

Selenium Webdriver Primer

Spensa Technologies, Inc
Sept. 22, 2016

Derrick Kearney
github.com/codedsk
tellask@gmail.com

Testing vs Checking

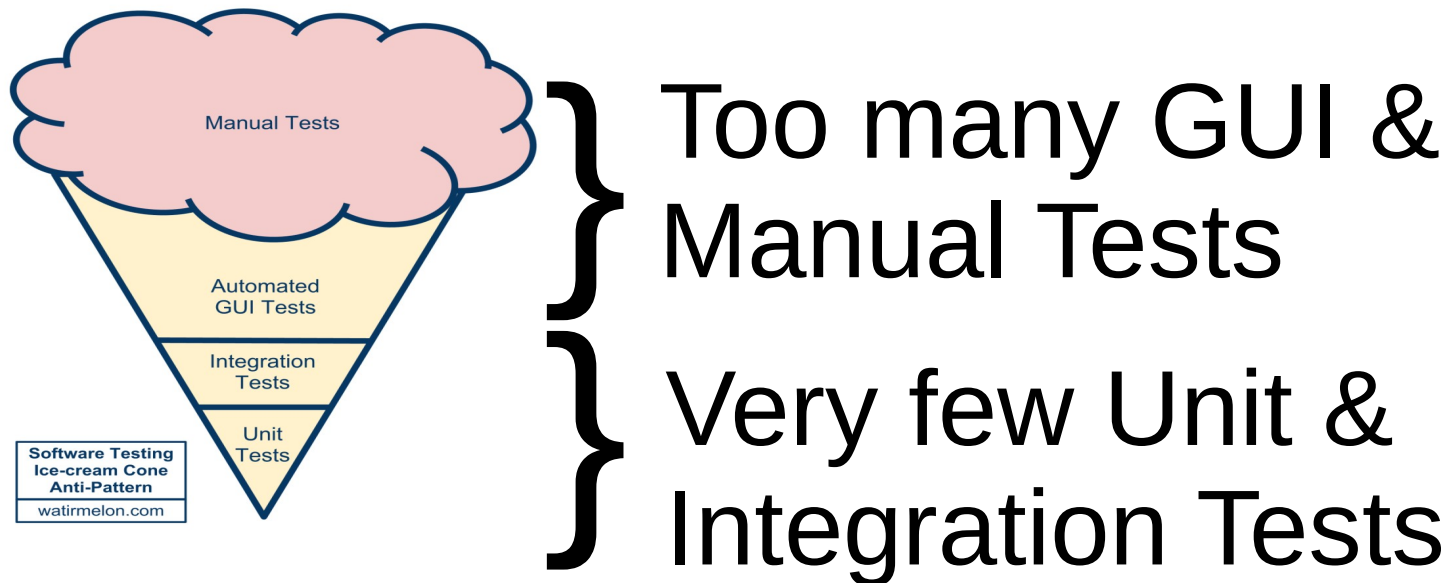
Testing is...

- * Exploring & Learning
- * Experimenting
- * Judgment & Values
- * Answers the question
“Is there a problem?”

Checking is...

- * Confirming
- * Pass / Fail
- * “It Works” vs “It Operates”

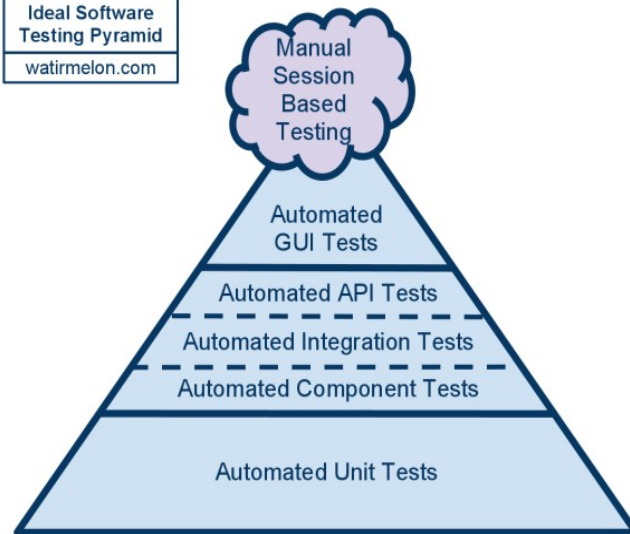
Typical Testing Pyramid



Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

Ideal Testing Pyramid

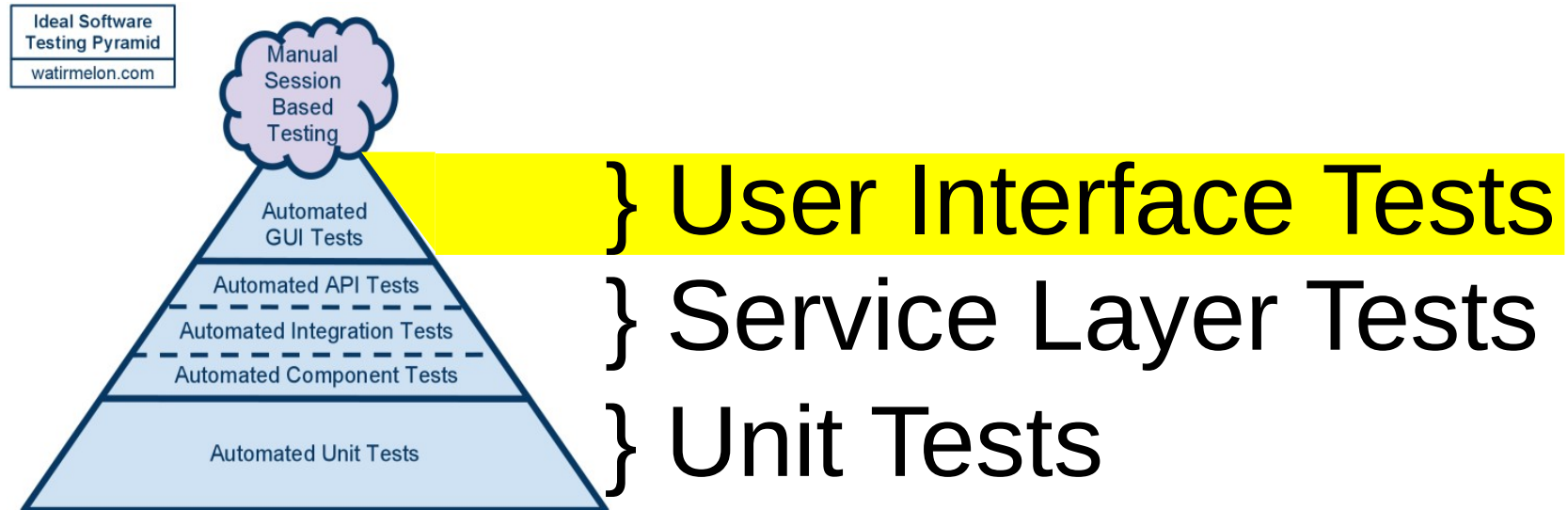
Ideal Software
Testing Pyramid
watirmelon.com



} User Interface Tests
} Service Layer Tests
} Unit Tests

Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

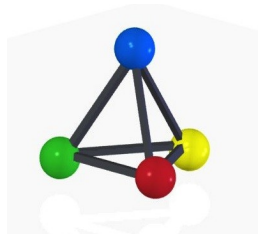
Ideal Testing Pyramid



Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

Automated Testing Tools

Web Crawlers



CrawlJax



IBM AppScan

Pros:

- Little Developer Coordination
- Semi directed exploration
- Web App is a black box

Cons:

- Time consuming to run
- No authority to judge results

Automated Testing Tools

Test Recorders



Selenium

Pros:

- Very customizable
- Produces specific targeted tests
- Programmable in a script



QuickTest Pro

Cons:

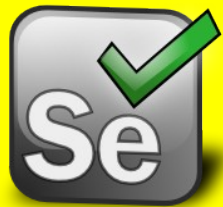
- Time consuming to build tests
- Allows for brittle test designs



Sahi Pro

Automated Testing Tools

Test Recorders



Selenium

- Developed By Jason Huggins and Simon Stewart at ThoughtWorks



QuickTest Pro

- Browser vendor support through W3C spec
- Automation written in “your favorite language”...



Sahi Pro

- Worldwide support Community

Automated Testing Tools

Test Recorders



Selenium

What can it do?



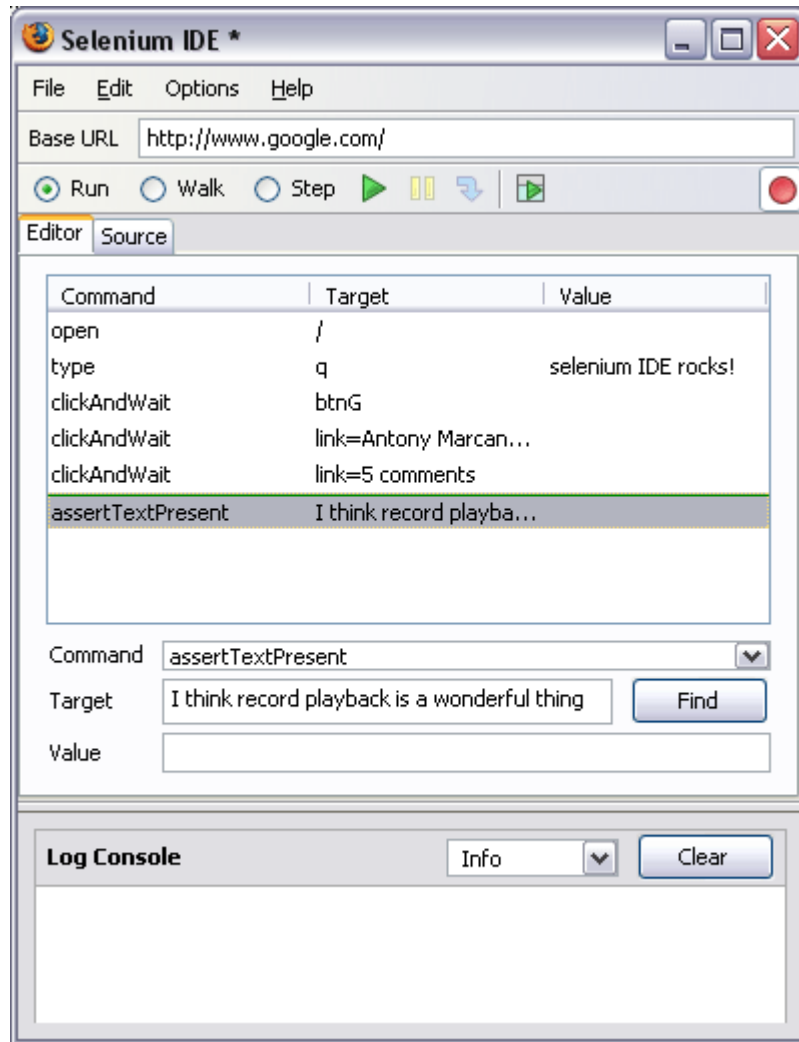
QuickTest Pro



Sahi Pro

- Open web browser
- Locate elements on page
- Type in or click elements
- Wait for triggered events
- Simulate moving the mouse

A brief word about Selenium IDE...

