# **Web scrapping**

# BCIT notes

## Web Scraping

This section offers a foundation in web scraping. Web scraping is the process of extracting content from web sites.

**Note: Please be careful. Most sites will not permit public or commercial use of scraped content without permission. If you are making scraped data public or if you are using the data for competitive or commercial gain then please obtain approval from the provider.**

Web scraping requires a creative approached to text parsing. Even experienced coders can struggle with the countless ways needed to extract real content from mountains of text. Getting comfortable parsing text takes practice. Parsing raw content will always be a continuous source of new and unique challenges. The experience gained though will definitely strengthen your coding skill. You have enough skill now to get started so I encourage you to practice.

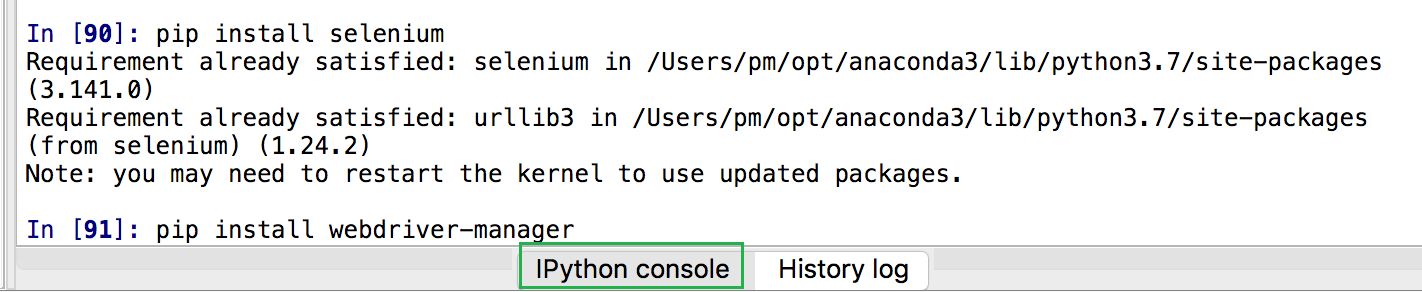
### Selenium

There are several ways to conduct web scraping with Python. I have chosen selenium for this course because it offers robust and simple mechanisms for scraping. Selenium is also a well-established and well-recognized framework in the developer community. You can easily find documentation for Selenium such as at:

<https://selenium-python.readthedocs.io/locating-elements.html>

To install Selenium and webdriver-manager run the following two commands in the IPython console of Spyder. You can also run these commands in the terminal window within PyCharm.

|  |
| --- |
| pip install selenium  pip install webdriver-manager |



Example : Installing WebDriver

This example invokes an install of the ChromeDriver. You will need to have Chrome installed on your computer for this to work. To get ChromeDriver go to

<https://chromedriver.chromium.org/> and get the latest stable build. Save it in a folder that you can find easily. Replace the path to chromedriver with the path that is displayed after the first run when the install occurs.

|  |
| --- |
| from selenium import webdriver  from webdriver\_manager.chrome import ChromeDriverManager  DRIVER\_PATH = "/Users/pm/.wdm/drivers/chromedriver/78.0.3904.70/mac64/chromedriver"  browser = None  # This loads webdriver from the local machine if it exists.  try:  browser = webdriver.Chrome(DRIVER\_PATH)  print("The path to webdriver\_manager was found.")  # If a webdriver not found error occurs it is then downloaded.  except:  print("webdriver not found. Update 'DRIVER\_PATH' with file path in the download.")  browser = webdriver.Chrome(ChromeDriverManager().install()) |

When you run a second time you will see the following output:

"The path to webdriver\_manager was found."

Exercise (1 mark)

Place the path to your chromedriver download here and keep it in a safe place. You will want to refer to it later:

