



CodeActually

Dr. Cindy Royal

Texas State University

School of Journalism and Mass Communication

Programming a Scraper Using Python

The previous scraping techniques we have covered are very powerful and may be all that you need. But let's take a look at how you might program your own scraper. There are other features that may allow you to have more control of the data you scrape. You can use any Web programming language to program a scraper, like Python, Ruby or PHP. We are going to look at an example using Python. You can often find a scraper and modify it for your own purposes.

This exercise uses the ScraperWiki platform (ScraperWiki.com). ScraperWiki provides a simulated development environment and scraping-related features. It has some new tools to allow you to extract PDFs. It also allows you to program your own scraper.

For this exercise, you will code in the browser. You don't have to have any programming skill to do this exercise. It is helpful to have knowledge of html and css concepts. These are basic concepts that all students who have taken the first few modules of a Web Design class should know.

- table – html code for introducing a table on an html page
- tr – html code for table row
- td – html code for table data
- div – a section of a Web page. It can be identified further with classes or ids.
- Basic understanding of how html is styled with attributes and inline and external css.

Note: Scraperwiki no longer allows automatic signup. But as students, I have arranged for you to have a free account.

Testing Existing Code in ScraperWiki.

1. Go to Scraperwiki and login. We will use one scraper for most of the testing. Click on New DataSet to start a scraper, then Code in your Browser. Choose Python for the language. Use the settings icon to Rename it (anything you want, something like "test"). We will be using the following tutorial to test existing code that we will later modify for use on other data sets: <https://scraperwiki.com/help/code-in-your-browser/>

2. Follow the instructions for creating a new scraper and copying and pasting the code. Replace the code that is originally provided with the code from the tutorial.

The site we are scraping is on the UPS Blog. This pulls the title, author, url and comments for each blog entry from this page. (Note: I had to remove the "category" line from the scraper to get it work. Categories are no longer represented on this page).

When all segments of the code are copied properly, it should look like this:

```
#!/usr/bin/env python

import scraperwiki
import requests
import lxml.html

html = requests.get("http://blog.ups.com").content
dom = lxml.html.fromstring(html)

for entry in dom.cssselect('.theentry'):
    post = {
        'title': entry.cssselect('.entry-
title')[0].text_content(),
        'author': entry.cssselect('.the-meta
a')[0].text_content(),
        'url': entry.cssselect('a')[0].get('href'),
        'comments': int( entry.cssselect('.comment-
number')[0].text_content() ),
    }
    print post
```

You will see the items if they are scraped properly in the output at the bottom when you Run the scraper.

Add this line to the bottom of the scraper to save it to the datastore:

```
scraperwiki.sql.save(['url'], post)
```

