**How to scrape Alibaba.com product data using Scrapy**

Scrapy is the most popular open source web scraping framework. Written in Python,  it has most of the modules you would need to efficiently extract, process, and store data from websites in pretty much any structured data format. Scrapy is best suited for web crawlers which scrapes data from multiple types of pages

In this tutorial, we will show you how to scrape product data from **Alibaba.com –**the world’s leading marketplace.

**Prerequisites**

**Install Python 3 and Pip**

We will use Python 3 for this tutorial.  To start, you need a computer with Python 3 and PIP .

Follow the guides below to install Python 3 and pip:

* Linux – <http://docs.python-guide.org/en/latest/starting/install3/linux/>
* Mac – <http://docs.python-guide.org/en/latest/starting/install3/osx/>
* Windows  – <https://www.scrapehero.com/how-to-install-python3-in-windows-10/>

**Install Scrapy**

pip3 install scrapy

You can find more details on installation here  – <https://doc.scrapy.org/en/latest/intro/>

**Create a Scrapy Project**

Let’s create a  scrapy project using the following command.

scrapy startproject scrapy\_alibaba

This command creates a Scrapy project with the Project Name  (scrapy\_alibaba) as the folder name.  It will contain all necessary files with proper structure and basic doc strings for each file, with a structure similar to

scrapy\_alibaba/ *# Project root directory*

scrapy.cfg  *# Contains the configuration information to deploy the spider*

scrapy\_alibaba/ *# Project's python module*

\_\_init\_\_.py

items.py      *# Describes the definition of each item that we’re scraping*

middlewares.py  *# Project middlewares*

pipelines.py     *# Project pipelines file*

settings.py      *# Project settings file*

spiders/         *# All the spider code goes into this directory*

\_\_init\_\_.py

**Create a Spider**

Scrapy has an built-in command called genspider to generate the basic spider template.

scrapy genspider <spidername> <website>

Let’s generate our spider

scrapy genspider alibaba\_crawler alibaba.com

and this will create a spiders/scrapy\_alibaba.py file for you with the initial template to crawl alibaba.com.

The code would like this

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

import scrapy

class AlibabaCrawlerSpider(scrapy.Spider):

name = 'alibaba\_crawler'

allowed\_domains = ['alibaba.com']

start\_urls = ['http://alibaba.com/']

def parse(self, response):

pass

The class AlibabaCrawler inherits the base class scrapy.Spider. The Spider class knows how to follow links and extract data from web pages but it doesn’t know where to look or what data to extract. We will add this information later.

Variable and functions

* **name** is the name of the spider which was given in the standard generation command.  
  We will use this name to start the spider from the command line.
* A list of **allowed\_domains** are the domains that spider is allowed to crawl
* **start\_urls** is the urls which the spider will start crawling when it is invoked.
* [**parse()**](https://doc.scrapy.org/en/latest/topics/spiders.html#scrapy.spiders.Spider.parse) is the Scrapy’s default callback method which is called for requests without an explicitly assigned callback. parse function get invoked after each start\_url is crawled. You can use this function to parse the response, extract the scraped data, and find new URLs to follow by creating new requests ([**Request**](https://doc.scrapy.org/en/latest/topics/request-response.html#scrapy.http.Request)) from them.

Now let’s run our basic spider to see what happens. You can run the spider using the command:

scrapy crawl alibaba\_crawler

The output would look  like this:

2018-11-06 15:28:52 [scrapy.utils.log] INFO: Scrapy 1.5.1 started (bot: scrapy\_alibaba)

2018-11-06 15:28:52 [scrapy.utils.log] INFO: Versions: lxml 4.2.5.0, libxml2 2.9.8, cssselect 1.0.3, parsel 1.5.1, w3lib 1.19.0, Twisted 18.7.0, Python 3.7.0 (default, Jun 28 2018, 08:04:48) [MSC v.1912 64 bit (AMD64)], pyOpenSSL 18.0.0 (OpenSSL 1.0.2p 14 Aug 2018), cryptography 2.3.1, Platform Windows-10-10.0.17134-SP0

