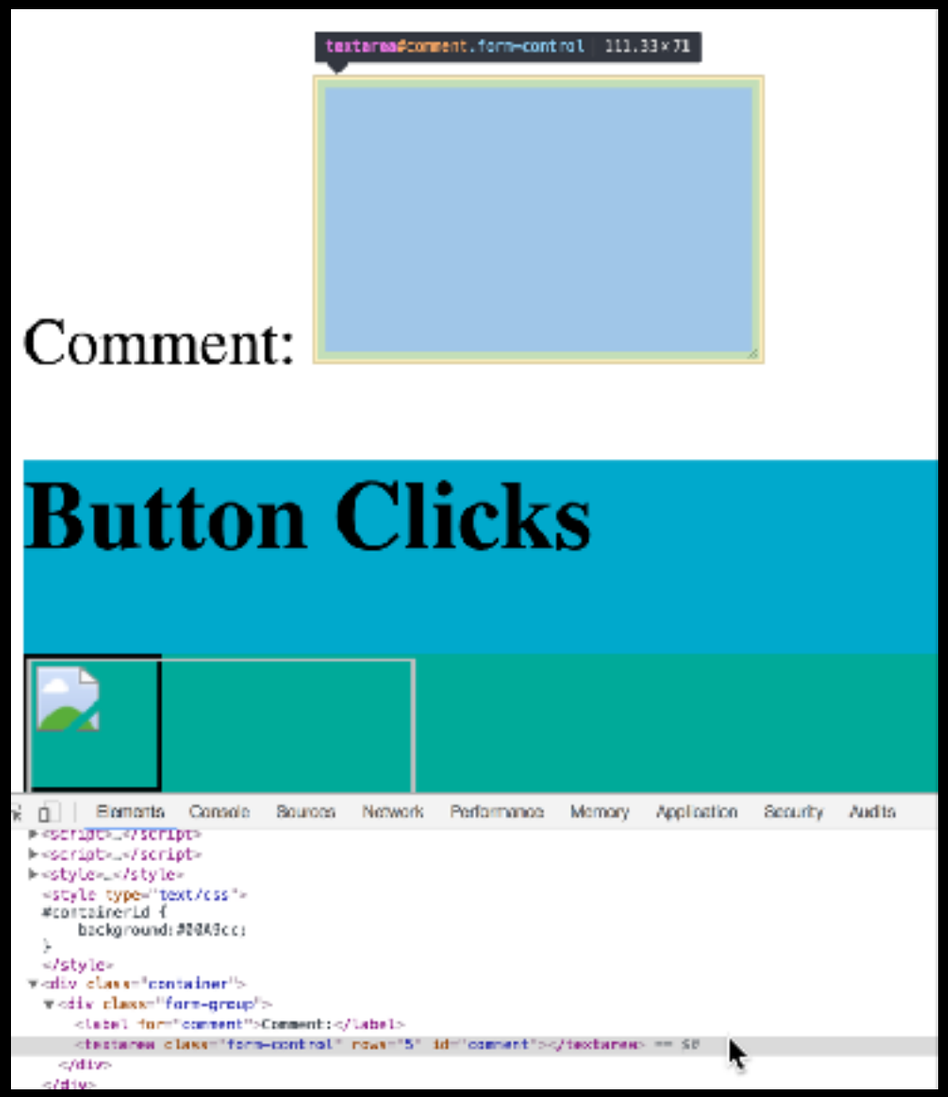
Selenium Locators: id

find_element(:id, <String>)



