Scraping with Chrome Scraper

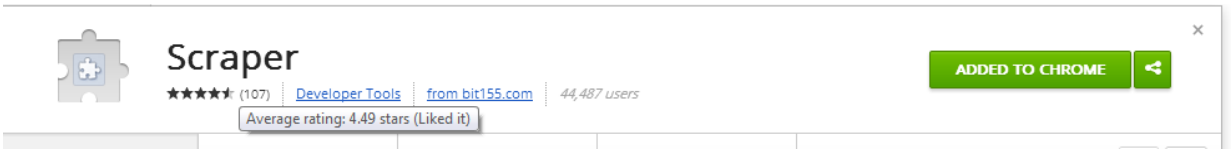
Sarah Cohen / The New York Times/ @sarahcnyt

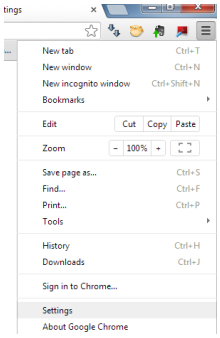
November 2013

Scraping may be the most important skill you'll learn this year. It's essential for collecting all kinds of content from the Internet, from the results of search forms to press releases. Chrome Scraper lets you convert a page you find on the Web into a spreadsheet or database. You'll eventually need to do more, but what you learn using this scraper will help you when you want to go to the next step.

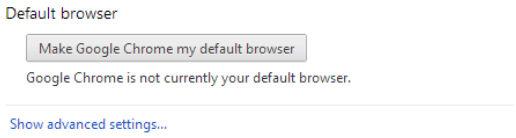
Getting started with Chrome Scraper

Make sure to have a recent version of Chrome on your computer, and add in the Scraper extension. The extension doesn't appear when you search the Chrome extension page. Get it here, or do a general Google search for Chrome Scraper. When you go to the installation page, it will look like this if it's been installed.



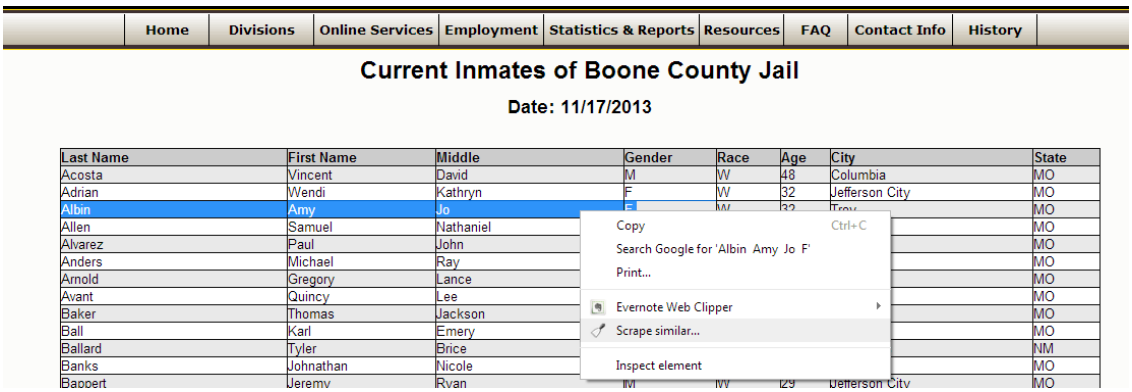


For this tutorial, you may want to make Chrome your default browser. Open a Chrome window and click on the upper right corner to get the menu and choose "Settings". The option is at the bottom of the page. Make it your default if it isn't already.