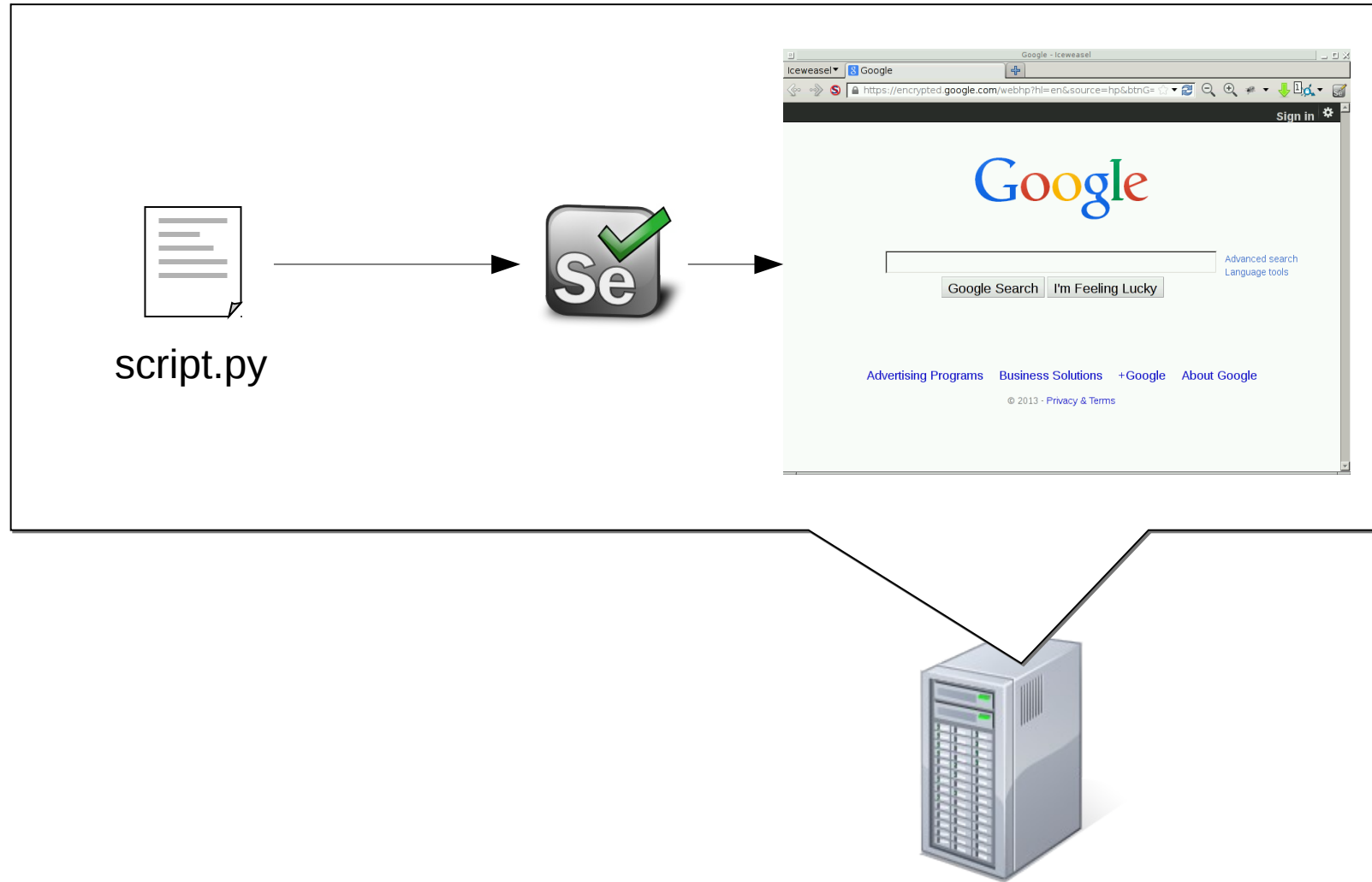


What can it do?

- Record actions, replay as tests
- Works in Firefox only
- Selenium 1.0 style tests
- Tests can be brittle

Try it out, then move on

How Selenium Works



[Login](#)[Register](#)

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for *you*.

[Login](#)[Register](#)

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for *you*.

[Login](#)[Register](#)

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for *you*.

[Login](#)[Register](#)

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for *you*.

Loading the Selenium Webdriver module:

```
from selenium import webdriver
```


Start a web browser:

```
from selenium import webdriver
```

```
url = 'http://hubzero.org'
```

```
browser = webdriver.Firefox()
```

Navigating to a web page:

```
from selenium import webdriver
```

```
url = 'http://hubzero.org'
```

```
browser = webdriver.Firefox()  
browser.get(url)
```

Closing a web browser:

```
from selenium import webdriver
```

```
url = 'http://hubzero.org'
```

```
browser = webdriver.Firefox()
```

```
browser.get(url)
```

```
...
```

```
browser.quit()
```

Finding elements on a web page:

```
from selenium import webdriver
```

```
url = 'http://hubzero.org'
```

```
browser = webdriver.Firefox()  
browser.get(url)
```

```
browser.find_element(...)  
browser.find_element_by_*(...)
```

```
browser.quit()
```

[Login](#)[Register](#)

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for *you*.

Login

Register

Search



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

HOME

GET STARTED

DOCUMENTATION

ABOUT

SUPPORT

You are here: Login

Login

Log in with your
Hub account.

Username:

No account?

Register. It's free.

Is this really free?

Yes! Use of HUBzero resources and tools is
free for registered users. There are no hidden
costs or fees.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

```
<label>
```

```
  Username:
```

```
  <input id="username" class="inputbox"  
        type="text" name="username">
```

```
</label>
```



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[Login](#)[Register](#)[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

Log in with your
Hub account.

Username:

No account?

Register. It's free.

Is this really free?

Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Id

Examples:

id='username'

<label>

Username:

<input id="username" class="inputbox"
type="text" name="username">

</label>

```
e = browser.find_element_by_id(id)
```



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[Login](#)[Register](#)[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

Log in with your
Hub account.

Username:

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Name

Examples:

name='username'

```
<label>  
  Username:  
  <input id="username" class="inputbox"  
        type="text" name="username">  
</label>
```

```
e = browser.find_element_by_name(name)
```


Hubzero
PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

HOME GET STARTED DOCUMENTATION ABOUT SUPPORT

You are here: Login

Login

Log in with your Hub account.

Username:

No account? Register. It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

<label>

Username:

```
<input id="username" class="inputbox" type="text" name="username">
```

</label>

Locator Type: Name

Examples:

```
loc = "//input[@id='username']"
```

```
loc = "//input[contains(@class,'inputbox')]"
```

```
e = browser.find_element_by_xpath(loc)
```



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[Login](#)[Register](#)[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

Log in with your
Hub account.

Username:

No account?

Register. It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is
free for registered users. There are no hidden

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Name

Examples:

loc = "input[id='username']"

loc = "#username"

loc = ".inputbox"

<label>

Username:

<input id="username" class="inputbox"
type="text" name="username">

</label>

```
e = browser.find_element_by_css_selector(loc)
```



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

Login Register

Search

HOME GET STARTED DOCUMENTATION ABOUT SUPPORT

You are here: Login

Login

Log in with your
Hub account.

Username:

[Forgot your Username?](#)

Password:

No account?

[Register.](#)

Is this really free?

Yes! Use of HUBzero resources and tools is
free for registered users. There are no hidden

Locator Types:

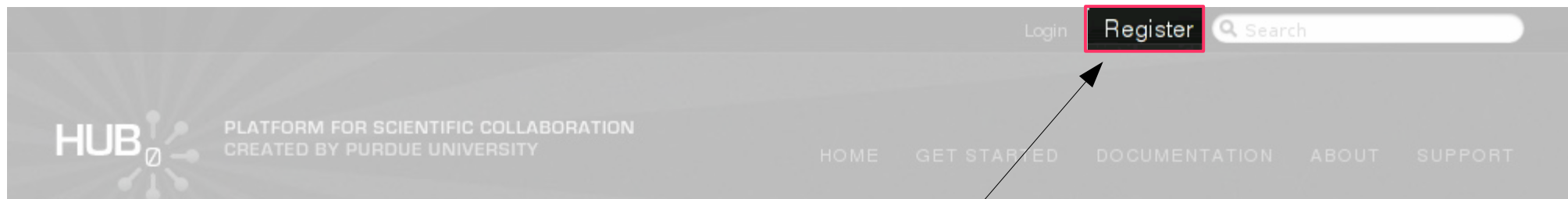
- Id
- Name
- Link
- Xpath
- CSS

```
<p>  
  <a href="/register">Register</a>  
  . It's free!  
</p>
```

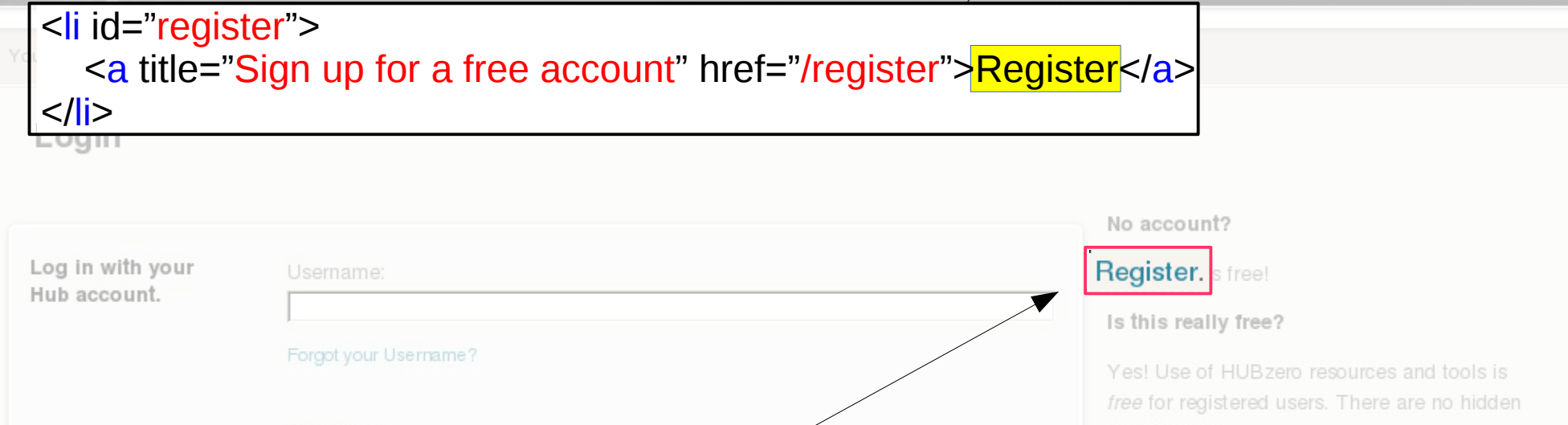
Locator Type: Link

Examples:
text = 'Register'

```
e = browser.find_element_by_link_text(text)
```



```
<li id="register">  
  <a title="Sign up for a free account" href="/register">Register</a>  
</li>
```



```
<p>  
  <a href="/register">Register</a>  
  . It's free!  
</p>
```

Locator Type: Link

Examples:
link=Register

Hub

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

HOME

GET STARTED

DOCUMENTATION

ABOUT

SUPPORT

Login

Register

Search

<li id="register">
 Register

Log in with your
Hub account.