Selenium Locators: id

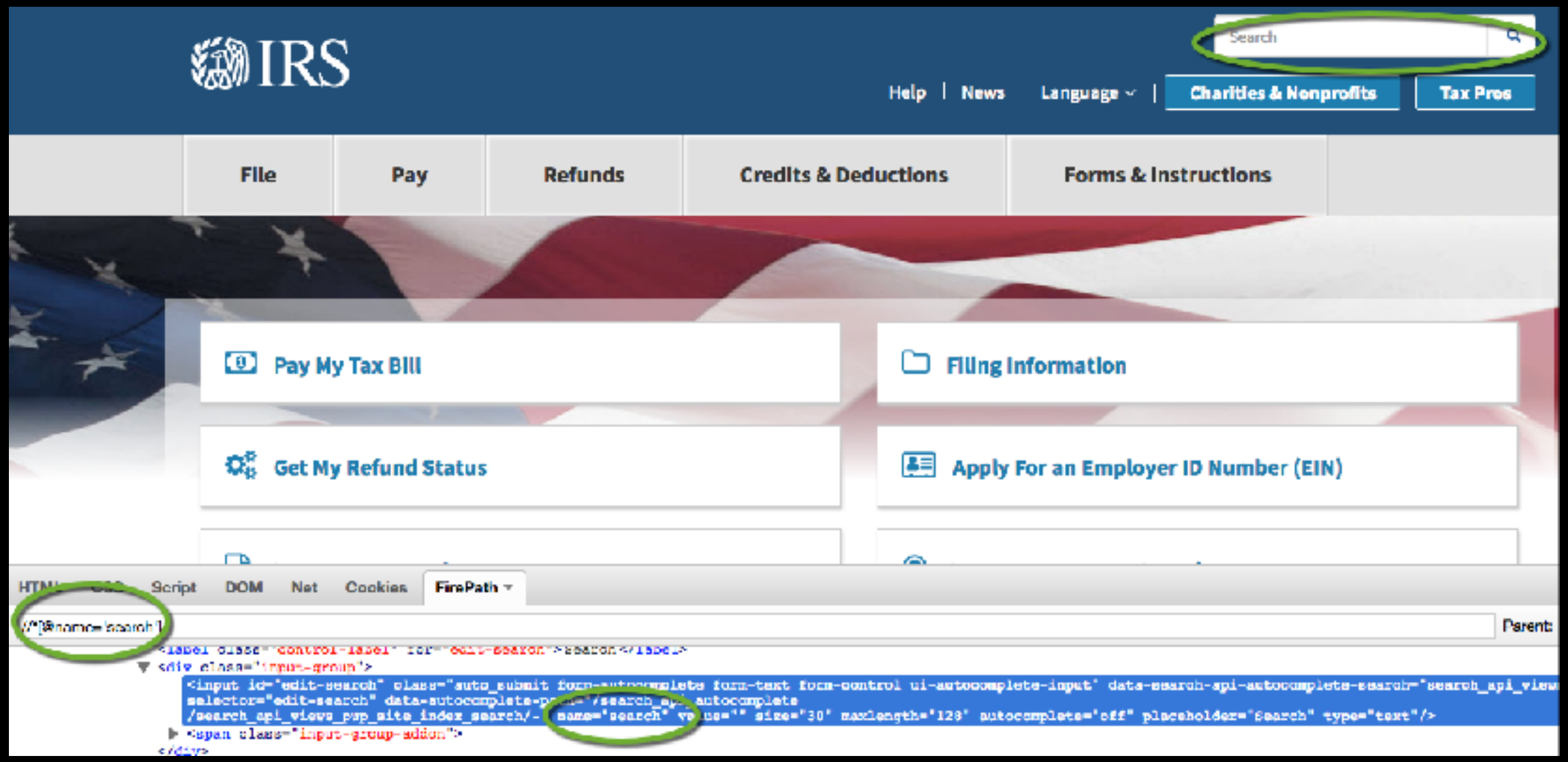
https://stark-bastion-95510.herokuapp.com/playground/



obj = b.find_element(:id, ‘comment’)