|  |
| --- |
| C:\\Datasets\\chromedriver |

### Selector Gadget

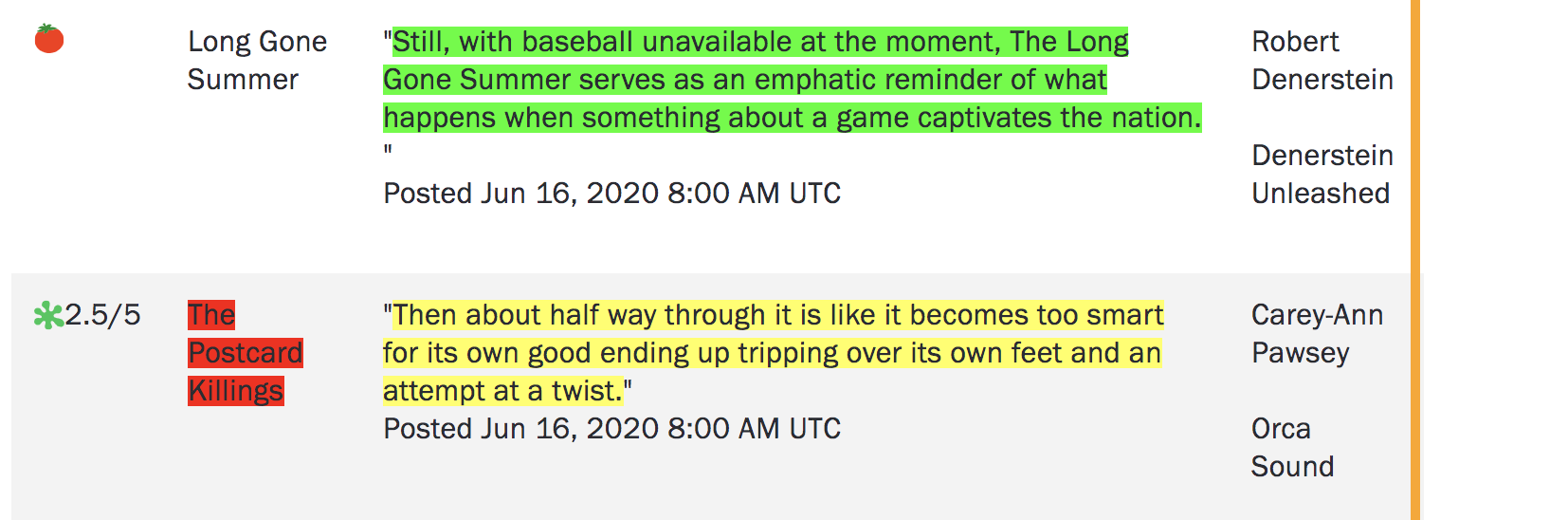
Selector Gadget is a Chrome plugin that allows us to easily obtain unique CSS selectors and XPath queries to extract targeted sections of a web site. Find the Selector Gadget plug-in for Chrome by Google and add it to your browser.

Example : Using Selector Gadget

Last day we used Python to perform sentiment analysis of movie reviews. Now let’s scrape some movie reviews. First, we can use Selector Gadget to help find a CSS selector to identify the desired movie text. In Chrome navigate to:

<https://www.rottentomatoes.com/critics/latest_reviews>

Open Selector Gadget but selecting the plugin at the top right of your Chrome tab. Then place your cursor on some movie review text. The green and yellow colours identify all of the content that is selected. Currently though we really just want the movie review text for this example. Click on the text in the *Movie* and *Critic* columns to deselect the undesired text. The undesired text should then be highlighted in red to indicate that it will not be selected.



Now that we only have the ‘Review’ text highlighted we can obtain the CSS selector. You can copy the selector from the box provided by Selector Gadget at the bottom left of the browser window. I have also pasted the selector here:

|  |
| --- |
| .critics-latest-reviews\_\_data-review .a |

### Selenium

We are going to start scraping soon. Selenium provides the following methods to locate a single element in a page. The two methods that we are going to use to locate the content identified by Selector Gadget are highlighted in green. Of course, you are welcome to experiment with the other methods as well.

find\_element\_by\_id

find\_element\_by\_name

find\_element\_by\_xpath

find\_element\_by\_link\_text

find\_element\_by\_partial\_link\_text

find\_element\_by\_tag\_name

find\_element\_by\_class\_name

find\_element\_by\_css\_selector

To find multiple elements (these methods will return a list):

find\_elements\_by\_name

find\_elements\_by\_xpath

find\_elements\_by\_link\_text

find\_elements\_by\_partial\_link\_text

find\_elements\_by\_tag\_name

find\_elements\_by\_class\_name

find\_elements\_by\_css\_selector

For more information see:

<https://selenium-python.readthedocs.io/locating-elements.html>

### BeautifulSoup

To simplify the parsing of our scraped content we will use BeautifulSoup to remove the HTML tags. BeautifulSoup reduces our workload considerably.

soup = BeautifulSoup(start, features="lxml")

print(soup.get\_text())

### Sleep

When we scrape with Selenium we are really extracting downloaded content that is delivered to our browser. To allow the browser enough time to load content before we scrape it we can use a *sleep()* function to pause the code.

# Sleep three seconds.

time.sleep(3)

Example : Scraping Rotten Tomatoes

Now that we have webdriver-manager from and the CSS selector that we obtained in , we can scrape the data. This code extracts the actual reviews from RottenTomatoes.com:

|  |
| --- |
| import time  from selenium import webdriver  from bs4 import BeautifulSoup  # driver = webdriver.Chrome(ChromeDriverManager().install())  DRIVER\_PATH = "/Users/pm/.wdm/drivers/chromedriver/78.0.3904.70/mac64/chromedriver"  URL = "https://www.rottentomatoes.com/critics/latest\_reviews"  browser = webdriver.Chrome(DRIVER\_PATH)  browser.get(URL)  # Give the browser time to load all content.  time.sleep(3)  content = browser.find\_elements\_by\_css\_selector(".critics-latest-reviews\_\_data-review .a")  for e in content:  start = e.get\_attribute('innerHTML')  # Beautiful soup allows us to remove HTML tags from our content if it exists.  soup = BeautifulSoup(start, features="lxml")  print(soup.get\_text())  print("\*\*\*") # Go to new line. |