The screenshot shows the ScraperWiki web interface. At the top, there's a header with the ScraperWiki logo, the user name 'CINDY ROYAL', and links for 'DOCS' and 'HELP'. Below the header is a navigation bar with buttons: 'Test by Cindy Royal', 'Code in your browser', 'View in a table', 'Download as spreadsheet', and 'More tools...'. The main area contains a Python script for scraping blog entries from ups.com. The script uses the requests library to fetch the page content and lxml.html to parse it. It iterates over 'theentry' elements and extracts title, author, url, and comment count. Below the script, there are buttons for 'Schedule', 'Run!', 'Clear', 'Help', and 'Report Bug'. The output of the script is displayed as a JSON array of dictionaries, each representing a blog entry with fields like 'url', 'author', 'comments', and 'title'. The output is truncated with '...' in the middle. At the bottom, it shows 'Finished run: 2014-02-20 20:55:16+00:00 Exit code: 0'.

```

1 #!/usr/bin/env python
2
3 import scraperwiki
4 import requests
5 import lxml.html
6
7 html = requests.get("http://blog.ups.com").content
8 dom = lxml.html.fromstring(html)
9
10 for entry in dom.cssselect('.theentry'):
11     post = {
12         'title': entry.cssselect('.entry-title')[0].text_content(),
13         'author': entry.cssselect('.the-meta a')[0].text_content(),
14         'url': entry.cssselect('a')[0].get('href'),
15         'comments': int(entry.cssselect('.comment-number')[0].text_content() ),
16     }
17     print post
18
Schedule Run! Clear Help Report Bug
[{"url": "http://blog.ups.com/2013/11/30/from-small-business-owners-why-its-important-to-shop-small/", "author": "Tracy Spahr", "comments": 0, "title": u'
{"url": "http://blog.ups.com/2013/11/26/my-business-runs-thanks-to-the-support-of-other-small-businesses/", "author": "Tracy Spahr", "comments": 0, "titl
{"url": "http://blog.ups.com/2013/11/21/what-you-can-learn-from-the-hi-tech-industry-and-its-post-sales-success/", "author": "David O'Leary", "comments":
{"url": "http://blog.ups.com/2013/11/21/living-the-dream-with-living-dreams-clothing/", "author": "Tracy Spahr", "comments": 0, "title": u'Living the Dre
{"url": "http://blog.ups.com/2013/11/14/preventing-flu-when-we-flew/", "author": "Susan Li", "comments": 0, "title": u'Preventing \u2018flu when we\
{"url": "http://blog.ups.com/2013/11/14/small-business-saturday-fly-connected-profile/", "author": "Tracy Spahr", "comments": 0, "title": u'Connecting wi
{"url": "http://blog.ups.com/2013/11/11/new-sba-partnership-helps-veterans-access-small-business-opportunities/", "author": "Wade Franklin", "comments":
Finished run: 2014-02-20 20:55:16+00:00 Exit code: 0

```

You can View in a Table and Export out to a spreadsheet using the buttons in Scraperwiki.

Take a look at the code. There are several elements.

- Identify Python as the language with a comment at the top of the page.
- Import the scraperwiki library. That allows the program to open the url. ScraperWiki provides various libraries that help with the scraping process including the requests and lxml.html libraries.
- Import the lxml library that does the scraping. This is a Python library for processing html. All you have to do is run the import statement and it is available.
- Establish the root of your document at the html tag.
- for loop: this loop goes through all the entries that you identify in the dom.cssselect area. You have to look at your code to find the marker elements that will allow the scraper to find the desired data. In this case, there is an element with a "theentry" class, and we want all items below it.
- Each of the other entries has it's own cssselect class to identify. Use the index [0] to pull the first element with that class.
- Then the "post" variable (object) is set up to extract the elements. Remember, arrays start with 0. We get the text.content of the cell. The last one gets an integer, if you need it to do that. In this case, the number of comments is a number, so we will leave it. Later, you may

have to remove the int function and associated parentheses, depending on the type of data you wish to scrape.

- The final line prints what is in the data variable (object).

You should see the data now in the console when you Run, if you created the program properly.

A few things about Python:

- The program is space sensitive, so tabbing matters. When you paste the save code under "print data", make sure it is indented in an identical manner.
- If you want to comment any lines of the code, precede each line with a # sign. Commenting allows you to add helpful instructions or descriptions to code that does not affect its ability to run. And, it can allow you to remove certain lines of code for testing.
- If you get any error messages in the console, read them. They will help you troubleshoot any problems.

The Datastore continues to add data, so you may need to Clear Data, if you have run the scraper multiple times before getting it to work properly.

Let's try one more

Now that you have executed the tutorial example, let's try one that pulls data from a table.

Go to this page with newspaper circulation:

<http://www.infoplease.com/ipea/A0004420.html>

The code below goes to this page, identifies the table and rows in which the data exist and finds table rows that are below a div with a class="center". It also uses the if statement `len(tds)==3` to make sure we get the correct table on the page, the one with three columns.

```
#!/usr/bin/env python
import scraperwiki
import requests
import lxml.html

html =
requests.get("http://www.infoplease.com/ipea/A0004420.html"
).content
dom = lxml.html.fromstring(html)

for tr in dom.cssselect("div[class='center'] tr"):
    tds = tr.cssselect("td")
    if len(tds)==3:
        data = {

            'Rank': tds[0].text_content(),
            'Newspaper': tds[1].text_content(),
            'Circulation': tds[2].text_content()

        }
    print data
```

Remember to add the Save line to save to the datastore.

```
scraperwiki.sql.save(['Rank'], data)
```

'Rank' reflects the first item and data is the variable in this scraper where the data is stored.