2018-11-06 15:28:52 [scrapy.crawler] INFO: Overridden settings: {'BOT\_NAME': 'scrapy\_alibaba', 'NEWSPIDER\_MODULE': 'scrapy\_alibaba.spiders', 'ROBOTSTXT\_OBEY': True, 'SPIDER\_MODULES': ['scrapy\_alibaba.spiders']}

2018-11-06 15:28:53 [scrapy.middleware] INFO: Enabled extensions:

['scrapy.extensions.corestats.CoreStats',

'scrapy.extensions.telnet.TelnetConsole',

'scrapy.extensions.logstats.LogStats']

2018-11-06 15:28:53 [scrapy.middleware] INFO: Enabled downloader middlewares:

['scrapy.downloadermiddlewares.robotstxt.RobotsTxtMiddleware',

'scrapy.downloadermiddlewares.httpauth.HttpAuthMiddleware',

'scrapy.downloadermiddlewares.downloadtimeout.DownloadTimeoutMiddleware',

'scrapy.downloadermiddlewares.defaultheaders.DefaultHeadersMiddleware',

'scrapy.downloadermiddlewares.useragent.UserAgentMiddleware',

'scrapy.downloadermiddlewares.retry.RetryMiddleware',

'scrapy.downloadermiddlewares.redirect.MetaRefreshMiddleware',

'scrapy.downloadermiddlewares.httpcompression.HttpCompressionMiddleware',

'scrapy.downloadermiddlewares.redirect.RedirectMiddleware',

'scrapy.downloadermiddlewares.cookies.CookiesMiddleware',

'scrapy.downloadermiddlewares.httpproxy.HttpProxyMiddleware',

'scrapy.downloadermiddlewares.stats.DownloaderStats']

2018-11-06 15:28:53 [scrapy.middleware] INFO: Enabled spider middlewares:

['scrapy.spidermiddlewares.httperror.HttpErrorMiddleware',

'scrapy.spidermiddlewares.offsite.OffsiteMiddleware',

'scrapy.spidermiddlewares.referer.RefererMiddleware',

'scrapy.spidermiddlewares.urllength.UrlLengthMiddleware',

'scrapy.spidermiddlewares.depth.DepthMiddleware']

2018-11-06 15:28:53 [scrapy.middleware] INFO: Enabled item pipelines:

[]

2018-11-06 15:28:53 [scrapy.core.engine] INFO: Spider opened

2018-11-06 15:28:53 [scrapy.extensions.logstats] INFO: Crawled 0 pages (at 0 pages/min), scraped 0 items (at 0 items/min)

2018-11-06 15:28:53 [scrapy.extensions.telnet] DEBUG: Telnet console listening on 127.0.0.1:6023

2018-11-06 15:28:54 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (302) to from

2018-11-06 15:28:55 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to from

2018-11-06 15:28:56 [scrapy.core.engine] DEBUG: Crawled (200) (referer: None)

2018-11-06 15:28:56 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (302) to from

2018-11-06 15:28:56 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to from

2018-11-06 15:28:56 [scrapy.core.engine] DEBUG: Crawled (200) (referer: None)

2018-11-06 15:28:57 [scrapy.downloadermiddlewares.redirect] DEBUG: Redirecting (301) to from

2018-11-06 15:28:57 [scrapy.core.engine] DEBUG: Crawled (200) (referer: None)

2018-11-06 15:28:57 [scrapy.core.engine] INFO: Closing spider (finished)

2018-11-06 15:28:57 [scrapy.statscollectors] INFO: Dumping Scrapy stats:

{'downloader/request\_bytes': 3914,

'downloader/request\_count': 8,

'downloader/request\_method\_count/GET': 8,

'downloader/response\_bytes': 28164,

'downloader/response\_count': 8,

'downloader/response\_status\_count/200': 3,

'downloader/response\_status\_count/301': 3,

'downloader/response\_status\_count/302': 2,

'finish\_reason': 'finished',

'finish\_time': datetime.datetime(2018, 11, 6, 9, 58, 57, 942393),

'log\_count/DEBUG': 9,

'log\_count/INFO': 7,

'response\_received\_count': 3,

'scheduler/dequeued': 3,

'scheduler/dequeued/memory': 3,

'scheduler/enqueued': 3,

'scheduler/enqueued/memory': 3,

'start\_time': datetime.datetime(2018, 11, 6, 9, 58, 53, 678737)}

2018-11-06 15:28:57 [scrapy.core.engine] INFO: Spider closed (finished)

Scrapy provides comprehensive information about the crawl, as you go through the logs, you can understand what’s happening in the spider.

