**SCRAPING WITHOUT PROGRAMMING USING IMPORT.IO**

Let me start this tutorial with a caveat: The tool you are about to see is very useful and cool, but limited. If you want to get to the ideal point of being able to process and analyze any potential dataset you encounter on the Web, I would recommend learning some programming. Once you get established in any programming environment – whether it’s Python, Ruby, or even R – learning how to scrape is not terribly difficult and becomes incrementally easier as you get the hang of it.

That being said, import.io can be your friend, as long as the target of interest is in a format within the limits of what this program can do, and I hope you’ll find this tutorial helpful for starters, and I hope it will encourage you to follow a path that will lead you to learn some programming.

Ok, now onto the show…

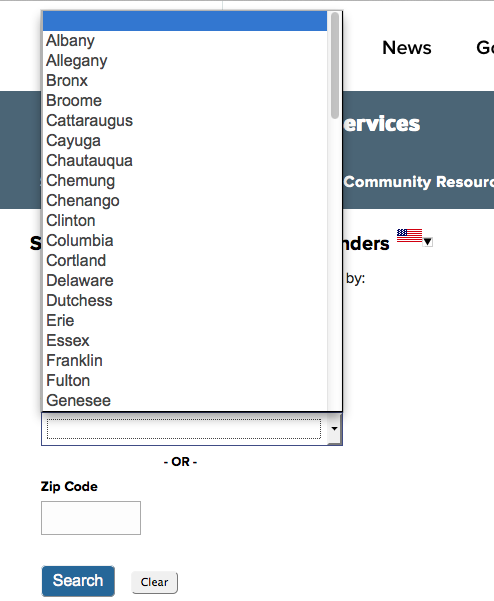
Like many states, New York keeps an online list of registered sex offenders and provides the public with a Web search engines. Often, reporters seek this data to analyze various aspects of sex offender monitoring – are sex offenders living close to schools, are they really living where they say they’re living, etc.

In this tutorial, we will use import.io to scrape sex offender registry and create a data file that you can analyze.

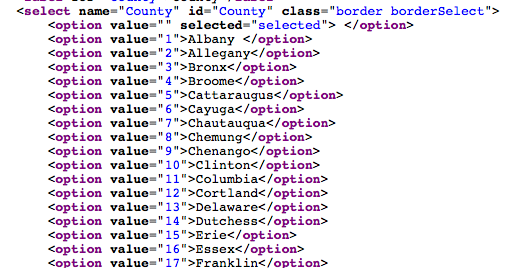
To get started, you will need to sign up for a free account and install the free [import.io software](https://www.import.io/download/). I am working on a MacBook so the images you will see in this tutorial are taken from the Mac version, but the functionality should be the same on a PC.

So before we build a scraper, let’s take a look at our target. The search interface for NY sex offenders is at: <http://www.criminaljustice.ny.gov/SomsSUBDirectory/search_index.jsp> .

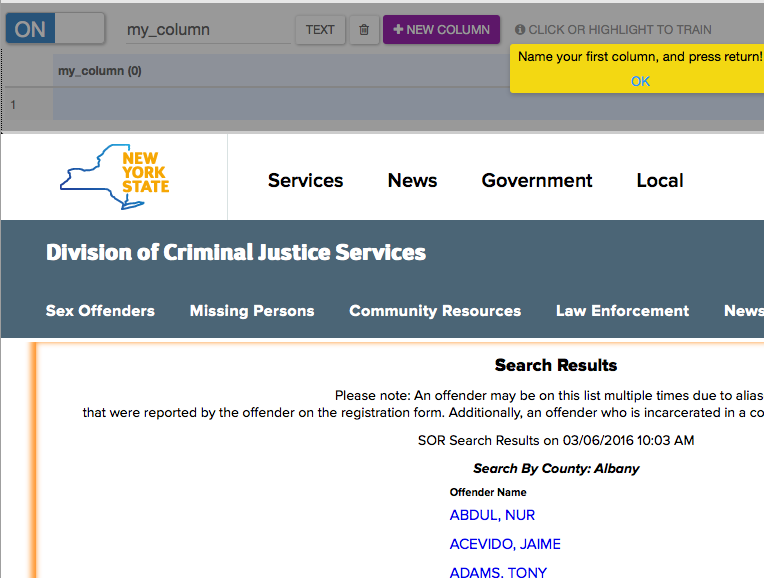
The first thing we need to think about with any scraping project is how we would go about gathering this information manually, and then try to envision the program that would automate that process.



