A Basic Crawling Tutorial

# Introduction

Many of you may need to crawl information on the Internet for your course project. This tutorial shows you some of the basic techniques to use python packages such as ***requests*** and ***Beautifu Soup*** as well as browser’s developer tool to get the information that you want. You may find this to be helpful whatever your case might be. But if you do face technical difficulties, please email me ([lukuang@udel.edu](mailto:lukuang@udel.edu)) and I would be more than happy to help you out.

# Pre-requisites

* Python >= 3.7
* requests
* beautifulsoup4

If you have python installed, you can install the two packages using the included requirements.txt file by executing the following command in the command line:

*pip install -r requirements.txt*

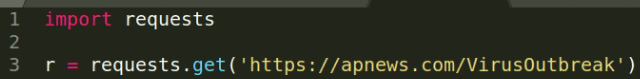
# Finding the seed/starting page

Before crawling the information, you need to identify where the information you want. This often exists as a seed/starting page. For instance, if you are interested in coronavirus news, a possibility could be [the AP page](https://apnews.com/VirusOutbreak) dedicated for that. If you want to get information from different movies, [rotten tomatoes](https://www.rottentomatoes.com/) could be a good choice. Some research by yourself will be needed to find these pages as they will be different for different projects. Then you can write code to **extract the links** to the information you want from the seed page, and then **follow each link to crawl the information** you need. Crawling coronavirus news and crawling movie information are going to be used as the two examples in the tutorial since they are likely to cover most of your use cases. Code is also included so that you run it yourself and build your crawler based on that.

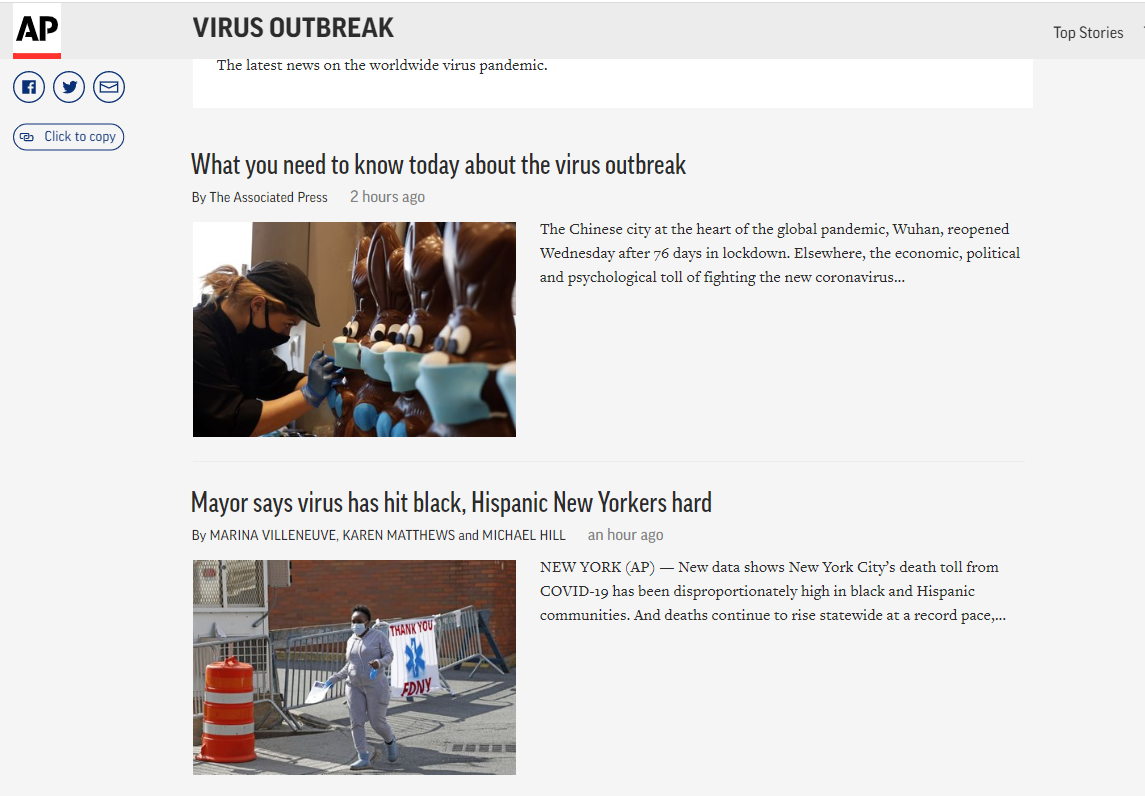
# Crawling coronavirus news

(You can reproduce the results in this section with the code ***crawl\_news.py***)

