



Published in Analytics Vidhya



Vandany Lubis (Danny)

Follow

Oct 2, 2021 · 10 min read · Listen



Save



How to Scrape Data from a Website using Python for Beginner

In the data science field, we are always dependent on data. There are many ways data can be collected. One of the many sources is data from websites. Websites are can be also a secondary source of data, there are many examples of such websites, for instance: data aggregation sites (ex. [Worldometers](#)), news websites (ex. CNBC), social media (ex. Twitter), e-commerce (ex. Shopee), and so on. Those websites are useful and they are provided the data necessary for data science projects.



Fig.1 Web Scrapping (Source: Data Science Central, 2020)

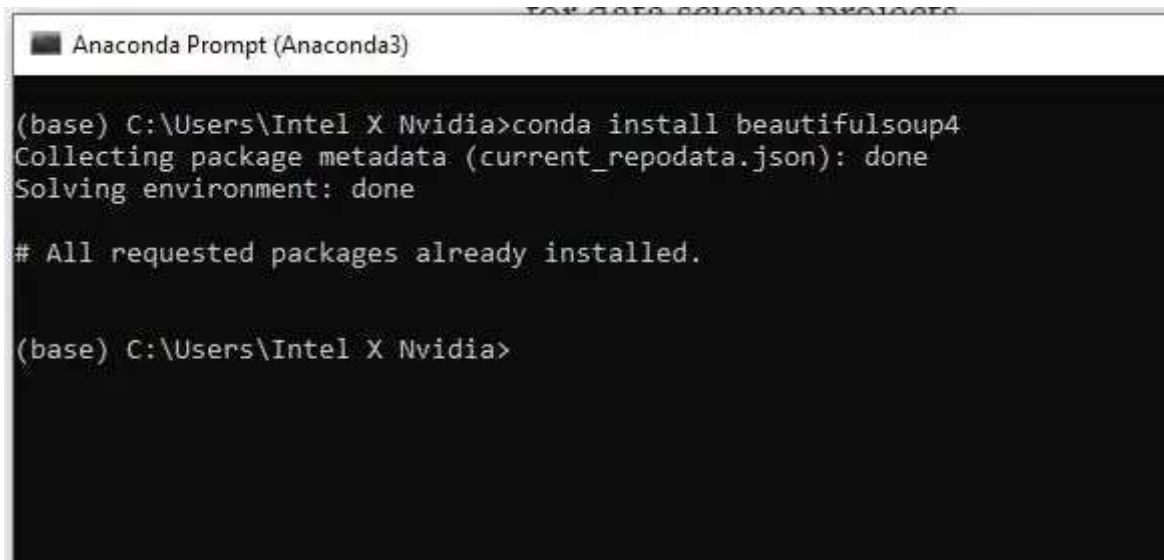
But how to collect the data? We cannot copy and paste manually one by one using our hands right? So the solution is using Python. In Python, there is a powerful library called BeautifulSoup and Selenium. Both of them are often used by data scientists to collect multiple formats of data. In this section, we will introduce to BeautifulSoup first.

STEP 1. INSTALLING LIBRARIES

First of all, we need to install required libraries, such as:

1. BeautifulSoup4
2. Requests
3. pandas
4. lxml

To install a library you can simply install it using **pip install [library name]** or **conda install [library name]** if you are using Anaconda Prompt. First, we want to install BeautifulSoup4.

A screenshot of the Anaconda Prompt (Anaconda3) window. The terminal shows the command 'conda install beautifulsoup4' being executed. The output indicates that the package metadata was collected successfully and the environment was solved. A message states that all requested packages are already installed. The prompt then returns to the command line.

```
Anaconda Prompt (Anaconda3)

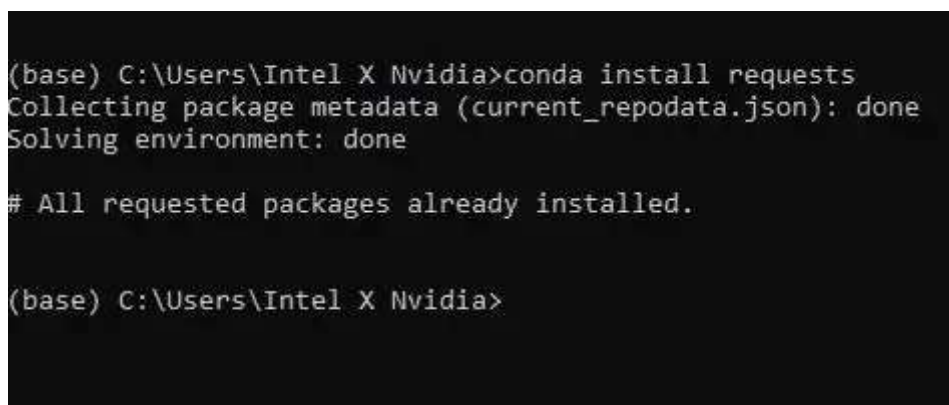
(base) C:\Users\Intel X Nvidia>conda install beautifulsoup4
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

(base) C:\Users\Intel X Nvidia>
```