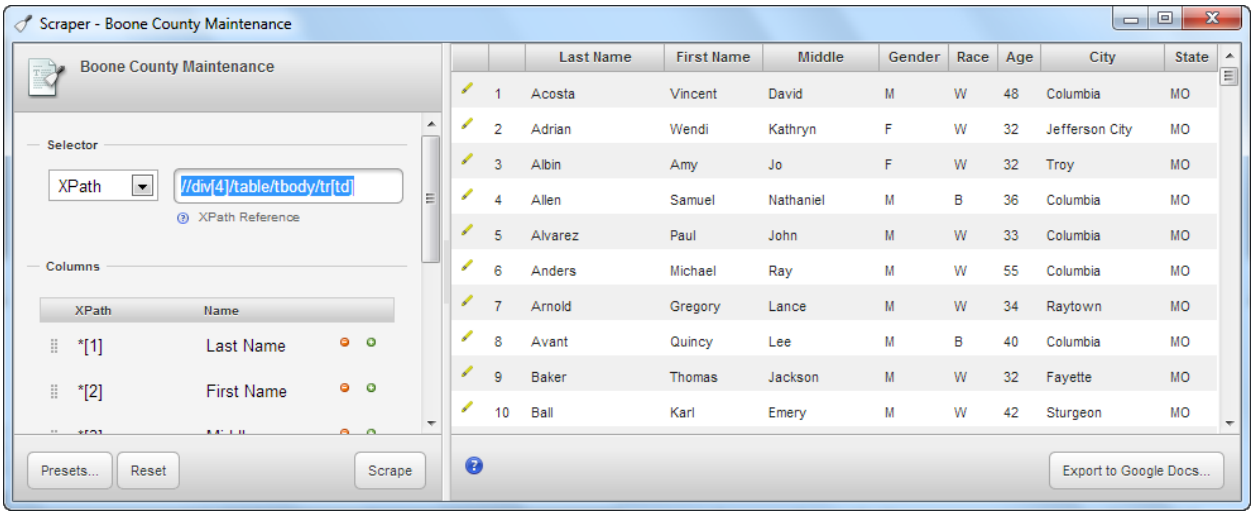
Scrape your first page

The Boone County, Mo., sheriff's office lists everyone who is in the jail right now. Select a few cells of one row in the table, right-click and choose "Scrape similar".



Last Name	First Name	Middle	Gender	Race	Age	City	State
Acosta	Vincent	David	M	W	48	Columbia	MO
Adrian	Wendi	Kathryn	F	W	32	Jefferson City	MO
Albin	Amy	Jo	F	W	32	Troy	MO
Allen	Samuel	Nathaniel	M	B	36	Columbia	MO
Alvarez	Paul	John	M	W	33	Columbia	MO
Anders	Michael	Ray	M	W	55	Columbia	MO
Arnold	Gregory	Lance	M	W	34	Raytown	MO
Avant	Quincy	Lee	M	B	40	Columbia	MO
Baker	Thomas	Jackson	M	W	32	Fayette	MO
Ball	Karl	Emery	M	W	42	Sturgeon	MO
Ballard	Tyler	Brice	M	W	19	Jefferson City	NM
Banks	Johnathan	Nicole	F	W	29	Jefferson City	MO
Bappert	Jeremy	Ryan	M	W	29	Jefferson City	MO

Like magic, you should be taken to a new window that looks like this:



	Last Name	First Name	Middle	Gender	Race	Age	City	State
1	Acosta	Vincent	David	M	W	48	Columbia	MO
2	Adrian	Wendi	Kathryn	F	W	32	Jefferson City	MO
3	Albin	Amy	Jo	F	W	32	Troy	MO
4	Allen	Samuel	Nathaniel	M	B	36	Columbia	MO
5	Alvarez	Paul	John	M	W	33	Columbia	MO
6	Anders	Michael	Ray	M	W	55	Columbia	MO
7	Arnold	Gregory	Lance	M	W	34	Raytown	MO
8	Avant	Quincy	Lee	M	B	40	Columbia	MO
9	Baker	Thomas	Jackson	M	W	32	Fayette	MO
10	Ball	Karl	Emery	M	W	42	Sturgeon	MO

The program has recognized that you selected part of a table, and that the headings should be the titles of each column. At this point, you can export to a Google Doc, or you can use CTL-A (select all) to copy and paste the table into Excel. It will have one extra column on the left that you can delete.

Before moving on, take a minute to understand what's happened. Zoom in on the left panel of the scraper.