The output shows all reviews that are currently on the page that we scraped:

|  |
| --- |
| ﻿…  \*\*\*  This movie is terrible.  \*\*\*  Dark and absorbing.  \*\*\*  What, if any, place this film will hold in the annals of cinema history is hard to say at this point, but Transsiberian is certainly a capable and engaging thriller.  \*\*\*  I can't say I didn't have fun in this movie.  \*\*\* |

Exercise (2 marks)

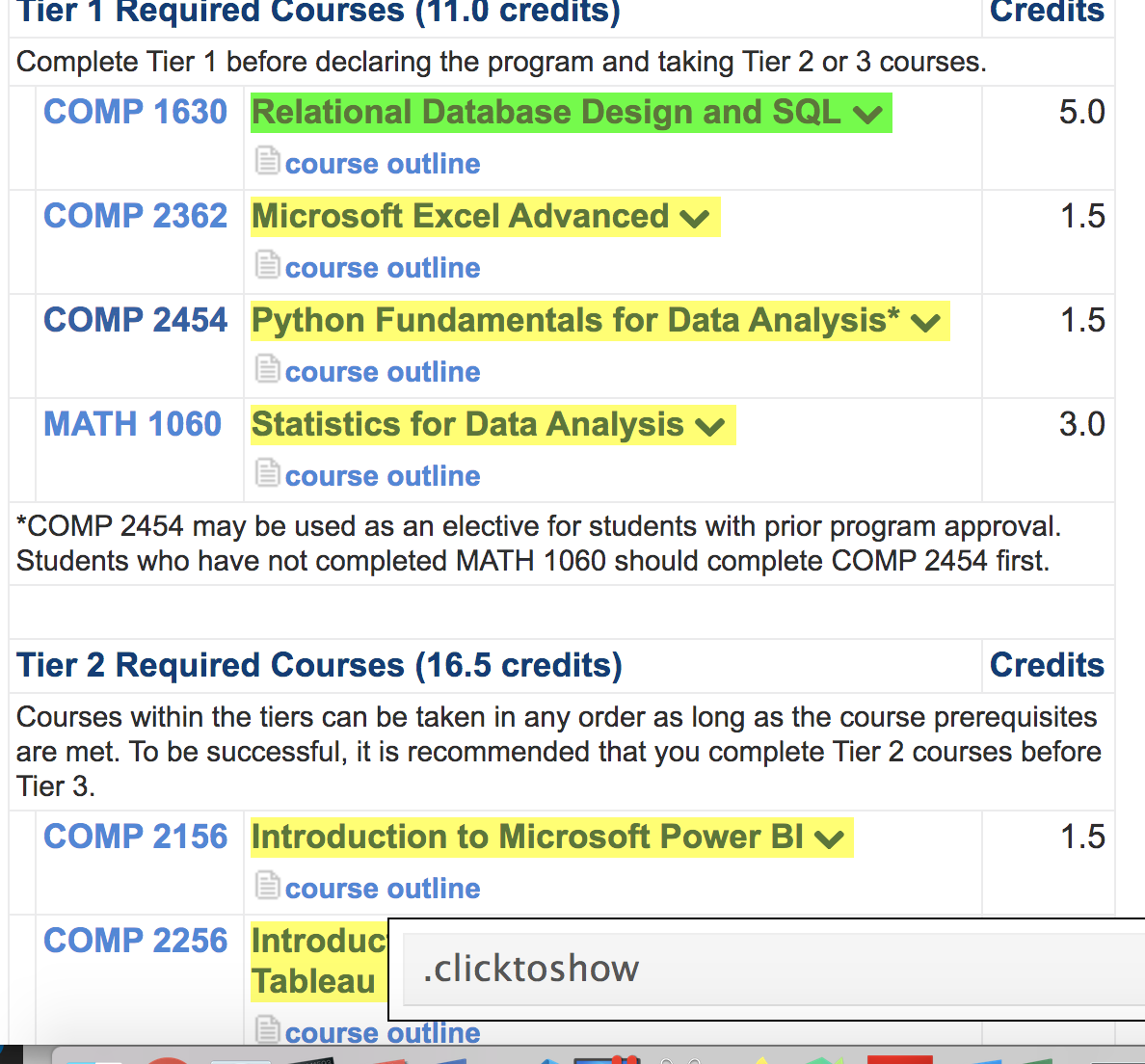
See if you can figure out how to only scrape the movie rating. Show your code here:

|  |
| --- |
| import time from selenium import webdriver from bs4 import BeautifulSoup   DRIVER\_PATH = "C:\\Datasets\\chromedriver" URL = "https://www.rottentomatoes.com/critics/latest\_reviews"  browser = webdriver.Chrome(DRIVER\_PATH) browser.get(URL)  # Give the browser time to load all content. time.sleep(3)  content = browser.find\_elements\_by\_css\_selector(".critics-latest-reviews\_\_score") for e in content:  start = e.get\_attribute('innerHTML')  # Beautiful soup allows us to remove HTML tags from our content if it exists.  soup = BeautifulSoup(start, features="lxml")  print(soup.get\_text())  print("\*\*\*") # Go to new line. |

Exercise (3 marks)

See if you can scrape the course names for the Applied Data Analytics Certificate. The URL is <https://www.bcit.ca/study/programs/5512cert#courses>

When you use Selector Gadget to obtain the content the values highlighted below will be selected.