Fig.2 Installing bs4

After we install **BeautifulSoup4 (bs4)** the next thing we can do is installing '**requests**'. '**Requests**' is the library whose task is asking permission from the hosting server if we want to fetch data from their website. Then, we also need to install **pandas** to create a dataframe and **lxml** to change the HTML format into a Python-friendly format.

A screenshot of the Anaconda Prompt (Anaconda3) window. The terminal shows the command 'conda install requests' being executed. The output indicates that the package metadata was collected successfully and the environment was solved. A message states that all requested packages are already installed. The prompt then returns to the command line.

```
(base) C:\Users\Intel X Nvidia>conda install requests
Collecting package metadata (current_repodata.json): done
Solving environment: done

# All requested packages already installed.

(base) C:\Users\Intel X Nvidia>
```

Fig.3 Installing 'requests'

STEP 2. IMPORT LIBRARIES

So after the important libraries are already installed the next thing we can do is open your favorite environment. In this tutorial, we suggest you to using Spyder 4.2.5 because for some steps we will encounter long output so Spyder is more comfortable to use than Jupyter Notebook.

Okay, once we open the Spyder the next thing we can do is importing the required library:

```
# Import library
from bs4 import BeautifulSoup
import requests
```

STEP 3. SELECTING PAGE

In this project, we will use webscraper.io. Since this website is made with HTML so it is easier to read the code and suitable for beginners. Now, in this section we will scrape this page:

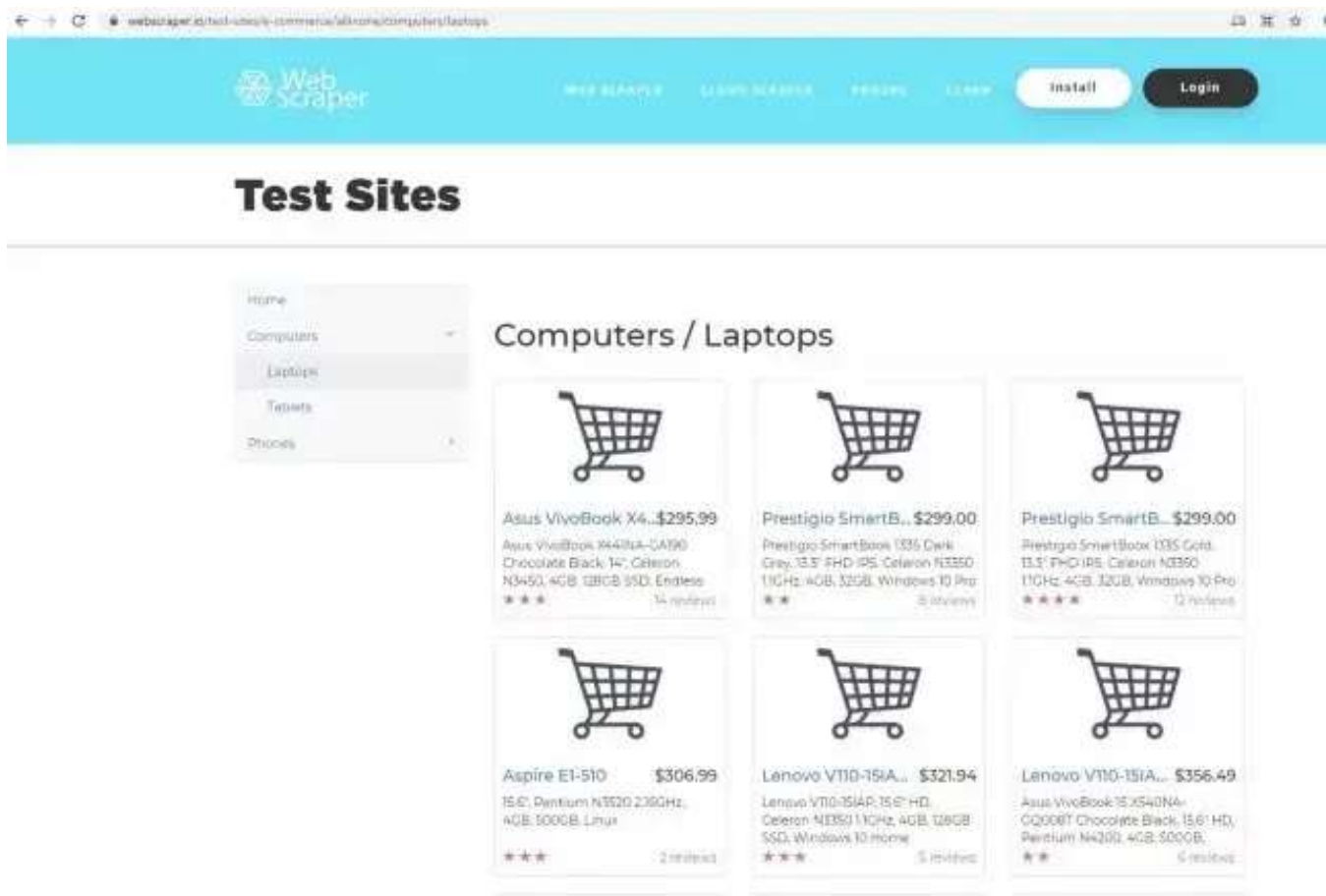


Fig.4 Webscraper.io

This page is a prototype of an e-Commerce website. In this project, we will scrape the data of computers and laptops, such as product name, price, description, and reviews.

STEP 4. REQUEST PERMISSION

After we select what page we want to scrape, now we can copy the page's URL and use **request** to ask permission from the hosting server that we want to fetch data from their site.

```
# Define URL
url = 'https://webscraper.io/test-sites/e-commerce/allinone/computers/laptops'

# Ask hosting server to fetch url
requests.get(url)
```

If the output is <Response [200]> so that means the server allows us to collect data from their website. To verify we can define the request.get function into text.

```
pages = requests.get(url)
pages.text
```

If you run this code the output will be the messy script code where it is not Python friendly. We need to use a parser so we can make it more readable.

```
# parser-lxml = Change html to Python friendly format
soup = BeautifulSoup(pages.text, 'lxml')
soup
```



Fig.5 Before and After using Parse

STEP 5. INSPECT ELEMENT

For every web scrapping project, we recommend you use Google Chrome because it is very handy and easy to use. Now we will learn how to inspect the script code of a website using Chrome. First, you need to right-click the page that you want to inspect, then click **Inspect** after that you will see this:

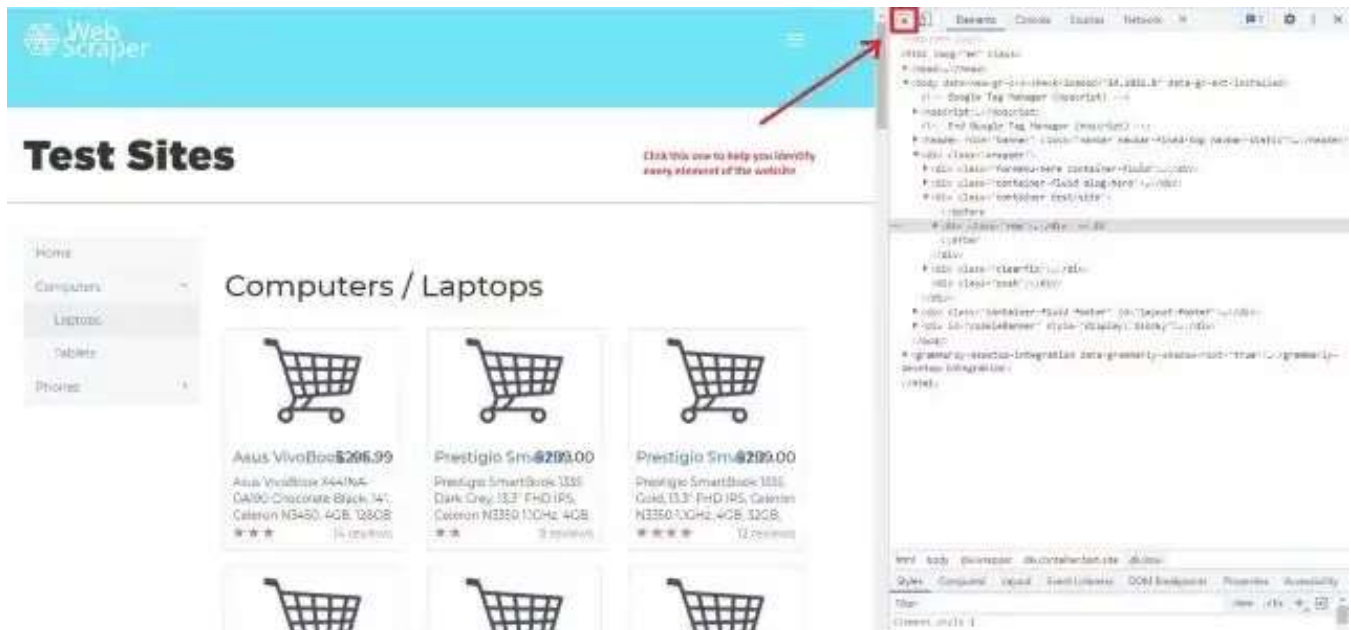


Fig.6 Inspect Element

Then you can click **Select an element in the page to inspect it**. After you click it you will notice if you move the cursor to each element of the page, the elements menu will give you the information of what is the script of the selected element.

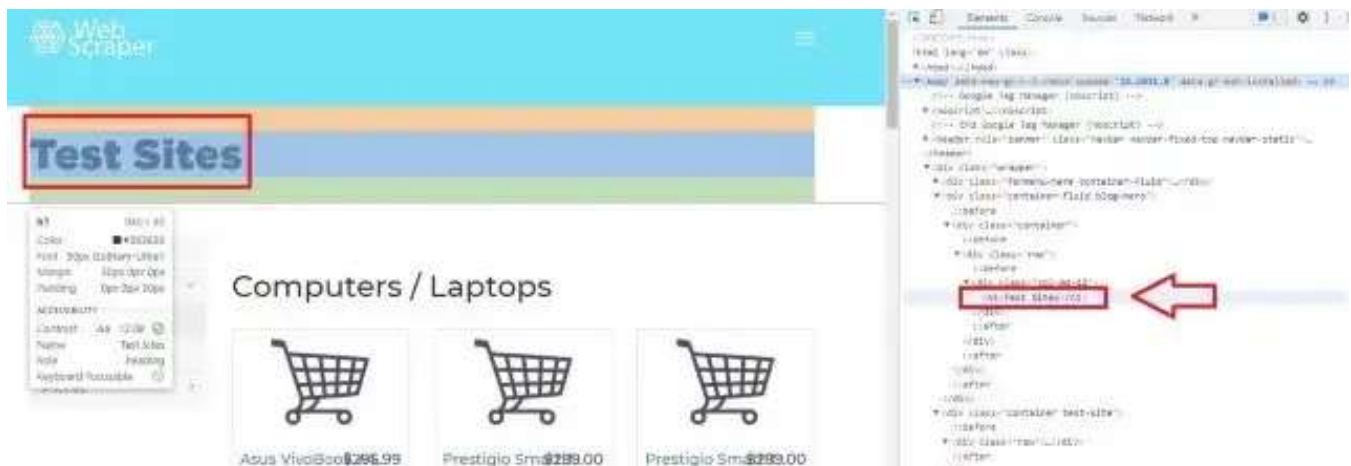


Fig.7 Example

For example, if we move the cursor to **Test Sites** the element will inform you that the **Test Sites** are located in the tag **h1**. In Python, if you want to inspect the elements of a website you can call the tags. The characteristic of tags is it always has **<** as a prefix and it is often purple-colored.