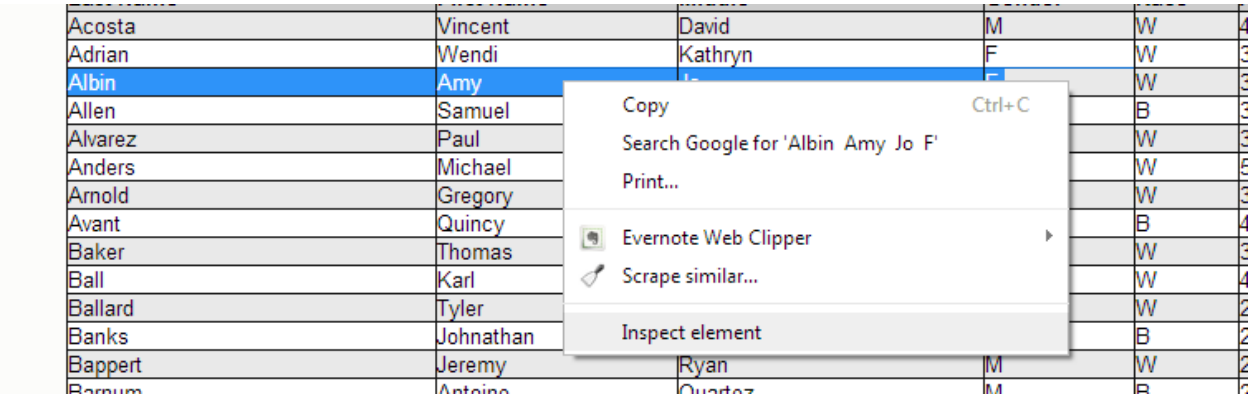
The scraper is using a language called XPath to try to guess what you wanted to scrape. (Every once in a while, it will go awry and say JQuery instead, and it won't work. Close out Chrome and try again from scratch.)

You need to know a little HTML to understand what's happening, but here's a basic translation:

Take the 4th " " tag in the page, then look for a <table> tag below it, then look for a "<tbody>" tag below that. Then take the rows <tr> and each cell in that row [td]. The *[1] means "first column", *[2] means "second column" and so on.

To verify that, let's look at the web page's structure. Go back to your Chrome window, and select the same cells if they aren't still selected. (They'll look different than this because the jail population is constantly changing.)

Right-click again, and choose "Inspect element".



You'll see a lot of HTML code, like this, which is all of the code needed to present the page to you with the layout, formatting, headings and sidebars.

```
▶ <div class="pageHeader">…</div>
▶ <div class="pageMenu">…</div>
▶ <div class="pageHeaderPrint">…</div>
  <br clear="all">
▼ <div class="canvas" id="canvas">
    <h2>Current Inmates of Boone County Jail</h2>
    <h3>Date: 11/17/2013</h3>
    <br>
  ▼ <table class="resultsTable" style="margin: 0 auto; width: 90%; font-size: small;">
    ▼ <tbody>
      ▶ <tr>…</tr>
      ▶ <tr class="A">…</tr>
      ▶ <tr class="B">…</tr>
      ▼ <tr class="A">
          <td valign="top" headers="LN">Albin </td>
          <td valign="top" headers="FN">Amy </td>
          <td valign="top" headers="MN">Jo </td>
          <td valign="top" headers="gender">F </td>
          <td valign="top" headers="race">W </td>
          <td valign="top" headers="age">32 </td>
```

If you study it, you'll see that this is the 4th , <table> <tbody> that you want. (Because the cells in the first row are tagged as "th", or table headings, rather than "td", or table cells, they are used as the names of the columns).

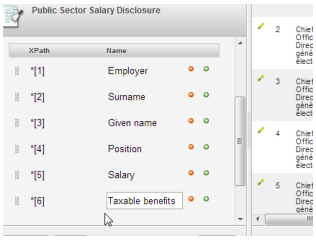
Understanding this will help when you try to learn more complex scraping. This structure, called the Document Object Model (or DOM), is like an upside-down tree. You can climb down and up, jump to other branches and find little tiny leaves way out at the end. Letting Chrome help you figure out where in the tree you want to start scraping will be helpful even if you use another programming language to actually do the work.