Username:

[Forgot your Username?](#)

No account?

Register. It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is
free for registered users. There are no hidden
costs.

<p>
 Register
 . It's free!
</p>

css="li[id='register'] > a"

css="a[title='Sign up for a free account']"

[Login](#)[Register](#)**HUB**PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)You are here: [Login](#)

Login

**Log in with your
Hub account.**

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)**No account?**[Register](#). It's free!**Is this really free?**

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

**Why is registration required for parts of
HUBzero?**

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```

Hub

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

Login

Register

Search

HOME

GET STARTED

DOCUMENTATION

ABOUT

SUPPORT

You are here: Login

Login

Log in with your Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

...

e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys("hctest")
...

Login



PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[Login](#)[Register](#)[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

Log in with your
Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```




PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[Login](#)[Register](#)[HOME](#)[GET STARTED](#)[DOCUMENTATION](#)[ABOUT](#)[SUPPORT](#)

You are here: [Login](#)

Login

Log in with your
Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

No account?

[Register](#). It's free!


Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```

[Login](#) | [Register](#)

 **HUB** PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#) [GET STARTED](#) [DOCUMENTATION](#) [ABOUT](#) [SUPPORT](#)

You are here: [Login](#)

Login

Log in with your Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

No account?

[Register.](#) It's free!

Is this really free?


Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

```
<label>
  Password:
  <input id="passwd" type="password"
    name="passwd">
</label>
```

```
...
e = browser.find_element_by_css_selector("input[id='passwd']")
e.clear()
e.send_keys("pass123")
...
```

[Login](#) | [Register](#)

 **HUB** PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

[HOME](#) [GET STARTED](#) [DOCUMENTATION](#) [ABOUT](#) [SUPPORT](#)

You are here: [Login](#)

Login

Log in with your Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

No account?
[Register](#). It's free!

Is this really free?
Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

```
...  
e = browser.find_element_by_css_selector("input[name='Submit']")  
e.click()  
...
```

Login

Log in with your Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

...

```
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")
```

```
e = browser.find_element_by_css_selector("input[id='passwd']")  
e.clear()  
e.send_keys("pass123")
```

```
e = browser.find_element_by_css_selector("input[name='Submit']")  
e.click()
```

...

Login automation script

```
from selenium import webdriver

url = 'http://hubzero.org'

browser = webdriver.Firefox()
browser.get(url)

e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys("hctest")

e = browser.find_element_by_css_selector("input[id='passwd']")
e.clear()
e.send_keys("pass123")

e = browser.find_element_by_css_selector("input[name='Submit']")
e.click()

assert(...)

browser.quit()
```

You are here: [Members](#) > [Derrick Kearney](#)

Derrick Kearney ▸ Dashboard

[Personalize Dashboard](#) ▾

Introduction

Welcome to your customizable dashboard page!

To get started, click the "Personalize Dashboard" button towards the top of this page. You will then be presented with a list of modules you may add to your page. At any time you may remove any unwanted modules or rearrange the current modules by drag-and-drop!

My Tools

Recent**Favorites****All Tools**[Hello World](#) [Structure](#) [Calculator](#) [Integer](#) [Workspace](#)

These are your most recently used tools.

My Sessions

☐ [Workspace](#) ☐ [Workspace](#) ☐ [Workspace \(11:17 pm\)](#)

Storage (manage)

88% of 30GB


Dashboard

[Profile](#)[Contributions](#) 14[Groups](#) 5[Favorites](#)[Messages](#) 701[Resume](#)[Blog](#)[Projects](#) 1

Derrick Kearney (dkearney)
Logout
My Account
701 New Message(s)
Search

HUB
PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

You are here: Members > Derrick Kearney



Derrick Kearney > Dashboard

Introduction

Welcome to your customizable dashboard page!

To get started, click the "Personalize Dashboard" button towards the top of this page. You will then be presented with a list of modules you may add to your page. At any time you may remove any unwanted modules or rearrange the current modules by drag-and-drop!

Dashboard

Profile

Contributions 14

Groups 5

Favorites

Messages 701

Resume

Blog

Projects 1

- [Dashboard](/members/1000/dashboard "My Dashboard")
- [Groups](/members/1000/groups "My Groups")

```

<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>

```

```
css="ul[id='page_menu'] > li:nth-child(4) > a"
```

```
css="ul[id='page_menu'] a[class='groups']"
```

```
css="#page_menu .groups"
```

Traversing tables and lists:

Printing the text of the first link, from all list items:

```
...
elist = browser.find_elements_by_css_selector('#page_menu li')

for e in elist:
    link = e.find_element_by_css_selector('a')
    print 'link text = %s' % (link.text())
...
```

```
<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>
```


Traversing tables and lists:

Printing the title attribute of the first link, from all list items:

```
...
elist = browser.find_elements_by_css_selector('#page_menu li')

for e in elist:
    link = e.find_element_by_css_selector('a')
    print 'link text = %s' % (link.get_attribute('title'))
...
```

```
<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>
```

Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

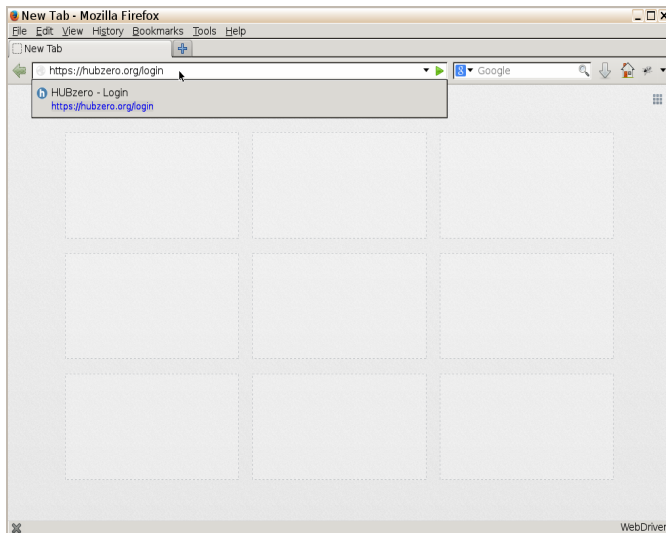
```
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

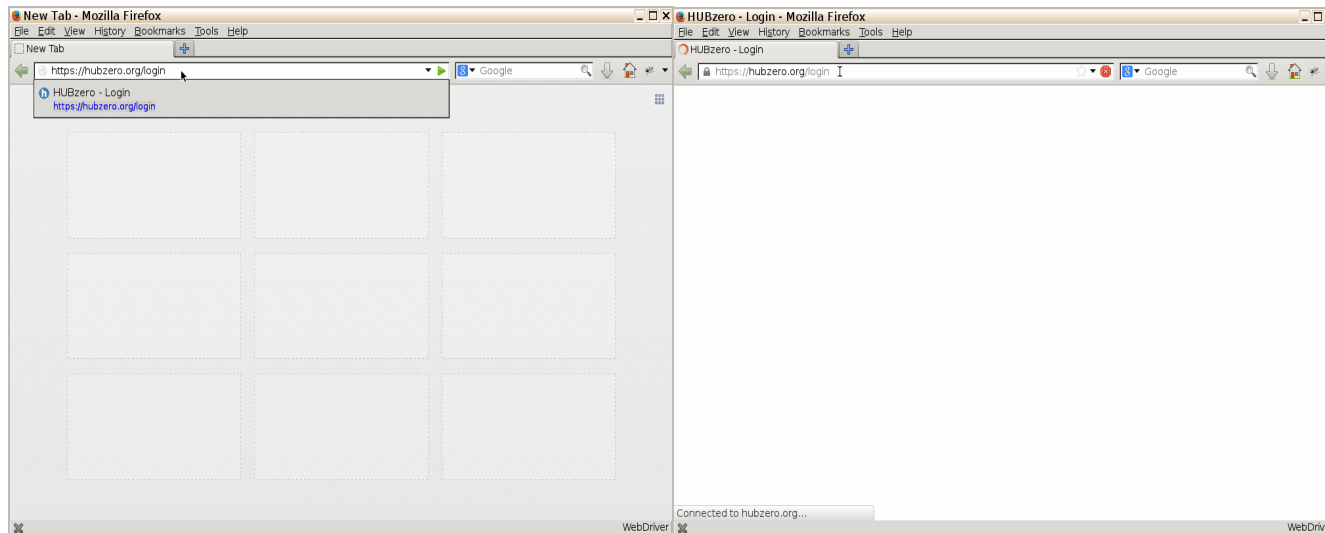
```
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



Implicit waits:

```
from selenium import webdriver
```

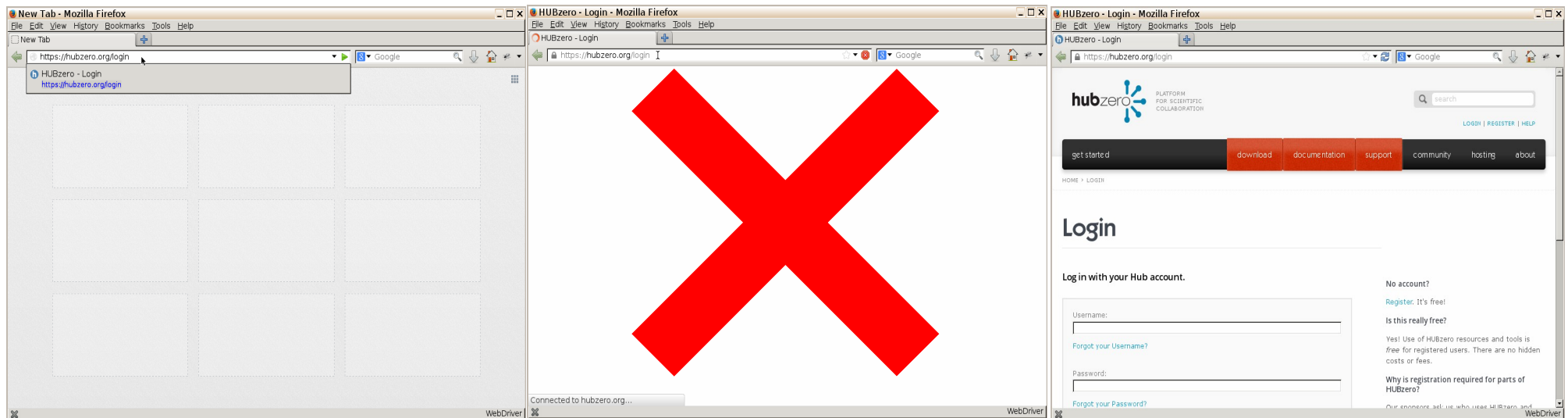
```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