Exercise

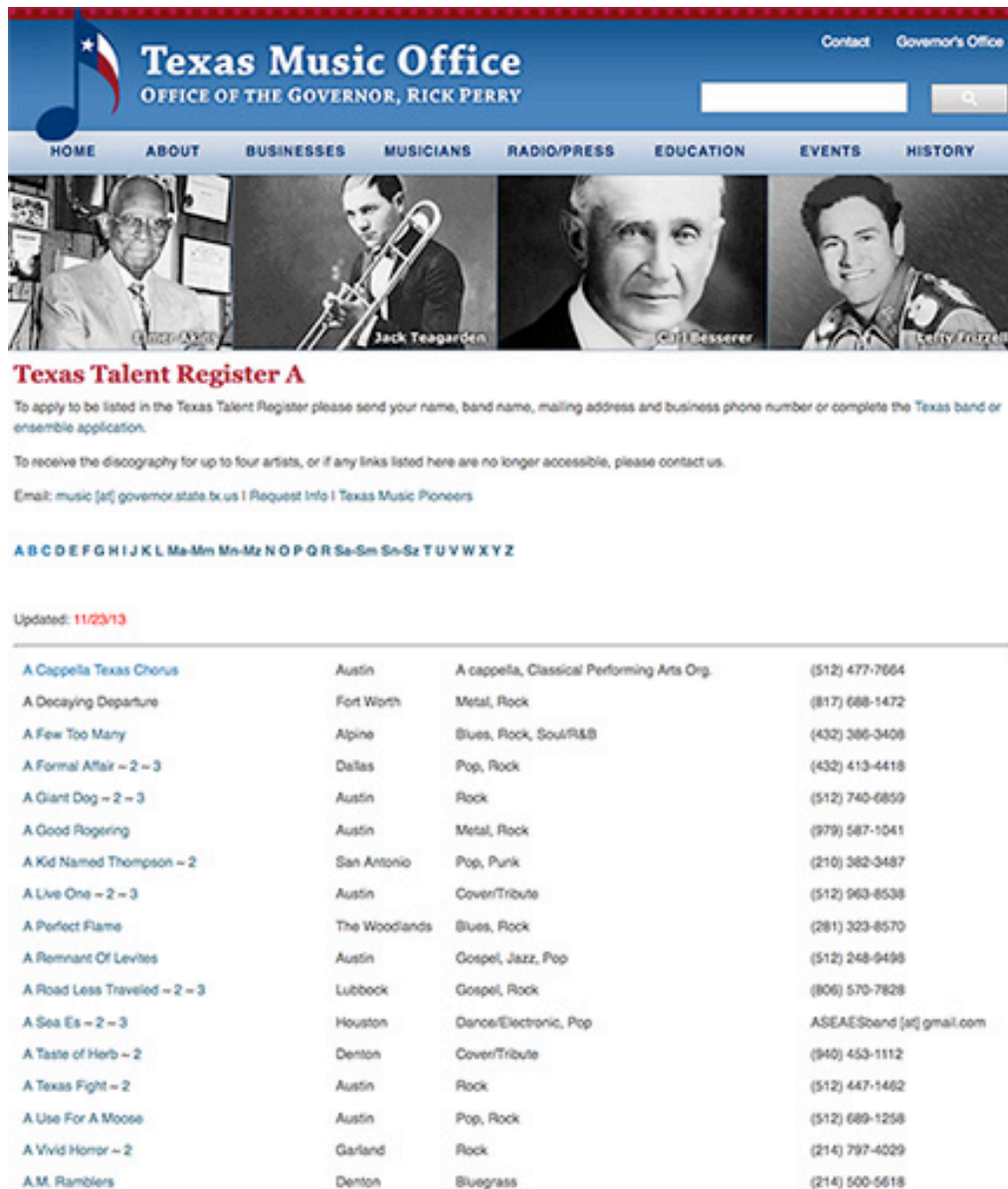
This one was fairly easy. But the data is kind of old on this page. Can you use this Wikipedia page with more updated data on circulation of newspapers in the world?