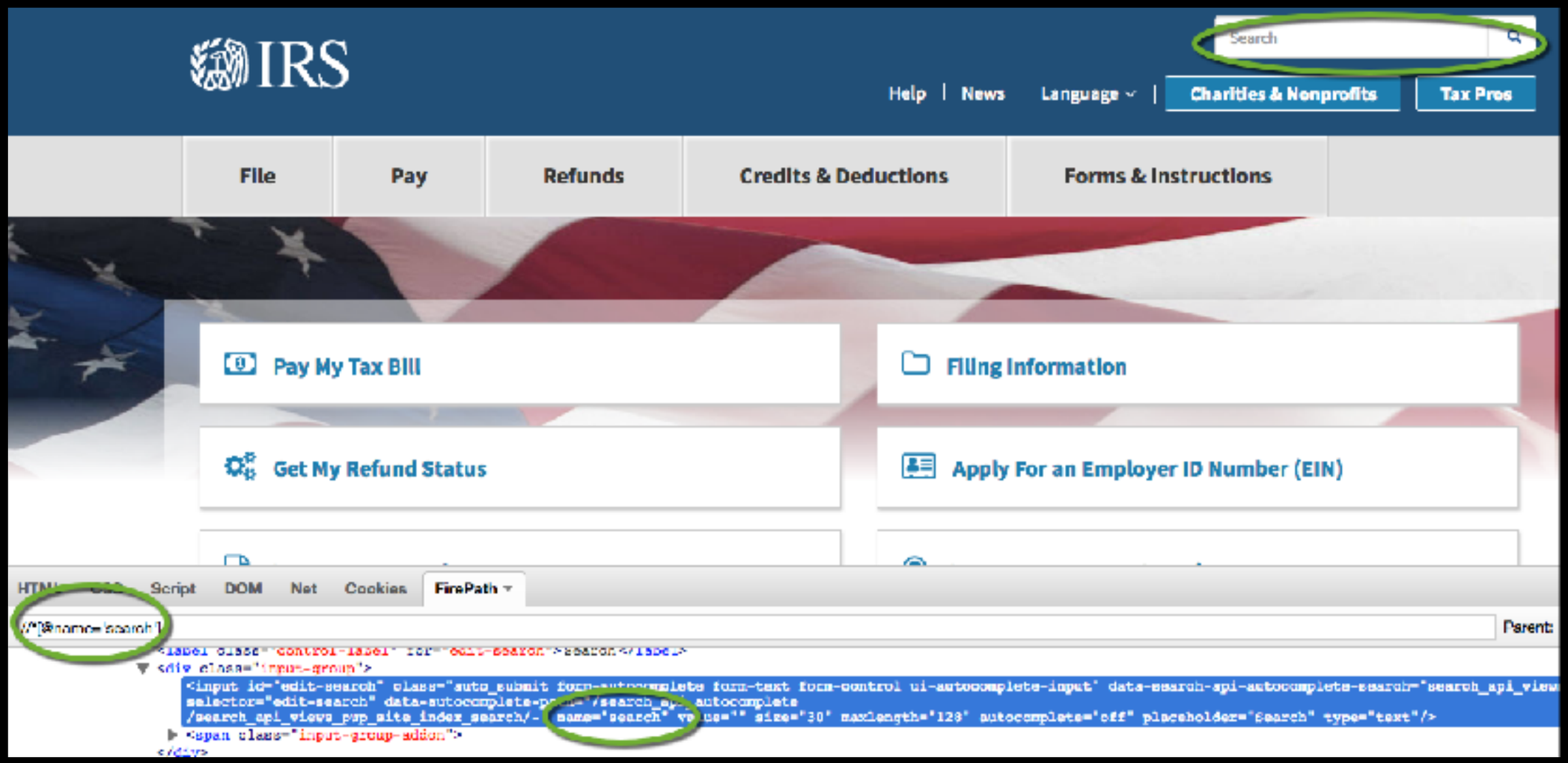
Selenium Locators: name

find_element(:name, <String>)