First, use requests to crawl the seed page (https://apnews.com/VirusOutbreak):



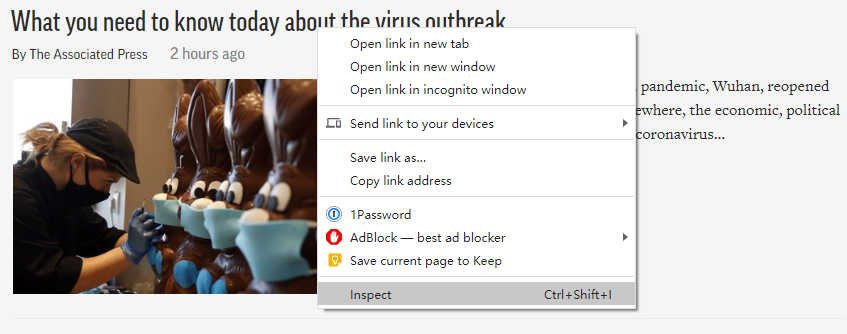
At the time of writing this tutorial, the page looks like this:



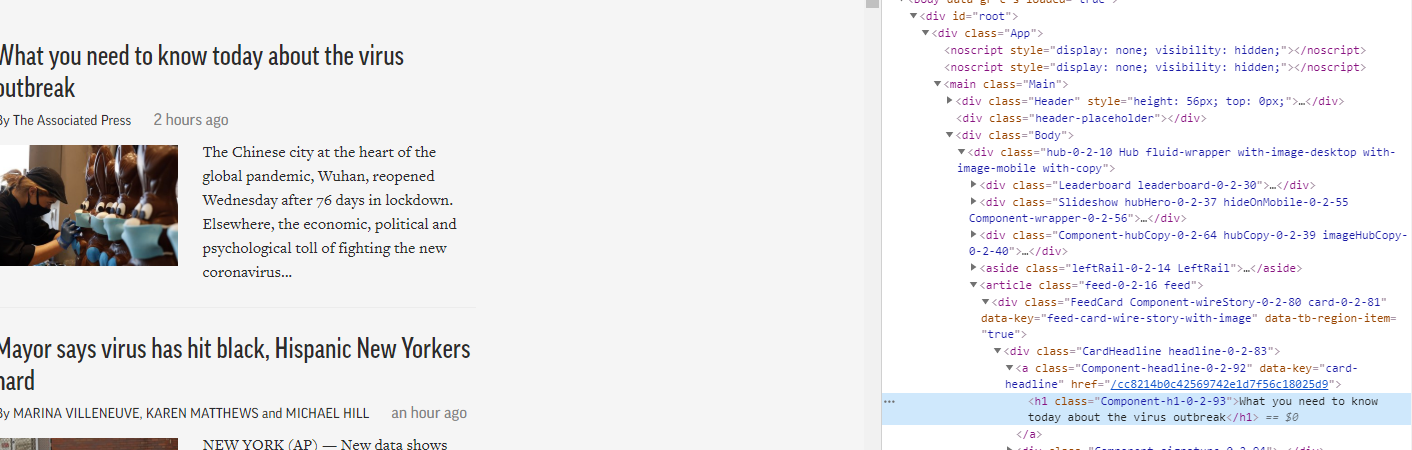
Although the content might be different when you are reading this, but the structure will likely to remain the same: there is a list of news articles that we want to crawl.