```
browser.get(url)
```

```
✗ e = browser.find_element_by_css_selector("input[id='username']")  
  e.clear()  
  e.send_keys('hctest')
```

```
browser.quit()
```



Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

→ **browser.implicitly_wait(30)**

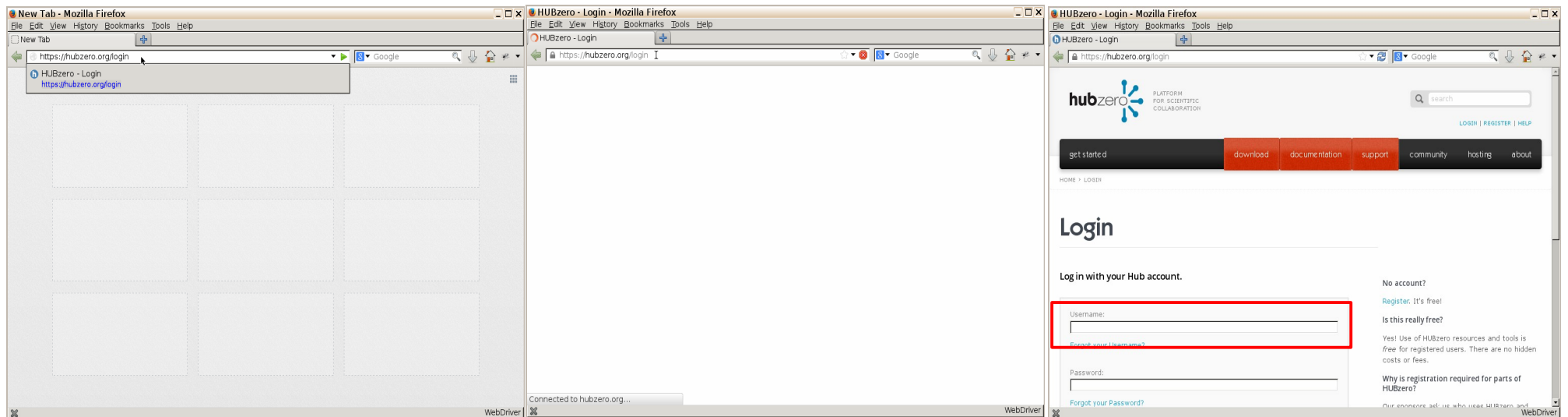
```
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



Explicit waits:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
browser.get(url)
```

```
wait = WebDriverWait(browser, 10)
wait.until(CONDITION)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys('hctest')
```

```
browser.quit()
```

Explicit waits:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
browser.get(url)
```

```
wait = WebDriverWait(browser, 10)
wait.until(EC.visibility_of_element_located(
    (By.CSS_SELECTOR, '#username')))
```

```
e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys('hctest')
```

```
browser.quit()
```

Dealing with AJAX:

Submit a Support Ticket

Username: OPTIONAL

Name: REQUIRED

hctestuser

E-mail: REQUIRED

hctestuser@hubzerro.org

Please answer the question: REQUIRED

Is night dark or light?

light

Problem: REQUIRED

test ticket

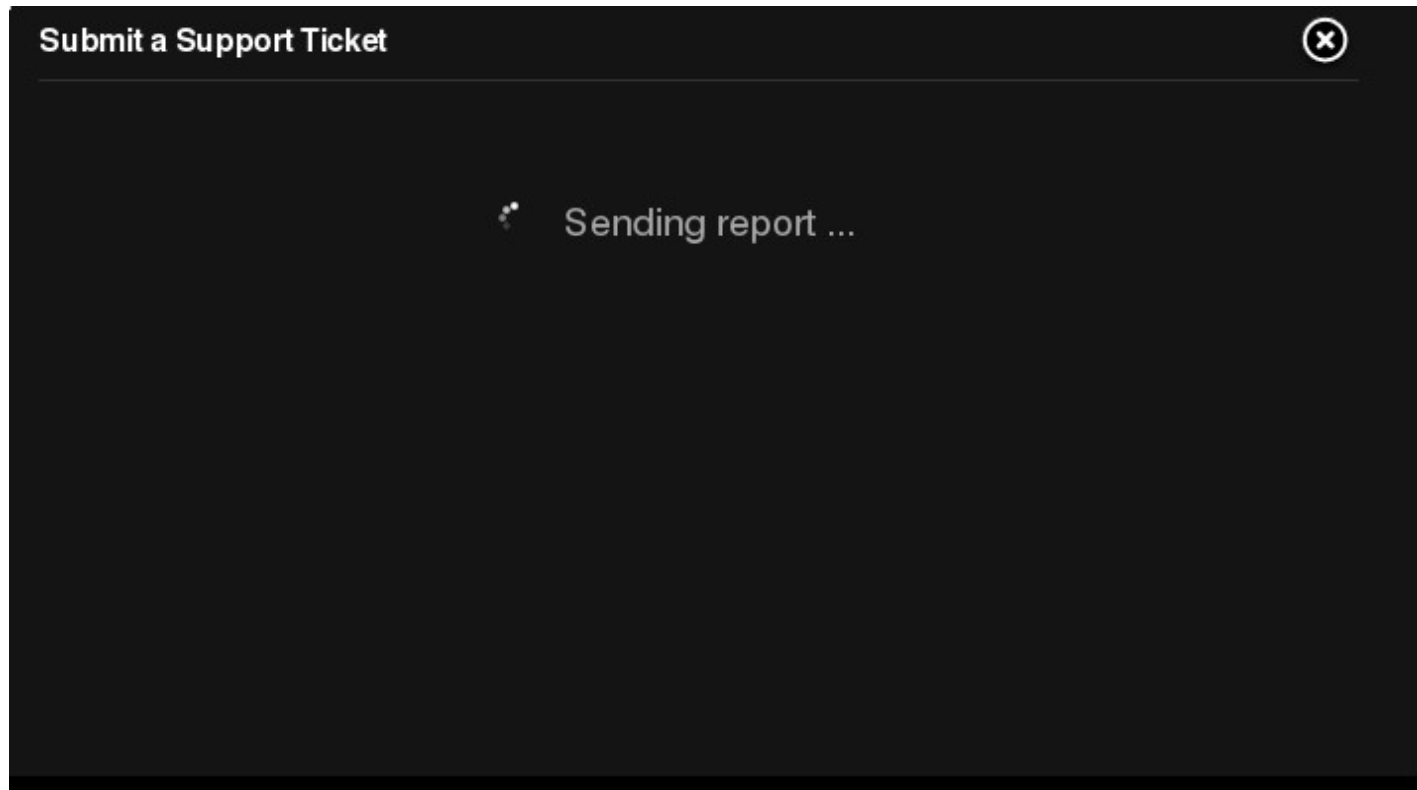
Attach a screenshot: OPTIONAL

Browse...

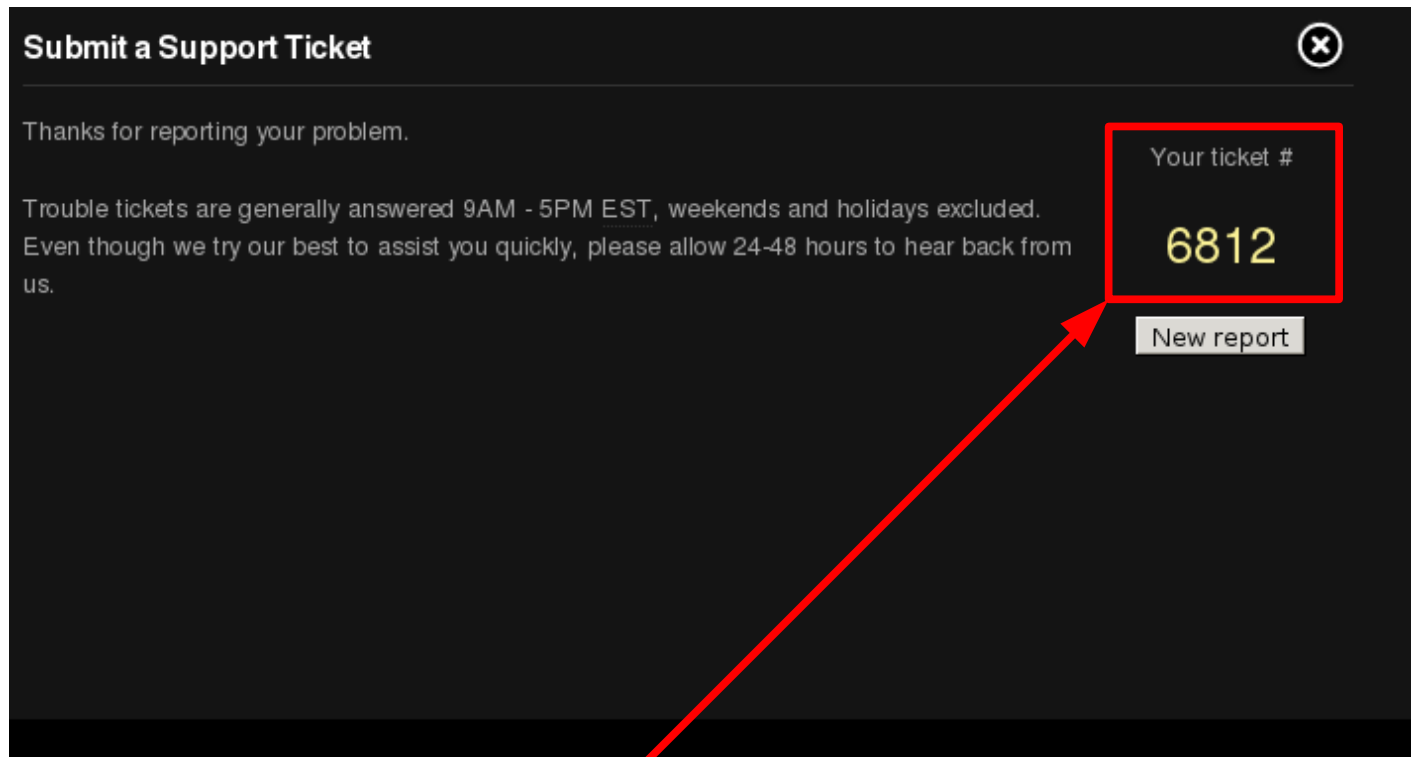
(.asf, .asx, .avi, .bmp, .csv, .doc, .docx, .eps, .gif, .gz, .html, .jpe, .jpeg, .jpg, .js, .log, .mov, .mp3, .mpeg, .mpg, .pdf, .png, .pps, .ppt, .pptx, .ra, .rm, .rtf, .swf, .tar, .tex, .tif, .tiff, .txt, .wav, .wmv, .xls, .xlsx, .xml, .xsl, .zip)

Submit

Dealing with AJAX:



Dealing with AJAX:



```
<div id="trSuccess">
  <div>
    <p>Your ticket #
      <span>
        <a title="View ticket" href="/support/ticket/6812">6812</a>
      </span>
    </p>
  </div>
</div>
```

Dealing with AJAX:

...

```
submit_ticket()
```

```
# wait for the page to refresh
```

```
wait = WebDriverWait(browser, 60)
```

```
e = wait.until(lambda browser :
```

```
    browser.find_element_by_id("trSuccess").is_displayed())
```

```
ticket_number = e.find_element_by_css_selector('a').text()
```

...

```
<div id="trSuccess">
```

```
  <div>
```

```
    <p>Your ticket #
```

```
      <span>
```

```
        <a title="View ticket" href="/support/ticket/6812">6812</a>
```

```
      </span>
```

```
    </p>
```

```
  </div>
```

```
</div>
```

QUESTIONS?
MOVIES?
MORE?