Another table, a little harder

Let's try again with the Ontario legislature's salary page.

Employer / Employeur	Surname / Nom de famille	Given Name / Prénom	Position / Poste	Salary Paid / Traitement	Taxable Benefits / Avantages imposables
Chief Electoral Officer / Directeur général des élections	BATTY	JONATHAN	Director, Election Finances & General Counsel / Directeur, Financement des élections et avocat général	$160,402.44	$229.68
Chief Electoral Officer / Directeur général des élections	ESSENSA	GREG	Chief Electoral Officer / Directeur général des élections	$188,303.44	$269.40
Chief Electoral Officer / Directeur général des élections	FLACH	LALITHA	Director, Operations / Directrice des opérations	$121,682.08	$174.12

This time Chrome wasn't so smart. It doesn't know that the top row is the titles and it ignores them. You can just type in the boxes in the left panel.

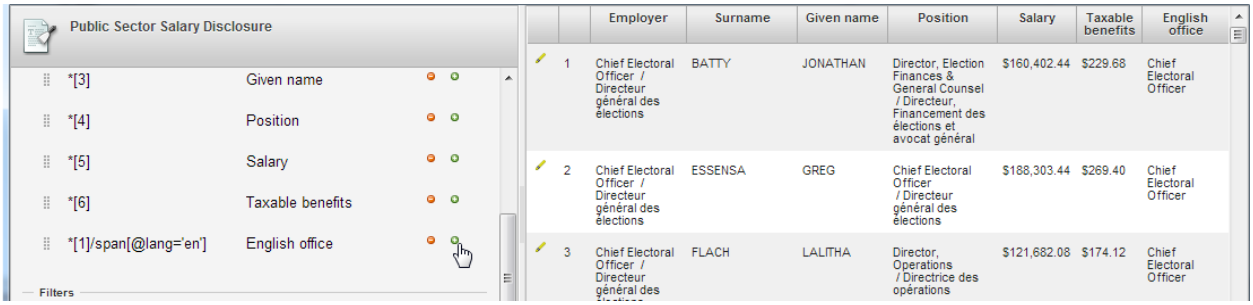


But wouldn't it be nice if we could split the English and French versions of the employer and position?

Go back to the web page and inspect one of those elements, and you should see something like this. (You might have to expand the little arrow to see it all.)

```
▼ <tbody>
  ▼ <tr>
    ▶ <td colspan="2" align="left" valign="top">
        <span lang="en">Chief Electoral Officer </span>
        <span lang="fr_ca"> / 
                          Directeur général des élections</span>
      </td>
      <td align="left" valign="top">BATTY</td>
      <td colspan="2" align="left" valign="top">JONATHAN</td>
```

We can use the structure of the page, which is hidden to us in the browser, to pick just the English version. In both the office and the position, there are two s. We want the one called English.

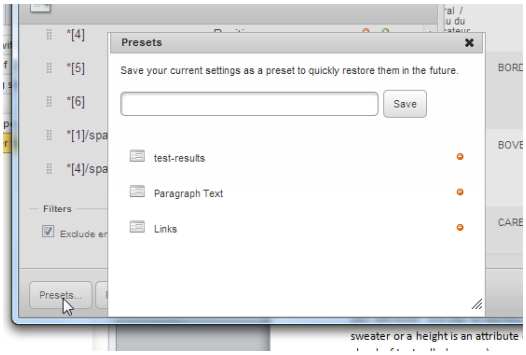


Add a row to the left panel by clicking on the little green plus sign on the last field.

Type in the code *[1]/span[@lang=”en”] . That tells Chrome to go to the first column (just like the first column that has both languages), but only take the text that is within the span with the attribute lang=”en”. In XPath, an attribute is preceded by an @ symbol.

(An “attribute” is a way to identify something on a web page, the same way a color is an attribute of a sweater or height is an attribute of a person. In this case, lang (short for language) is an attribute of a chunk of text called a span. The value is that we want is “en” -- the equivalent of saying that you want a “red” sweater.)

Try picking out the English portion of the position on your own.