http://en.wikipedia.org/wiki/List_of_newspapers_in_the_world_by_circulation

- Modify the scraper to pull the data you want from this page.
- Change the url from which the scraper pulls,
- Change the identifier in the dom.cssselect statement,
- Change the number (len) of the tds in the table
- Identify the index for the data items you want to pull. You can add new data items, if you wish.
- Once you get comfortable with this code, you will start recognizing how to modify.

Comprehensive Example

Now, let's try this with some other data. We will reuse the code from this scraper. Copy the code from the last scraper and create a New Dataset. Call this one "music." We are going to pull the names and cities of musicians from the Texas Music Office site that lists musicians whose names start with the letter "A."



The screenshot shows the Texas Music Office website. The header includes the Texas Music Office logo, the text "OFFICE OF THE GOVERNOR, RICK PERRY", and navigation links for Contact, Governor's Office, and a search bar. Below the header is a navigation menu with links for HOME, ABOUT, BUSINESSES, MUSICIANS, RADIO/PRESS, EDUCATION, EVENTS, and HISTORY. A banner image features four musicians: Luther Youngblood, Jack Teagarden, Cliff Besserer, and Larry Fitzgerald. Below the banner is the "Texas Talent Register A" section, which includes instructions on how to apply to be listed in the register and contact information for requests. A list of musicians is provided, organized by city and then by name. The list includes names like A Cappella Texas Chorus, A Decaying Departure, A Few Too Many, A Formal Affair - 2 - 3, A Giant Dog - 2 - 3, A Good Rogering, A Kid Named Thompson - 2, A Live One - 2 - 3, A Perfect Flame, A Remnant Of Levites, A Road Less Traveled - 2 - 3, A Sea Es - 2 - 3, A Taste Of Herb - 2, A Texas Fight - 2, A Use For A Moose, A Vivid Horror - 2, and A.M. Ramblers. Each entry includes the city, the musician's name, and a phone number or email address.

Texas Music Office
OFFICE OF THE GOVERNOR, RICK PERRY

Contact Governor's Office

HOME ABOUT BUSINESSES MUSICIANS RADIO/PRESS EDUCATION EVENTS HISTORY

Texas Talent Register A

To apply to be listed in the Texas Talent Register please send your name, band name, mailing address and business phone number or complete the Texas band or ensemble application.

To receive the discography for up to four artists, or if any links listed here are no longer accessible, please contact us.

Email: music[at]governor.state.tx.us | Request Info | Texas Music Pioneers

Updated: 11/23/13

A Cappella Texas Chorus	Austin	A cappella, Classical Performing Arts Org.	(512) 477-7664
A Decaying Departure	Fort Worth	Metal, Rock	(817) 688-1472
A Few Too Many	Alpine	Blues, Rock, Soul/R&B	(432) 386-3408
A Formal Affair - 2 - 3	Dallas	Pop, Rock	(432) 413-4418
A Giant Dog - 2 - 3	Austin	Rock	(512) 740-6859
A Good Rogering	Austin	Metal, Rock	(979) 587-1041
A Kid Named Thompson - 2	San Antonio	Pop, Punk	(210) 382-3487
A Live One - 2 - 3	Austin	Cover/Tribute	(512) 963-8538
A Perfect Flame	The Woodlands	Blues, Rock	(281) 323-8570
A Remnant Of Levites	Austin	Gospel, Jazz, Pop	(512) 248-9498
A Road Less Traveled - 2 - 3	Lubbock	Gospel, Rock	(806) 570-7828
A Sea Es - 2 - 3	Houston	Dance/Electronic, Pop	ASEAESband[at]gmail.com
A Taste Of Herb - 2	Denton	Cover/Tribute	(940) 453-1112
A Texas Fight - 2	Austin	Rock	(512) 447-1462
A Use For A Moose	Austin	Pop, Rock	(512) 689-1258
A Vivid Horror - 2	Garland	Rock	(214) 797-4029
A.M. Ramblers	Denton	Bluegrass	(214) 500-5618

