Exercise Using Selenium

"Scrape" URLs from a page of search results

Revised 6 Nov 2023 for Python Selenium 4.15.0

Selenium

Browser automation.

Not just testing.

`https://selenium.dev/`

We will use Selenium WebDriver

- programmatically control a web browser

Selenium Example

Goal:

Use duckduckgo.com to find links to Kasertsart U.

Print the top 10 links.

Software Needed

- Python 3.x

- Selenium WebDriver: `pip install selenium`

- Driver for the Web browser you use:

  Firefox driver, called "geckodriver"

  https://github.com/mozilla/geckodriver/releases

  Chrome & Chromium driver *(brittle)*:

  https://sites.google.com/chromium.org/driver/

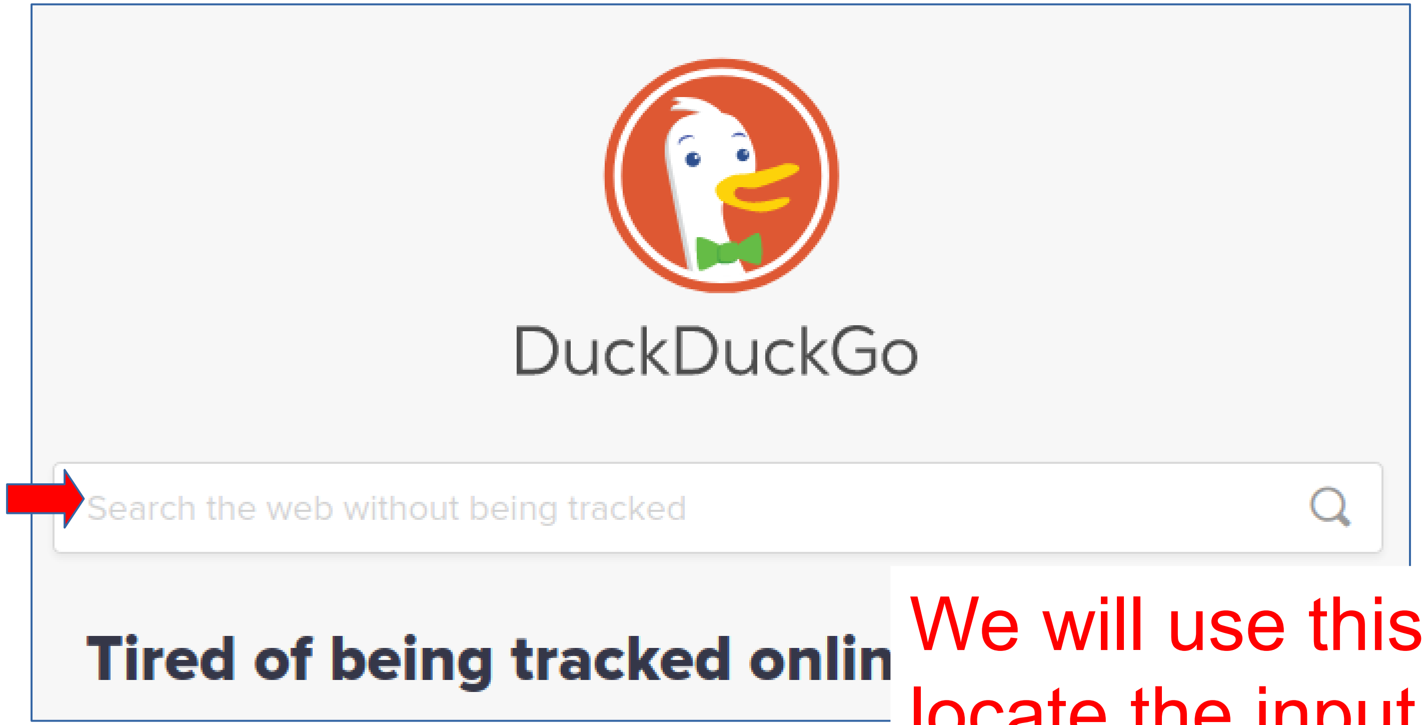
  Safari driver: already in `/usr/bin/safaridriver`

Try it! Get a web page

```
from selenium import webdriver


# get the duckduckgo search page
url = "https://duckduckgo.com"
# browser: a WebDriver object
browser = webdriver.Firefox()
# or: browser = webdriver.Chrome()
browser.get( url )
```

Get the id of the search input box


DuckDuckGo
Search the web without being tracked
Tired of being tracked onlin

We will use this "id" to locate the input field

Firefox: right-click in search box -> "Inspect Element".

<input id='**searchbox_input**' name="**q**" type="text" ...>

Find the input field

Find the page element for the searchbox input field.

```
from selenium.webdriver.common.by import By
from selenium.webdriver.common.keys import Keys
# Find an "element" for the search box
element_id = 'searchbox_input'
element = browser.find_element(By.ID,
                               element_id)
```

Enter some text and press ENTER

`send_keys` simulates typing into a field.

```
# Enter the search text & press ENTER
element.send_keys("Kasetsart Univer")
element.send_keys(Keys.ENTER)
# Run It! (if you are using a script)
# the browser should display results
```

Inspect the Page & Identify Links

We need a way for Selenium to "find" the hyperlinks on the results page.