STEP 6. ACCESSING TAGS

For example, we want to access the element of **h1** using Python, we can simply type this:

```
# Access h1 tag
soup.h1
```

The output will be:

```
soup.h1
Out[11]: <h1>Test Sites</h1>
```

Not only the single line tag you can also access class tags, for example:

```
# Access header tag
soup.header

# Access div tag
soup.div
```

Remember to define **soup** before, because it is important to change HTML into a Python-friendly format.

You can access a specific tag from nested tags. **The nested tags mean tags inside tags.** For example tag **<p>** is located inside tag **<header>**. But when you access a specific tag from **<header>**, Python always showing results from the first index. We will learn how to access multiple tags from nested tags later.

```
# Access string from nested tags
soup.header.p
```

Output:

```
soup.header.p
Out[10]: <p>Web Scraper</p>
```

You can also access a string from a specific tag from nested tags. You can simply add a **string** to the code.

```
# Access string from nested tags
soup.header.p
soup.header.p.string
```

Output:

```
soup.header.p
soup.header.p.string
Out[12]: 'Web Scraper'
```

Okay, now we understand how to access tags from the nested tags. So the next thing we can learn is how to access attributes in tags. To do this we can use **attrs**, this is the feature of the BeautifulSoup. The output of **attrs** is a dictionary.

```
# Access 'a' tag in <header>
a_start = soup.header.a
a_start

# Access only the attributes using attrs
a_start.attrs
```

Output:

```
Out[16]:
{'data-toggle': 'collapse-side',
 'data-target': '.side-collapse',
 'data-target-2': '.side-collapse-container'}
```

We can access a specific attribute, remember Python treats the attribute as a dictionary, so we reckon **data-toggle**, **data-target**, and **data-target-2** as a **key**. Now,

for example, we will access **'data-target':**

```
a_start['data-target']
```

Output:

```
a_start['data-target']
Out[17]: '.side-collapse'
```

We can also add a new attribute, remember the changes is only affect the website locally, not the website itself globally.

```
a_start['new-attribute'] = 'This is the new attribute'
a_start.attrs
a_start
```

Output:

```
a_start['new-attribute'] = 'This is the new attribute'
a_start.attrs
a_start
Out[18]:
<a data-target=".side-collapse" data-target-2=".side-collapse-
container" data-toggle="collapse-side" new-attribute="This is the
new attribute">
<button aria-controls="navbar" aria-expanded="false" class="navbar-
toggle pull-right collapsed" data-target="#navbar" data-target-
2=".side-collapse-container" data-target-3=".side-collapse" data-
toggle="collapse" type="button">
...
</a>
```

STEP 7. ACCESSING SPECIFIC ATTRIBUTES OF TAGS

We have learned that in one tag it could be many sub-tags, for example, If we run **soup.header.div** there are many sub-tags under the <div>, remember we only call the <div> under the <header> so other tag under <header> will not included.

Output:

```
soup.header.div
Out[26]:
<div class="container">
<div class="navbar-header">
<a data-target=".side-collapse" data-target-2=".side-collapse-
container" data-toggle="collapse-side" new-attribute="This is the
new attribute">
<button aria-controls="navbar" aria-expanded="false" class="navbar-
toggle pull-right collapsed" data-target="#navbar" data-target-
2=".side-collapse-container" data-target-3=".side-collapse" data-
toggle="collapse" type="button">
...
</div>
```

So as we can see there are many attributes under one tag, the question is how to access only the attribute that we want. In BeautifulSoup there is a function called **'find'** and **'find_all'**. To make it clearer we will show you how to use both functions and what is different from each other. As an example, we will find the price of each product. To identify what is the code of price you just simply move your cursor to the price indicator.

After we move the cursor, we can identify that the price is located on tag **h4** and class **pull-right price**