1. Let's go to the Texas Music Office Site.
<http://governor.state.tx.us/music/musicians/talent/talent>
2. The first page lists all the A musicians.
3. First, replace the url in the appropriate line of the scraper.
4. Now, take a look at the code on the html page. Use Chrome to Inspect Element in your browser. Where does the data begin? How can we identify this area? There may be several different ways to do it, but we are looking for any identifier code that will allow us to refine the query. In this case, I found a div that had an inline style. You put the styles in [] square brackets. The tr that we are looking for is within that div.
Try:

```
"div[style='text-align: center'] tr"
```

5. The "tds" variable will be the same. We are looking for table cell data.
6. The if statement should be changed to the number of columns for this table. This table has 9 columns. This helps identify exactly which table you wish to scrape on the page. If there is only one table on the page, you can run the scraper without the "if statement," but it doesn't hurt to be specific.
7. Adjust the data items you are pulling out. You can modify the integer conversion, if your info is all text. Note: even though circulation is in numbers, if the page has commas in the numbers, they will be pulled as strings. We can adjust later in a spreadsheet to remove.
8. Make sure the scraper is saved. You can view in the table and download for a spreadsheet.

Your code should look something like this. It is very similar to what we did with the newspaper circulation data, with just a few modifications.

```
#!/usr/bin/env python

import scraperwiki
import requests
import lxml.html

html =
requests.get("http://governor.state.tx.us/music/musicians/talent/talent").content
dom = lxml.html.fromstring(html)

for tr in dom.cssselect("div[style='text-align: center']tr"):
    tds = tr.cssselect("td")
    if len(tds)==4:
        data = {

            'Name': tds[0].text_content(),
            'City': tds[1].text_content(),
            'Genre': tds[2].text_content(),
            'Phone': tds[3].text_content()

        }
    print data
```

If you want to Save to the Datastore, don't forget to add this:

```
scraperwiki.sql.save(['Name'], data)
```

You may need to use the Clear button to clear the dataset to get a fresh set of data in the datastore. Then rerun the scraper.

Take a look at the results. We have all the data on the page, but some things are funny or missing:

- Weird characters with links on each name
- We only scraped the page with the A's. How do we get it to scrape all the other pages without having to run this individually on each page?
- Didn't scrape links
- Multiple genres/links for each band – we will handle this in a spreadsheet.

Let's improve it.

With a few more lines of code and our experience with loops and arrays, we can add some additional functionality to this scraper.

First we need to identify if there is any consistent naming scheme for each of the pages on the Texas Music Office site. If you look at the subsequent pages by clicking on the letters (B, C, etc.), you will see the naming scheme.

A - <http://governor.state.tx.us/music/musicians/talent/talent>

B - <http://governor.state.tx.us/music/musicians/talent/talentb/>

C - <http://governor.state.tx.us/music/musicians/talent/talentc/>

and so on... You can see that for the M's and S's, there is more than one page, but we can accommodate.

Let's make an array that holds all this information. Each letter is an element of the array as a string.

```
letters=["", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l", "m",  
        "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y", "z", "m2", "  
        s2"]
```

We also need to create a variable to hold the site name to make it easier to concatenate the letters to the end on each iteration of the loop.

```
site = http://governor.state.tx.us/music/musicians/talent/talent
```

Then everything goes in a for loop that performs the scrape and moves to the next page.

```
for letter in letters:  
    newsite = site + letter
```

We also want to give each band a unique id, so we are adding a counter that will be incremented for each band.

```
#initialize id
id = 0
```