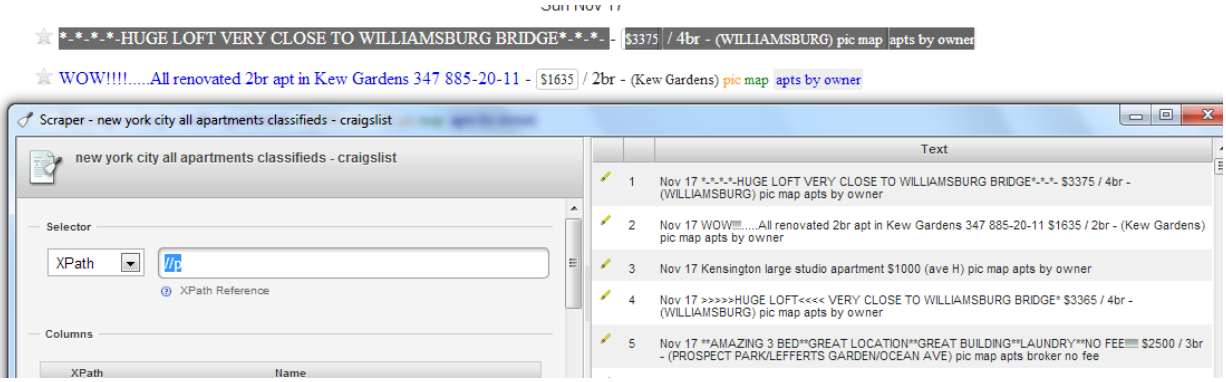


Once you have what you want, you might want to save your work so you can repeat it once the data is updated. Click on “Presets” and give your scraper a name. If you’re signed into Chrome, it will be saved as part of your profile. If not, it will only work on the machine you’re on right now.

Hidden structure

You don’t need a table to scrape using the structure of the page, but you’ll have to do more of the work yourself when it isn’t obvious. Let’s try a page of Craigslist results, this one for apartments in New York City.

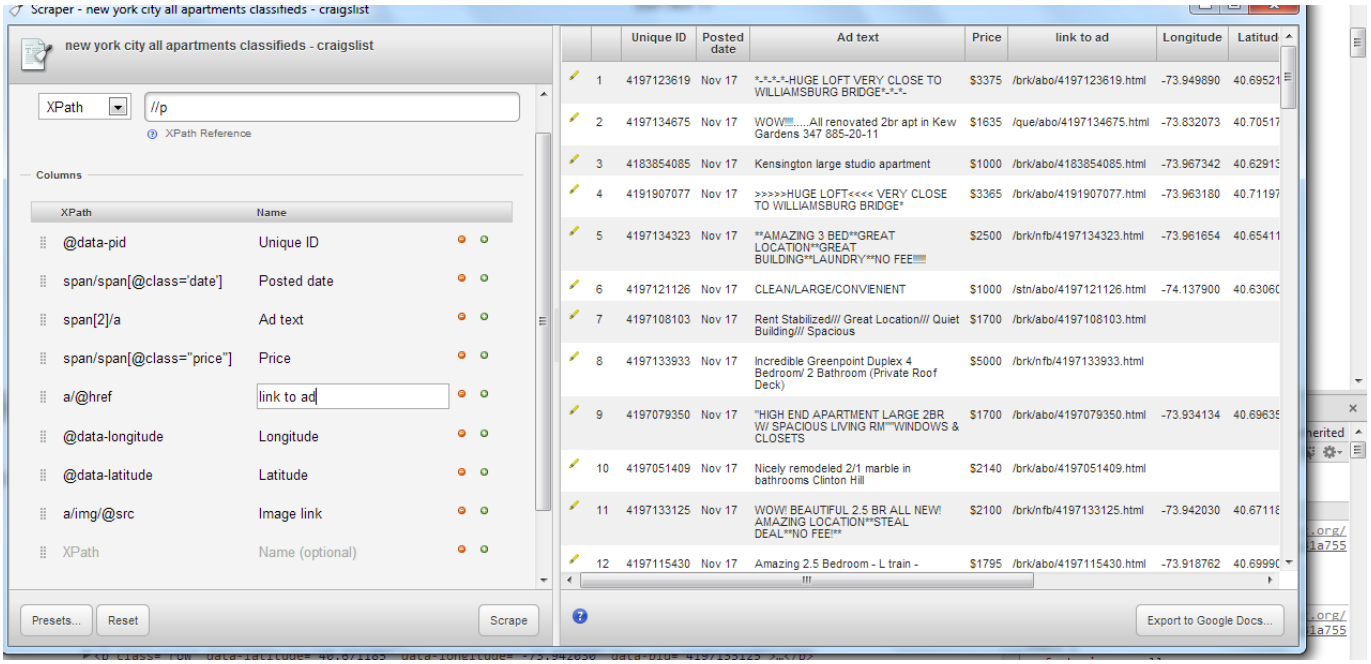
When you choose the first result and scrape, all it gives you is the list of results, separated by the <p> tag.