`browser.find_element`(by=________, "value to find")

`By.CLASS_NAME`	`By.NAME`
`By.ID`	`By.TAG_NAME`
`By.LINK_TEXT`	`By.PARTIAL_LINK_TEXT`
`By.CSS_SELECTOR`	

You can omit the parameter name **`"by="`**

Inspect one search result

Select a search result. Right Click -> Inspect.

The source looks like:

```
<div class="ikg2IXiCD14iVX7AdZo1">
<h2 class="LnpumSThxEWMIsDdAT17 CXMyPcQ6nDv47DKFeywM">
<a href="https://www.ku.ac.th/en/community-home"
rel="noopener" target="_self"
class="eVNpHGjtxRBq_gLOfGDr LQNqh2U1kzYxREs65IJu" data-
testid="result-title-a" data-handled-by-react="true">
<span class="EKtkFWMYpwzMKOYr0GYm
LQVY1Jpkk8nyJ6HBWKAk">News and Activities - Kasetsart
University</span>
</a></h2>
</div>
```

The `class` *names are random. Nothing we can reliably search.*

Find all the "a" tags

```
# Find all "a" elements on page
links = browser.find_elements(
                    By.TAG_NAME, "a")
len(links)
125

# Too many! Find links with text "Kaset.."
match = browser.find_elements(
        By.PARTIAL_LINK_TEXT, "Kasetsart")
len(match)
13
```

Getting Data from a WebElement

`browser.find_element` and **`browser.find_elements`** returns **`WebElements`** that are parts of the page DOM.

You can:

- get "attributes" or text from each WebElement

- search its child WebElements - the DOM is a tree

Print value of the "href=" attribute of the first matches.

```
>>> match[0].get_attribute('href')
'https://duckduckgo.com/?q=Kasetsart
  %20University&t=h_'    (not what we want)
>>> match[2].get_attribute('href')
'https://en.wikipedia.org/wiki/
  Kasetsart_University'  (yes!)
```

"Click" on elements

When you locate a "clickable" web element like a button or hyperlink, you can click to activate it.

```
>>> match[2].click()
```

The browser should open the link you clicked.

Exercise: print first 10 URLs

Print the URLs of the first 10 hyperlinks on the DuckDuckGo search results page.

- omit hyperlinks that refer to duckduckgo.com

- some "a" tags may not have an "href" attribute.

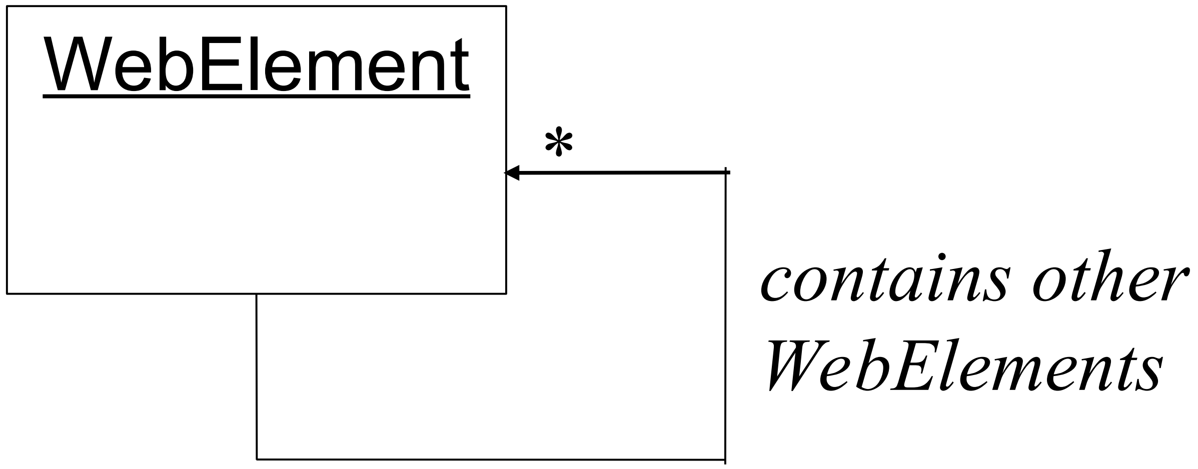
  Use try - except to catch this.

Composite Design Pattern

`WebElement` is the primary object for interacting with a web page using Selenium.

`WebElement` may contain other `WebElements.`

`WebDriver` contains many of the same methods as WebElement



Headless Browsing

You can use a browser without opening the U.I. window.

This is called *headless mode*.

Headless mode is needed when running tests on a C.I. server, like Github Actions.

Headless mode is *faster*, too.

https://developer.mozilla.org/en-US/docs/Mozilla/Firefox/Headless_mode

References

Good Selenium Tutorial in Python

Uses the previous version of selenium, so some methods may no longer work.

https://blog.testproject.io/2019/07/16/web-ui-testing-python-pytest-selenium-webdriver

The same author has other good testing tutorials:

https://blog.testproject.io/2019/07/16/

Exercise

Write a unit test for this:

When I search DuckDuckGo for Kasetsart University,

then at least 1 of the top-10 search results contains a link to `https://www.ku.ac.th/`*(anything)*

1. Use `setUp` to create the browser instance.

2. Write a unit test method go perform the test above.

Wrong: testing for exact match of "https://www.ku.ac.th/", because KU's home page could change.

3. Once your test works, add headless mode to setUp and rerun. My intro to Selenium shows how to.