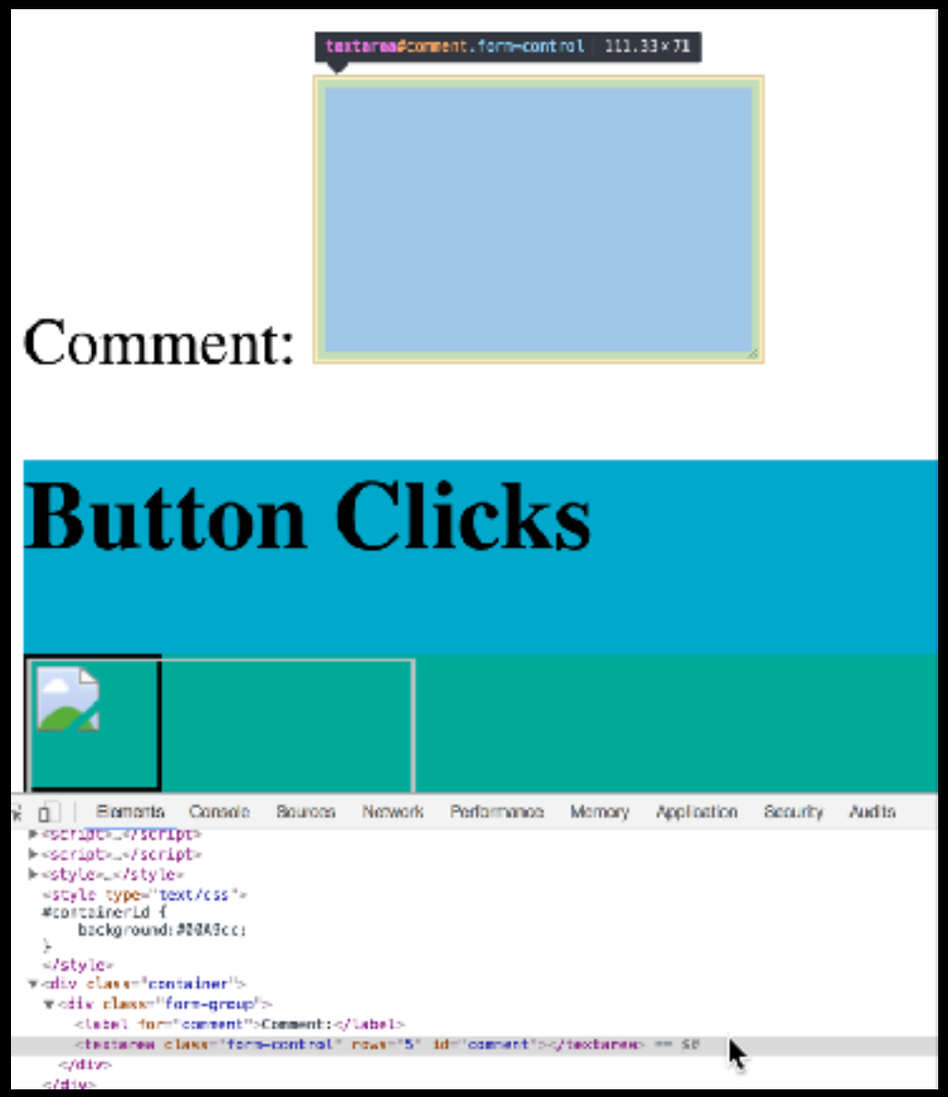
Selenium Locators: **css**

find_element(:css, <String>)