Selenium Webdriver Primer

Spensa Technologies, Inc
Sept. 22, 2016

Derrick Kearney
github.com/codedsk
tellask@gmail.com

Testing vs Checking

Testing is...

- * Exploring & Learning
- * Experimenting
- * Judgment & Values
- * Answers the question
"Is there a problem?"

Checking is...

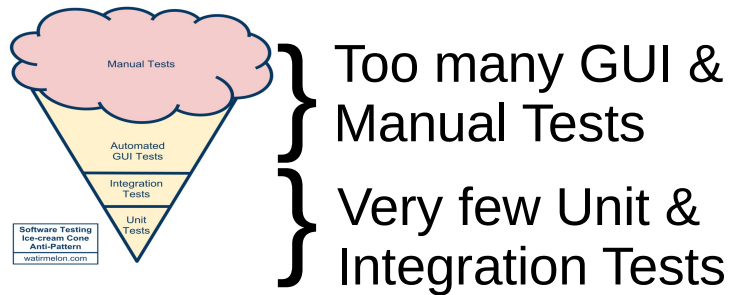
- * Confirming
- * Pass / Fail
- * "It Works" vs "It Operates"

There is an ongoing conversation within Testing communities that questions whether testing can be automated. Some people say that testing cannot be automated, but checking can. And that the automation scripts people write and call tests are really just checks.

As you are exploring, you may see these terms being used. It took me a while to understand the difference. Here is what I think is going on.

When people talk about Testing, they say it is the use of tools to gain knowledge about how the system works. It involves exploring and learning. They say you can't automate testing because testing involves having values and using judgment to decide what should happen next. It answers the question "Is there a problem?"

Typical Testing Pyramid



Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

3

Another thing you may run into is the Testing Pyramid. It is important to consider what kind of testing you want to be doing and how much time, effort, or resources you want to spend at each level.

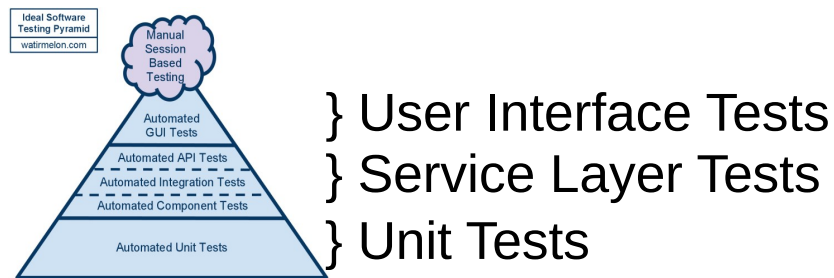
There are always things to test. This pattern gives us an idea of the resource allocation that seems to have worked for other people.

The testing pyramid splits resources into checking the business logic, which include the graphical user interface and public facing api, and checking the technology logic, which includes your core functions and libraries that get called by the business logic.

Here, we see the dreaded Ice-cream Cone Antipattern.

A lot of resources are focused on building automated checks for the graphical user interface and manually testing the business level, while too few resources are allocated toward making sure the technology layer works

Ideal Testing Pyramid



Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

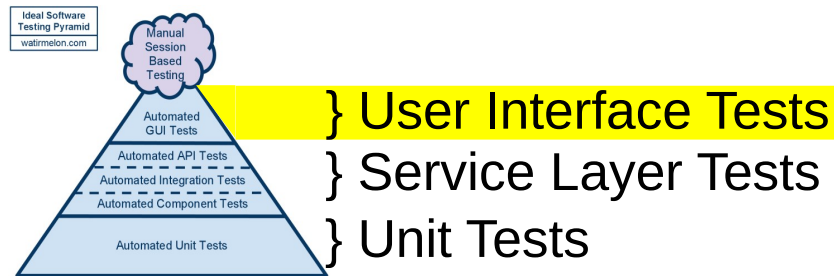
4

Ideally, your organization's testing pyramid would look like this.

Right away we notice that the bottom layer of the Pyramid, the most broad layer and the largest by area, is unit testing, followed slightly above it by component and integration testing. These show that you should spend the majority of your testing effort building a broad base of unit and component tests that check that the pieces of your program operate as you expect them to. This follows along with the pieces of your software you expect to use the most. Many of the hard errors you'll find are going to be deep in the software and these functions will be used the most, in the technology layer.

Above the unit and component tests are what are called the business facing tests. The business layer portion of the pyramid is smaller. The idea is that if you do the legwork of making sure the lowest level pieces of your software work properly, you don't need to spend as many resources checking if the pieces above work properly because you

Ideal Testing Pyramid



Ideal Automated Testing Pyramid & Ice-cream Cone Antipattern -
<http://watirmelon.com/2012/01/31/introducing-the-software-testing-ice-cream-cone/>

5

Today we are going to be focusing on building tests for the business layer and in particular, the graphical user interface.

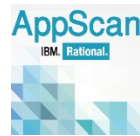
There are many tools that are available to perform testing at the graphical user interface level and they usually fall within two categories.

Automated Testing Tools

Web Crawlers



CrawlJax



IBM AppScan

Pros:

- Little Developer Coordination
- Semi directed exploration
- Web App is a black box

Cons:

- Time consuming to run
- No authority to judge results

6

The first category of tools are web crawlers. These include tools like Crawljax and IBM AppScan. Web Crawlers need little developer coordination. This allows people to quickly get started running the programs.

Because these tools are crawling a website, they can get into a bit of trouble navigating to places they shouldn't be.

These tools usually allow the user to set a blacklist of urls not to visit.

The best thing is that they treat the web application as a black box. The user doesn't have to do much work to get them up and running. Unfortunately, because they need to traverse and exhaust the whole website graph, they can take a really long time to run. Additionally there is no authority to tell if a behavior that is experienced while navigating it correct or incorrect for the web application.

Automated Testing Tools

Test Recorders



Selenium

Pros:

- Very customizable
- Produces specific targeted tests
- Programmable in a script



QuickTest Pro

Cons:

- Time consuming to build tests
- Allows for brittle test designs



Sahi Pro

On the other hand there are other tools that fall into the category of test recorders. These include applications like Selenium Webdriver, QuickTest Pro, and Sahi Pro. Test recorders allow the user to provide explicit web application navigation based on actions that were previously recorded. This makes them very customizable and able to produce targeted tests. With some, you can incorporate them in a script.

Automated Testing Tools

Test Recorders



Selenium

- Developed By Jason Huggins and Simon Stewart at ThoughtWorks



QuickTest Pro

- Browser vendor support through W3C spec

- Automation written in “your favorite language”...



Sahi Pro

- Worldwide support Community

8

Of these tools, I only have experience with Selenium Webdriver and that is the one we will be talking about today.

It was developed by Jason Huggins and later combined with another tool by Simon Stewart at ThoughtWorks.

It works with many of the major browsers by using the browser's own internal automation. There is a team of people working on a W3C specification, that includes browser vendors. Once that is completed it will help standardize the protocols and capabilities that some vendors may provide that others do not.

Selenium officially supports a hand full of programming languages like Python, Java, C#, and Ruby. Other language bindings are provided by 3rd parties.

There is a huge community to provide support for the library. They hold conferences around the world, there are mailing lists and google hangouts, and the documentation is pretty

Automated Testing Tools

Test Recorders



Selenium

What can it do?



QuickTest Pro



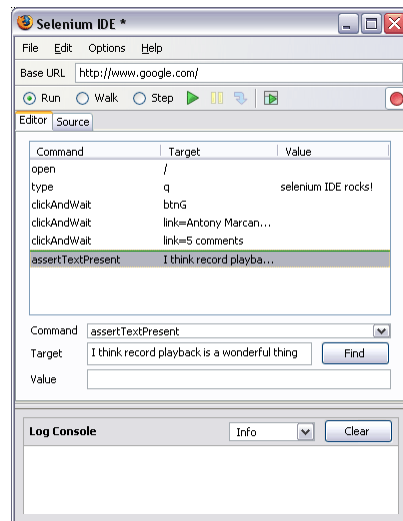
Sahi Pro

- Open web browser
- Locate elements on page
- Type in or click elements
- Wait for triggered events
- Simulate moving the mouse

9

Most of the stuff you'll end up doing with Selenium will be pretty simple. Much of the automation will be Opening a web browser, navigating to a URL, locating an element on the web page, interacting with the element by typing into it or clicking it or reading it, waiting for an event to be triggered, or moving the mouse over objects and evaluating the result.

A brief word about Selenium IDE...



What can it do?

- Record actions, replay as tests
- Works in Firefox only
- Selenium 1.0 style tests
- Tests can be brittle

Try it out, then move on

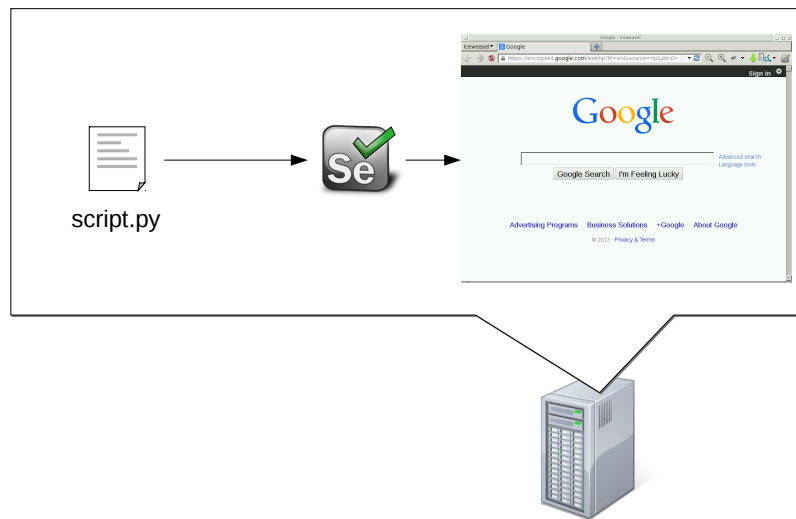
10

Let me take some time here to mention Selenium IDE. So far I've been talking about Selenium WebDriver, otherwise known as Selenium 2.0.