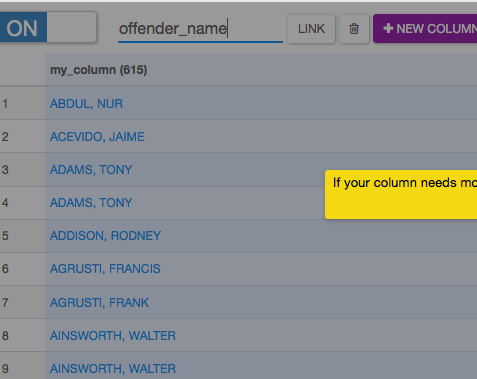
If we wanted to grab info on all offenders, the best way would be to select each county, one at a time. If you view the source of this page, you’ll see that each county has a numerical code, from 1 to 62, with a 64 for “out of country” and 99 representing “unknown”.



Ok, so let’s get to the scraping. Run import.io, and create a new extractor. A special Web browser will open, and we’ll enter our url of the search page. Then within that browser, select the first county, Albany, and click search. This will produce a list of offender names that we want to scrape, so in the import.io browser, click the button that allows you to start the extraction.



Select the first name on the list, and import.io will ask if you want just this one, or the entire list. Tell it you want all rows and it will ask you to give this field a name, so let’s call it offender\_name:

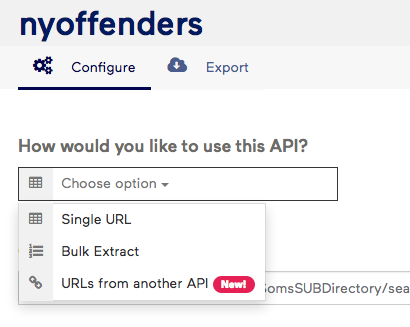


If there were more columns in this preliminary page, we could add more by hitting the new column button. But in our case, our approach is going to be to download all of the names first, and then click through and grab the details of each offender later, so for now click “Done”.

This will take us back to the main page, where all of the offender names will be highlighted in yellow. We can give our “API” a name – I called it nyoffenders, and then I clicked Publish. This then takes you to your import.io control panel, where you now have saved a routine that will scrape the Albany sex offender page whenever you want, and allow you to download the results.

There are still two more major steps – grabbing the other counties, and then getting the details about each offender.

To incorporate all of the counties in our scrape, we need to configure our import.io project to conduct a “bulk extract”, which you can select from your project control panel:

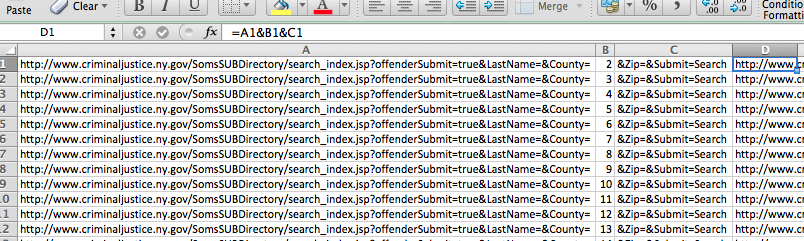


This will prompt impot.io to ask you for a list of URLs that you want to scrape. Which takes us back to our initial observation – each county has a unique numeric ID and if you notice, when you clicked on Albany, it went to a URL that included that ID, as in:

http …. \_index.jsp?offenderSubmit=true&LastName=&County=1&Zip=&Submit=Search

So what we need to do is create a list of URLS and swap in the different county values, 2 through 64 and 99. For this, we can use Excel. Copy and paste the first part of the URL, up until the number, into column A. Copy and paste the last part of the URL, after the number into column C. In column B, enter the number two, hover over the bottom right corner of the cell, and drag down – Excel will autocount – until you get to row 64. Then enter 99 in row 65. Then copy and paste down the URL components in columns A and C. Finally, enter a formula in column D to put it altogether: =A1&B1&C1 – and copy and paste that down to the bottom.

