**What is Web Scraping?**

*Web scraping is a computer software technique of extracting information from websites. This technique mostly focuses on the transformation of unstructured data (HTML format) on the web into structured data (database or spreadsheet).*

We will use Python for scraping because of its ease .It has a library known as **‘Beautiful Soup’** which assists this task.

*There is no universal solution for web scraping because the way data is stored on each website is usually specific to that site. In fact, if you want to scrape the data, you need to understand the website’s structure.*

Explore the website you want to scrape.

Accordingly, plan your code to pick the attribute values from the corresponding html elements in the website page which will provide you the information you wish to extract out.

Let us take the example of the website<http://www.spareshub.com>

We will hereby follow the analogy as per the example of a product page like <https://spareshub.com/car-brands/ashok-leyland/gasket-for-ashok-leyland-light-commercial-vechile-leyland-iveco-full-set-with-1-5mm-thick-chg.html>

We would like to get the product name, product price, image url, product details and Product description from this page.

Basics of HTML:

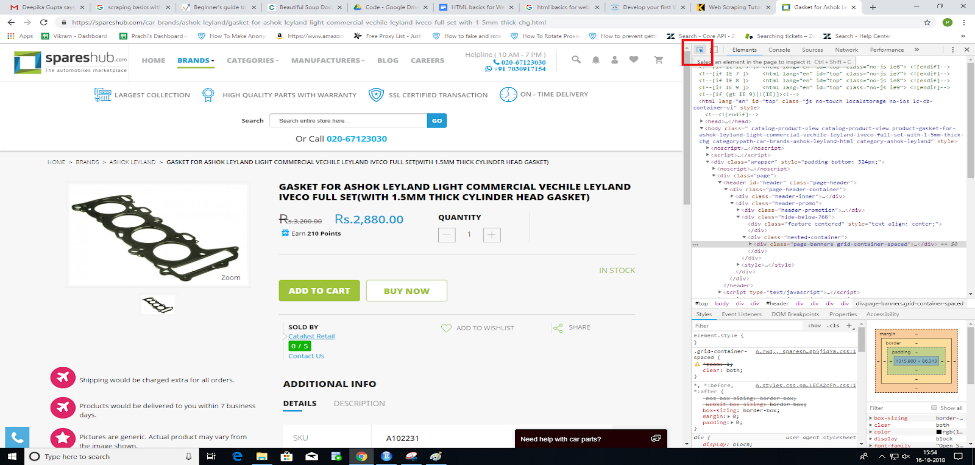
**Some prerequisite knowledge of HTML is needed for moving forward with scraping the webpage with Python code. Let us go through the most important HTML concepts associated with scraping:**

For finding the unique defining attributes for a particular element in a page, right-click and choose ‘Inspect’ for opening your browser’s inspector to inspect the webpage.



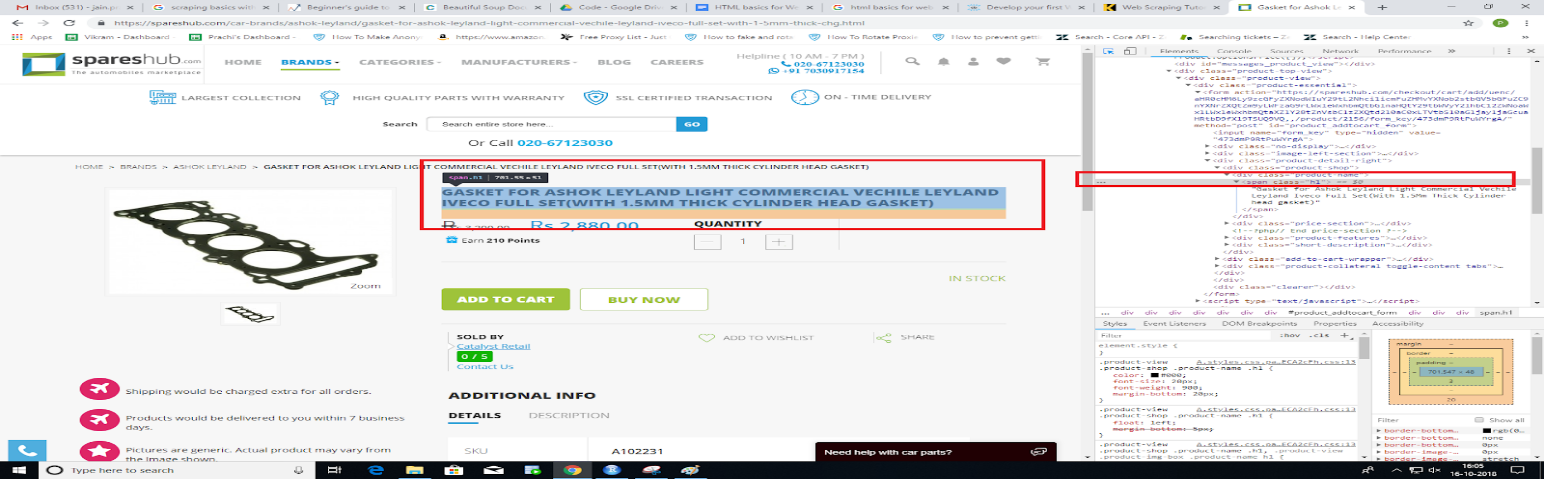
Next, use the inspector cursor to click on a section of the website that you want to control.

The inspector cursor can be taken from this portion:



When you have clicked that HTML element, the HTML code that creates that section will be highlighted on the right.

In the photo below, I have clicked on the product name section.