* The spider is initialized with the bot name “scrapy\_alibaba” and prints all packages used in the project with version numbers.
* Scrapy looks for spider modules which are located in the /spiders directory. Setting default values to variables such as CONCURRENT\_REQUESTS\_PER\_DOMAIN, CONCURRENT\_REQUESTS, SPIDER\_MODULES,  
  DOWNLOAD\_TIMEOUT
* Loaded all the components like middlewares, extensions, and pipelines which are needed to handle the requests
* Used the URLs provided in start\_urls and retrieved the HTML content of the page. Since we didn’t specify any callbacks for the start\_urls the response is received at the parse() function. We also did not write any lines to handle the response received, so the spider finished with stats like the pages scraped in the crawl, bandwidth used in bytes, the number of items scraped, status code counts etc.

**Extract Data from Alibaba.com**

For this tutorial we will only extract the following fields from any search result page of alibaba.com:

1. Product Name
2. Price Range (in US dollars)
3. Minimum Order
4. Seller Name
5. Seller Response Rate
6. Number of years as a seller on Alibaba
7. Transactional Level

You could go further and scrape the product details based on filters and orders. But for now, we’ll keep it simple and stick to these fields.

When you search for any keyword, say “earphones” in Alibaba you will see that the result page has a URL similar– [https://www.alibaba.com/trade/search?fsb=y&IndexArea=product\_en&CatId=&SearchText=**earphones**&viewtype=G](https://www.alibaba.com/trade/search?fsb=y&IndexArea=product_en&CatId=&SearchText=earphones&viewtype=G), where the parameter SearchText having the keyword you searched for.

To find where each field is located in the HTML, open a browser and search for a product on Alibaba.com. We are using Google Chrome, but the steps would be similar for most of the browsers. Right-click on anywhere on the page and choose – Inspect Element. The browser will open the developer toolbar and show the HTML Content of the Web page formatted nicely.

Here is a portion of the HTML code that holds a single product’s information. We are extracting the data from this section of the HTML.  
For better readability, we removed some tags.

<div class="item-main">

 <div class="item-img">

    <div class="place"></div>

    <div class="item-img-inner">

       <a href="//www.alibaba.com/product-detail/240-hours-standby-blue-tooth-earphone\_60795361346.html?s=p">

          <div class="icon-wrap">

          </div>

       </a>

    </div>

 </div>

 <div class="item-info">

    <h2 class="title two-line">

      <a target="\_blank"  data-p4plog="60795361346" href="//www.alibaba.com/product-detail/240-hours-standby-blue-tooth-earphone\_60795361346.html?s=p" title="**240 hours standby blue tooth earphone OEB-E35S**">

              240 hours standby blue tooth <strong>earphone</strong> OEB-E35S

       </a>

    </h2>

    <div class="pmo ">

       <div class="price">

          <b>

**US $10.00**

          </b>

          / Piece

       </div>

**<div class="min-order">**

**<b>1 Piece</b>**

**(Min. Order)**

**</div>**

    </div>

    <div class="stitle util-ellipsis">

       <div class="s-gold-supplier-year-icon">1<span class="unit"> YR </span></div>

       <a title="Shanghai Hue Industry Co., Ltd." href="//huesh.en.alibaba.com/company\_profile.html#top-nav-bar" data-domdot="id:2679,mn:allpage,pid:60795361346"

          data-p4plog="60795361346" target="\_blank">Shanghai Hue Industry Co., Ltd.</a>

    </div>

    <div class="sstitle">

       <div class="supplier">

          <a data-domdot="id:2639,ext:'type=gs|scene=pdgallery'" href="//fuwu.alibaba.com/gps/buyer.htm" target="\_blank">