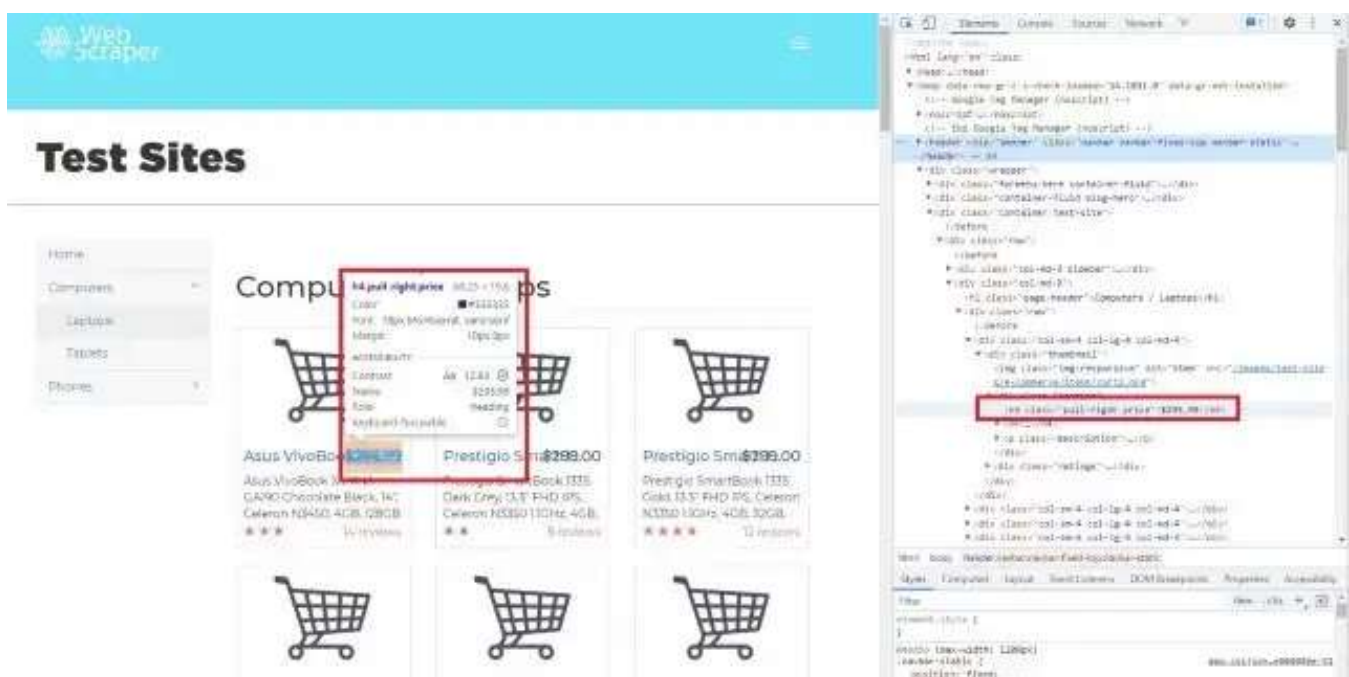


Fig.8 Price Indicator

Now we want to search the string of h4 using find:

```
# Searching specific attributes of tags
soup.find('h4', class_='pull-right price')
```

Output:

```
Out[28]: <h4 class="pull-right price">$295.99</h4>
```

As we can see \$295.99 is the attribute (string) of the h4. But how if we use find_all.

```
# Using find_all
soup.find_all('h4', class_='pull-right price')
```

Output:

```
Out[29]:
[<h4 class="pull-right price">$295.99</h4>,
 <h4 class="pull-right price">$299.00</h4>,
 <h4 class="pull-right price">$299.00</h4>,
 <h4 class="pull-right price">$306.99</h4>,
 <h4 class="pull-right price">$321.94</h4>,
 <h4 class="pull-right price">$356.49</h4>,
 ...
 </h4>]
```

Did you notice the difference between **find** and **find_all**?

Yes, you are correct, **find** is to use if you want to search specific attributes because it only generates one result. However, if you want to scrape data in large quantities (for example prices, product name, description, etc.) using **find_all** is the right way.

We can also slice the results of find_all, for example in this case we want to see only index 3 to 5.

```
# Slicing the results of find_all
soup.find_all('h4', class_='pull-right price')[2:5]
```

Output:

```
Out[32]:
[<h4 class="pull-right price">$299.00</h4>,
 <h4 class="pull-right price">$306.99</h4>,
 <h4 class="pull-right price">$321.94</h4>]
```

[!] Remember in Python index start from 0 and the last index is not included

STEP 8. USING FILTER

We can also find multiple tags:

```
# Using filter to find multiple tags
soup.find_all(['h4', 'a', 'p'])
soup.find_all(['header', 'div'])
soup.find_all(id = True) # class and id are special attribute so it
can be written like this
soup.find_all(class_ = True)
```