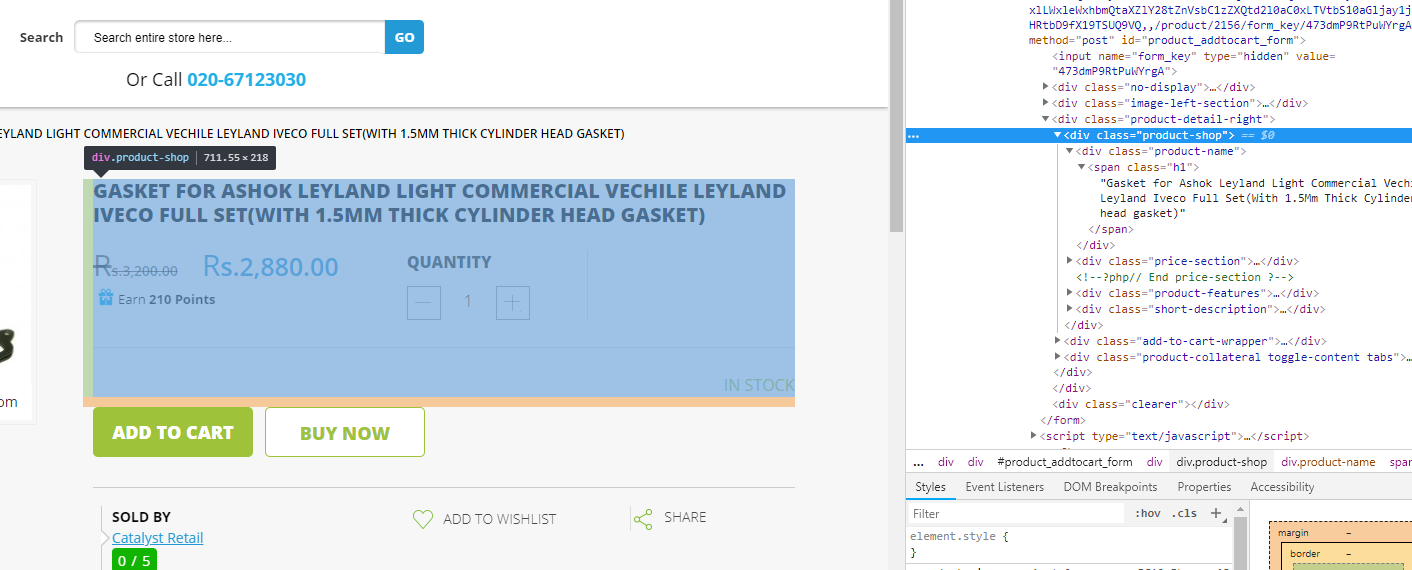


Some HTML elements which generally hold scrapable information of interest in the HTML pages are:

**Div**

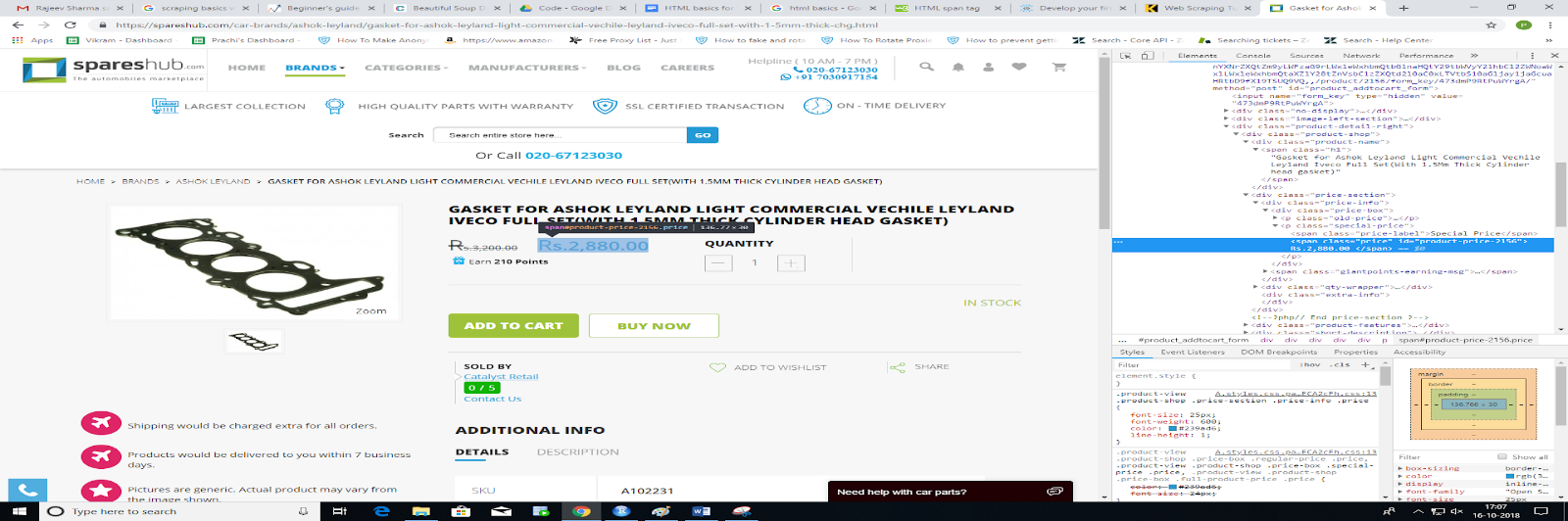
The <div> tag defines a division or a section in an HTML document.

The <div> element is often used as a container for other HTML elements



**Span**

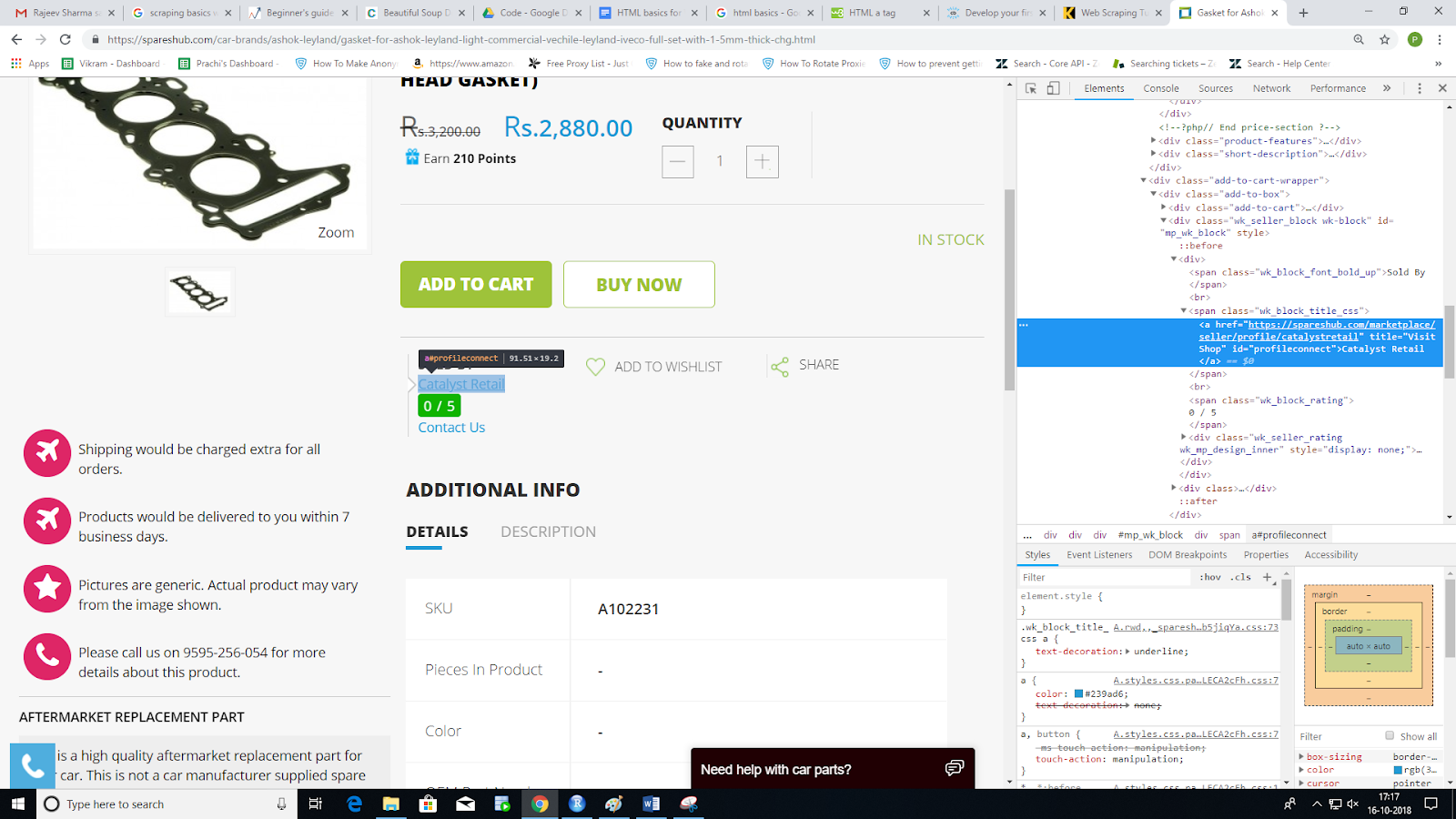
The <span> tag provides a way to add a hook to a part of a text or a part of a document.It is generally found inside <div> elements.