It will look like this:



To test it, copy and paste one of the URLs you generated into a new browser window with a regular Web browser. If you get some county’s sex offender list, you’re good to go. If not, go back over your work and debug.

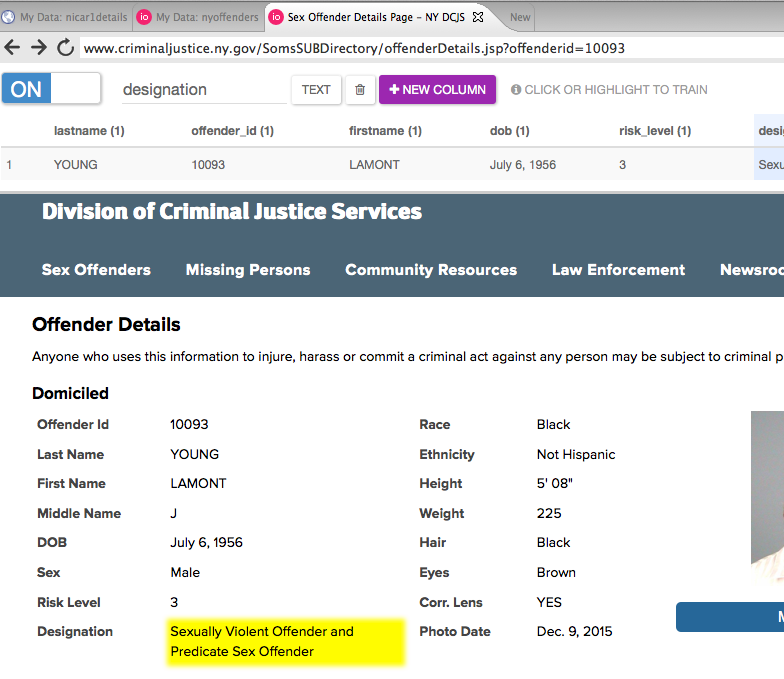
Once you have a good Excel list, you can copy and paste it into import.io and click “run queries”. The program will go out and grab all of the data from the URLs we fed it. If you get a “failed query”, click on it and tell it to retry. This usually happens because the program moves faster than the Web site can supply the information.

From here, you can then download the list of names as a CSV file – there should be several thousand. But we’re not done yet – we need to get some details about each offender. So instead, let’s save these results as an import.io dataset that we can connect to another operation we’ll create to grab the details. IMPORTANT: If you do not remember to save this as an import.io dataset, you will be able to re-run the bulk extraction, but you will not be able to link the names with the detail extractor.

The way import.io thinks of this is as a “chain” of extractors. There is one extractor for generating the list of offenders. And a second extractor, which we need to build, for harvesting details about each person which we can “chain” to the first one.

So let’s build the detailed extractor. From the control panel, select “new” and “extractor” and navigate to the main sex offender page, select Albany, and then pick the first offender on the list, which we’ll use as a template for scraping all of the offenders.

Turn “ON” the extractor and start grabbing columns. Click NEW COLUMN, then highlight the data, give the column a name and click enter. Repeat for any column you want. IMPORTANT: 1) Make sure in this case you select “single row” because we’re creating a template based on one person and 2) Make sure you hit NEW COLUMN before selecting data, otherwise, import.io will put the data into the previous column you created.