          <i class="supplier-icon ui2-icon-svg ui2-icon-svg-xs ui2-icon-svg-gold-supplier"></i>

          </a>

          <a data-ta="item-tips-action" data-ta-price="36,000" target="\_blank"

             rel="nofollow" href="//tradeassurance.alibaba.com?tracelog=from\_list\_item">

          <i class="supplier-icon ui2-icon-svg ui2-icon-svg-xs ui2-icon-svg-trade-assurance"></i>

          </a>

       </div>

       <a

          class="diamond-level-group"

          target="\_blank" href=" //huesh.en.alibaba.com/company\_profile/transaction\_level.html ">

       <i class="ui2-icon-svg diamon-icon ui2-icon-svg-xs ui2-icon-svg-diamond-level-half-filled"></i>

       </a>

    </div>

    <div class="

       item-extra item\_\_offer-global-impression-wrapper item-flexible

       always-hidden   ">

       <div class="m-gallery-offer-global-impression m-offer-global-impression\_\_matrix-wrapper full-expand">

       </div>

    </div>

    <div class="hr"><span></span></div>

    <div class="contact">

       <a class="ui2-button ui2-button-default ui2-button-primary ui2-button-small csp" data-domdot="id:2675,mn:allpage,pid:60795361346"

          data-p4plog="60795361346" href="//message.alibaba.com/msgsend/contact.htm?action=contact\_action&amp;appForm=s\_en&amp;chkProductIds=60795361346&amp;chkProductIds\_f=IDX1ZMTWI\_ZYptpWONExXcc0dDuHMZ57MZF0IFgXKQcA0V4gDR3TIcwUVm7LLdrVhbbc&amp;tracelog=contactOrg&amp;mloca=main\_en\_search\_list" data-role="contact-supplier" rel="nofollow" target="\_blank">

       <i class="ui2-icon ui2-icon-message ico-cs"></i> Contact Supplier

    </div>

    <div class="action">

       <div class="compare-action">

       </div>

       <div class="more-entry-wrap">

          <a class="more-entry">···</a>

          <div class="more-entry-container-wrap"></div>

       </div>

    </div>

 </div>

</div>

**Download a page using Scrapy Shell**

Scrapy shell is an interactive shell where you can try and debug your code without running the full spider. You can launch scrapy shell using the command scrapy shell in your terminal.

You just have to provide it a URL and Scrapy Shell will download the HTML content of the URL. It will let you interact with the same objects that your spider handles in its callbacks, including the response object. We will use the fetch command to get the response from the URL  
– <https://www.alibaba.com/trade/search?fsb=y&IndexArea=product_en&CatId=&SearchText=earphones>

In [2]:fetch("https://www.alibaba.com/trade/search?fsb=y&IndexArea=product\_en&CatId=&SearchText=earphones")

[scrapy.core.engine] INFO: Spider opened

[default] INFO: Spider opened: default

[scrapy.core.engine] DEBUG: Crawled (200) <GET https://www.alibaba.com/trade/search?fsb=y&IndexArea=product\_en&CatId=&SearchText=earphones> (referer: None)

Scrapy shell successfully grabbed the URL and stored the HTML response in the response object.

**Construct XPath Selectors for the Product List**

If you look at the code of the product list above, you can see that each product grid item is enclosed in a div with the class attribute as item-main.  Lets write an XPath to select all grid items first.

In [3]:response.xpath("//div[@class='item-main']")

Out[3]:

[<Selector xpath="//div[@class='item-main']" data='<div class="item-main"> <div class'>,***# First product***

<Selector xpath="//div[@class='item-main']" data='<div class="item-main"><div class'>,***# Second product***

. . .

<Selector xpath="//div[@class='item-main']" data='<div class="item-main"><div class'>,***# 36th product***

]

As you can see above, the xpath() method returns a selector object for each of the  36 products in the listing page. We’ll store this in an array called products.