For additional information because class and id are special attributes so you can write class_ and id instead of 'class' or 'id'

By using a filter we can collect the data that we want from the website, in this case, we want to collect name, price, reviews, and descriptions. So we need to define the variables first.

```
# Filter by name
name = soup.find_all('a', class_='title')

# Filter by price
price = soup.find_all('h4', class_ = 'pull-right price')

# Filter by reviews
reviews = soup.find_all('p', class_ = 'pull-right')
```

```
# Filter by description
description = soup.find_all('p', class_ = 'description')
```

Output by name:

```
[<a class="title" href="/test-sites/e-commerce/allinone/product/545"
title="Asus VivoBook X441NA-GA190">Asus VivoBook X4...</a>,
  <a class="title" href="/test-sites/e-commerce/allinone/product/546"
title="Prestigio SmartBook 133S Dark Grey">Prestigio SmartB...</a>,
  <a class="title" href="/test-sites/e-commerce/allinone/product/547"
title="Prestigio SmartBook 133S Gold">Prestigio SmartB...</a>,
  ...
</a>]
```

Output by price:

```
[<h4 class="pull-right price">$295.99</h4>,
  <h4 class="pull-right price">$299.00</h4>,
  <h4 class="pull-right price">$299.00</h4>,
  <h4 class="pull-right price">$306.99</h4>,
  ...
</h4>]
```

Output by reviews:

```
[<p class="pull-right">14 reviews</p>,
  <p class="pull-right">8 reviews</p>,
  <p class="pull-right">12 reviews</p>,
  <p class="pull-right">2 reviews</p>,
  ...
</p>]
```

Output by description:

```
[<p class="description">Asus VivoBook X441NA-GA190 Chocolate Black,
14", Celeron N3450, 4GB, 128GB SSD, Endless OS, ENG kbd</p>,
  <p class="description">Prestigio SmartBook 133S Dark Grey, 13.3"
FHD IPS, Celeron N3350 1.1GHz, 4GB, 32GB, Windows 10 Pro + Office
365 1 gadam</p>,
  <p class="description">Prestigio SmartBook 133S Gold, 13.3" FHD
```

```
IPS, Celeron N3350 1.1GHz, 4GB, 32GB, Windows 10 Pro + Office 365 1
gadam</p>,
...
</p>]
```

STEP 9. CLEANING OUTPUT RESULTS FROM FILTER

As we can see the results of the output are still in the HTML format so we need to clean this and get only the string. To do this we can simply use the **text** function.

Text can be used to filter strings from the HTML code however we need to define a new variable, for example:

```
# Try to call price
price1 = soup.find('h4', class_ = 'pull-right price')
price1.text
```

Result:

```
Out[55]: '$295.99'
```

As we can see the output result are only the string from the code, but this is not enough. In the next step, we will learn how to scrape all the strings and make them into the list.

STEP 10. CREATE A FOR LOOP TO MAKE STRING INTO LIST

This is the interesting part, to make all the strings into the list we need to create a for a loop.

```
# Create for loop to make string from find_all list
product_name_list = []
for i in name:
    name = i.text
    product_name_list.append(name)

price_list = []
for i in price:
    price = i.text
    price_list.append(price)
```

```
review_list = []
for i in reviews:
    rev = i.text
    review_list.append(rev)

description_list = []
for i in description:
    desc = i.text
    description_list.append(desc)
```

STEP 11. CREATE A DATAFRAME FROM LIST

Okay after we created a for loop and all the strings were added to the lists, so the last step is to create a dataframe from the list. To create a dataframe we need to import pandas.

```
# Create dataframe
# Import library
import pandas as pd

tabel = pd.DataFrame({'Product Name':product_name_list,
    'Price': price_list,
    'Reviews':review_list,
    'Description':description_list})
```

After the dataframe is created now we can use this data to make a data science project, we can put it into machine learning, getting valuable insight from it, or anything.