**Anchor**

The <a> tag defines a hyperlink, which is used to link from one page to another.

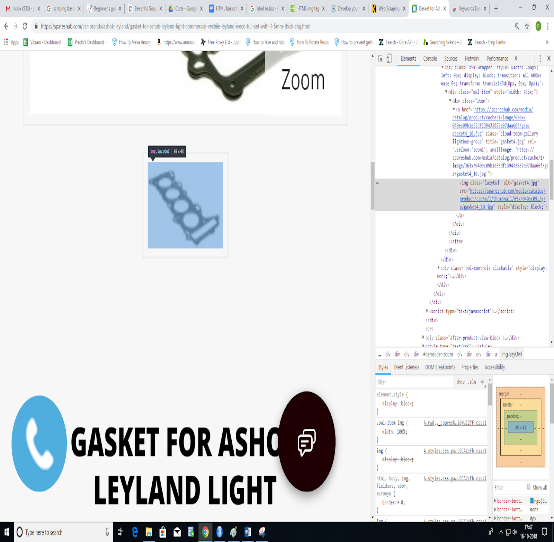
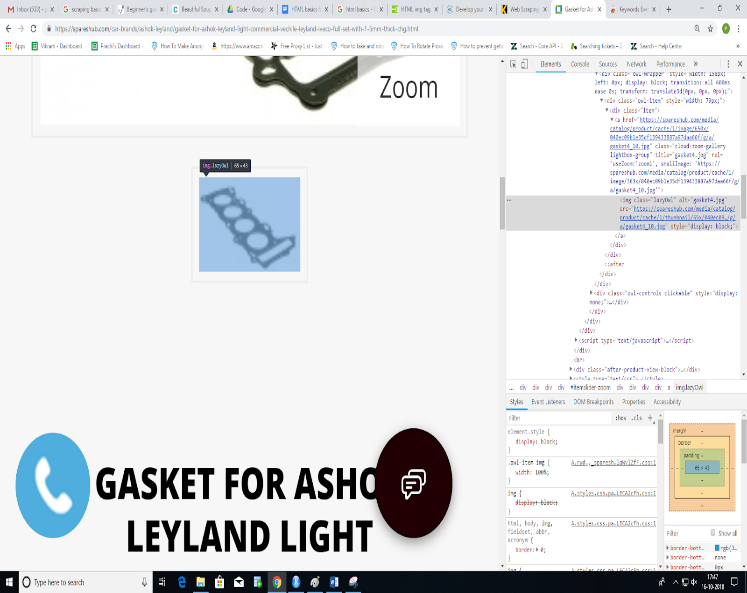
The most important attribute of the <a> element is the **href** attribute, which indicates the link's destination.



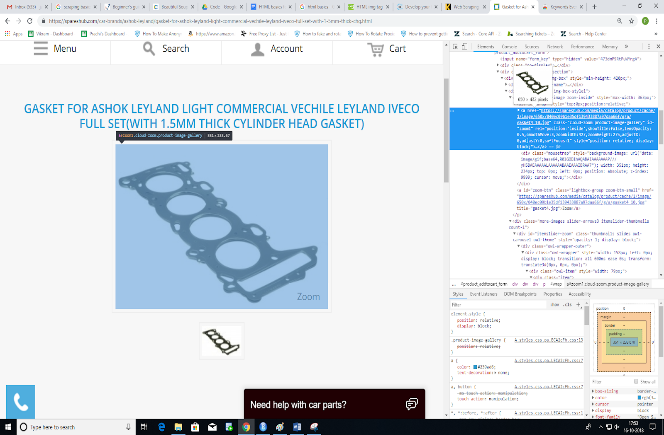
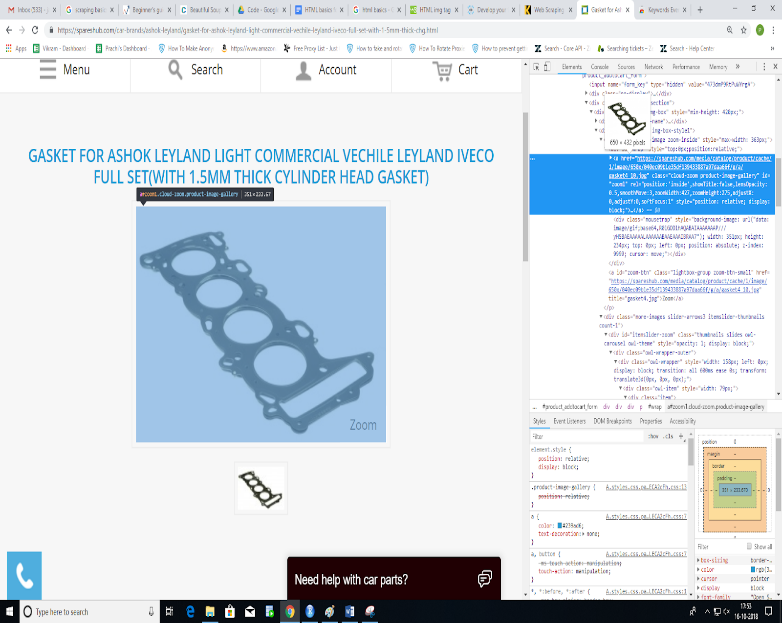
**Image**

The <img> tag defines an image in an HTML page.

The most important attribute of the <a> element is the **src** attribute, which indicates the image's source.