So far we have this:

```
#!/usr/bin/env python

import scraperwiki
import requests
import lxml.html

#initialize id
id = 0

# loops through pages ending in the element in letters
# all letters of alphabet. Also a couple with 2 on the end
letters=["","b","c","d","e","f","g","h","i","j","k","l","m",
,"n","o","p","q","r","s","t","u","v","w","x","y","z","m2","s2"]

site =
"http://governor.state.tx.us/music/musicians/talent/talent"
for letter in letters:
    newsite = site + letter

    html = requests.get(newsite).content
    dom = lxml.html.fromstring(html)

    for tr in dom.cssselect("div[style='text-align: center'] tr"):
        tds = tr.cssselect("td")

        if len(tds)==4:

            # increments id
            id = id + 1
```

```
# writes the content into a list
data = {
    'id' : id,
    'name' : tds[0].text_content(),
    'city' : tds[1].text_content(),
    'genre': tds[2].text_content(),
    'phone': tds[3].text_content()
}
print data
```

One more thing will make this much better. The Texas Music Office uses numbers and ~ to indicate if a band has more than one Web address. But we don't need those numbers and symbols in the band name. Add this in the if statement that checks the length.

```
band = tds[0].text_content()
# this checks to see if there is a ~ in the
band name and removes rest of string
if '~' in band:
    i = band.index('~')
    band = band[:i]
else:
    band
```

The screenshot shows the ScraperWiki web interface. At the top, there's a navigation bar with the ScraperWiki logo, a user profile for 'CINDY ROYAL', and links for 'DOCS' and 'HELP'. Below this is a toolbar with buttons for 'Test music' (by Cindy Royal), 'Code in your browser', 'Download as spreadsheet', 'View in a table', and 'More tools...'. The main area is a code editor with a dark background, showing Python code that scrapes music data. The code includes a comment about writing content into an object and joining/separating a URL list. The code defines a 'data' dictionary with keys for 'id', 'name', 'city', 'genre', and 'phone', and uses BeautifulSoup's 'text_content()' method to extract data from table rows. The code also includes a conditional statement to remove tilde symbols from the band name. The code is executed, and the results are displayed at the bottom. The results are a list of dictionaries, each representing a music entry with fields for genre, city, id, phone, and name. The entries include bands like Doug Burr, Kim Burrell, Brison Bursey Band, Aaron Burton, Bury the Past, Bus Stop Stallions, and Elvis T. Busboy & the Blues Butchers.

```
# writes the content into an object; joins and separates url_list
data = {
    'id' : id,
    'name' : band,
    'city' : tds[1].text_content(),
    'genre': tds[2].text_content(),
    'phone': tds[3].text_content(),
}

print data
scraperwiki.sql.save(['id'],data)
```

```
{'genre': 'Folk/Acoustic, Rock', 'city': 'Denton', 'id': 1524, 'phone': '(940) 594-5543', 'name': 'Doug Burr '}
{'genre': 'Gospel, Jazz', 'city': 'Houston', 'id': 1525, 'phone': '(713) 384-3011', 'name': 'Kim Burrell'}
{'genre': 'Country, Rock', 'city': 'Huntsville', 'id': 1526, 'phone': '(830) 626-8005', 'name': 'Brison Bursey Band '}
{'genre': 'Blues, Folk/Acoustic', 'city': 'Dallas', 'id': 1527, 'phone': '(214) 476-1054', 'name': 'Aaron Burton '}
{'genre': 'Rock', 'city': 'Beaumont', 'id': 1528, 'phone': '(337) 515-5113', 'name': 'Bury the Past'}
{'genre': 'Rock, Soul/R&B', 'city': 'Austin', 'id': 1529, 'phone': '(512) 731-4870', 'name': 'Bus Stop Stallions '}
{'genre': 'Blues, Soul/R&B', 'city': 'Arlington', 'id': 1530, 'phone': '(682) 429-9941', 'name': 'Elvis T. Busboy & the Blues Butchers'}
```

Scraping the Links

The links pose an interesting challenge. Not only do we need to get the url associated with an `<a>` link, but in some cases, we need to scrape more than one.