Show your code here:

|  |
| --- |
| import time from selenium import webdriver from bs4 import BeautifulSoup   DRIVER\_PATH = "C:\\Datasets\\chromedriver" URL = "https://www.bcit.ca/study/programs/5512cert#courses"  browser = webdriver.Chrome(DRIVER\_PATH) browser.get(URL)  # Give the browser time to load all content. time.sleep(3)  content = browser.find\_elements\_by\_css\_selector(".clicktoshow") for e in content:  start = e.get\_attribute('innerHTML')  # Beautiful soup allows us to remove HTML tags from our content if it exists.  soup = BeautifulSoup(start, features="lxml")  print(soup.get\_text())  print("\*\*\*") # Go to new line. |

Show a screenshot of your output in the Python console window here:

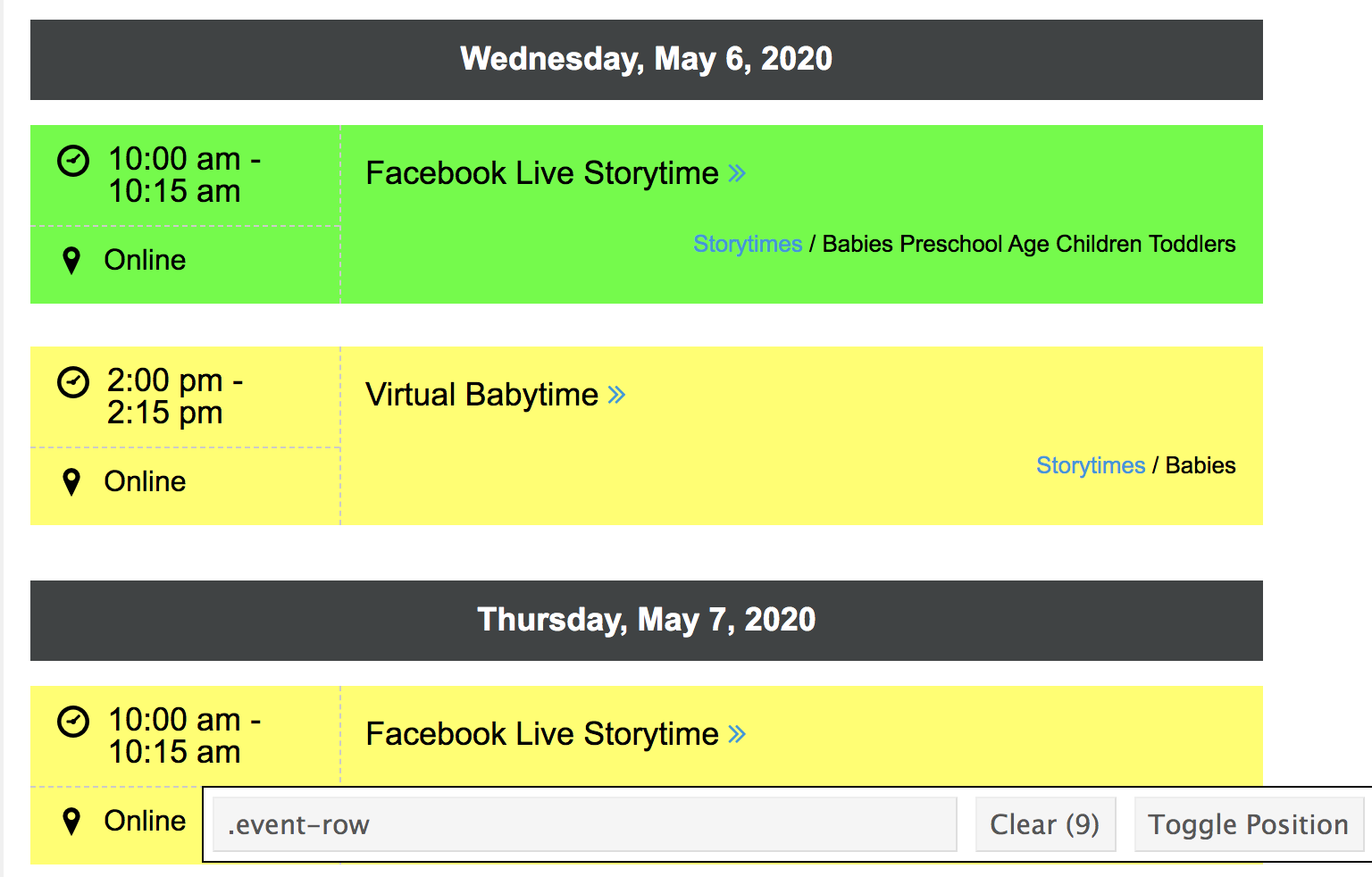
|  |
| --- |
|  |

Example : Extracting Rows of Content

Until now we have scraped neighbouring reviews and ratings separately but we really should consider extracting all related content together to ensure our data attributes are properly related.