That's fine, but what we want to separate out the pieces of the ad, like the price and the link to a picture? Go back to the web browser and inspect the element (I've expanded all the little arrows for this row):

```
<h4 class="ban">…</h4>
<p class="row" data-latitude="40.695217" data-longitude="-73.949890" data-pid="4197123619">
  <a href="/brk/abo/4197123619.html" class="i" data-id="0:00F0F_hv2vKVIPQ50">
    <img alt src="http://images.craigslist.org/00F0F_hv2vKVIPQ50_50x50c.jpg">
  </a>
  <span class="star v" title="save this post in your favorites list"></span>
  <span class="pl">
    <span class="date">Nov 17</span>
    <a href="/brk/abo/4197123619.html">*-*-*-*-HUGE LOFT VERY CLOSE TO WILLIAMSBURG BRIDGE*-*-*-</a>
  </span>
  <span class="12">
    <span class="price">$3375</span>
    " / 4br - "
    <span class="pnr">…</span>
    <a class="gc" href="/abo/" data-cat="abo">apts by owner</a>
  </span>
</p>
<p class="row" data-latitude="40.705172" data-longitude="-73.832073" data-pid="4197134675">…</p>
<p class="row" data-latitude="40.629132" data-longitude="-73.967342" data-pid="4183854085">…</p>
<p class="row" data-latitude="40.711972" data-longitude="-73.963180" data-pid="4191907077">…</p>
<p class="row" data-latitude="40.654112" data-longitude="-73.961654" data-pid="4197134323">…</p>
<p class="row" data-latitude="40.630600" data-longitude="-74.137900" data-pid="4197121126">…</p>
```

It turns out there's a lot of information in there that we didn't see on the page – a latitude and longitude, a unique ID, a link to the picture and a link to the original item. Here's an example of some of the items you can pull out of that structure:



Each of these is “relative” to the paragraph it belongs to – that’s why none of them begin with “//” or “/”. For items that are attributes of the paragraph itself, we just need to specify their names, preceded by an @ symbol. For others, we’ll have to give it a little more information about where to find them.

Two examples are the ad text and the price. To find the ad text, we have to look below the paragraph to the second span section to get the text of the “a” tag below that. To get the price, we have to go two spans down, but only where the class=”price”.

Try pulling out more information from this page on your own. If you run into trouble, just Google “XPath” and whatever tag you’re trying to get. You’ll usually find it pretty quickly.

(To remove a row, press the little red minus sign. Use the little shaded area to the left of the XPath to move the column to another position.)

Where to go from here

The obvious next question is, “how do I automatically cycle through all of the pages of results?” Chrome’s scraper doesn’t actually interact with the Web, so it won’t do that for you.

Outwit Hub ($60) and Helium Scraper ($199) will automate some of the simpler problems. But they have a pretty steep learning curve and language all their own. In the long run, it might more efficient to begin learning how to program using Ruby, Python or another language. Paul Bradshaw’s “Scraping for Journalists” book has a good first tutorial, using ScraperWiki as a way to avoid the headaches of installing software.

Just remember that all of the tools out there use some version of what you learned here as the basis for getting structured data from a web page.