This nifty code will get you the first link. Put this code below the code that increments the "id".

```
# get url (if it exists)
url_cells = tds[0].cssselect('a')

if url_cells:
    url = url_cells[0].get('href')

else:
    url = ''
```

This code finds the `<a>` tags in the table cells indicated by the first `<td>` `tds[0]`. If one exists, then it gets the content of the href of the first `<a>`

Then make sure you add this line to the data object, so it will print this information.

```
'url': url
```

(be sure to include a comma at the end of the line above this line)

This method only gets the first url. But some of the bands have more than one url

But we really should try to get all the links. Replace the code above with this:

```
# finds urls
url_cells = tds[0].cssselect('a')

# create a list to hold urls
url_list = []

# get multiple urls (if they exist)
if url_cells:
    for cell in url_cells:
        url_list.append(cell.get('href'))
```

What this code does is creates a list, then finds all the <a> tags and records the result of the href.

Modify the url line in the data object to look like this.

```
'url': ', '.join(url_list)
```

This takes the items in the url_list and the join separates them with a comma, so they can be read into the datastore and spreadsheet.

The screenshot shows the ScraperWiki web interface. At the top, there's a navigation bar with the ScraperWiki logo, a user profile for 'CINDY ROYAL', and links for 'DOCS' and 'HELP'. Below this is a toolbar with buttons for 'Test music' (by Cindy Royal), 'Code in your browser', 'Download as spreadsheet', 'View in a table', and 'More tools...'. The main area is a code editor showing Python code that extracts links from a table. The code includes comments and uses BeautifulSoup's CSS selector to find all 'a' tags and append their hrefs to a list. Below the code editor, there's a status bar with 'Schedule', 'Stop!', and 'Running...' buttons. At the bottom, a table of extracted data is shown, with columns for city, name, url, phone, genre, and id. The table contains several rows of data, including 'Mother Falcon', 'The Mother Truckers', 'Mount Righteous', 'Mount Vernon Music', 'Mountains in the Moon', 'Mouse.About', 'Movin' Melvin Brown', and 'Movin' On Band'.

```
# section to handle links
url_cells = tds[0].cssselect('a')
url_list = []

if url_cells:
    for cell in url_cells:
        url_list.append(cell.get('href'))

# writes the content into an object; joins and separates url_list
data = {
    'id': id,
    'name': band,
    'city': tds[1].text_content(),
    'genre': tds[2].text_content(),
    'phone': tds[3].text_content(),
    'url': ', '.join(url_list)
```

{'city': 'Austin', 'name': 'Mother Falcon', 'url': 'http://www.motherfalconmusic.com/', 'phone': '(512) 825-3905', 'genre': 'Classical, Pop', 'id': 512}
{'city': 'Austin', 'name': 'The Mother Truckers ', 'url': 'http://www.themothertruckers.com/, http://www.myspace.com/themothertruckers', 'phone': '(615)
{'city': 'Grapevine', 'name': 'Mount Righteous ', 'url': 'http://www.mountrighteous.com/, http://kenran.com/', 'phone': '(817) 573-6100', 'genre': 'Pop
{'city': 'Mount Vernon', 'name': 'Mount Vernon Music', 'url': 'http://www.mountvernonmusic.org', 'phone': '(903) 563-3780', 'genre': 'Classical, Jazz',
{'city': 'Austin', 'name': 'Mountains in the Moon', 'url': 'http://www.myspace.com/mountainsinthemoon', 'phone': '(281) 734-4760', 'genre': 'Pop, Rock'
{'city': 'Austin', 'name': 'Mouse.About ', 'url': 'http://www.mouseabout.net/, http://twitter.com/mouseabout/, http://www.sonicbids.com/mouseabout, htt
{'city': 'Austin', 'name': 'Movin' Melvin Brown ', 'url': 'http://www.movinmelvin.com/, http://www.iloveyouday.com/', 'phone': '(512) 339-9030', 'genre
{'city': 'Houston', 'name': 'Movin' On Band', 'url': '', 'phone': '(832) 527-6000', 'genre': 'Blues, Country, Rock', 'id': 519}

Improving the “letters” array

That array with the letters in it works, but it is pretty long. We can improve it by iterating through the alphabet with a function.