Sometimes, the images are given as links,so we can also get image source from its href attribute:

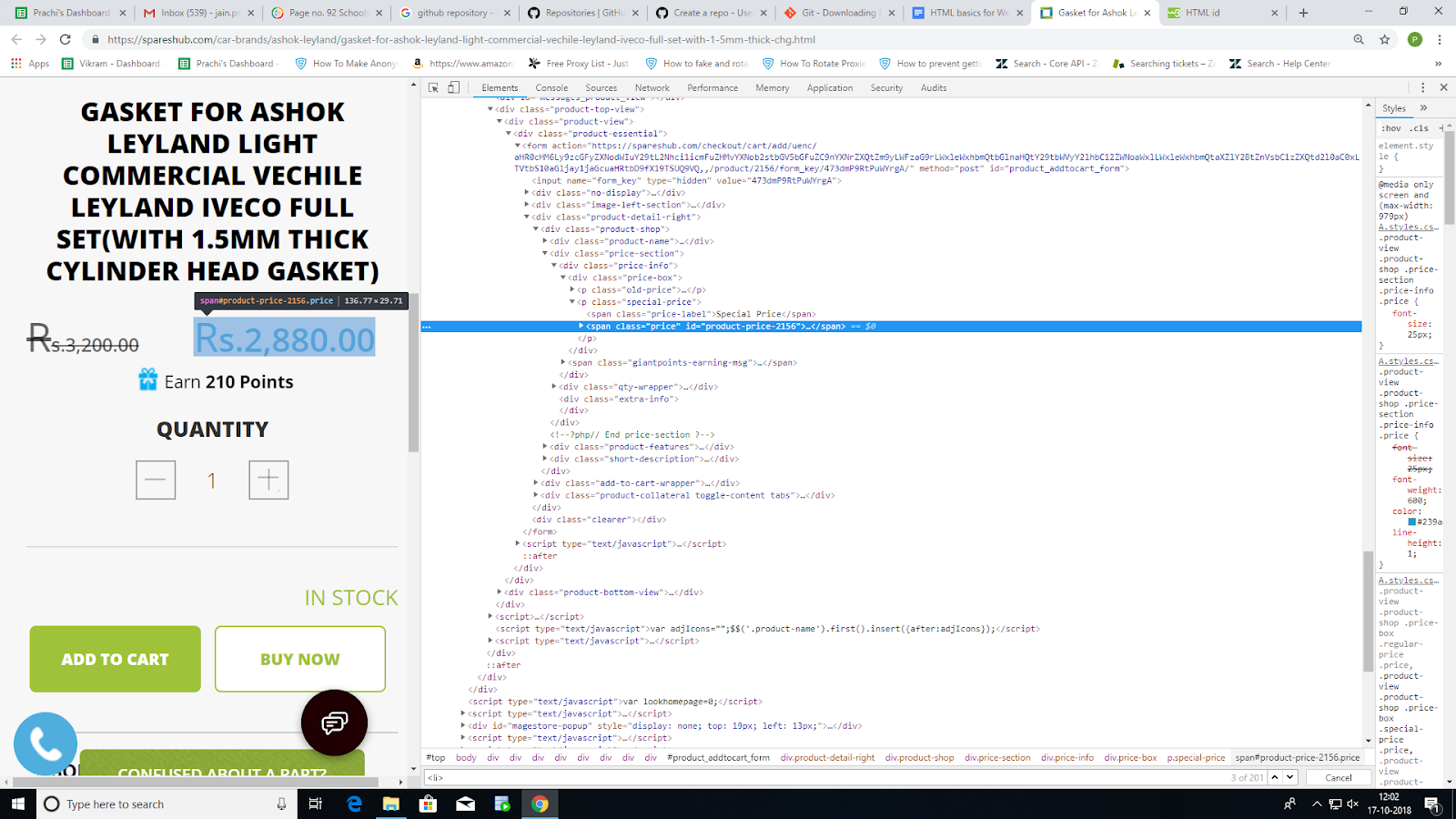


Now that we have selected the target HTML element, we would have to take up an identifier associated with this in the HTML source code document of the page.

This purpose is served by HTML attributes:

**HTML id**

The id attribute specifies a unique id for an HTML element (the value must be unique within the HTML document).