tabel - DataFrame				
Index	Product Name	Price	Reviews	Description
0	Asus VivoBook X4...	\$295.99	14 reviews	Asus VivoBook X441NA-GA190 Chocolate Black,...
1	Prestigio SmartB...	\$299.00	8 reviews	Prestigio SmartBook 1335 Dark Grey, 13.3" F...
2	Prestigio SmartB...	\$299.00	12 reviews	Prestigio SmartBook 1335 Gold, 13.3" FHD IP...
3	Aspire E1-510	\$306.99	2 reviews	15.6", Pentium N3520 2.16GHz, 4GB, 500GB, Linux
4	Lenovo V110-15IA...	\$321.94	5 reviews	Lenovo V110-15IAP, 15.6" HD, Celeron N3350 1.1GHz, 4GB, 128GB SSD, Windows 10 Home
5	Lenovo V110-15IA...	\$356.49	6 reviews	Asus VivoBook 15 X540NA-GQ008T Chocolate Bl...
6	Hewlett Packard...	\$364.46	12 reviews	Hewlett Packard 250 G6 Dark Ash Silver, 15.6" HD, Core i3-6006U, 4GB, 128GB SSD, Windows 10 Home
7	Acer Aspire 3 A3...	\$372.70	2 reviews	Acer Aspire 3 A315-31 Black, 15.6" HD, Celeron N3350, 4GB, 128GB SSD, Windows 10 Home
8	Acer Aspire A315...	\$379.94	0 reviews	Acer Aspire A315-31-C33J Black 15.6", HD, Core i3-6006U, 4GB, 128GB SSD, Windows 10 Home
9	Acer Aspire E51-...	\$379.95	9 reviews	Acer Aspire E51-572 Black, 15.6" HD, Core i3-6006U, 4GB, 128GB SSD, Linux
10	Acer Aspire 3 A3...	\$391.48	10 reviews	Acer Aspire 3 A315-31 Black, 15.6" HD, Pentium N3520, 4GB, 128GB SSD, Windows 10 Home
11	Acer Aspire 3 A3...	\$393.88	9 reviews	Acer Aspire 3 A315-21, 15.6", AMD A4-9120, 4GB, 128GB SSD, Linux
12	Asus VivoBook Ma...	\$399.00	4 reviews	Asus VivoBook Max X541NA-GQ041 Black Chocol...
13	Asus VivoBook E5...	\$399.99	3 reviews	Asus VivoBook E502NA-GQ022T Dark Blue, 15.6" HD, Core i3-6006U, 4GB, 128GB SSD, Windows 10 Home
14	Lenovo ThinkPad...	\$404.23	12 reviews	Lenovo ThinkPad E31-80, 13.3" HD, Celeron 3120, 4GB, 128GB SSD, Windows 10 Home
15	Acer Aspire 3 A3...	\$408.98	10 reviews	Acer Aspire 3 A315-31 Black, 15.6" HD, Pentium N3520, 4GB, 128GB SSD, Windows 10 Home
16	Lenovo V110-15IS...	\$409.63	9 reviews	Lenovo V110-15ISK, 15.6" HD, Core i3-6006U, 8GB, 128GB SSD, Windows 10 Home
17	Acer Aspire E51-...	\$410.46	14 reviews	Acer Aspire E51-732 Black, 17.3" HD+, Celeron, N3350, 4GB, 1TB, Windows 10 Home
18	Asus VivoBook 15...	\$410.66	4 reviews	Asus VivoBook 15 X540NA-GQ026T Chocolate Bl...
19	Packard 255 G2	\$416.99	2 reviews	15.6", AMD E2-3800 1.3GHz, 4GB, 500GB, Windows 8.1
20	Asus EeeBook R41...	\$433.30	1 reviews	Asus EeeBook R416NA-FA014T, 14" FHD, Pentium N3520, 4GB, 128GB SSD, Windows 10 Home
21	Acer Aspire 3 A3...	\$436.29	1 reviews	Acer Aspire 3 A315-51, 15.6" HD, Core i3-6006U, 4GB, 128GB SSD, Windows 10 Home

Fig.9 Dataframe

That's the tutorial I gave, hopefully, it will be useful for you guys especially for you who are learning web scraping. See you again in the next project.

Data Science

Data Engineering

Web Scraping

Tutorial

Database

Sign up for Analytics Vidhya News Bytes

By Analytics Vidhya

Latest news from Analytics Vidhya on our Hackathons and some of our best articles! [Take a look.](#)

Your email



Get this newsletter

By signing up, you will create a Medium account if you don't already have one. Review our [Privacy Policy](#) for more information about our privacy practices.

Open in app

Sign up

Sign In