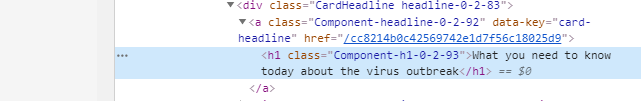
Web pages are written by a language called [Hypertext Markup Language](https://developer.mozilla.org/en-US/docs/Web/HTML) (HTML). You do not need to have a deep understanding of it to extract the information from an HTML page. It is good enough to know that a webpage consists of many HTML elements, and they are like Lego blocks. Each HTML element represents a part of a HTML, such as a link, a paragraph of text, etc. Now the objective becomes how to find the HTML elements to extract the articles’ links. In order to do that, you will need the developer tool of your browser. I use Chrome as an example and this process should be similar for different browsers.

First, it is clear that by clicking on the title of an article (e.g. “What you need to know today about the virus outbreak.”), you will be directed to the article. Therefore, the link to the article is associated with the title. Right click on the title and click on inspect:

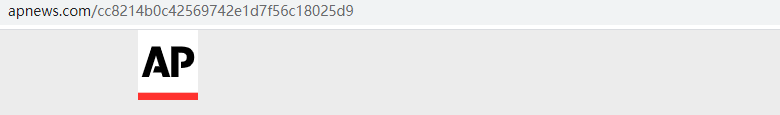


Then, you could see a window pops up on the right and the part related to the title is highlighted.





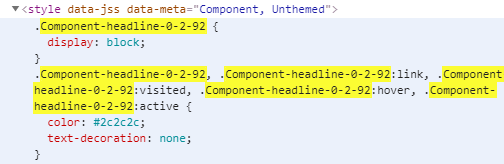
In HTML, a link is usually in a link element that starts with a tag ‘<a>’ and ends with a tag ‘</a>’. But the highlighted part is only a text element (e.g. h1) and there is no link. However, since we know that clicking on the title will direct us to the article page, it is clear that the link should not be far away. It turns out that the highlighted text part is a child of the link we are looking for as the text element is contained by the tags of link one line above and one line below it (For your project, you might also need to do a bit of search from the clickable part of the page to the actual link as well). For a link element, the “href” attribute contains the actual link. We can click on the link of href to check whether this is the link of the article. It turns out that it is. However, if you look at the url of the new page, it is not exactly the same as shown in the href.



As can be seen, there is a “apnews.com” prefix added to it. This means that as long as we can identify all the link elements for the articles, we can take the string from the href attribute, add the prefix, and we get all the article links on the seed page. The problem becomes how we identify the relevant link elements. Not all links on the seed page are article links. For instance, there is a link to AP’s Twitter page.

To identify the link elements that we need, attributes other than href will be used. I usually use ‘id’ and ‘class’ attributes since they are often used to control the appearance of an element on the webpage, and it is usually the case that the things you want from a webpage appear in a different way compared to other things on the same page. In this case, the titles of the news articles look the same and different from the Twitter icon for AP’s Twitter page. In this section, the class attribute is used. As can be seen in the screenshot of the link element, its class is set to “Component-headline-0-2-92”. Can this be used to uniquely identify **ALL** the news links? In order to confirm that, click on the link element to make sure we are interacting with the HTML of the page, use the search short cut “ctrl+F” (command + F on MAC), and search for “Component-headline-0-2-92” to double check if this value correlates uniquely with the article links and there are no missing articles using it. Therefore, we can safely use it to extract the article links.

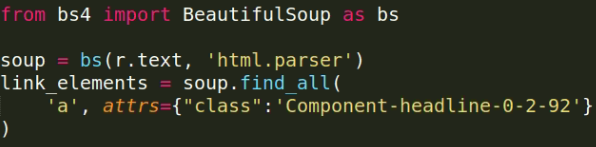
There are several things to be noted here. First, it is not always the case that a single value for an attribute can catch all the things you are looking for. Therefore, if there are things that are missing, find out the id or class attributes that can capture the them use the same procedure described above. Second, when performing search, you can see something below:



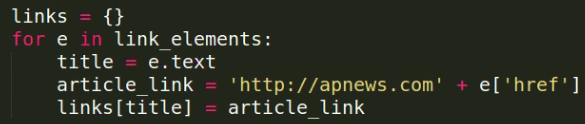
This part controls the appearance of the link (as you can see, the part is in a “style” element). When you click on it, nothing on the webpage is highlighted, which means that it does not correlates to any part of the page content. Thus, for the sake of crawling, this part can be ignored.