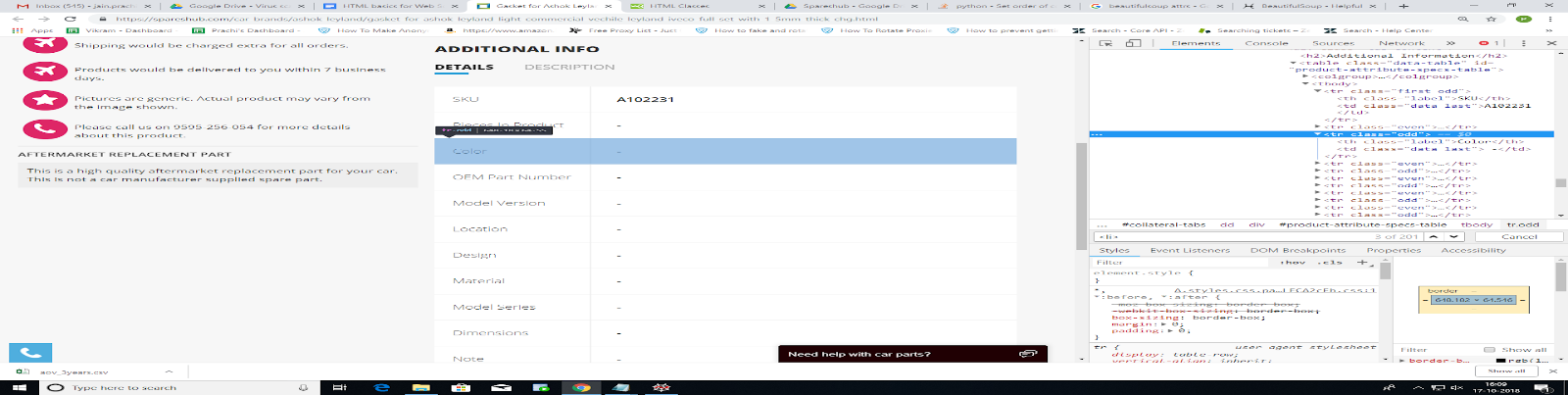


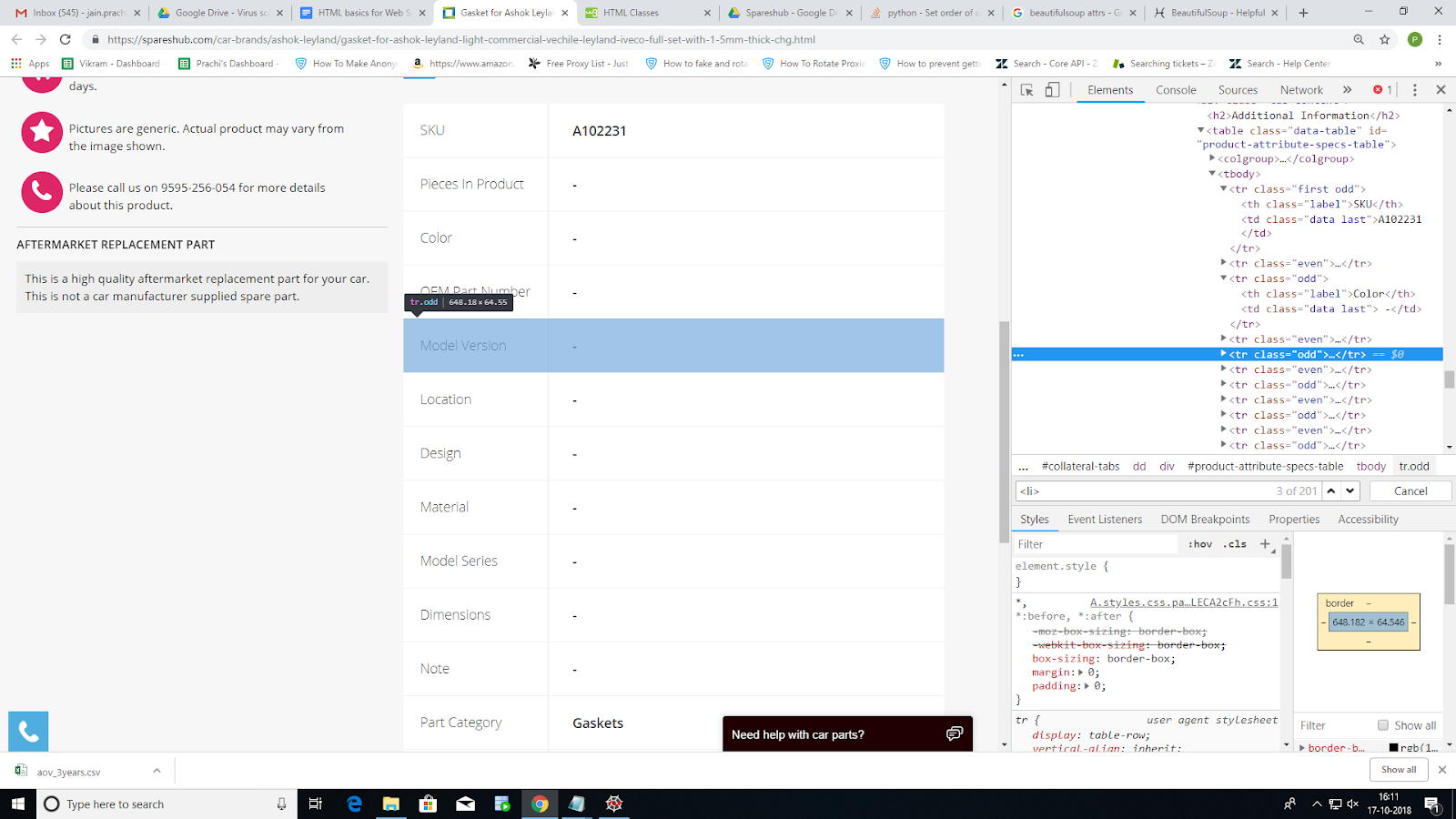
So, this id will be unique in the html page of a single product and can be used as it is to scrape this element across different product pages(holds good in case of Spareshub).

**HTML Class**

The HTML class attribute is used to define equal styles for elements with the same class name(i.e. within the same html document,there can be multiple elements with same class name).

So, all HTML elements with the same class attribute will have the same format and style.

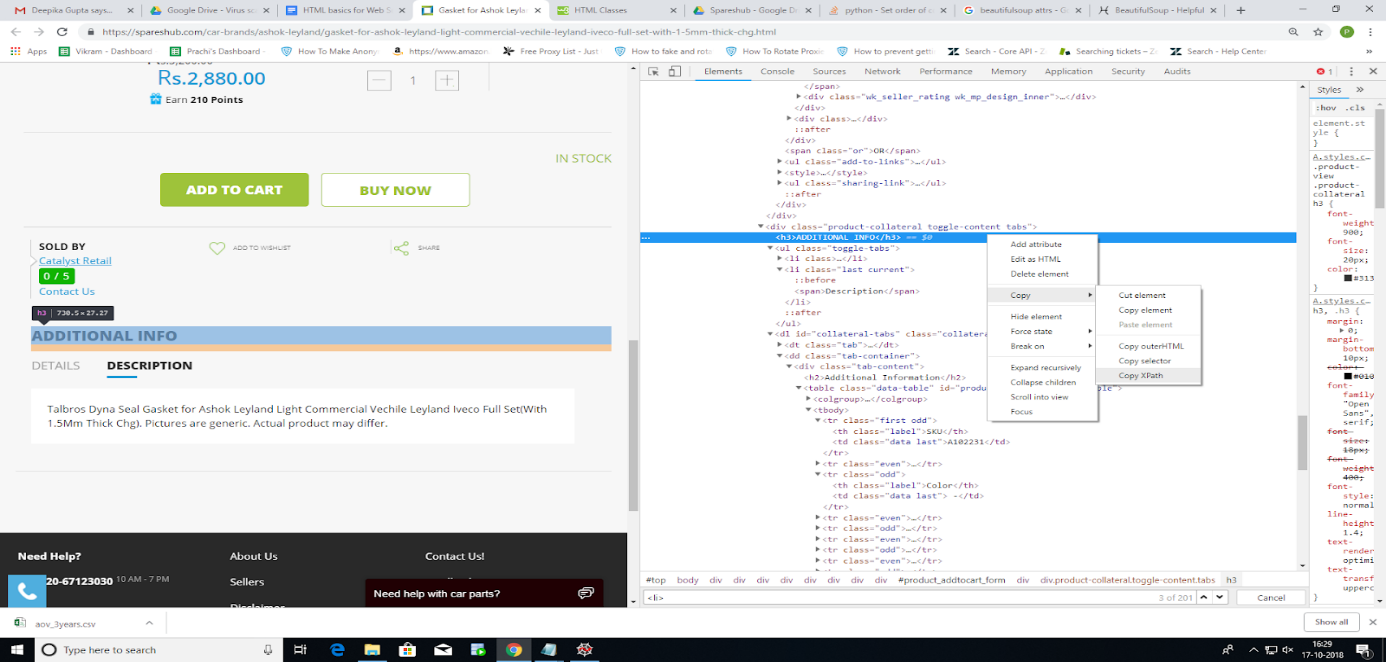




In the example shown above, every Product specification placed at odd position has the same class name i.e. class = “odd”. So, this can be used in a loop(i.e. using for loop in Python) to fetch the details from the elements having same class names .