This example demonstrates how to extract table row-by-row for events that are hosted by the Vancouver Public Libraries. We will extract the event name, date, time and location from their site for each event. To do this, in Chrome navigate to <https://vpl.bibliocommons.com/events/search/index>

Then, using Selector Gadget identify only the rows which contain the event name, date, time and location so your highlighting is similar to the following.



The CSS selector that appears should appear as:

|  |
| --- |
| .event-row |

Now try running our scraping code from with the updated hyperlink and CSS selector. There are some hidden characters in the output which causes unwanted new lines, tabs and spaces. There might be also be some hidden characters, special characters or unwanted lines and spaces. We need to be creative in the way that we extract our text.

|  |
| --- |
| ﻿Time  10:00 am -10:15 am    tv    Online event  Online event  \*\*\*    Career and Job Search Tour    Wednesday, June 17, 2020  10:30 am - 12:00 pm    tv    Online event  Online event |

Example : Custom Parsing

We want to remove the unwanted and hidden characters and we want to isolate each section of data in our output from . If we are not careful though, we could end up stripping out all spaces. The final result may end up with *allwordsthatarestucktogether*.

To avoid this problem, we can create a string of content that is separated with a common delimiter. The delimited string can later be used to split the different sections of text into an array. We can then use indexes to access the desired portions of our array. To create this delimited string:

* First strip out the new line and tab characters.
* Remove all groupings of two or more consecutive spaces.
* Inspect your output after the changes and make custom changes with the *replace()* function.

This code performs the described tasks:

# Remove hidden characters for tabs and new lines.

rawString = re.sub(r"[\n\t]\*", "", rawString)

# Replace two or more consecutive empty spaces with '\*'.

rawString = re.sub('[ ]{2,}', '\*', rawString)

# Replace old string with new string.

rawString.replace("Old String", "New String\*")

To fully implement a revised solution, start with and import the regular expression library.

|  |
| --- |
| import re |

Then replace the for-loop with this new version. The comments describe each section of code:

|  |
| --- |
| for e in content:  textContent = e.get\_attribute('innerHTML')  # Beautiful soup removes HTML tags from our content if it exists.  soup = BeautifulSoup(textContent, features="lxml")  rawString = soup.get\_text().strip()    # Remove hidden characters for tabs and new lines.  rawString = re.sub(r"[\n\t]\*", "", rawString)    # Replace two or more consecutive empty spaces with '\*'  rawString = re.sub('[ ]{2,}', '\*', rawString)    # Fine tune the results so they can be parsed.  rawString = rawString.replace("Location", "Location\*")  rawString = rawString.replace("Registration closed", "Registration closed\*")  rawString = rawString.replace("Registration required", "Registration required\*")  rawString = rawString.replace("In Progress", "\*In Progress\*")  rawString = rawString.replace("\*/\*", "/")  rawString = rawString.replace("Full\*","\*Full\*")    print(rawString)  print("\*\*\*") |

The output shows a more easily manageable series of strings.

|  |
| --- |
| Facebook Live Storytime\*Tuesday, June 16, 2020  \*10:00 am - 10:15 am\*tv\*Online eventOnline eventStorytimes/Babies\*Preschool Age Children\*Toddlers\*Storytimes/Babies\*Preschool Age Children\*Toddlers\*Time\*10:00 am -10:15 am\*tv\*Online eventOnline event  \*\*\*  ESL Conversation Practice Online\*Tuesday, June 16, 2020  \*11:00 am - 12:00 pm\*tv\*Online eventOnline eventWaitlist\*Registration required\*Meetups/Adults\*Registration required\*Meetups/Adults\*Time\*11:00 am -12:00 pm\*tv\*Online eventOnline event  \*\*\*  Career and Job Search Tour\*Tuesday, June 16, 2020  \*2:00 pm - 3:30 pm\*tv\*Online eventOnline event\*1 seat(s) remainingClasses & Workshops/Adults\*Newcomers\*1 seat(s) remainingClasses & Workshops/Adults\*Newcomers\*Time\*2:00 pm -3:30 pm\*tv\*Online eventOnline event  \*\*\*  Facebook Live Storytime\*Wednesday, June 17, 2020  \*10:00 am - 10:15 am\*tv\*Online eventOnline eventStorytimes/Babies\*Preschool Age Children\*Toddlers\*Storytimes/Babies\*Preschool Age Children\*Toddlers\*Time\*10:00 am -10:15 am\*tv\*Online eventOnline event  \*\*\*  … |

Example : Cleaning Up Our Output

Now that the scraped content is more manageable we can format it. If you examine the output from you will notice that some of the data is repeated. Some sections may still not be ideal for parsing. However, the data that we desire can be parsed easily so we won’t try to perfect the scraping code any further.