Selenium Locators: **css**

https://stark-bastion-95510.herokuapp.com/playground/

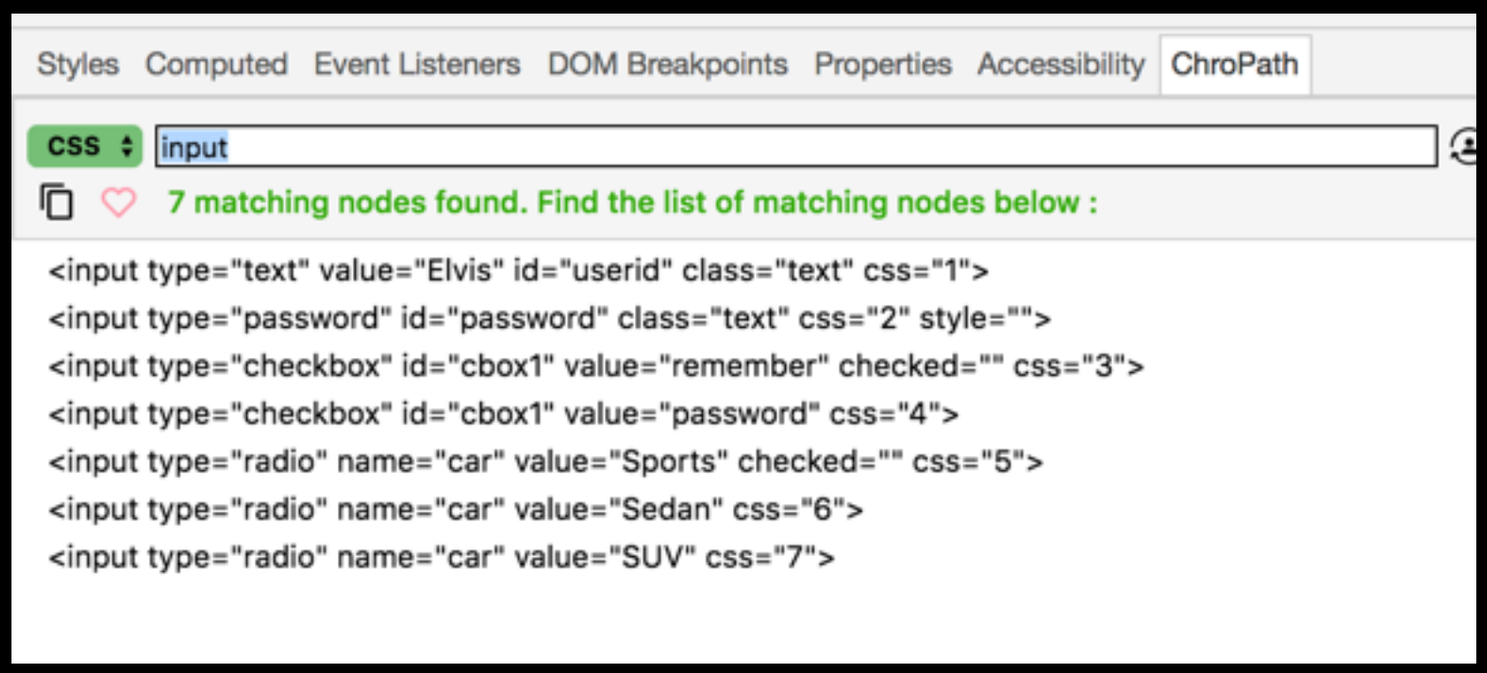


obj = b.find_element(:css, '#comment')

Selenium Locators

Exploring CSS with Attributes

1. https://stark-bastion-95510.herokuapp.com/playground/
2. Inspect (chropath)
3. Enter CSS to find all “input” elements



Selenium Locators: link_text

find_element(:link_text, <String>)

Selenium Locators: xpath

https://www.w3schools.com/xml/xpath_syntax.asp

Selecting Nodes

XPath uses path expressions to select nodes in an XML document. The node is selected by following a path or steps. The most useful path expressions are listed below:

Expression	Description
nodename	Selects all nodes with the name "*nodename*"
/	Selects from the root node
//	Selects nodes in the document from the current node that match the selection no matter where they are
.	Selects the current node
..	Selects the parent of the current node
@	Selects attributes

xpath - XML Path Language

“//” - selects nodes in the document (page) that match the selection no matter where they are