**Xpath(Can be used with Python html package)**

This identifier is useful when we need to fetch details about an element uniquely when no id or class is specified.This can be found out by right clicking on the element tag in the inspector,and then going to Copy -> Copy XPath

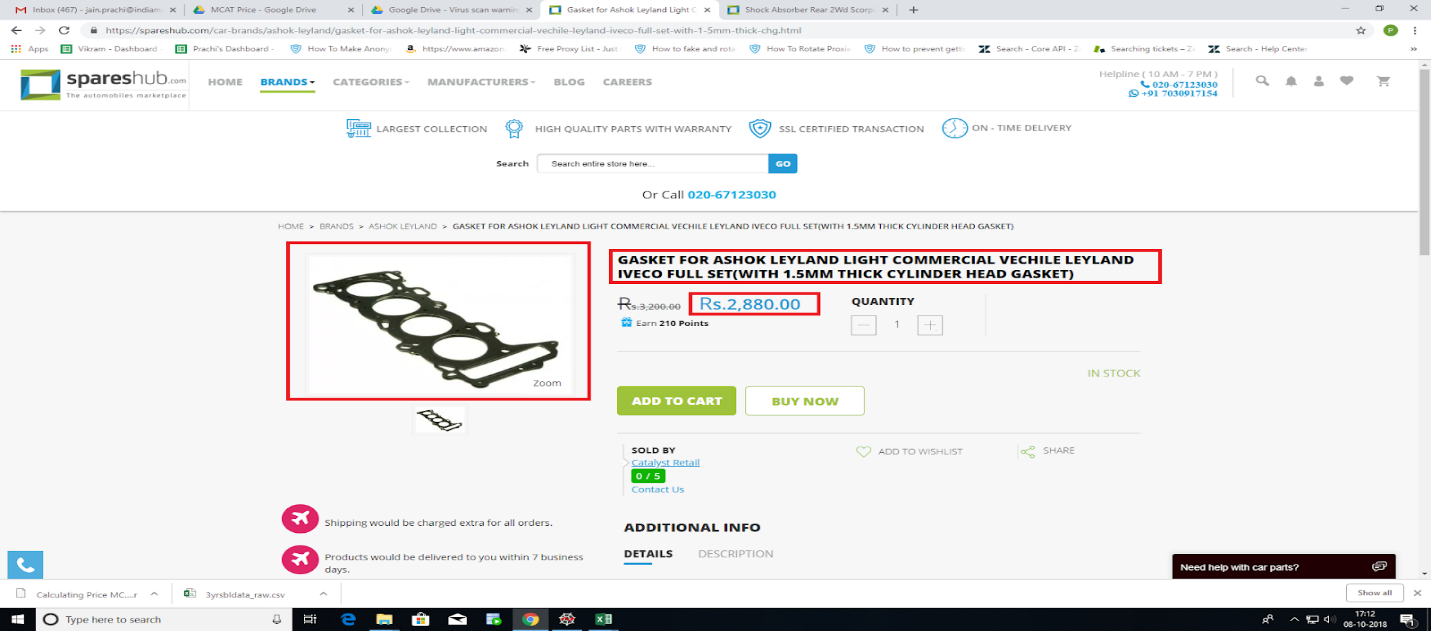


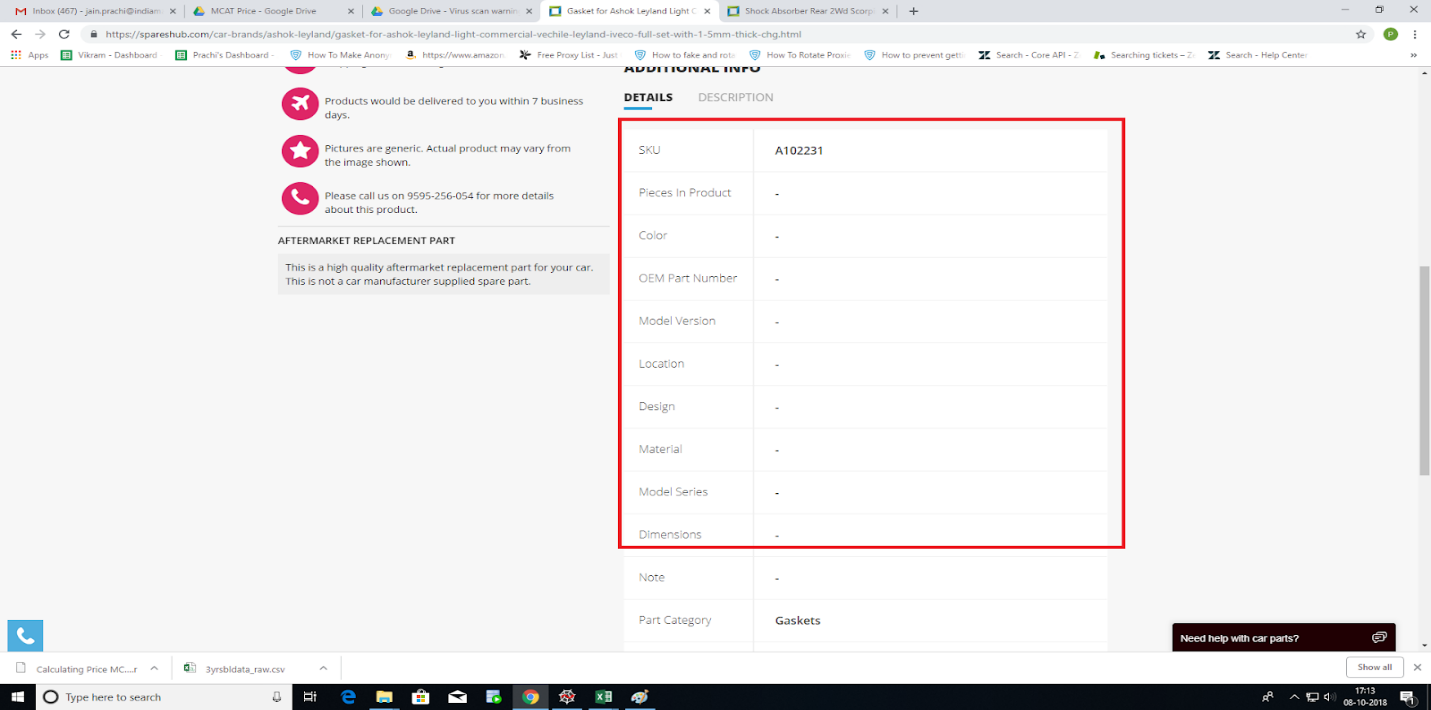
You can refer the link <https://www.w3schools.com/tags/default.asp> for more clarity and  further insights on all HTML tags and their usage.