Below the line where we initialize the id and above where we set up the site variable, include this function replacing the original array. Be careful with the indentation.

```
# loops through pages ending in the element in letters
# all letters of alphabet and extra characters
letters = ["", "m2", "s2"]
def letter_list(start, stop):

    for c in range(ord(start),ord(stop)+1):
        letters.append(chr(c))

letter_list('b', 'z') //calls the function
```

Here’s what we did:

- Created an array to hold the empty string, “m2” and “s2”
- Set up a function that requires two arguments – start and stop
- Within a for loop, we convert the letters to ordinal (“ord”), so we can use the range function. You have to add 1 (+1) to the stop to get it to run through “z” and not stop before.
- Convert each letter “c” back to a character and then append to the letters array.
- Call the letter_list function and provide the arguments for “b” through “z”.
- This gets called for every letter and scrapes each page.

Final Code

The code below is the commented final scraper. Do you understand what each segment is doing?

```
import scraperwiki
import requests
import lxml.html

#initialize id
id = 0

# loops through pages ending in the element in letters
# all letters of alphabet. Also a couple with 2 on the end
letters = ["", "m2", "s2"]
def letter_list(start, stop):

    for c in range(ord(start),ord(stop)+1):
        letters.append(chr(c))

letter_list('b', 'z')

site = "http://governor.state.tx.us/music/musicians/talent/talent"
for letter in letters:
    newsite = site + letter

    html = requests.get(newsite).content
    dom = lxml.html.fromstring(html)

    for tr in dom.cssselect("div[style='text-align: center'] tr"):
        tds = tr.cssselect("td")

        band = tds[0].text_content()

        # this checks to see if there is a ~ in band name/removes rest
        if '~' in band:
            i = band.index('~')
            band = band[:i]
        else:
            band

        # increments id
        id = id + 1
```



```

# finds urls
url_cells = tds[0].cssselect('a')

# create a list to hold urls
url_list = []

# get multiple urls (if they exist)
if url_cells:
    for cell in url_cells:
        url_list.append(cell.get('href'))

# writes the content into an object;
# joins and separates url_list
data = {
    'id' : id,
    'name' : band,
    'city' : tds[1].text_content(),
    'genre' : tds[2].text_content(),
    'phone' : tds[3].text_content(),
    'url' : ', '.join(url_list)
}
print data
scrapewiki.sql.save(['id'],data)

```

Don't worry that the columns appear to be out of order. It really doesn't matter in a database, which is what we will ultimately be using this in. And, it is easy to change the order of the columns in Excel or Google Spreadsheets. It's not all that hard if you break things down and go step by step.

Note: This scraper worked nicely until I encountered a problem in the html of the site. Apparently, some new entries had been made on the site, but the html was not closed properly. This threw an `IndexError`. I added a `try` and `except` statement around the code starting inside the 2nd for loop, with the exception happening at the very end to handle problems of this nature.

`try:`
... all the code inside the 2nd for loop

```
except IndexError:  
    print "{error}"
```

Separating Multiple Genres and Links

It's easy when you get the data into Excel to separate lines with multiple genres. Our genre and links columns separate each with a comma. Make sure your genre column is the last one in the spreadsheet. Select the entire column. Then go to Data, Text To Columns and go through the Wizard to identify the comma as the separator. This will spread the different genres out to their own column. You can then use them this way or combine later into one column to sort. Do the same for the links column, making sure you move it to the end before running the Text To Columns feature.

Troubleshooting

- If you see any errors in the console, read them and try to correct the problem on the line number that is indicated.
- In some cases, you won't get an error, but you don't get any data either. Check your dom.cssselect markers and the number of columns you indicated (if needed).
- Keep trying until you see data accumulating in the data tab when you run the program.

Extra Credit: Pick a site of your own that has data in an html table and use the code to scrape it. What did you have to do to make this work? Explain.