

Chapter 9

Your First Multi-Page Scrape

Skills you will learn: How to make requests to multiple URLs using For loops and by altering the URL

In this tutorial, we will pick up from the detailed example from Chapter 9, though we will make a few changes because we will be scraping the original, live site, and not the demonstration site at dataprof.ca.

Here is the code we ended up with in Chapter 9.

```
1. #!/usr/bin/env python
2. import urllib2
3. from bs4 import BeautifulSoup
4. import unicodcsv
5. output = open('complete file path/myoutput.txt', 'w')
6. writer = unicodcsv.writer(output, delimiter='\t',
encoding='utf-8')
7. writer.writerow(['date', 'vendor', 'description', 'value'])
8. URL = "http://dataprof.ca/DataJournalist/scrapingla.
html"
9. response = urllib2.urlopen(URL)
10. HTML = response.read()
11. soup = BeautifulSoup(HTML)
12. for eachrow in soup.findAll('tr'):
13.     data = []
14.     cells = eachrow.findAll('td')
15.     for eachdataitem in cells:
16.         data.append(eachdataitem.text)
17.     writer.writerow(data)
```

In the above, be sure to replace 'complete file path' in line 5 with the path to the file on your computer. Line numbers are for reference only. Don't type them!

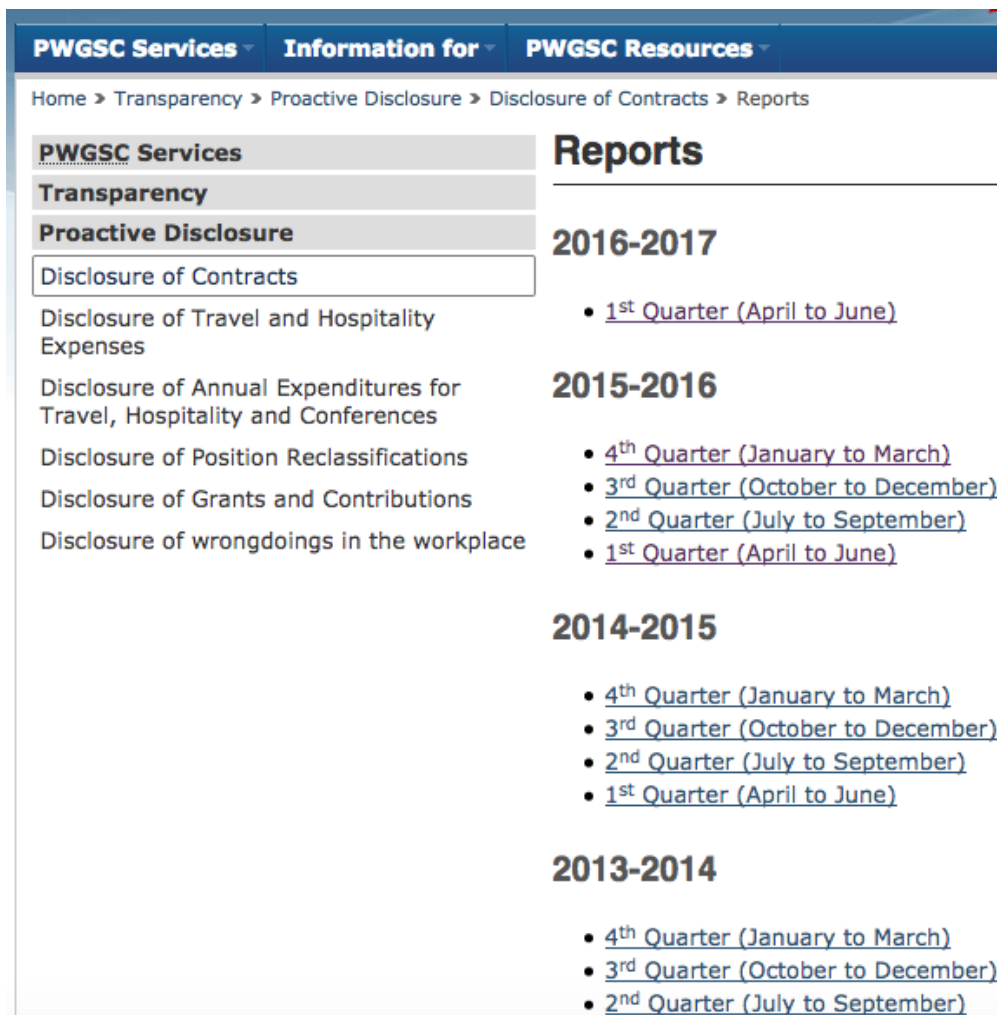
This code does a lot, but there is more than one webpage. In fact, there are pages dating back to 2004, and we don't want to have to run the script separately for each one, though we could.

