

# Web scraping with Python Workshop

Denver McNeney

Centre for the Study of Democratic Citizenship  
McGill University

[denver.mcneney@mcgill.ca](mailto:denver.mcneney@mcgill.ca)

April 4, 2017

## 1 Introduction

- Required Resources
- Installing Python & Packages

## 2 Webscraping

- Scraping Overview
- Scraping with Python & Selenium

## 3 Lab Exercise

## 4 Conclusion

- Words of Caution

## 1 Introduction

- Required Resources
- Installing Python & Packages

## 2 Web scraping

- Scraping Overview
- Scraping with Python & Selenium

## 3 Lab Exercise

## 4 Conclusion

- Words of Caution

## 1 Introduction

- Required Resources
- Installing Python & Packages

## 2 Webscraping

- Scraping Overview
- Scraping with Python & Selenium

## 3 Lab Exercise

## 4 Conclusion

- Words of Caution

## 1 Introduction

- Required Resources
- Installing Python & Packages

## 2 Webscraping

- Scraping Overview
- Scraping with Python & Selenium

## 3 Lab Exercise

## 4 Conclusion

- Words of Caution

# Download Materials

- To download slides, scripts, and example materials, visit:  
<http://www.dmcneney.com/>
- To download Python, visit:  
<https://www.python.org/downloads/>

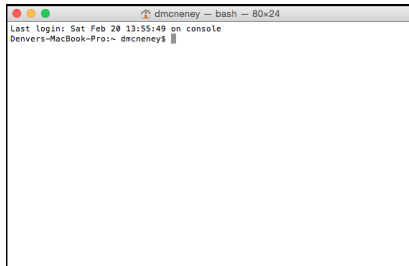
# Download Materials

- Download Python 3.6.1
- OS X
  - Open Terminal
- Windows
  - Open Command Prompt



# Download Materials

- Download Python 3.6.1
- OS X
  - Open **Terminal**
- Windows
  - Open **Command Prompt**





# Updating Python Installer

## Windows

```
python C:\PATHTOFILE\get-pip.py
```

## OS X

```
sudo easy_install pip
```

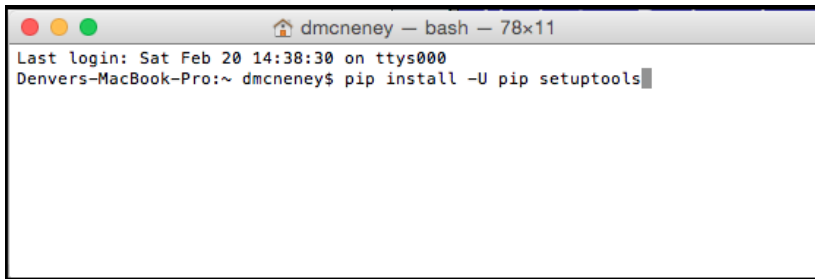
# Updating Python Installer

## Windows

```
python -m pip install -U pip setuptools
```

## OS X

```
pip install -U pip setuptools
```

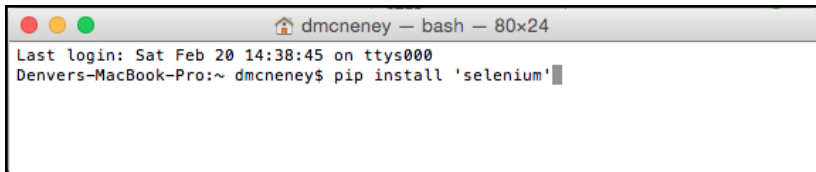


A screenshot of a macOS terminal window. The title bar shows a home icon, the username 'dmcneney', and the shell 'bash' with a window size of '78x11'. The terminal text shows the last login time as 'Sat Feb 20 14:38:30 on ttys000' and the current command being executed: 'Denvers-MacBook-Pro:~ dmcneney\$ pip install -U pip setuptools'. The cursor is at the end of the command line.