A long time ago there was a Selenium 1.0. People don't speak about it now a days unless they have to. It worked by running a little server in the background and your script would send JSON command to the server, and the server would then talk to the browser. I think this caused some problems like CSRF. But now there is version 2.0 (and soon version 3.0?)

Newer versions of the Selenium binaries have been backwards compatible but I think I heard that will be ending with the newest releases. Selenium 1.0 is dying. So if you are getting started with it, try it out, get to understand how to locate element on the web page, but then move on to using WebDriver.

How Selenium Works



The general way Selenium works is that you write a script or a program that interacts with the Selenium library, and that launches a web browser that has an open port listening for automation commands. Automation commands are accepted by the browser, the browser performs actions and then sends some results back to your program.

Hub

PLATFORM FOR SCIENTIFIC COLLABORATION
CREATED BY PURDUE UNIVERSITY

HOME

GET STARTED

DOCUMENTATION

ABOUT

SUPPORT

Login

Register

Search

You are here: Login

Log in with your Hub account.

Username:

hctest

Forgot your Username?

Password:

Forgot your Password?

☐ Remember Me

No account?

[Register](#). It's free!

Is this really free?

Yes! Use of HUBzero resources and tools is *free* for registered users. There are no hidden costs or fees.

Why is registration required for parts of HUBzero?

Our sponsors ask us who uses HUBzero and what they use it for. Registration helps us answer these questions. Usage statistics also focus our attention on improvements, making the HUBzero experience better for you.

Login

Then we would locate the password field and type our password into it

Loading the Selenium Webdriver module:

```
from selenium import webdriver
```

These steps can also be done in Selenium. Our first step is to import the Selenium WebDriver libraries into our script.

Start a web browser:

```
from selenium import webdriver  
  
url = 'http://hubzero.org'  
  
browser = webdriver.Firefox()
```

Next we launch a browser by instantiating the webdriver dot browser name class.

Navigating to a web page:

```
from selenium import webdriver  
  
url = 'http://hubzero.org'  
  
browser = webdriver.Firefox()  
browser.get(url)
```

Next we navigate to our destined url using the `get()` method of our browser object

Closing a web browser:

```
from selenium import webdriver

url = 'http://hubzero.org'

browser = webdriver.Firefox()
browser.get(url)
...
browser.quit()
```

And when we are all done, we'll need to close the browser, so we add a call to the `browser.quit()` method here.

Finding elements on a web page:

```
from selenium import webdriver

url = 'http://hubzero.org'

browser = webdriver.Firefox()
browser.get(url)

browser.find_element(...)
browser.find_element_by_*(...)

browser.quit()
```

Opening and immediately closing a browser isn't fun. Remember, our original goal was to login to the website.

After we open the browser and navigate to the login web page, we need to look for the username and password fields on the web page. The browser object provides a few functions to help us out. There are the `find_element` and `find_element_by_*` functions.

The image shows a screenshot of the HUB (Platform for Scientific Collaboration) login page. The page has a header with the HUB logo and navigation links. A box on the right side of the page lists 'Locator Types' with a bulleted list: Id, Name, Link, Xpath, and CSS. Below the login form, the HTML code for the username field is displayed, showing a label and an input field with attributes id='username', class='inputbox', type='text', and name='username'.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

```
<label>
  Username:
  <input id="username" class="inputbox"
    type="text" name="username">
</label>
```

And we are interested in finding the username field.

There are a number of ways to locate elements on the web page. They all deal with figuring out a way to name the element.

This naming of elements is often referred to as locator strategies.

The HTML for our username field looks like this. It is an input field with an id of username, a class attribute of inputbox, a type of text, and a name of username.

The image shows a screenshot of the HUB login page. A red box highlights the 'Username:' label and its corresponding input field. A red line connects this box to a list of 'Locator Types' (Id, Name, Link, Xpath, CSS) and another red line connects it to an HTML snippet. The HTML snippet shows the following code:

```
<label>  
  Username:  
  <input id="username" class="inputbox"  
        type="text" name="username">  
</label>
```

To the right of the HTML snippet, the text 'Locator Type: Id' is displayed, followed by 'Examples:' and 'id='username''. Below this, a yellow box contains the code: `e = browser.find_element_by_id(id)`.

We could use the input element's id to identify the element. Id's on a web page are supposed to be unique so they tend to be good locators to use.

To use the element's id, you can hand the id to the `find_element_by_id` method. And it would find and return an object that represents the username element.

The image shows a screenshot of the HUB (Platform for Scientific Collaboration) login page. The page has a header with the HUB logo and navigation links. The main content area is titled 'Login' and contains a form with a 'Username:' label and an input field. To the right of the form, there are links for 'No account?', 'Register', and 'is this really free?'. Below the form, there is a code snippet for the HTML structure of the login form. A red box highlights the 'name="username"' attribute in the code snippet. A red line connects this box to a 'Locator Types' list on the right, where 'Name' is highlighted with a red box. Another red line connects the 'Name' locator type to the 'Locator Type: Name' section, which provides examples and a code snippet for using the 'find_element_by_name' method.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Name

Examples:
name='username'

```
e = browser.find_element_by_name(name)
```

`<label>
Username:
<input id="username" class="inputbox"
type="text" name="username">
</label>`

Similarly, we could use the name attribute and hand that to the `find_element_by_name` method. If there is only one element on the web page with that name attribute, it will return the element we want.

The image shows a screenshot of the HUB (Platform for Scientific Collaboration) login page. The page has a header with the HUB logo and navigation links. A search bar is in the top right. The main content area has a 'Login' section with a 'Username:' label and an input field. To the right of the input field, there is a box titled 'Locator Types:' containing a list: 'Id', 'Name', 'Link', 'Xpath', and 'CSS'. The 'Xpath' item is highlighted with a red border. Below the input field, there is an example of HTML code for the 'Username:' label and input field. To the right of the HTML code, there is a section titled 'Locator Type: Name' with examples of XPath locators. At the bottom right, there is a yellow box containing the Selenium code: `e = browser.find_element_by_xpath(loc)`.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Name

Examples:

```
loc = "//input[@id='username']"
loc = "//input[contains(@class,'inputbox')]"
```

```
e = browser.find_element_by_xpath(loc)
```

Selenium also understands XPATH locators. We could use the XPATH language to build a locator that uses one of the input field's attributes to identify it, and then hand the locator to the `find_element_by_xpath` method.

The image shows a screenshot of the HUB (Platform for Scientific Collaboration) login page. The page has a header with the HUB logo and navigation links. The main content area includes a login form with a 'Username:' label and an input field. To the right of the input field, there is a 'Locator Types:' list with options: Id, Name, Link, Xpath, and CSS. The 'CSS' option is highlighted with a red box. Below the login form, there is a 'Locator Type: Name' section with examples of locators. A yellow box contains the Selenium code snippet: `e = browser.find_element_by_css_selector(loc)`.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Name

Examples:

```
loc = "input[id='username']"  
loc = "#username"  
loc = ".inputbox"
```

```
e = browser.find_element_by_css_selector(loc)
```

Selenium also understand CSS selectors. So we could use a CSS selector and hand that to the `find_element_by_css_selector` to get an object representing our element on the web page.

The image shows a screenshot of the HUBzero website's login page. The page has a header with the HUBzero logo and navigation links. The main content area includes a login form with fields for 'Username' and 'Password', and a 'Register' link. A red box highlights the 'Register' link text. A red line connects this box to a list of 'Locator Types' on the right, which includes 'Id', 'Name', 'Link', 'Xpath', and 'CSS'. The 'Link' type is highlighted with a red box. Below the list, the text 'Locator Type: Link' is shown, followed by an example: 'Examples: text = 'Register''. At the bottom, a code snippet is shown: 'e = browser.find_element_by_link_text(text)'.

Locator Types:

- Id
- Name
- Link
- Xpath
- CSS

Locator Type: Link

Examples:
text = 'Register'

```
e = browser.find_element_by_link_text(text)
```

Now, there is also a `find_element_by_link_text` method. You can include the text of a link and ask it to give you back an element, but using link text is a bad idea. It tends to make brittle locators. As soon as someone changes the text of the link, your automation will fail.

The screenshot shows the HUBzero website. At the top, there is a header with a 'Register' link highlighted in a red box. Below the header, there is a 'Login' section with a 'Register' link highlighted in a red box. The 'Register' link in the header is associated with the following HTML code snippet:

```
<li id="register">
  <a title="Sign up for a free account" href="/register">Register</a>
</li>
```

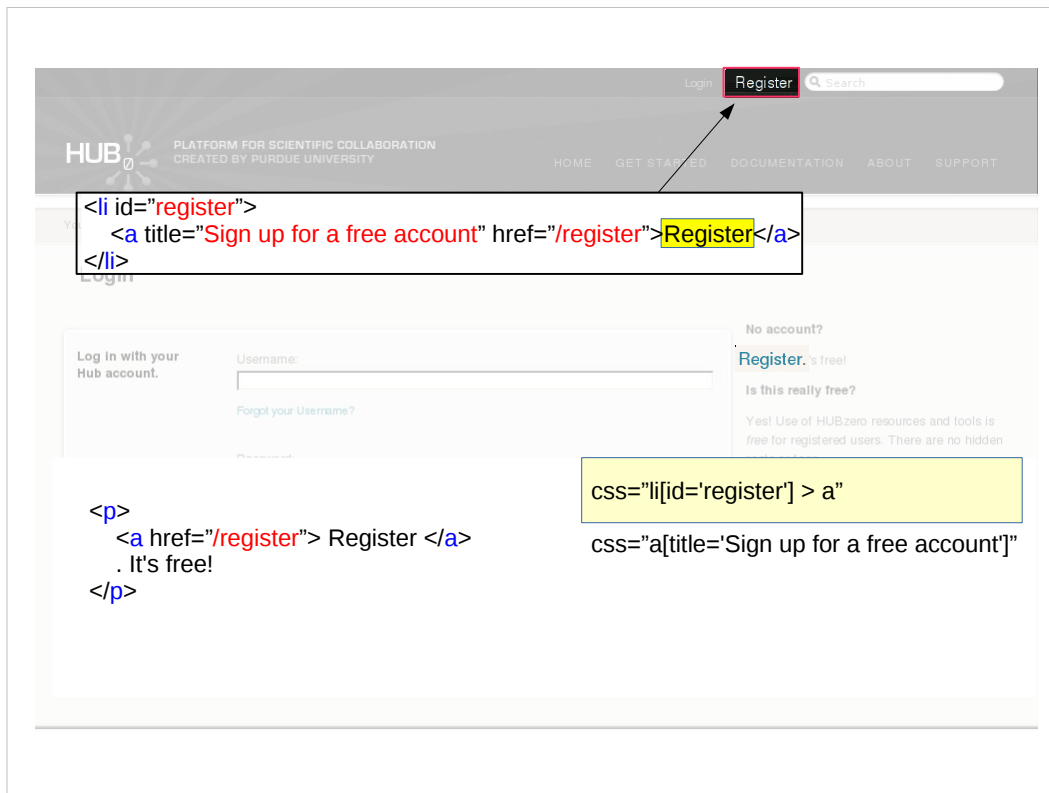
The 'Register' link in the 'No account?' section is associated with the following HTML code snippet:

```
<p>
  <a href="/register">Register</a>
  . It's free!
</p>
```

Locator Type: Link

Examples:
link=Register

Also, what happens if your web page has two links with the same text, like in this example. There is a “Register” link at the top of the page, and another one in the middle. Which do you get back?



