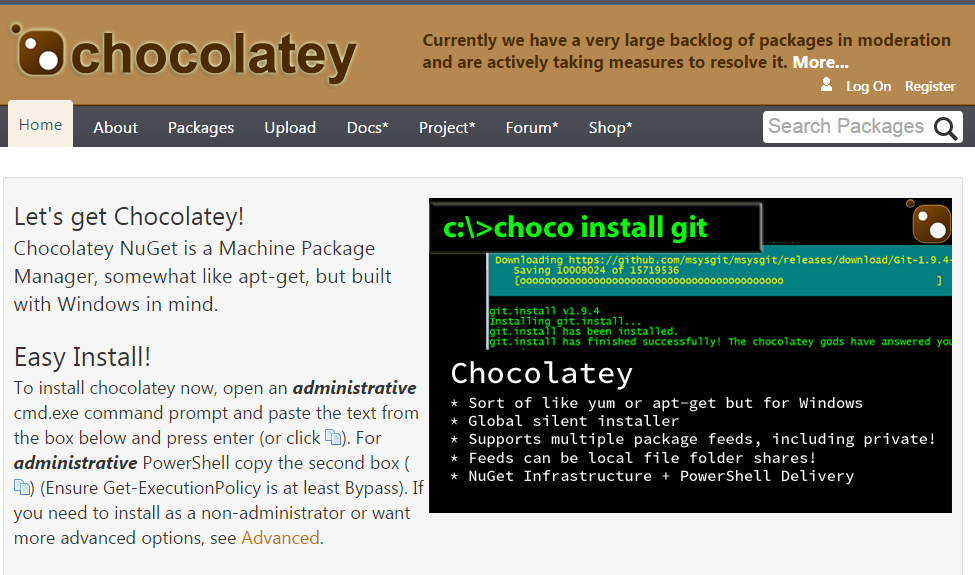
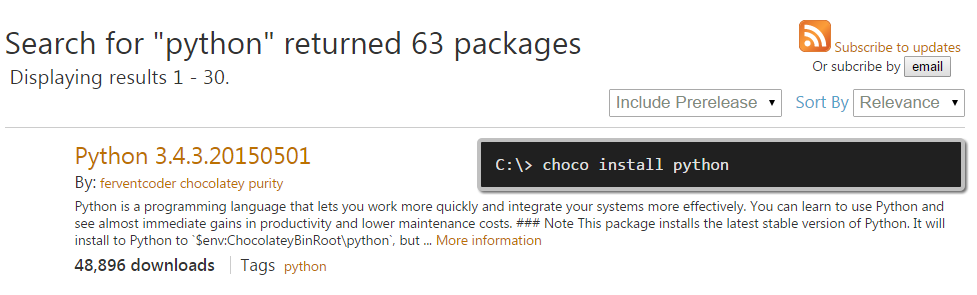
# Grabbing multiple “clickable” pdf files from a single page

Having decided that the Firefox add-on [DownThemAll](https://addons.mozilla.org/en-US/firefox/addon/downthemall/) could not achieve what I wanted, I had wanted to use [import.io](https://import.io) to accomplish this task, but having completed a quick dirty test using their Authenticated Extractor tool, I could not seem to get their tool working. A quick [Google](https://www.google.co.uk/webhp?sourceid=chrome-instant&ion=1&espv=2&es_th=1&ie=UTF-8#safe=off&q=import+io+extractor+from+a+onclick+button+on+a+page) led me to the following solution:

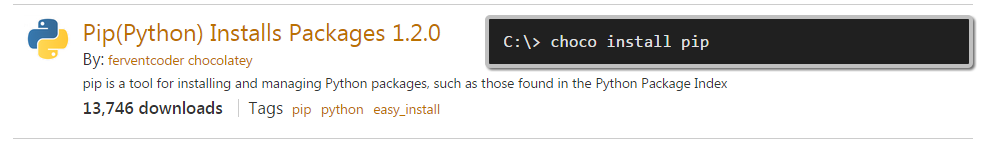
First off, you will need to have installed Mozilla Firefox, Python and PIP for this to work. For me, the easiest way to do this was to use chocolatey.org:

[](https://chocolatey.org)

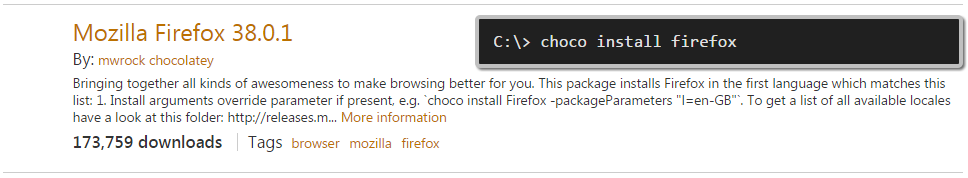
Followed by python:



Then use chocolatey to install PIP (python) Installs Packages:



Having installed Python, the next step is to use it to install Firefox:

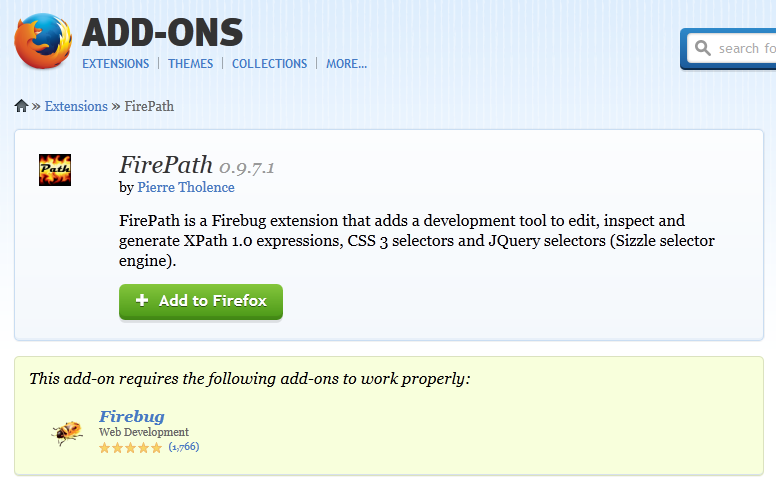


Note: Prior to writing this article I had originally gotten the correct XPath using Google Chrome but herein follow the instructions to perform the same trick using Firefox:

Install the FireBug add-on:



Then the FirePath add-on:



This add-on provides a simple right-click “Copy XPath” function used later on in the guide.

Once these installations are completed, the next stage is to install the tools needed by Python to automate the data grab:

1. Open a command prompt as administrator then type the following:
2. pip install –U selenium

This command installs the Python bindings for [selenium](https://pypi.python.org/pypi/selenium)- a package that automates web browser interaction from Python.

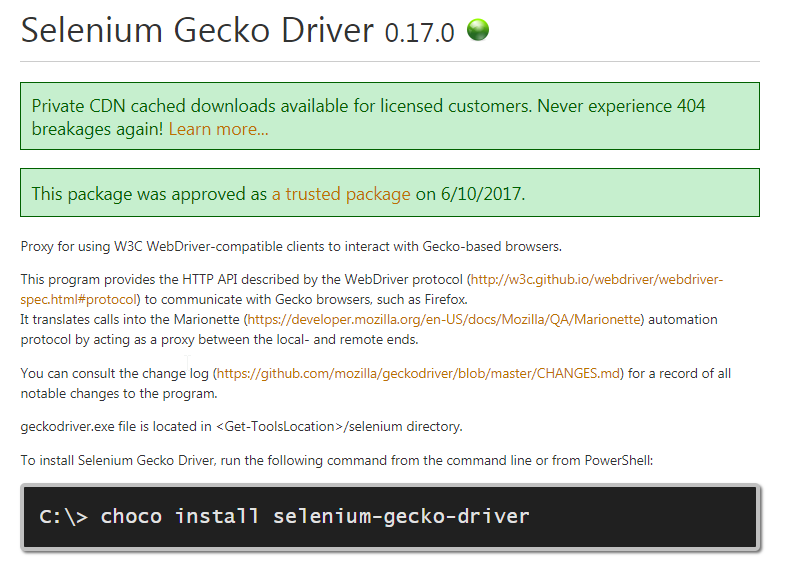
Once selenium has finished installing, you can run the following command from the Python command line:

## What is Selenium?

*Selenium automates browsers*. That is it! What you do with that power is entirely up to you. Primarily, it is for automating web applications for testing purposes, but is certainly not limited to just that. Boring web-based administration tasks can (and should!) also be automated as well.

Selenium has the support of some of the largest browser vendors who have taken (or are taking) steps to make Selenium a native part of their browser. It is also the core technology in countless other browser automation tools, APIs and frameworks.

**UPDATE 2017-06-29: A recent change in Firefox means you also need to perform a couple more steps:**

1. ****
2. add the folder that geckodriver.exe resides in to the PATH variable, on this system:

C:\tools\selenium\

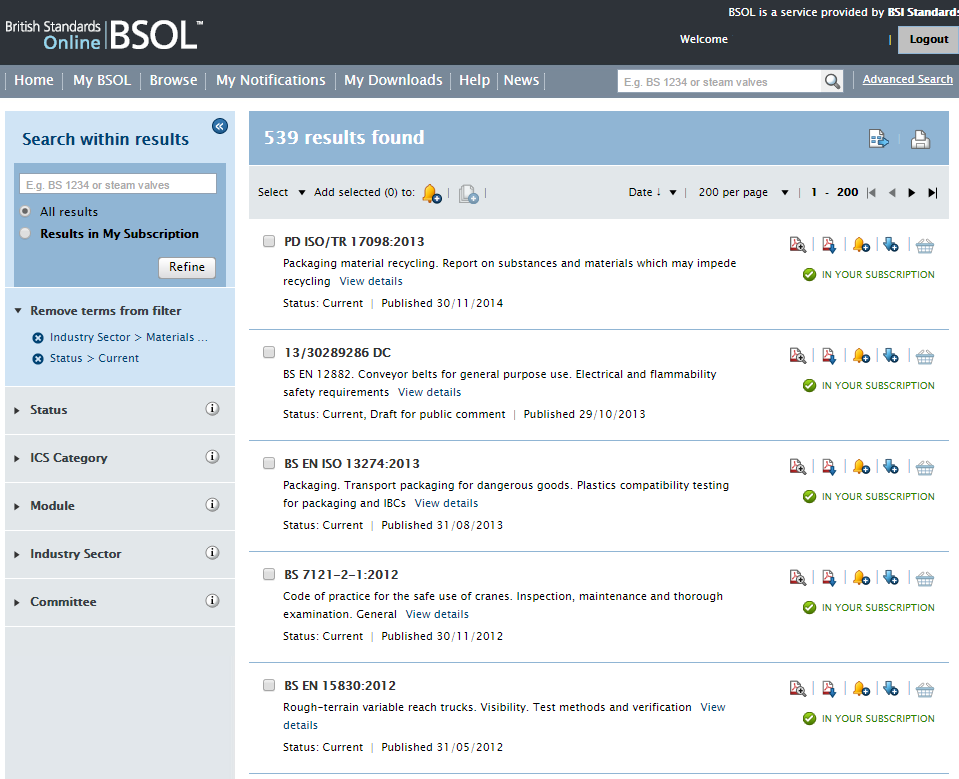
1. if you had cmd prompt running, exit and restart and then the following will work