# Download Python Packages

## Installing Packages

```
pip install 'Package Name'
```



A screenshot of a macOS terminal window. The title bar shows three colored window control buttons (red, yellow, green) on the left, a home icon followed by the text 'dmcneney — bash — 80x24' in the center, and standard macOS window controls on the right. The terminal content shows the last login time as 'Sat Feb 20 14:38:45 on ttys000' and the current command prompt as 'Denvers-MacBook-Pro:~ dmcneney\$'. The command 'pip install 'selenium'' is entered at the prompt, with a cursor at the end of the line.

# Download Python Packages

## Installing Packages

```
pip install 'Package Name'
```

## Required Packages

- “selenium”
- “requests”

# Webscraping: What is it?

- Automatic extraction and parsing of online information to create structured database
- Two kinds:
  - Web APIs (Application Program Interface) → Website or database creates interface for data requests that return JSON or XML files
  - Screen or Pseudo-Manual Scraping → Need to either extract from html or interact with website using bots to download materials

# Webscraping: What is it?

- Automatic extraction and parsing of online information to create structured database
- Two kinds:
  - Web APIs (Application Program Interface) → Website or database creates interface for data requests that return JSON or XML files
  - Screen or Pseudo-Manual Scraping → Need to either extract from html or interact with website using bots to download materials

# Webscraping: What is it?

- Automatic extraction and parsing of online information to create structured database
- Two kinds:
  - Web APIs (Application Program Interface) → Website or database creates interface for data requests that return JSON or XML files
  - Screen or Pseudo-Manual Scraping → Need to either extract from html or interact with website using bots to download materials

# Webscraping: What is it?

- Automatic extraction and parsing of online information to create structured database
- Two kinds:
  - Web APIs (Application Program Interface) → Website or database creates interface for data requests that return JSON or XML files
  - Screen or Pseudo-Manual Scraping → Need to either extract from html or interact with website using bots to download materials



# Downloading Files with Python

- Data occasionally available across a number of websites
- No APIs exist
- Need to interact with web-browser beyond just copying text (i.e. click links, select dropdown menus, enter search terms, etc.)

## Roll Call Data

All the roll call matrices and codebooks listed in the links below are in a common format. Questions about the layout Roll Call Matrices and codebooks for Congresses 109 - 113 were compiled by Jeff Lewis and Keith Poole.

[113th House Roll Call Data](#)  
[113th Senate Roll Call Data](#)  
[112th House Roll Call Data](#)  
[112th Senate Roll Call Data](#)  
[111th House Roll Call Data](#)  
[111th Senate Roll Call Data](#)  
[110th House Roll Call Data](#)  
[110th Senate Roll Call Data](#)  
[109th House Roll Call Data](#)  
[109th Senate Roll Call Data](#)

Roll Call Matrices and codebooks for Congresses 102 - 108 were compiled by Keith Poole and Nolan McCarty.

[108th House Roll Call Data](#)  
[108th Senate Roll Call Data](#)  
[107th House Roll Call Data](#)  
[107th Senate Roll Call Data](#)  
[106th House Roll Call Data](#)  
[106th Senate Roll Call Data](#)  
[105th House Roll Call Data](#)  
[105th Senate Roll Call Data](#)  
[104th House Roll Call Data](#)  
[104th Senate Roll Call Data](#)  
[103rd House Roll Call Data](#)  
[103rd Senate Roll Call Data](#)  
[102nd House Roll Call Data](#)  
[102nd Senate Roll Call Data](#)

# Downloading Files with Python

- Data occasionally available across a number of websites
- No APIs exist
- Need to interact with web-browser beyond just copying text (i.e. click links, select dropdown menus, enter search terms, etc.)

## Roll Call Data

All the roll call matrices and codebooks listed in the links below are in a common format. Questions about the layout of the Roll Call Matrices and codebooks for Congresses 109 - 113 were compiled by Jeff Lewis and Keith Poole.