We know the first item in the delimited sentence is the event name. The second item in the sentence always appears to contain the date. The third item in delimited sentence contains the time. The location appears in two places. Each delimited sentence always contains the event location at the very end so I will use that one.

To build this example, append this code at the end of the for-loop in .

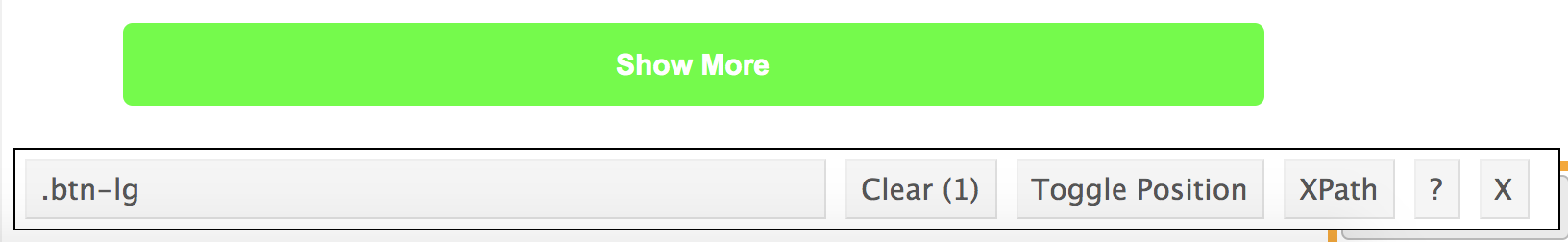
|  |
| --- |
| #print(rawString)  eventArray = rawString.split('\*')    EVENT\_NAME = 0  EVENT\_DATE = 1  EVENT\_TIME = 2  eventName = eventArray[EVENT\_NAME]  eventDate = eventArray[EVENT\_DATE].strip() # remove leading and trailing spaces  eventTime = eventArray[EVENT\_TIME].strip() # remove leading and trailing spaces  location = eventArray[len(eventArray)-1]  print("Name: " + eventName)  print("Date: " + eventDate)  print("Time: " + eventTime)  print("Location: " + location)  print("\*\*\*") |

When running the code now we are able to obtain our desired output:

|  |
| --- |
| \*\*\*  Name: Facebook Live Storytime  Date: Thursday, June 18, 2020  Time: 10:00 am - 10:15 am  Location: Online eventOnline event  \*\*\*  Career and Job Search Tour\*Thursday, June 18, 2020  \*10:30 am - 12:00 pm\*tv\*Online eventOnline event\*34 seat(s) remainingClasses & Workshops/Adults\*Newcomers\*34 seat(s) remainingClasses & Workshops/Adults\*Newcomers\*Time\*10:30 am -12:00 pm\*tv\*Online eventOnline event  \*\*\*  Name: Career and Job Search Tour  Date: Thursday, June 18, 2020  Time: 10:30 am - 12:00 pm  Location: Online eventOnline event  \*\*\*  Book Beat\*Thursday, June 18, 2020  \*2:00 pm - 2:15 pm\*tv\*Online eventOnline eventStorytimes/School Age Children\*Storytimes/School Age Children\*Time\*2:00 pm -2:15 pm\*tv\*Online eventOnline event  \*\*\*  Name: Book Beat  Date: Thursday, June 18, 2020  Time: 2:00 pm - 2:15 pm  Location: Online eventOnline event  \*\*\* |

### Clicking

If you inspect the Vancouver Public Library events page you will notice that you can load more content by clicking the “Show More” button. Other sites may allow us to see more content by providing pagination links.



Selenium allows us automate clicks to elements. Notice here that the method used to locate the element is singular because we are only identifying only one element. In other words, find\_element**s**\_by\_css\_selector is not used:

|  |
| --- |
| ﻿button = browser.**find\_element**\_by\_css\_selector(".btn-lg")  for i in range(0,20):  button.click() |

Example : Clicking

In this example we will use selenium to automate the display of much more content before we scrape it. To build this example, replace the *sleep()* instruction in with the following code. This code will click the “Show More” button twenty times before scraping begins.

|  |
| --- |
| # Give the browser time to load all content.  time.sleep(4)  button = browser.find\_element\_by\_css\_selector(".btn-lg")  for i in range(0,20):  button.click()  '''  If you see the following error increase the sleep time:  ElementClickInterceptedException: element click intercepted:  '''  print("Count: ", str(i))  time.sleep(4)  print("done loop") |

Our output shows nicely formatted content event descriptions for the week ahead.