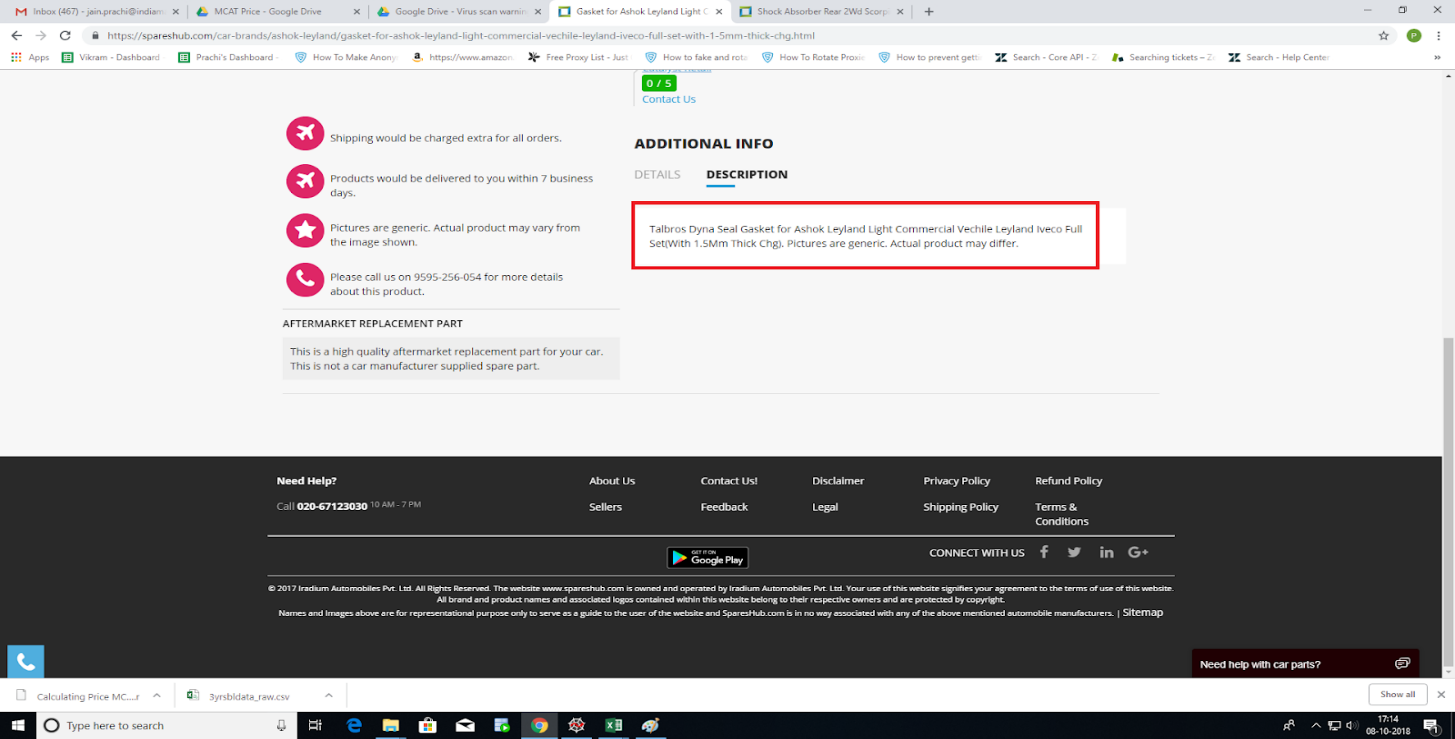
**Implementation with Python:**

We will be taking up the Product page <https://spareshub.com/shock-absorber-rear-2wd-scorpio-tc-crde-m-hawk-eagle.html> on Spareshub.

We are interested in scraping relevant information like Product Name,Price, Image URL,Product Specifications and Product Description as shown below:







Below is the Python code for the above purpose and line by line explanation of the code will follow after it:

# -\*- coding: utf-8 -\*-

"""

Created on Thu Oct  4 15:53:40 2018

@author: prachi

"""

from lxml import html

import requests

from bs4 import BeautifulSoup

url  = 'https://spareshub.com/shock-absorber-rear-2wd-scorpio-tc-crde-m-hawk-eagle.html'

headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.90 Safari/537.36'}

page = requests.get(url,headers = headers,verify=False)

if page.status\_code != 200:

       print('Failed to retrieve articles with error{}'.format(page.status\_code))

       exit()

page\_response = page.text

parser = html.fromstring(page.content)

productname = parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[1]/div[1]')

name = productname[0].text\_content().strip()

productprice=parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[1]/div[2]/div[1]/div')

price = productprice[0].text\_content().strip()

productdesc = parser.xpath('//\*[@id="collateral-tabs"]/dd[2]/div/div')

productdesc = productdesc[0].text\_content().strip()

soupvar = BeautifulSoup(page.content, "html.parser")

imglink = soupvar.find('a', attrs={'id': 'zoom1'})

img = imglink.get('href')

productspecs = parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[3]')

a = productspecs[0].text\_content().strip().split('\n')

a = [x.strip(' ') for x in a]

a= list(filter(None,a))

a=a[3:]

data = dict(zip(a[::2], a[1::2]))

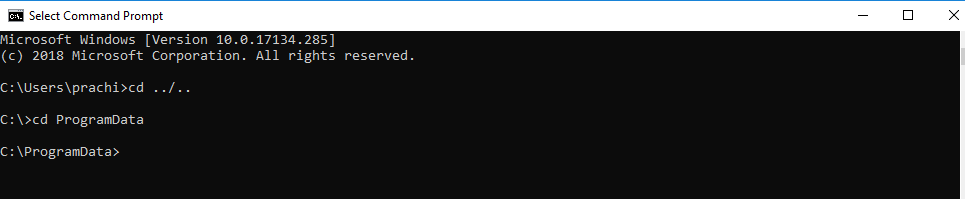
There are various functionalities provided by Python packages which are helpful for web scraping. Some of them are **BeautifulSoup(** it is found in the ‘bs4’ module) and **lxml** modules ( we generally require the html functionality from the lxml module because we are working on fetching html source code to find our required elements).Others include modules like Scrapy,etc.

The ‘requests’ module is needed to make url requests to the web for getting a webpage response.

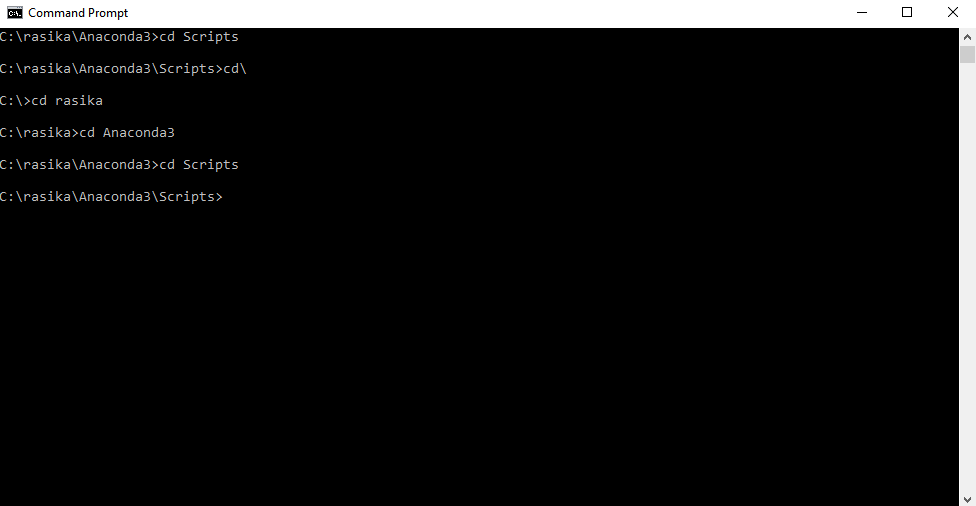
So first of all, we need to install all these packages into our computer system.