[113th House Roll Call Data](#),  
[113th Senate Roll Call Data](#),  
[112th House Roll Call Data](#),  
[112th Senate Roll Call Data](#),  
[111th House Roll Call Data](#),  
[111th Senate Roll Call Data](#),  
[110th House Roll Call Data](#),  
[110th Senate Roll Call Data](#),  
[109th House Roll Call Data](#),  
[109th Senate Roll Call Data](#).

Roll Call Matrices and codebooks for Congresses 102 - 108 were compiled by Keith Poole and Nolan McCarty.

[108th House Roll Call Data](#),  
[108th Senate Roll Call Data](#),  
[107th House Roll Call Data](#),  
[107th Senate Roll Call Data](#),  
[106th House Roll Call Data](#),  
[106th Senate Roll Call Data](#),  
[105th House Roll Call Data](#),  
[105th Senate Roll Call Data](#),  
[104th House Roll Call Data](#),  
[104th Senate Roll Call Data](#),  
[103rd House Roll Call Data](#),  
[103rd Senate Roll Call Data](#),  
[102nd House Roll Call Data](#),  
[102nd Senate Roll Call Data](#).

# Downloading Files with Python

- Data occasionally available across a number of websites
- No APIs exist
- Need to interact with web-browser beyond just copying text (i.e. click links, select dropdown menus, enter search terms, etc.)

## Roll Call Data

All the roll call matrices and codebooks listed in the links below are in a common format. Questions about the layout Roll Call Matrices and codebooks for Congresses 109 - 113 were compiled by Jeff Lewis and Keith Poole.

[113th House Roll Call Data](#)  
[113th Senate Roll Call Data](#)  
[112th House Roll Call Data](#)  
[112th Senate Roll Call Data](#)  
[111th House Roll Call Data](#)  
[111th Senate Roll Call Data](#)  
[110th House Roll Call Data](#)  
[110th Senate Roll Call Data](#)  
[109th House Roll Call Data](#)  
[109th Senate Roll Call Data](#)

Roll Call Matrices and codebooks for Congresses 102 - 108 were compiled by Keith Poole and Nolan McCarty.

[108th House Roll Call Data](#)  
[108th Senate Roll Call Data](#)  
[107th House Roll Call Data](#)  
[107th Senate Roll Call Data](#)  
[106th House Roll Call Data](#)  
[106th Senate Roll Call Data](#)  
[105th House Roll Call Data](#)  
[105th Senate Roll Call Data](#)  
[104th House Roll Call Data](#)  
[104th Senate Roll Call Data](#)  
[103rd House Roll Call Data](#)  
[103rd Senate Roll Call Data](#)  
[102nd House Roll Call Data](#)  
[102nd Senate Roll Call Data](#)

# Python: Selenium

Open “vancouver.py” in a text editor **that is not Word**.

## Python Code

```
import selenium
import os
from selenium import webdriver
import time
import requests
from selenium.webdriver.chrome.options
    import Options
```

# Python: Selenium

## Python Code

```
chrome_profile = webdriver.ChromeOptions()
profile = {'download.default_directory': '/Users/dmcneney/Desktop/Vancouver',
          'download.prompt_for_download': False,
          'download.directory_upgrade': True,
          'plugins.plugins_disabled': ['Chrome PDF Viewer']}
```

# Python: Selenium

## Python Code

```
chrome_profile.add_experimental_option('
    prefs', profile)
chrome_profile.add_argument('--disable-
    extensions')
browser = webdriver.Chrome(executable_path='
    /Users/dmcneney/Desktop/chromedrivermac',
    chrome_options=chrome_profile)
browser.implicitly_wait(30)
```

# Python: Selenium

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

The screenshot displays a web application interface on the left and its corresponding HTML structure on the right. The interface, titled "ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)", includes a "Search in:" section with radio buttons for "All City of Vancouver web pages", "City Council Meetings and Agendas", "Park Board Meetings and Agendas", and "Policy and Procedures". Below this, there are search criteria fields: "should contain", "in the body", "the words", "dated", and "sorted by relevance". The HTML code on the right shows a table with a search input field. A context menu is open over the input field, showing options like "Copy", "Copy XPath", and "Copy element".