To scrape more than one page, we will have to make the scraper go back to each page in turn, and that's a perfect job for a loop. What we are doing is a very common approach to developing this kind of script; figure out a way to tackle one page, then find a way to run that code over and over again.

In Chapter 9, we used a page we served on our own website, but for this tutorial, we will scrape the original website on the Government of Canada Servers. With the federal government's changes to its public-facing websites, the data used for this tutorial is now on an archived site that you can find at <https://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng&SCR=Q&Sort=0>

One of the first steps to a decision on how to do this is to take a look at what request is being sent to the web server when one of the main pages is fetched.

So let's use the element inspector in Firefox developer tools to have a look at requests being sent to the server when we click on different links on the page that lists all of the quarterly reports. As a reminder, it looks like this:



The screenshot shows a web page with a blue header containing three navigation tabs: "PWGSC Services", "Information for", and "PWGSC Resources". Below the header is a breadcrumb trail: "Home > Transparency > Proactive Disclosure > Disclosure of Contracts > Reports". On the left side, there is a vertical menu with the following items: "PWGSC Services", "Transparency", "Proactive Disclosure", and "Disclosure of Contracts" (which is highlighted with a white border). Below this menu are several links: "Disclosure of Travel and Hospitality Expenses", "Disclosure of Annual Expenditures for Travel, Hospitality and Conferences", "Disclosure of Position Reclassifications", "Disclosure of Grants and Contributions", and "Disclosure of wrongdoings in the workplace". The main content area is titled "Reports" and is divided into sections for different years: "2016-2017", "2015-2016", "2014-2015", and "2013-2014". Each year section contains a list of quarterly reports, with the 1st quarter of each year being highlighted in blue.

PWGSC Services Information for PWGSC Resources

Home > Transparency > Proactive Disclosure > Disclosure of Contracts > Reports

PWGSC Services
Transparency
Proactive Disclosure
Disclosure of Contracts

Disclosure of Travel and Hospitality Expenses
Disclosure of Annual Expenditures for Travel, Hospitality and Conferences
Disclosure of Position Reclassifications
Disclosure of Grants and Contributions
Disclosure of wrongdoings in the workplace

Reports

2016-2017

- [1st Quarter \(April to June\)](#)

2015-2016

- [4th Quarter \(January to March\)](#)
- [3rd Quarter \(October to December\)](#)
- [2nd Quarter \(July to September\)](#)
- [1st Quarter \(April to June\)](#)

2014-2015

- [4th Quarter \(January to March\)](#)
- [3rd Quarter \(October to December\)](#)
- [2nd Quarter \(July to September\)](#)
- [1st Quarter \(April to June\)](#)

2013-2014

- [4th Quarter \(January to March\)](#)
- [3rd Quarter \(October to December\)](#)
- [2nd Quarter \(July to September\)](#)

So we can see that there are many links on the page, one for each quarter. When clicked, these take us to the detail page.

PWGSC Services Information for PWGSC Resources			
Home > Transparency > Proactive Disclosure > Disclosure of Contracts > Reports > 2016-2017 - 1 st Quarter (April to June)			
PWGSC Services Transparency Proactive Disclosure Disclosure of Contracts Disclosure of Travel and Hospitality Expenses Disclosure of Annual Expenditures for Travel, Hospitality and Conferences Disclosure of Position Reclassifications Disclosure of Grants and Contributions Disclosure of wrongdoings in the workplace		2016-2017 - 1st Quarter (April to June) N.B.: The contract date represents the date that the contract is recorded in the departmental financial system. List of contracts for the trimester, which includes the date, vendor, description and value	
Contract Date	Vendor Name	Description of Work	Contract Value
2008-04-22	THYSSENKRUPP ELEVATOR (CANADA) LTD.	859 - Other Business Services not Elsewhere Specified	\$112,072.99
2008-04-22	THYSSENKRUPP ELEVATOR (CANADA) LTD.	859 - Other Business Services not Elsewhere Specified	\$181,839.89
2008-04-22	CAPITAL ELEVATOR LTD.	859 - Other Business Services not Elsewhere Specified	\$73,635.29
2008-07-15	NORR LIMITED	421 - Architectural Services	\$10,358,526.57
2008-09-04	NORR LIMITED	421 - Architectural Services	\$40,770,897.54
2009-01-17	SIEMENS BUILDING TECHNOLOGIES LTD.	460 - Protection Services	\$569,639.03
2009-02-17	FT3 ARCHITECTURE LANDSCAPE	859 - Other Business Services not Elsewhere Specified	\$5,248,488.67