It is better to choose a different, more flexible locator like an id or class, something that won't change often.

This example shows why it is important to instrument your code for automation. If you don't build in the hooks for automation, but using good id and class names, locating elements becomes difficult.

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, the text "PLATFORM FOR SCIENTIFIC COLLABORATION CREATED BY PURDUE UNIVERSITY", and navigation links: HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT. There are also links for Login and Register, and a search bar.

The main content area is titled "Login". It contains a login form with fields for Username and Password, and links for "Forgot your Username?" and "Forgot your Password?". To the right of the form, there is a section for "No account?" with a "Register. It's free!" link, and a section for "Is this really free?" with a paragraph of text. Below the form, there is a "Login" button.

Overlaid on the login form is a Selenium script snippet:

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```

Let's get back to our example. We are going to use a css selector to locate our element

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, navigation links (HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT), and a search bar. The main content area is titled "Login" and contains a "Log in with your Hub account." section. This section has a "Username:" label followed by a text input field, a "Forgot your Username?" link, a "Password:" label followed by a text input field, and a "Forgot your Password?" link. A red box highlights the "Username:" label and its corresponding input field. To the right of the login form is a "No account?" section with a "Register. It's free!" link and a "Is this really free?" section with explanatory text. At the bottom of the login form is a "Login" button. Two code annotations are present: one above the username input field showing the HTML structure of the label and input, and another below the password input field showing Selenium code to find and interact with the username input field.

Username:

```
<label>
Username:
<input id="username" class="inputbox"
type="text" name="username">
</label>
```

```
...
e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys("hctest")
...
```

Login

Our first step is to locate the element

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, the tagline "PLATFORM FOR SCIENTIFIC COLLABORATION CREATED BY PURDUE UNIVERSITY", and navigation links: HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT. There are also links for Login and Register, and a search bar.

The main content area is titled "Login". It contains a login form with fields for Username and Password. A red box highlights the Username field. Below the Username field is a link "Forgot your Username?". Below the Password field is a link "Forgot your Password?".

On the right side of the login form, there is a section titled "No account?" with a link "Register. It's free!". Below that is a section titled "Is this really free?" with text: "Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs or fees." Below that is a section titled "Why is registration required for parts of HUBzero?".

At the bottom of the login form is a "Login" button.

Overlaid on the bottom of the login form is a Selenium script snippet:

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```

Next we clear it out, remove any text that may have been in the field.

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, the tagline "PLATFORM FOR SCIENTIFIC COLLABORATION CREATED BY PURDUE UNIVERSITY", and navigation links: HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT. There are also links for Login, Register, and a search bar.

The main content area is titled "Login". It features a login form with the following elements:

- A message: "Log in with your Hub account."
- A "Username:" label above a text input field containing "hctest". This field is highlighted with a red border.
- A "Forgot your Username?" link below the username field.
- A "Password:" label above a password input field.
- A "Forgot your Password?" link below the password field.
- A "No account? Register. It's free!" link.
- A section titled "Is this really free?" with text: "Yes! Use of HUBzero resources and tools is free for registered users. There are no hidden costs or fees."
- A section titled "Why is registration required for parts of HUBzero?"

Overlaid on the bottom of the login form is a Selenium script snippet:

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
...
```

The "Login" button is located at the bottom of the form.

And lastly, we use the `send_keys()` method to type text into the field.

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, navigation links (HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT), and a search bar. The main content area is titled 'Login' and contains a form for logging in with a Hub account. The form has two input fields: 'Username' (containing 'hctest') and 'Password' (containing '*****'). A red box highlights the password input field. To the right of the form, there is a 'No account?' section with links for 'Register' and 'Is this really free?'. Below the form, there is a 'Login' button. Two code snippets are overlaid on the image. The first snippet, located near the password field, shows the HTML structure of the password label and input field. The second snippet, located below the form, shows Selenium code to find the password input field by CSS selector, clear it, and send keys.

```
<label>
Password:
<input id="passwd" type="password"
name="passwd">
</label>
```

```
...
e = browser.find_element_by_css_selector("input[id='passwd']")
e.clear()
e.send_keys("pass123")
...
```

We do the same thing with the password field. First we locate the element representing the input field, then we clear it out, and last we send keys to it.

The screenshot shows the HUBzero login page. The header includes the HUBzero logo, the tagline "PLATFORM FOR SCIENTIFIC COLLABORATION CREATED BY PURDUE UNIVERSITY", and navigation links: HOME, GET STARTED, DOCUMENTATION, ABOUT, SUPPORT. A search bar is also present. The main content area is titled "Login" and includes a "You are here: Login" breadcrumb. The login form has fields for "Username:" (containing "hctest") and "Password:" (containing "*****"). There are links for "Forgot your Username?" and "Forgot your Password?". To the right of the form, there is a "No account?" section with a "Register. It's free!" link, and a "Is this really free?" section with a paragraph of text. At the bottom of the form, there is a "Login" button. Two code snippets are overlaid on the image: one above the form showing the HTML for the submit button, and another below the form showing the Selenium code to click it.

HTML snippet:

```
<p class="submit">  
<input class="button" type="submit"  
value="Login" name="Submit">  
</p>
```

Selenium code snippet:

```
...  
e = browser.find_element_by_css_selector("input[name='Submit']")  
e.click()  
...
```

Our last step is to locate the “Login” button on the page. From the HTML for the widget, we see we can use the field's name attribute as a locator. Once we find the element, we can call the click method to press it.

Log in with your Hub account.

Username:

[Forgot your Username?](#)

Password:

[Forgot your Password?](#)

☐ Remember Me

Login

```
...  
e = browser.find_element_by_css_selector("input[id='username']")  
e.clear()  
e.send_keys("hctest")  
  
e = browser.find_element_by_css_selector("input[id='passwd']")  
e.clear()  
e.send_keys("pass123")  
  
e = browser.find_element_by_css_selector("input[name='Submit']")  
e.click()  
...
```

So all together, our automation for the page looks like this.

Locate, clear, and type into the username field
Locate, clear, and type into the password field
Then locate and click the login button.

After you have logged in, you can check that login was successful by locating an element that is only on the web page when you have successfully logged in is present on the page.

Alternatively, you would check for a login failure message, but this may slow down all of your successful logins.

Login automation script

```
from selenium import webdriver

url = 'http://hubzero.org'

browser = webdriver.Firefox()
browser.get(url)

e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys("hctest")

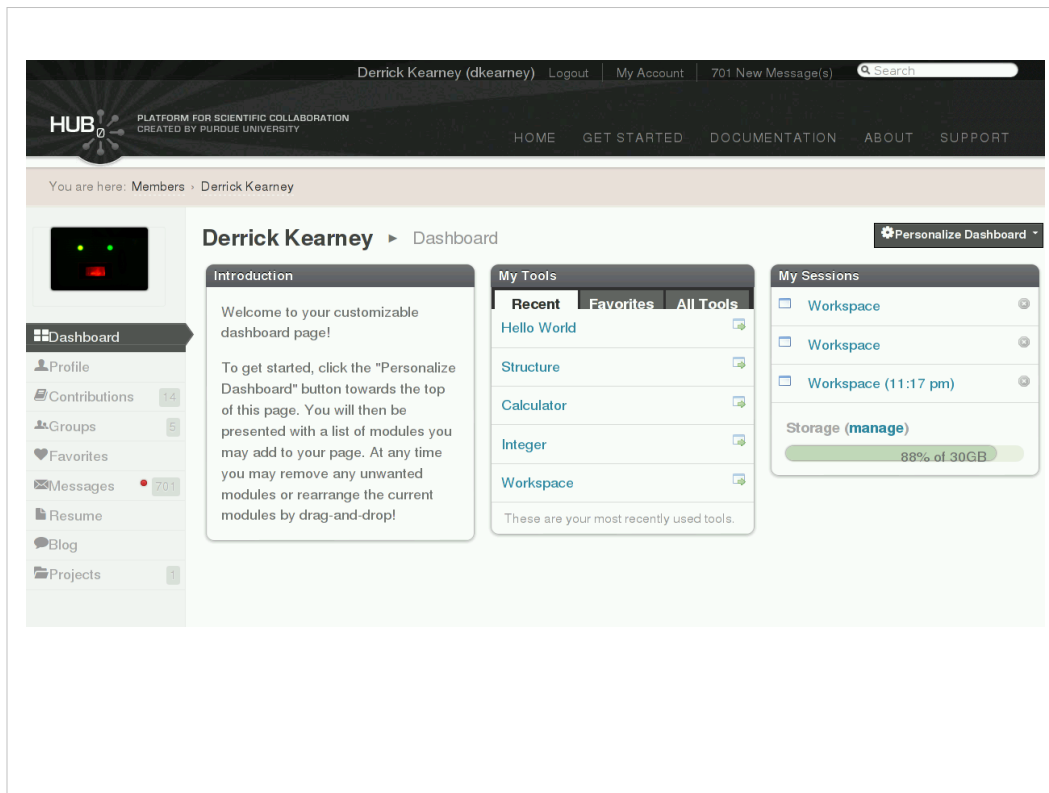
e = browser.find_element_by_css_selector("input[id='passwd']")
e.clear()
e.send_keys("pass123")

e = browser.find_element_by_css_selector("input[name='Submit']")
e.click()

assert(...)

browser.quit()
```

Here is a look at the whole login automation script.



Here is another example of using locators to find elements. After logging into the website, I get a menu on the left side of the web page. How could I read the link names and cycle through the links until I found one I was interested in?

The screenshot shows a web application interface for Derrick Kearney. The side menu on the left contains several links: Dashboard, Profile, Contributions, Groups (highlighted with a red box), Favorites, Messages, Resume, Blog, and Projects. A red arrow points from the 'Groups' link to a code snippet showing the HTML structure of the menu. The code snippet shows a list of links, with the 'Groups' link highlighted. Below the code snippet, three CSS selectors are listed in yellow boxes.