# Python: Selenium

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

The screenshot displays a web application's search interface on the left and its corresponding HTML structure on the right. The search interface, titled "ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)", includes a "Search in:" section with radio buttons for "All City of Vancouver web pages", "City Council Meetings and Agendas", "Park Board Meetings and Agendas", and "Policy and Procedures". Below this, there are three rows of search criteria, each with a dropdown menu for "should contain", "must contain", or "must not contain", and a text input field. The "dated" section has radio buttons for "Anytime", "in the last week", and "on or after 12 January 2009 and before 20 January 2009". The "and show" section has a dropdown for "10 results", a dropdown for "sorted by relevance", and checkboxes for "with summaries" and "without grouping". The HTML structure on the right shows a table with a legend and a search input field. A context menu is open over the search input field, showing options like "Add Attribute", "Edit as HTML", "Copy", "Copy XPath", "Cut element", "Copy element", and "Paste element".



# Python: Selenium

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

The screenshot displays a web application's search interface on the left and its corresponding HTML structure on the right. The search interface, titled "ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)", includes a "Search in:" section with radio buttons for "All City of Vancouver web pages", "City Council Meetings and Agendas", "Park Board Meetings and Agendas", and "Policy and Procedures". Below this, there are three rows of search criteria, each with a dropdown menu for "should contain", "must contain", or "must not contain", and a text input field. The "dated" section has radio buttons for "Anytime", "in the last week", and "on or after 12 January 2009 and before 20 January 2009". The "and show" section has a dropdown for "10 results", a dropdown for "sorted by relevance", and checkboxes for "with summaries" and "without grouping". The HTML structure on the right shows a table with a legend and a search input field. A context menu is open over the search input field, showing options like "Add Attribute", "Edit as HTML", "Copy", "Copy XPath", "Cut element", "Copy element", and "Paste element".

# Python: Selenium

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

The screenshot displays a web interface on the left and an HTML DOM tree on the right. The web interface, titled "ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)", includes a "Search in:" section with radio buttons for "All City of Vancouver web pages", "City Council Meetings and Agendas", "Park Board Meetings and Agendas", and "Policy and Procedures". Below this are search filters: "should contain", "must contain", and "must not contain", each with dropdowns for "in the body" and "the words". There are also date filters: "dated" with "Anytime", "in the last week", and "on or after 12 January 2009 and before 20 January 2009". At the bottom, it says "and show 10 results sorted by relevance with summaries without grouping".

The HTML DOM tree on the right shows a table structure. A context menu is open over an input field with attributes: `input type="text" name="tx0" id="constraints0" size="45" value=""`. The menu options include "Add Attribute", "Edit as HTML", "Copy", "Copy outerHTML", "Copy selector", "Copy XPath", "Cut element", "Copy element", and "Paste element". The "Copy XPath" option is highlighted.

# Python: Selenium

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

The screenshot displays a web interface for searching through various collections. On the left, under 'ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)', there are radio buttons for 'All City of Vancouver web pages', 'City Council Meetings and Agendas', 'Park Board Meetings and Agendas', and 'Policy and Procedures'. Below this, a 'Search in:' section has three checkboxes. Further down, there are search filters: 'should contain', 'must contain', and 'must not contain', each with dropdowns for 'in the body' and 'the words'. There are also date filters: 'dated' with 'Anytime', 'in the last week', and 'on or after 12 January 2009 and before 20 January 2009'. At the bottom, 'and show' is set to '10 results', 'sorted by relevance', 'with summaries', and 'without grouping'.

On the right, a snippet of HTML code is shown. A context menu is open over an input field, with options: 'Add Attribute', 'Edit as HTML', 'Copy', 'Copy XPath', 'Copy selector', 'Copy outerHTML', 'Cut element', 'Cut element', 'Copy element', and 'Paste element'. The 'Copy XPath' option is highlighted.