If we were to conceptualize our scraper, then, it has to go from the main page to a detail page, scrape that page of the data, then return to the main page and repeat until done. We're going to have to send a request to the server for each page we want to scrape. To figure out what to send, we can use the element inspector. You'll recall the element inspector from the tutorial **Using Development Tools to Examine Webpages**.

Using the inspector, we can see the links for the first three of the listed detail reports.

```

<ul>
  <li>
    <a href="https://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q3.txt"></a>
  </li>
  <li>
    <a href="https://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q2.txt"></a>
  </li>
  <li>
    <a href="https://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q1.txt"></a>
  </li>

```

Let's take a look at a couple more. This one is from 2011-12,

```

<ul>
  <li>
    <a href="http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201011Q4.txt">
  </li>
  <li>
    </li>

```

And this one is the very first link from 2004-05:

```
</li>
  <a href="http://www.tpsgc-pwgscc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL200405Q1.txt">
</li>
/nl>
```

An examination shows that the links are identical, except for the part that follows PF=CL

We can see that the fiscal year is represented by six digits, such as 200405 for 2004-05 and the quarter is represented by the letter Q followed by one of the digits 1 to 4. For now, we will assume that this pattern holds for all of the possible links. There is little reason to believe it would vary, so it's a pretty good bet. We'll find out anyway when we try to run our script.

When we send requests to the server, we are going to have to send the part of the URL that doesn't change, and then each time we send the request, substitute the part that does. It changes in two ways. For each new fiscal year, there is a new six-digit string. And within each year, there are the four combinations of Q1,Q2,Q3,Q4. This sounds like a job for two For loops, embedded one inside the other.

The first For loop would loop through each of the possible fiscal years, and the second would loop through each of the possible quarters.

Now, we could, as an alternative approach, grab all of the links in the page using Beautiful Soup, but the links are so consistent that we can accomplish the same goal by altering the URL. Besides, there are many additional links in the page, other than links for contract reports, so the approach we are going to follow should be simpler.

A straightforward approach is to put all of the possible fiscal year variants into a Python list. It would look like this:

```
years=[200405,200506,200607,200708,200809,200910,201011,201112,201213,201314,201415,201516,201617]
```

For now, we've made the elements in the list numbers. That's easier than inserting quotations around each element to turn them into strings. We can convert them to strings later when we actually construct the URLs.

We can do the same thing with the quarters.

```
quarters=[1,2,3,4]
```

Now we have to figure out how to make our URLs.

Alright, let's write some code to create the URLs one after another.

```
for fiscalYear in years:
    for quarter in quarters:
        url = 'http://www.tpsgc-pwgsc.gc.ca/cgi-
            bin/proactive/cl.pl?lang=eng;SCR=L; Sort=0;PF=CL' +
            str(fiscalYear) + 'Q' + str(quarter) + '.txt'
```

In your code editor, the url = line will be one line of code that doesn't wrap, but to put it all on the page, we've wrapped it here. Let's walk through what this does.

The first line, for fiscalYear in years: will loop, or iterate, through each item in the list years. The code in the indented block below will then run each time Python gets to a new year. That code happens to be another loop, this time looping through each of the quarters in the loop quarters. It will loop through all of the quarters before going back to the second element in years, and do this until there are no years left.