“@” - Selects attributes

Examples:

//*[@id=‘username’]

//input[@name=‘password’]

xpath - XML Path Language

Operators:

“and”

Examples:

//*[@make=‘porsche’ and @model=‘carrera’]

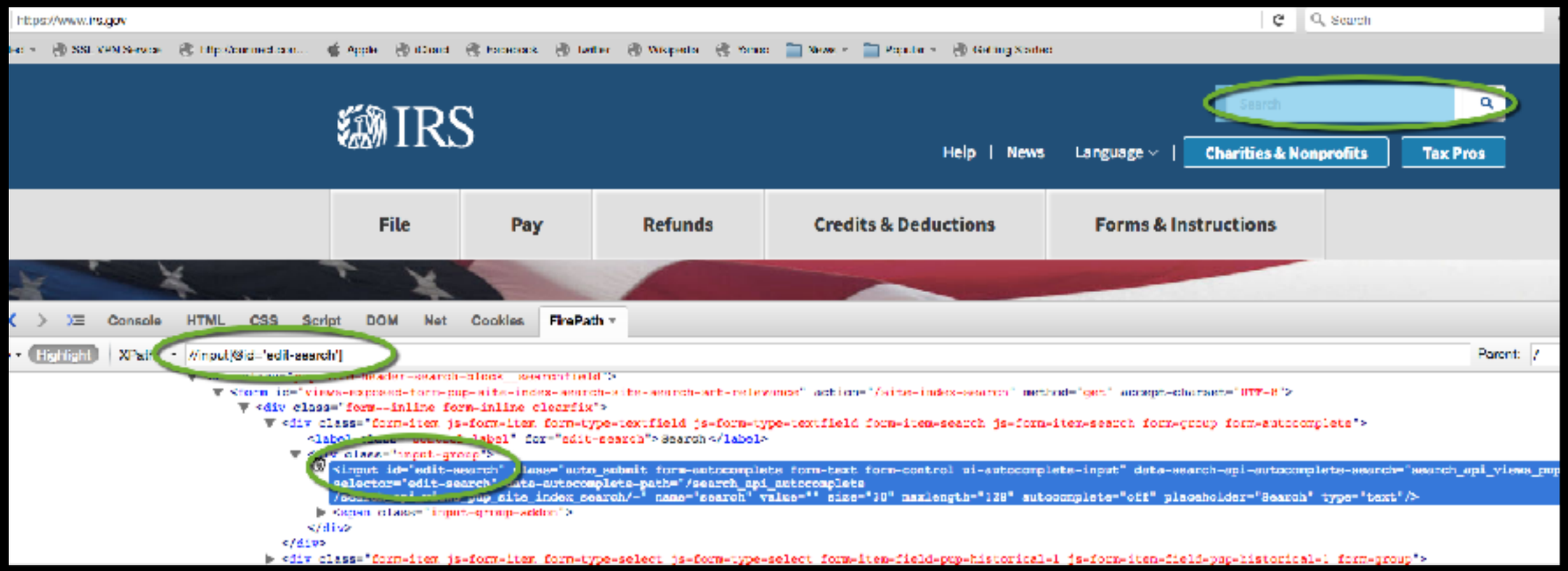
xpath regarding … performance?

If a solid attribute such as “id” and “name” is available .. that is preferable.

If a solid CSS locator can be leveraged .. preferable.

Ensuring that you have a reliable and scalable locator will save a lot of headaches (**refactoring**/**flaky** tests).