Another thing that is important to note is that the value of the class attribute might change depends on your system set up, such as the operating system. If the included code does not work for you, please do the element searching procedure detailed above by yourself and see if the class name is different or not on your end.

After the right class attribute value is found, the next step is to use ***Beautiful Soup*** to extract the links. ***Beautiful Soup*** is a powerful tool to parse HTML data. Using the following code to extract the link elements:

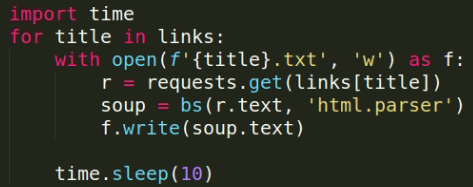


In this code, “r” is the output of the crawl of the seed page that we get from the earlier code, and “r.text” is the full HTML of the page. ***Beautiful Soup*** is used to parse the HTML and we use the function “find\_all” to find all link elements (“a”) that has the attribute “class” equals to value we find previously. We then extract the links and the article titles from the link elements:



In this code “e.text” returns the text part of the element, which is the title. On the other hand, we use the value of the attribute “href” and add the prefix to it to create the article links. Please note that the “http://” at the beginning. This is needed for the ***requests*** to work properly.

The final step is to use the extracted links to crawl the articles:

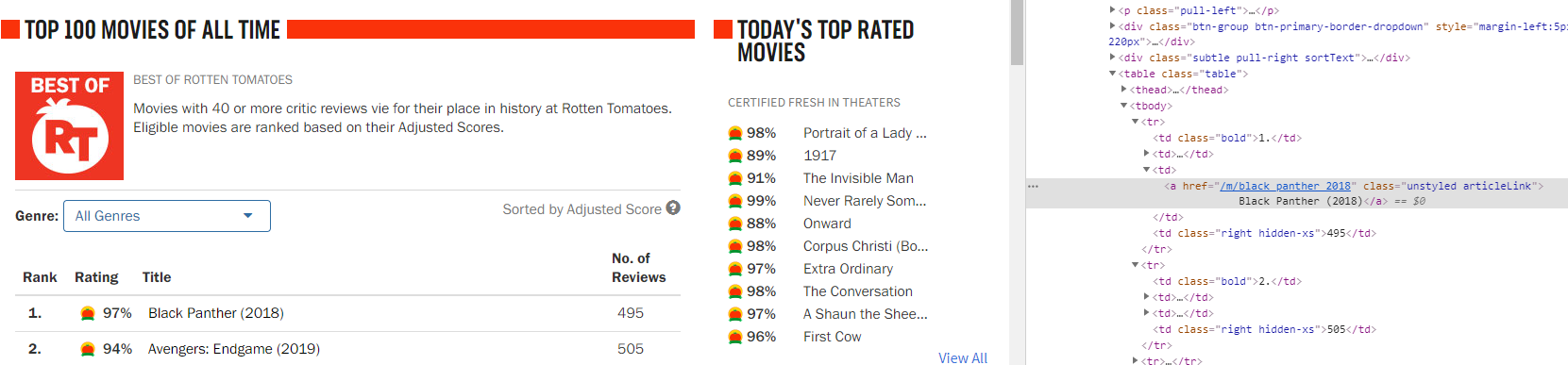


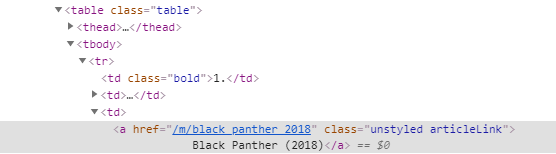
The article titles are used as file names and the links are used to crawl the HTML of the article through ***requests***. ***Beautiful soup*** is then used to parse the HTML and get only the text content out since that is the only part that we are interested in. The “time.sleep(10)” line, as its name suggests, pauses 10 seconds at the end of each crawl iteration. It is used to avoid sending to many requests to a website in a short period of time, which may require too much resources from the website server and they may decide to ban you from visit the website temporary.