This can be done with the help of the command Prompt.

Go to the path where your anaconda package is installed.



Generally, it is found at **C:\ProgramData** but sometimes can also be in your user path, for example:



Then we will go to the path Anaconda3/Scripts to find the pip file and use it to install the required modules as follows:



Now store the target html URL enclosed in single or double quotes into a variable as shown below:

url='<https://spareshub.com/shock-absorber-rear-2wd-scorpio-tc-crde-m-hawk-eagle.html>'

Python Requests does not force you to use request headers while sending requests but there are few smart websites that do not let you read anything important unless certain headers are set in it. The least we can do for making our requests legitimate is to set a User-agent.

headers = {'User-Agent': 'Mozilla/5.0 (X11; Linux x86\_64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/42.0.2311.90 Safari/537.36'}

page = requests.get(url,headers = headers,verify=False)

The requests.get() sends a GET request to the url passed along with the headers,if any.Other optional parameters like username,password,timeout(in seconds),etc can also be passed in the get request according to the url which is being requested for.

The above command returns a response object which is named above as ‘page’.

Next, we check if the status code of the response object received is not equal to 200(successful).

If this condition becomes true, we will print the error and simply exit from the further execution of the code.

if page.status\_code != 200:

       print('Failed to retrieve articles with error{}'.format(page.status\_code))

       exit()

If the response status code is one from the 400 or 500 family, it is an error fetching the response.

For more information about response status codes,visit <https://developer.mozilla.org/en-US/docs/Web/HTTP/Status>

If the code did not exit in the above line, it means the response was successful (with status code 200) and the subsequent lines of code will execute.

page\_response = page.text

parser = html.fromstring(page.content)

‘page.content’ will give the response in encoded form and can be passed to the fromstring function of the ‘html’ package to make an html parser.

If you want to see the actual text result of the HTML page, you can read the ‘.text’ property of this object.

productname=parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[1]/div[1]')

name = productname[0].text\_content().strip()

The html parser we created by the name ‘parser’ is used to utilise the xpath().Inside this function, we can pass the xpath of the target element(shown in grey) which we had copied from the html page inspector.The fetched element ResultSet is stored in a variable ‘productname’.

Hence, the productname[0] is the only element tag in the productname ResultSet and we can find the actual product name by using text\_content() on it. We then store it in a variable ‘name’.

Similarly, ‘productprice’ and ‘productdescription’ are found in the following lines using their corresponding xpath copied from html inspector.

productprice=parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[1]/div[2]/div[1]/div')

price = productprice[0].text\_content().strip()

productdesc = parser.xpath('//\*[@id="collateral-tabs"]/dd[2]/div/div')

productdesc = productdesc[0].text\_content().strip()

soupvar = BeautifulSoup(page.content, "html.parser")

The BeautifulSoup() is used to create a soup variable ‘soupvar’ which provides other useful functions. BeautifulSoup() takes the page.content as an argument as well as the type of parser being used, which is the ‘html.parser’ here(there are other options available like xml parsers,etc).

imglink = soupvar.find('a', attrs={'id': 'zoom1'})

img = imglink.get('href')

The soupvar variable now has the complete HTML code of the URL. So, with the help of .find function on soupvar, we can find a particular element having certain specified attributes to identify them. For example, we are finding the link (‘a’ i.e. anchor tag) where attrs (i.e. attribute) ‘id’ is having value ‘zoom1’(as found by the html inspector).

The attribute can be anything like class,title,etc. for elements like div,span,img,etc. as the first parameter in find().

Hence, ‘imglink’ is the element tag retrieved and we can use the .get() on it to get the required value from it.Here, we want the image url from the image, so we pass ‘href’ attribute in it and store it in a variable ‘img’.

Note: The **find\_all(‘<tag-name>’)** will help you get all the elements in the HTML page for the tagname passed.The retrieved elements can then be used to find information from the tags using loops. For more details on find\_all(), see this [link](https://www.crummy.com/software/BeautifulSoup/bs4/doc/#find-all).

productspecs=parser.xpath('//\*[@id="product\_addtocart\_form"]/div[3]/div[3]')

a = productspecs[0].text\_content().strip().split('\n')

a = [x.strip(' ') for x in a]

a= list(filter(None,a))

a=a[3:]

data = dict(zip(a[::2], a[1::2]))

The ‘productspecifications’ are found in a similar fashion using xpath and stored in a variable ‘a’.

Then ‘a’ is processed to remove unwanted rows fetched and strip() removes the leading and trailing spaces from each row of data fetched. Finally, we are using the dict() to zip together the specifications like SKU’s as keys and their values on the pages as values in the python dictionary ‘data’.

Once all the required information is scraped using the above functions as well as other text processing commands and loop statements, we can store this data into a file.

This can be done with the help of Python dataframes.

Note: First install package ‘pandas’’ using pip in cmd.

**C:\ProgramData\Anaconda3\Scripts>**pip install pandas

And import it in the starting lines of the code,

import pandas as pd

For example, the data scraped above i.e. name , price, productdesc, img, data can be saved as:

#It creates an empty dataframe(virtual table) with the specified column names

Dataframe1 = pd.DataFrame(columns = {‘ProductName’,’ProductPrice’,’ProductDescription’,’ProductImageURL’,’ProductSpecifications’})

#Now setting the values in the columns row wise where i can range from **0 to any length**

Dataframe1.set\_value(i,‘ProductName’,name)

Dataframe1.set\_value(i,’ProductPrice’,price)

Dataframe1.set\_value(i,’ProductDescription’,productdesc)

Dataframe1.set\_value(i,’ProductImageURL’,img)

Dataframe1.set\_value(i,’ProductSpecifications’,data)

#Saving the dataframe to a csv file with specified name

Dataframe1.to\_csv(“ScrapedData.csv”,index=False)

For more details on dataframes, go to <http://pandas.pydata.org/pandas-docs/version/0.23.4/generated/pandas.DataFrame.html>

As already mentioned earlier:

*There is no universal solution for web scraping because the way data is stored on each website is usually specific to that site. In fact, if you want to scrape the data, you need to understand the website’s structure.*