**from** selenium **import** webdriver

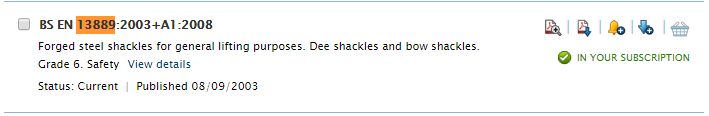
browser **=** webdriver**.**Firefox**()**

browser**.**get**(**'http://yourchosenURL'**)**

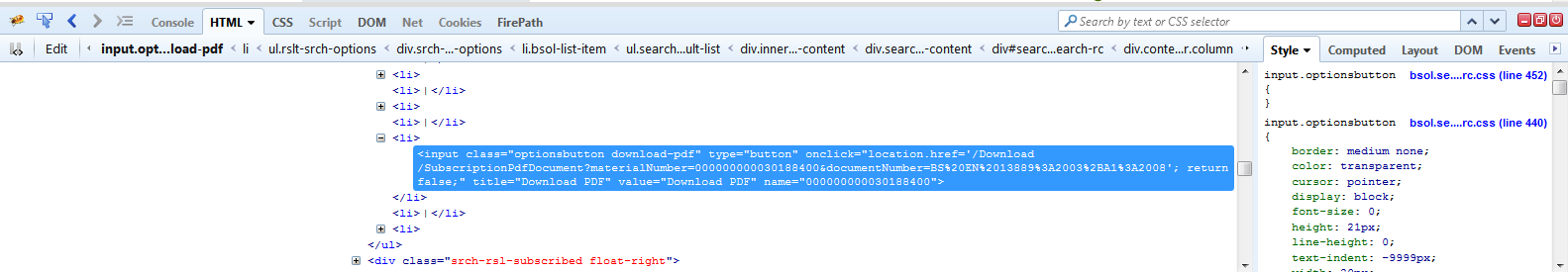
This will load the page you selected into the Firefox browser (you may need to log in):



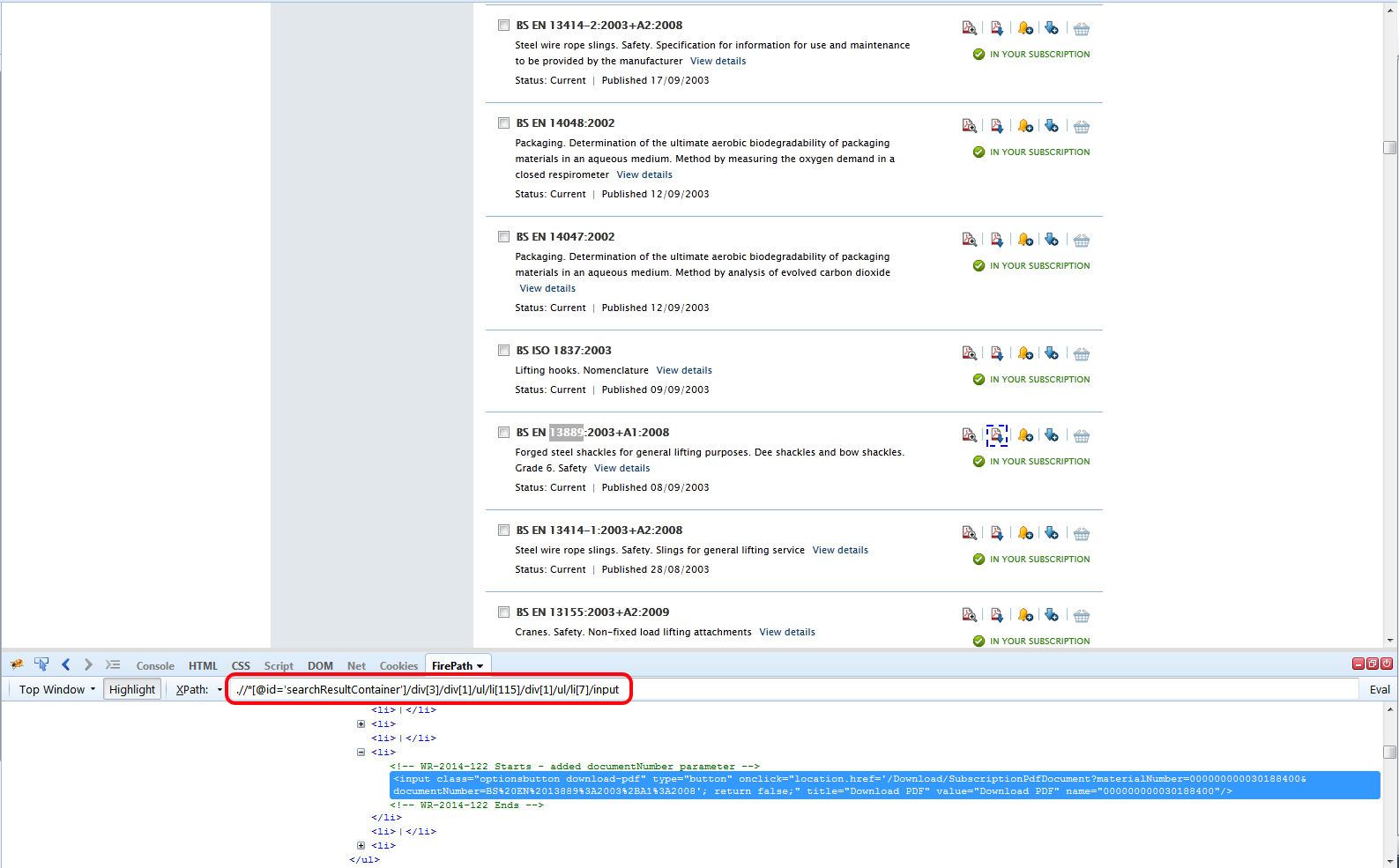
The next step is to navigate to the area where you would otherwise click the button:



Then right-click the item (in this case the download pdf icon) – select “Inspect in FirePath” and the FirePath window appears:



The XPath information appears in the FirePath Console:



For this particular example, the resultant text string looks like this:

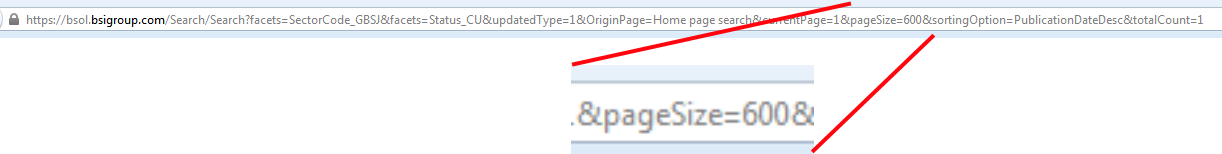
**.//\*[@id='searchResultContainer']/div[3]/div[1]/ul/li[115]/div[1]/ul/li[7]/input**

The key part of this is the [###] that appears about midway through the string (115 in the image above). This is the unique identifier needed by selenium for the next stage of automation:

From the results page shown earlier, we can see that there are 539 entries in this specific category, and the page breaks them down into 200 per page:



On the example site, we can modify this to include every result by changing the pageSize=200 value to 600:



This value gives us the extents of the range we need to loop through so we can now begin to use Python in conjunction with selenium to grab the clickable files:

**Note: during my initial testing I used Google Chrome to grab the XPath information and this is the resultant string: ‘//\*[@id="searchResultContainer"]/div[3]/div[1]/ul/li[115]/div[1]/ul/li[7]/input’**

**The only difference between this and the output string from Firefox is that the Firefox string has single speech marks surrounding the id=’searchResultContainer’ section instead of the double speech marks copied from Chrome.**

**It is important to remember to swap these for “ “ otherwise you will receive “Syntax Error” messages at the Python command prompt.**

Still at the Python command prompt, type the following (the Tab before “xpath=” is important!):

**for** x **in** range**(**1**,**200**):**

xpath**=**' .//\*[@id=”searchResultContainer”]/div[3]/div[1]/ul/li[' **+** str**(**x**)** **+** ']/div[1]/ul/li[7]/input'

browser**.**find\_element\_by\_xpath**(**xpath**).**click**()**

Hitting Enter twice will begin collecting the required files.

At this point it is a good idea to have already set up a default save location with Firefox because once the tool is running there is no way of pausing whilst you do so and for hundreds of links this cause you to miss some of them.