# Crawling movie information

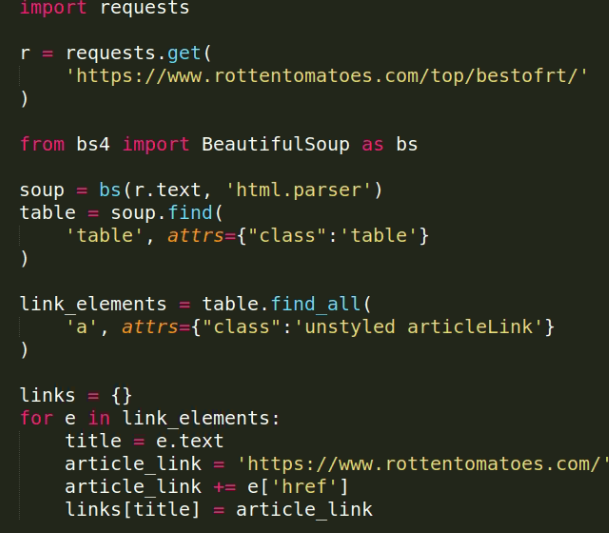
(You can reproduce the results in this section with the code ***crawl\_movie.py***)

In this section, we will go through how to crawl information about movies on Rotten Tomatoes. More specifically, we will crawl the synopses of [the top 100 movies of all time](https://www.rottentomatoes.com/top/bestofrt/) that are ranked by the website. Similar to the previous section, we first visit the website and use the developer tool to identify the id or class attributes that can be used to identify the links to the 100 movies.



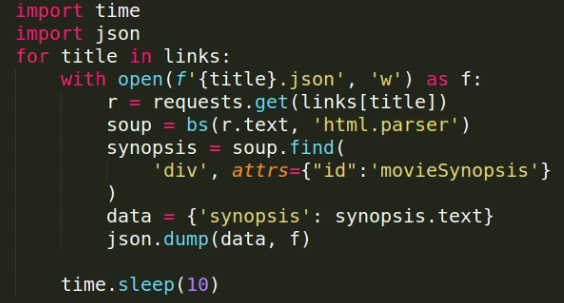


Different from the previous section, use the inspect tool on the first movie “Black [Panther (2018)](https://www.rottentomatoes.com/m/black_panther_2018)” take us directly to the link element. Moreover, there is a space in the class attribute. HTML uses space to separate multiple values for an attribute. This means that the link has two class values: “unstyled” and “articleLink”. Since all the 100 movies have their class set to exactly “unstyled articleLink”, we can directly use this string to match the class attribute. However, there are other links that are not the top 100 movies but have the same class value. In order to avoid extracting unwanted links, we need to, starting from the links that we want, trace upwards to their parent, or parent of parent, etc. to find the parent that can be uniquely identified from its element type or class/id attribute values. This parent can be extracted first so that the movie links can be subsequently taken out. After such process, the element “table” and class value “table” (shown on the top of the screenshot) fits the requirements. Below is the code to crawl the seed page and use the table as well as link elements to extract the movie links:



Most of it is similar to the code in previous section and the only major difference is that the table element is extracted first using function “find”. “final\_all” is not used since there is only one such table element fits the criteria. The link elements then are extracted with the help of the class attribute value from the table. Appropriate prefix is added to the links.

The final step is to use the links to crawl the information of the movies that we need. Unlike news articles, the information of the movie is structured, and you might only need some parts of it. For the purpose of the tutorial, let us assume that we only need the synopsis of the movies. By using the developer tool, this information is contained in an element that as “movieSynopsis” as the value of its id. Therefore, it is used to extract the synopsis. The code of crawling the movie page HTML, extract the synopsis, and save the data is shown below:



This is also similar to previous section. The two major differences are that only synopsis is saved instead of the full text of the page, and the data is saved with json format. Json is used since you might also need to store information besides synopsis, such as ratings and cast. Json is a good choice if you want to store such structured data and it is easy to use.

# Final Notes

The two examples illustrate how you could build your crawler. However, different websites have different HTML structure and sometimes the structure is so complex that it is not easy to write the ***Beautiful Soup*** code to extract the part of the page that you want. If you face such difficulties, please let me know and I will help you with that.