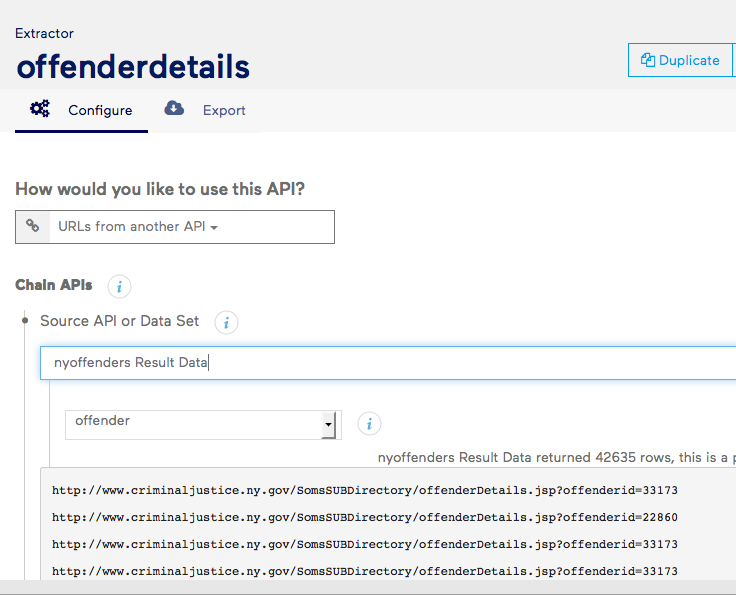


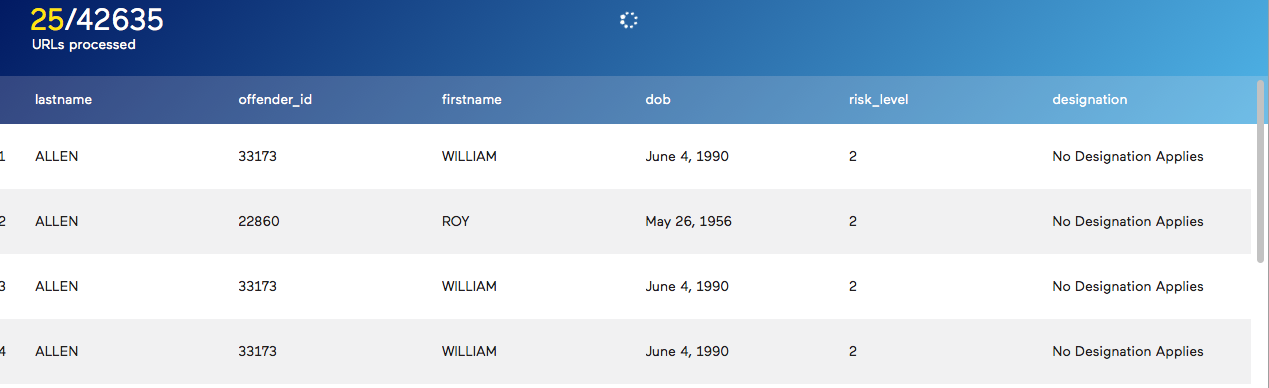
Once you have all of the fields you want, click “DONE”, and PUBLISH and give this template a name – I called mine offenderdetails.

Now the moment of truth – we need to tell this new template where to get the list of offenders to scrape. It needs to get it from our first extractor.

So in the control panel for offender details, let’s enable the “URLS from Another API” feature and link offender details to our import.io dataset.

And now when you hit “run queries”, you get … details about ever offender from all counties:





Pretty cool, no? Ok, so in closing, let me get back to reality a bit. The reason this worked so well is because we were dealing with a Web site that had a logical structure (County=> Offender => Offender Detail) and a URL driven architecture – each county had a unique URL, each offender had one too.

In the real world, not every Web site is built this way – the underlying URLs are hidden from view, or the pagination uses a system that is more complicated than just changing a county ID or a page number.

This is where programming comes in, but I hope by working with import.io, you have some scraping successes and get a better sense of what you need to learn to do more.