Selenium Locator: xpath



SearchContext - find_element

```
#find_element(how, what) ⇒ Element
```

Parameters:

- **how** (Symbol, String) — The method to find the element by
- **what** (String) — The locator to use

```
class: 'class name',
class_name: 'class name',
css: 'css selector',
id: 'id',
link: 'link text',
link_text: 'link text',
name: 'name',
partial_link_text: 'partial link text',
tag_name: 'tag name',
xpath: 'xpath'
```

Exercise. Find an element with ‘css’.

```
…
drv.get(‘https://stark-bastion-95510.herokuapp.com/playground’)
comment = drv.find_element(css:  “#comment”)
comment.send_keys(“STP Conference 2018”)
…
```

Exercise. Find an element with ‘xpath’.

```
…
drv.get(‘https://stark-bastion-95510.herokuapp.com/playground’)
comment = drv.find_element(xpath:  “//textarea[@id=‘comment’]”)
comment = drv.find_element(xpath:  “//textarea[@id=‘foo’ or @id=‘comment’]”)
comment = drv.find_element(xpath:  “//*[@id=‘comment’]”)

comment.send_keys(“STP Conference 2018”)

…
```

Exercise 3. Find all links on a page (find_elements)

/selenium101/exercises/3/exercise.find_elements.rb

```
elements = drv.find_elements( [LOCATOR ] )
...
puts elements[0].attribute('text')
puts elements[1].attribute('alt')
```

Ref: Exercise3 - "exercise.find_elements.rb". Highlights target element.

Waiting on elements

Explicit Waits

An explicit wait is code you define to wait for a certain condition to occur before proceeding further in the code. The worst case of this is Thread.sleep(), which sets the condition to an exact time period to wait. There are some convenience methods provided that help you write code that will wait only as long as required. WebDriverWait in combination with ExpectedCondition is one way this can be accomplished.

Explicit waits

Use the Wait class to explicitly wait for some condition:

```
wait = Selenium::WebDriver::Wait.new(timeout: 3)
wait.until { driver.find_element(id: "cheese").displayed? }
```

http://www.seleniumhq.org/docs/04_webdriver_advanced.jsp

Implicit Wait

Implicit Waits

An implicit wait is to tell WebDriver to poll the DOM for a certain amount of time when trying to find an element or elements if they are not immediately available. The default setting is 0. Once set, the implicit wait is set for the life of the WebDriver object instance.

Implicit waits

WebDriver lets you configure implicit waits, so that a call to `#find_element` will wait for a specified amount of time before raising a `NoSuchElementError`:

```
driver = Selenium::WebDriver.for :firefox
driver.manage.timeouts.implicit_wait = 3 # seconds
```

“waits” - Which should I use?

WARNING: Do not mix implicit and explicit waits. Doing so can cause unpredictable wait times. For example setting an implicit wait of 10 seconds and an explicit wait of 15 seconds, could cause a timeout to occur after 20 seconds.

Use “explicit”

- Waits on elements based on conditions with respect to the timeout
- Handles AJAX elements better than implicit

Demo: Exercise 3. - “common/utils3.rb”

Run “exercise.wait2.rb” - “getClickable”

Working with Frames

To find an element, located inside a frame (e.g. <iframe>), you must be “switched” into the frame that contains that element.

E.g.
drv.switch_to.frame [id|name]

Ref: exercise4/5_frames*

```
drv.navigate.to('https://stark-bastion-95


drv.switch_to.frame('elements')

src = drv.find_element(id: 'userid2')

```

Ruby Spec

rspec-core - Test Runner, reporter, manage tests

rspec-expectations - express conditions (assertions)

http://rspec.info/documentation/

RSpec - Simple Example

./basics/frameworks/

Exercises:

* basics/frameworks/rspec/0 (run.sh or run.bat)
* basics/frameworks/rspec/1 (rake)
* basics/frameworks/rspec/2 (rake)
* basics/frameworks/rspec/3 (‘rake’)

Page Objects

Ref: ./basics/pageObjects

Peter Kim
LinkedIn : peterkim777
Twitter : peter_kim777
Email : h20dragon@outlook.com

Joshua Peeling
LinkedIn : Joshua-peeling
Email : jpeeling13@outlook.com

Join DCAST.io