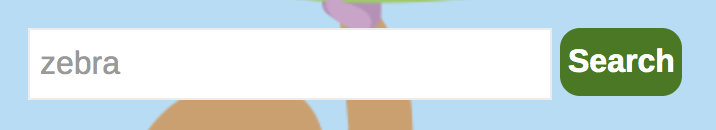
### Sending Keys

It is possible to send keys to input boxes. To automate a search for a specific topic on the AnimalFactGuide website we could find the search textbox with the CSS selector and use Selenium’s *send\_keys()* function to input the word “Zebras”.

# Find the search input.

search = browser.find\_element\_by\_css\_selector("#s")

search.send\_keys("Zebra")



After sending keys we can then click the button to trigger the search:

# Find the search button - this is only enabled when a search query is entered

button = browser.find\_element\_by\_css\_selector("#searchsubmit")

button.click() # Click the button.

Clicking on the search button brings up the search results which we can then scrape.

Example : Sending Keys

Here is the full program for scraping the headers after automatically searching on **zebra** at https://animalfactguide.com/links/" with Selenium.

Note: In this example I am searching on more than one item during each pass through the loop. You may want to do this too. When it is possible to isolate the data with a different selector, parsing is much easier in comparison to obtaining an entire row of mixed content like in the Vancouver Public Library example.

|  |
| --- |
| import time  import re  from selenium import webdriver  from bs4 import BeautifulSoup  # driver = webdriver.Chrome(ChromeDriverManager().install())  DRIVER\_PATH = "/Users/pm/.wdm/drivers/chromedriver/81.0.4044.138/mac64/chromedriver"  URL = "https://animalfactguide.com/links/"  browser = webdriver.Chrome(DRIVER\_PATH)  browser.get(URL)  time.sleep(3)  # Find the search input.  search = browser.find\_element\_by\_css\_selector("#s")  search.send\_keys("Zebra")  # Find the search button - this is only enabled when a search query is entered  button = browser.find\_element\_by\_css\_selector("#searchsubmit")  button.click() # Click the button.  # Extracts text from scraped content.  def extractText(data):  text = data.get\_attribute('innerHTML')  soup = BeautifulSoup(text, features="lxml")  content = soup.get\_text()  return content  titles = browser.find\_elements\_by\_css\_selector(".entry-title a")  descriptions = browser.find\_elements\_by\_css\_selector(".entry-summary p")  titleList = []  descriptionList = []  for i in range(0, len(titles)):  # extract title and add to list.  title = extractText(titles[i])  titleList.append(title)  # extract description and add to list.  description = extractText(descriptions[i])  descriptionList.append(description)  # Show the content.  for i in range(0, len(descriptionList)):  print("\n\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*")  print("Title: " + titleList[i])  print("Description: " + descriptionList[i]) |

The output is not in an ideal format but with further experimentation and manipulation we should be able to accurately extract the desired content:

|  |
| --- |
| \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Title: PHOTOS: Baby Zebra at Taronga Western Plains Zoo  Description: …Magner. “Zina is staying close by her mother’s side at present but does enjoy a gallop around the paddock in the morning.” Photo by Taronga Western Plains Zoo. Photo by Taronga Western Plains Zoo. In the wild, plains zebras (or common zebras) inhabit the grasslands of eastern and southern Africa. To learn more about Zina, see the Taronga Western Plains Zoo website. To learn more about zebras, see our article, Plains Zebra…. Continue reading →  \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*  Title: Grevy’s Zebra Born  Description: …g them a spot on the endangered species list. Add to that number one more as the Denver Zoo welcomes a baby male zebra. The baby, named Lakota, was born on November 27th. He is the third Grevy’s zebra to ever be born at the zoo. Grevy’s zebras differ from other zebras in their appearance. Gervy’s zebras have thinner stripes and white stomachs. They also have longer legs and larger, rounder ears. In the wild they are found in Kenya and Ethiopia. Fo… Continue reading →  … |

# iNeuron notes:

## Review scrapper with pivotal deployment

project link

C:\Users\tresa\Dropbox\data science\python\downloaded projects\reviewScrapper

Open pycharm

New project, conda env, python 3.6, new window.

Pycharm terminal: verify envmt

r.c project folder> show in explorer

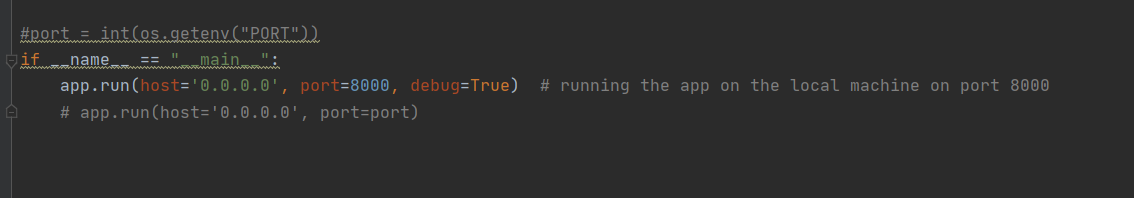
copy files from downloaded projects into our project folder.