```
<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>
```

css="ul[id='page_menu'] > li:nth-child(4) > a"

css="ul[id='page_menu'] a[class='groups']"

css="#page_menu .groups"

The HTML for the side menu looks like this. And if I wanted to access the 4th link in the menu, the Groups link, I would need to write a locator to identify it.

The first locator I may try is `ul > li 4th child > a`. This is a brittle locator, if the website is redesigned and another element is placed in the list, the Groups link may move to 5th position or 3rd position.

The next locator fixes these problem by using the class attribute.

We can rewrite the second locator to make it shorter as seen in the third locator.

Traversing tables and lists:

Printing the text of the first link, from all list items:

```
...
elist = browser.find_elements_by_css_selector("#page_menu li")

for e in elist:
    link = e.find_element_by_css_selector('a')
    print 'link text = %s' % (link.text())
...
```

```
<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>
```

If we wanted to print the text of each link in the list, we could use the `find_elements_by_css_selector` method to return all elements that match our locator, not just the first. There are a whole host of complimentary methods that return a list of element objects instead of a single element object.

In a for loop, we can loop through our list of element objects, and for each object, call the `text` method to get the text of the widget.

Traversing tables and lists:

Printing the title attribute of the first link, from all list items:

```
...
elist = browser.find_elements_by_css_selector("#page_menu li")

for e in elist:
    link = e.find_element_by_css_selector('a')
    print 'link text = %s' % (link.get_attribute('title'))
...
```

```
<ul id="page_menu">
  <li class="active">
    <a class="dashboard"
      href="/members/1000/dashboard"
      title="My Dashboard"> Dashboard </a>
  </li>
  ...
  <li class="">
    <a class="groups"
      href="/members/1000/groups"
      title="My Groups"> Groups </a>
  </li>
  ...
</ul>
```

Similarly, if we wanted to print the title attribute of the first link for all list items, we could use the `find_elements_by_css_selector` to get the list of matching element objects. Next, in our for loop, iterate over each element object, calling the `get_attribute()` method.

This pretty much concludes the basics of using the Selenium WebDriver library. There are a few more topics that I think are helpful to know about.

Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

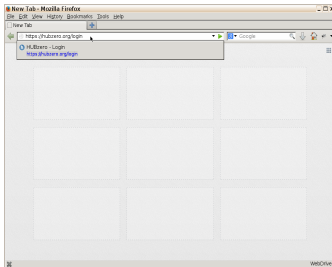
```
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



The first being waiting strategies.

If you run enough scripts, you'll eventually see errors like this happening.

You open your browser, navigate to a url

Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

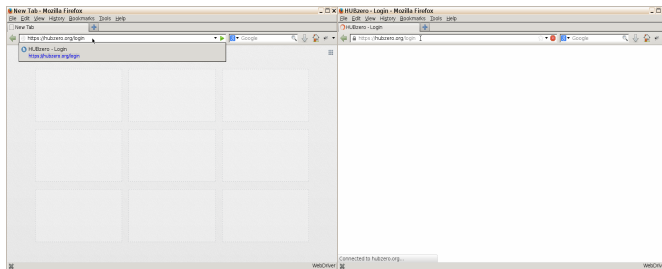
```
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



And before the web page finishes loading, your script is looking for an element on the page.

Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

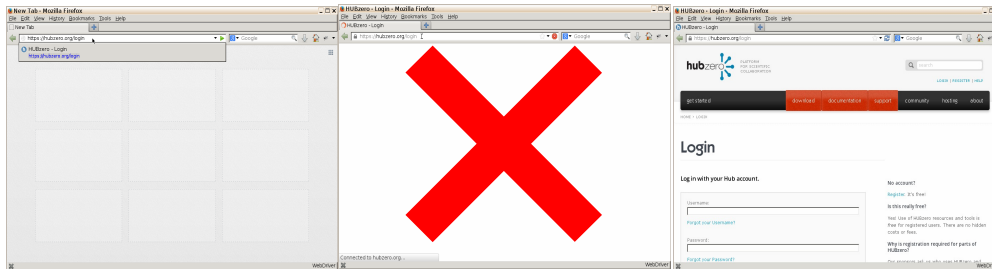
```
browser.get(url)
```

```
✗ e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



It fails to find the element and your automation dies.
Just as it dies, the web page finishes loading.

With modern web pages and ajax, it is really hard to tell when a web page has “finished” loading. There is no “finished” tag that you can look for.

Implicit waits:

```
from selenium import webdriver
```

```
url = 'https://hubzero.org/login'
```

```
browser = webdriver.Firefox()
```

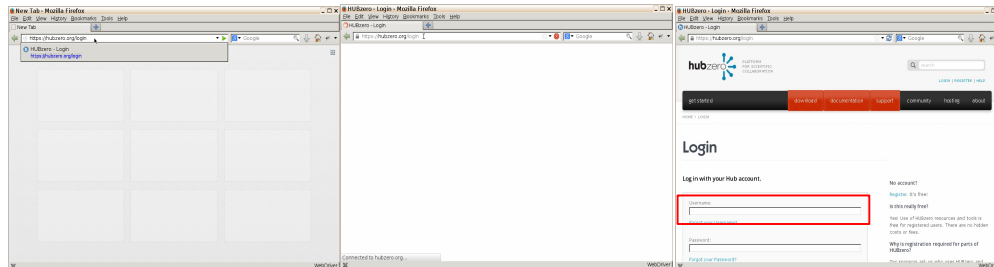
```
→ browser.implicitly_wait(30)  
browser.get(url)
```

```
e = browser.find_element_by_css_selector("input[id='username']")
```

```
e.clear()
```

```
e.send_keys('hctest')
```

```
browser.quit()
```



To help give your automation a little flexibility, you can add what is called an implicit wait to your element lookup. You only need to call the implicit wait method once for the browser object. It adds a polling strategy to the lookup methods. So if the `find_element` method doesn't find the element on the first try, it will sleep and try again every half second or so until it reaches the timeout you set.

Here the wait is implicit because you did not have to set it explicitly for each `find_element` call. The wait is implied.

Explicit waits:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

url = 'https://hubzero.org/login'

browser = webdriver.Firefox()
browser.get(url)

wait = WebDriverWait(browser, 10)
wait.until(CONDITION)

e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys('hctest')

browser.quit()
```

Alternatively, there are explicit wait as well. To use an explicit wait, you need to include the `WebDriverWait` class and you can also include the `expected_condition` class. The `WebDriverWait` class creates the explicit wait object that will wait until either some condition is met, or the timeout is passed.

Explicit waits:

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC

url = 'https://hubzero.org/login'

browser = webdriver.Firefox()
browser.get(url)

wait = WebDriverWait(browser, 10)
wait.until(EC.visibility_of_element_located(
    (By.CSS_SELECTOR, '#username')))

e = browser.find_element_by_css_selector("input[id='username']")
e.clear()
e.send_keys('hctest')

browser.quit()
```

In this case, we can use one of the provided conditions from the expected condition class to check for the visibility of an element. If the element is visible on the page, we are ready to move on and locate the element.

Note there are some conditions that check visibility vs existence of an element. An element can exist on the web page without being visible.

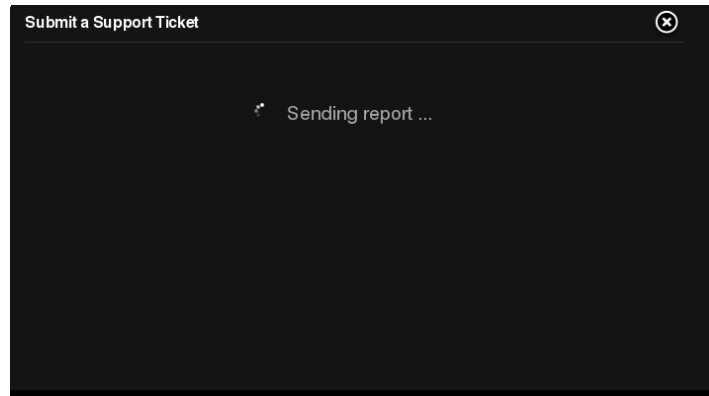
Dealing with AJAX:

The screenshot shows a web form titled "Submit a Support Ticket" with a close button in the top right corner. The form is organized into two columns. The left column contains four fields: "Username:" (optional), "Name:" (required), "E-mail:" (required), and "Please answer the question:" (required). The right column contains a "Problem:" field (required) and an "Attach a screenshot:" field (optional). The "Username:" field is empty. The "Name:" field contains "hctestuser". The "E-mail:" field contains "hctestuser@hubzerro.org". The "Problem:" field contains "test ticket". The "Please answer the question:" field contains "light". The "Attach a screenshot:" field has a "Browse..." button and a list of supported file formats: (.asf, .asx, .avi, .bmp, .csv, .doc, .docx, .eps, .gif, .gz, .html, .jpe, .jpeg, .jpg, .js, .log, .mov, .mp3, .mpeg, .mpg, .pdf, .png, .pps, .ppt, .pptx, .ra, .rm, .rtf, .swf, .tar, .tex, .tif, .tiff, .txt, .wav, .wmv, .xls, .xlsx, .xml, .xsl, .zip). A "Submit" button is located at the bottom right of the form.

One place I've used explicit waits is when dealing with dynamic Javascript from our website's support ticket mechanism.

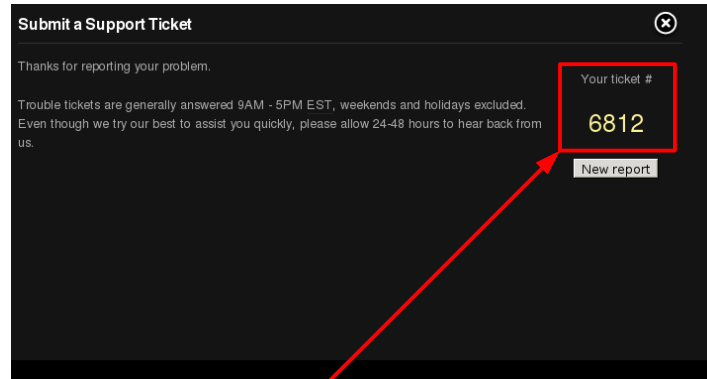
After you fill out a support ticket,

Dealing with AJAX:



You get a spinner like this while the web site stores your support ticket and generates a ticket number for you.

Dealing with AJAX:



```
<div id="trSuccess">
  <div>
    <p>Your ticket #
      <span>
        <a title="View ticket" href="/support/ticket/6812">6812</a>
      </span>
    </p>
  </div>
</div>
```

Then you get a web page like this, where the support ticket number is inside this trSuccess div. Without some kind of waiting, this can be a hard problem to solve.

Dealing with AJAX:

```
...  
  
submit_ticket()  
  
# wait for the page to refresh  
wait = WebDriverWait(browser, 60)  
e = wait.until(lambda browser :  
                browser.find_element_by_id("trSuccess").is_displayed())  
  
ticket_number = e.find_element_by_css_selector('a').text()  
...  
  
<div id="trSuccess">  
  <div>  
    <p>Your ticket #  
      <span>  
        <a title="View ticket" href="/support/ticket/6812">6812</a>  
      </span>  
    </p>  
  </div>  
</div>
```

With explicit waits, I don't need to increase my implicit wait, which affects all failing element lookups. I can keep my implicit wait timeout low and use the explicit wait with a longer timeout here.

For my condition, I write my own lambda function which returns true or false as to whether the trSuccess div is displayed in the web page.

Once it is displayed, I jump out of my explicit wait, and grab the ticket number.

**QUESTIONS?
MOVIES?
MORE?**