Inside the indented code block below the second loop, more happens. We set a new variable called URL, and populate it with the part of the URL that doesn't change:

```
http://www.tpsgc-pwgsc.gc.ca/cgi-
bin/proactive/cl.pl?lang=eng;SCR=L; Sort=0;PF=CL
```

and the part that does

```
str(fiscalYear) + 'Q' + str(quarter) + '.txt'
```

The part that changes is made up of the fiscalYear, which comes from the outermost loop, the letter Q, the quarter, taken from the second, embedded loop, and the file extension .txt, which is the last part of the URL. We're using the str() method to convert the fiscalYear and quarter elements into text, so they can be concatenated with everything else. The + sign is the concatenation operator, so each time you see it, that means another piece of text is being added to what came before. The result is a complete URL that can be sent to the server. If you want to test it out, just add a print statement and run it.

```
years=[200405,200506,200607,200708,200809,200910,201011,201112,2
01213,201314,201415,201516,201617]
```

```
quarters=[1,2,3,4]
```

```
for fiscalYear in years:
    for quarter in quarters:
        url = 'http://www.tpsgc-pwgsc.gc.ca/cgi-
            bin/proactive/cl.pl?lang=eng;SCR=L; Sort=0;PF=CL' +
            str(fiscalYear) + 'Q' + str(quarter) + '.txt'
        print url
```


http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201516Q2.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201516Q3.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201516Q4.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q1.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q2.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q3.txt
http://www.tpsgc-pwgsc.gc.ca/cgi-bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL201617Q4.txt

If you look through, you can see how the URL has been changed each time. If you want, try one of the URLs out in your browser, to make sure it works. At the time of writing this tutorial, there were no reporters for 2016-17, quarters 2 to 4, so the last three URLs would fail. But as new reports are posted, they will work. And as long as the page design doesn't change, we can make our script work for future years by adding them to the first list. Be cautioned that you can expect sites to change over time, so a script written today may not work next year.

So far, we have just printed the URLs to the screen, but you can probably see that incorporating this new looping functionality into our previous script is going to be easy.

```
#!/usr/bin/env python
import urllib2
from bs4 import BeautifulSoup
import unicodedsv
import time
output = open('fullPathToFile.txt', 'w')
writer = unicodedsv.writer(output, delimiter='\t',
encoding='utf-8')
writer.writerow(['date', 'vendor', 'description', 'value'])
years=[200405,200506,200607,200708,200809,200910,201011,
201112,201213,201314,201415,201516,201617]
quarters=[1,2,3,4]
for fiscalYear in years:
    for quarter in quarters:
        url = 'http://www.tpsgc-pwgsc.gc.ca/cgi-
        bin/proactive/cl.pl?lang=eng;SCR=L;Sort=0;PF=CL' +
        str(fiscalYear) + 'Q' + str(quarter) + '.txt'
        response = urllib2.urlopen(url)
        time.sleep(5)
        HTML = response.read()
        soup = BeautifulSoup(HTML, "html.parser")
        for eachrow in soup.findAll('tr'):
            data =[]
            cells = eachrow.findAll('td')
            for eachdataitem in cells:
                data.append(eachdataitem.text)
            writer.writerow(data)
            print data
```

You may wish to compare this script with the script at the beginning of this tutorial. The biggest change is that we have added the years and quarters lists, and integrated the loop to iterate through all of the possible URLs. These relatively small changes make our script do a lot more.

An important point: if you are copying and pasting the script above into a coding text editor, be sure to clean up any indenting issues, and ensure that the single and double quotation marks are plain quotation marks and not the curly smartquotes used in typography.

One other really important change is found in lines 5 and 15. In line 5, we import the time module, which like urllib2 is one of the Python standard library modules. In line 15 we use the sleep method of time to add in a five second delay between each request to the server.

Your computer is capable of sending dozens if not hundreds of requests to the server every second; you are limited only by the speed at which Python can iterate through the loop. The `time.sleep(5)` line puts in a five-second delay before the next request. As noted in Chapter 9 of *The Data Journalist*, this not only makes your script behave a bit more like a real person using a browser (though what real person would look at each contract page for just a few seconds), but more importantly, it is just polite. The government of Canada server we are sending requests to here is probably pretty capable, and can handle many requests per second. But smaller servers might actually be taken down by a barrage of traffic in a short time. By spacing out our requests, we ensure we are being good Net citizens. There is a longer discussion of the ethics and practice of web scraping in Chapter 9 of *The Data Journalist*.