Pycharm terminal:

pip install -r requirements.txt

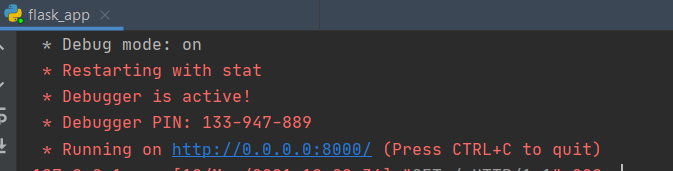
in webpage, press f12, search for elements in html code.

After finishing code, run in local machine first.



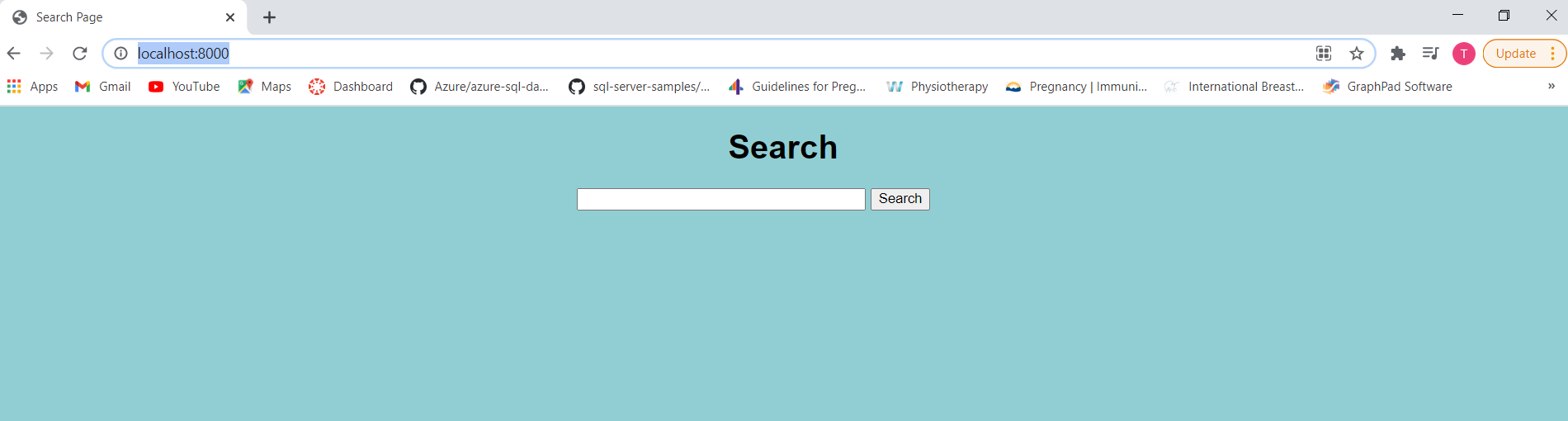
In python console, type

exit() and then run.

Click on the link, 

Then change ip address to localhost.

<http://localhost:8000/>



After successful execution in local system, we need to host it over the cloud.

Cloud options are

* Heroku
* Pivotal
* Aws
* Azure
* GCP
* Oracle
* IBM
* Alibaba

1. stop execution of application in local system.

2. create account in pivotal if you don’t have one. Vtres…

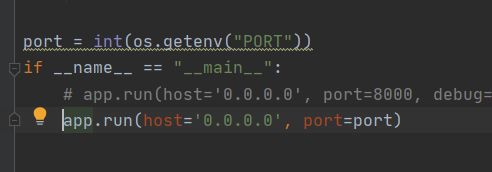
3. to connect local to cloud, download and install cli.

<https://github.com/cloudfoundry/cli/wiki/V7-CLI-Installation-Guide>

after installation, in pycharm terminal, type

cf

4. in cloud, we need to use the port provided by the environment. So modify code accordingly.



5. create requirements.txt file. This can be used to install all the packages in our server environment

pip freeze

list all the installations in the current environment.

Now copy it into a file using following command in pycharm terminal

pip freeze > requirements.txt

6. create file runtime.txt. This will contain the python version used in the project.

python-3.6.9

7. create procfile. Here specify the file the server needs to execute, and the resources needed based on load.

web: python flask\_app.py --master --processes 4 --threads 2

8. create manifest.yml

---  
applications:  
 name: reviewscrapper  
 memory: 1.5GB  
 disk\_quota: 1.5GB  
 random-route: true  
 parameters:  
 memory: 1.5GB  
 buildpack: 'python\_buildpack'

9. push code to web

cf login

if first time, specify api end point also.

cf login -a <https://api.run.pivotal.io>

choose space

cf push