- Selenium finds site elements through their attributes:
  - Name
  - ID
  - XPath
  - (Partial) Link Text
  - CSS Selector

◀ ◻ ▶ ◀ ◻ ▶ ◀ ≡ ▶ ◀ ≡ ▶ ≡ ↺ 🔍 ↻

# Python: Selenium

## Python Code

```
browser.find_element_by_name("NAME")
browser.find_element_by_id("ID")
browser.find_element_by_xpath("XPATH")
browser.find_element_by_partial_link_text("
    LINK TEXT")
browser.find_element_by_css_selector("CSS
    SELECTOR")
```

# Python: Selenium

- Once Selenium has found a web-element, it can interact with it in several ways:
  - (Double-)Click
  - Send Keys
  - Clear

## Python Code

```
.click()  
.send_keys()  
.clear()
```

# Python: Selenium

ADDITIONAL SEARCH OPTIONS (CHOOSE A COLLECTION TO SEARCH)

Search in:

- ☐ All City of Vancouver web pages
- ☐ City Council Meetings and Agendas
- ☐ Park Board Meetings and Agendas
- ☐ Policy and Procedures

should contain  in the body  the words

```

<tbody>
  <tr>
    <td> </td>
    <td valign="top">
      <select name="op0" id="constraints0"></select>
      <select name="f10" id="constraints0"></select>
      <select name="ty0" id="constraints0"></select>
      <input type="text" name="tx0" id="constraints0" size="45" value=""
        maxlength="512">
    </td>
  </tr>
</tbody>
</table>
<table width="95%"></table>
<table width="95%"></table>
<table width="95%" border="0" cellpadding="0" cellspacing="0" border="0">

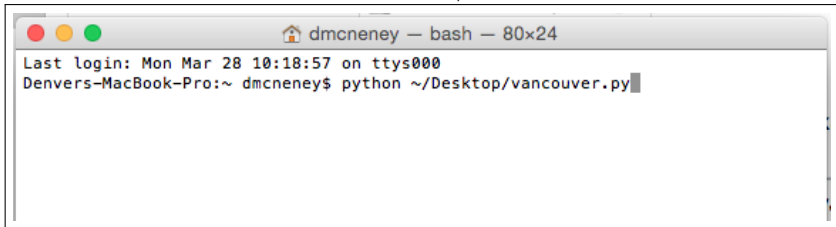
```

## Python Code

```
browser.find_element_by_name("tx0").send_keys("Environment")
```

# Python: Selenium

To run Python Script, open Terminal/Command Prompt:



A screenshot of a macOS Terminal window. The title bar shows a home icon, the username 'dmcneney', and the shell 'bash' with window dimensions '80x24'. The terminal text shows the last login as 'Mon Mar 28 10:18:57 on ttys000' and the current command prompt 'Denvers-MacBook-Pro:~ dmcneney\$ python ~/Desktop/vancouver.py' with a cursor at the end of the command.

```
dmcneney — bash — 80x24
Last login: Mon Mar 28 10:18:57 on ttys000
Denvers-MacBook-Pro:~ dmcneney$ python ~/Desktop/vancouver.py
```



# Lab Exercise

Open “vancouver2.py in a Text Editor and create a script that scrapes Meeting Minutes from the City of Vancouver website for a search term and dates of your choosing.

## Python Code

```
browser.find_element_by_name("NAME")
browser.find_element_by_id("ID")
browser.find_element_by_xpath("XPATH")
browser.find_element_by_partial_link_text("
    LINK TEXT")

.click()
.sendKeys()
.clear()
```

# Webscraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

# Webscraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

# Web scraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

# Web scraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

# Web scraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

# Web scraping: Don't Be a Jerk

- Always check to see if data available for download in structured format from website/database
- Don't crush sites:
  - Wait between data requests
  - Don't over-parallelize
  - Only download what you need. (Don't scrape just to scrape)
- Be aware of terms and conditions and what your scraping looks like to administrators.

2.2 Use of the Online Services via mechanical, programmatic, robotic, scripted or any other automated means is strictly prohibited. Unless otherwise agreed to by LN in writing, use of the Online Services is permitted only via manually conducted, discrete, individual search and retrieval activities.