In [4]: products = response.xpath("//div[@class='item-main']")

Let’s get the data for the first product

**Product Name**

If your at the HTML snippet, you can see that the product heading is present as

<div class="item-info">

    <h2 class="title two-line">

      <a href="//www.alibaba.com/product-detail/240-hours-standby-blue-tooth-earphone\_60795361346.html?s=p" title="**240 hours standby blue tooth earphone OEB-E35S**">

           240 hours standby blue tooth

<strong>

earphone

</strong>

OEB-E35S

       </a>

    </h2>

Our XPath Selector would be

In [6]: XPATH\_PRODUCT\_NAME = ".//div[@class='item-info']//h2[contains(@class,'title')]//a/@title"

In [7]: products[0].xpath(XPATH\_PRODUCT\_NAME).extract()

Out[7]: [u'**240 hours standby blue tooth earphone OEB-E35S**']

**Price**

The product price can be found as

<div class="price">

<b>

**US $10.00**

    </b>

        / Piece

</div>

The XPath Selector would look like

In [9]: XPATH\_PRODUCT\_PRICE = ".//div[@class='item-info']//div[@class='price']/b/text()"

In [10]: products[0].xpath(XPATH\_PRODUCT\_PRICE).extract()

Out[10]: [u'\n US $10.80-$12.80\n ']

Likewise, we’ll find and write XPaths for all our data points from the HTML response.

**Seller Year:**

In [12]: XPATH\_SELLER\_YEARS = ".//div[@class='item-info']//div[@class='stitle util-ellipsis']//div[contains(@class,'supplier-year')]//text()"

In [14]: products[0].xpath(XPATH\_SELLER\_YEARS).extract()

Out[14]: ['1', ' YR ']

**Seller Response Rate**

In [15]: XPATH\_RESPONSE\_RATE = ".//div[@class='item-info']//div[@class='sstitle']//div[@class='num']/i/text()"

In [16]: products[0].xpath(XPATH\_RESPONSE\_RATE).extract()

Out[16]: ['91.4%']

We’ve skipped adding few fields. Please see the code below for the full fields and xpaths list. We will first select all the search results and then iterate over the variable products and extract the above info from all the selector objects that we had previously extracted from the HTML.

Putting it all together, the parser function would look similar to

def parse(self, response):

"""Function to process alibaba search results page"""

search\_keyword = response.meta["search\_text"]

parser = scrapy.Selector(response)

products = parser.xpath("//div[@class='item-main']")

*#iterating over search results*

for product in products:

*#Defining the XPaths*

XPATH\_PRODUCT\_NAME = ".//div[@class='item-info']//h2[contains(@class,'title')]//a/@title"

XPATH\_PRODUCT\_PRICE = ".//div[@class='item-info']//div[@class='price']/b/text()"

XPATH\_PRODUCT\_MIN\_ORDER = ".//div[@class='item-info']//div[@class='min-order']/b/text()"

XPATH\_SELLER\_YEARS = ".//div[@class='item-info']//div[@class='stitle util-ellipsis']//div[contains(@class,'supplier-year')]//text()"

XPATH\_SELLER\_NAME = ".//div[@class='item-info']//div[@class='stitle util-ellipsis']//a/@title"

XPATH\_SELLER\_RESPONSE\_RATE = ".//div[@class='item-info']//div[@class='sstitle']//div[@class='num']/i/text()"

XPATH\_TRANSACTION\_LEVEL = ".//div[@class='item-info']//div[@class='sstitle']//a[@class='diamond-level-group']//i[contains(@class,'diamond-level-one')]"

XPATH\_TRANSACTION\_LEVEL\_FRACTION = ".//div[@class='item-info']//div[@class='sstitle']//a[@class='diamond-level-group']//i[contains(@class,'diamond-level-half-filled')]"

XPATH\_PRODUCT\_LINK = ".//div[@class='item-info']//h2/a/@href"

raw\_product\_name = product.xpath(XPATH\_PRODUCT\_NAME).extract()

raw\_product\_price = product.xpath(XPATH\_PRODUCT\_PRICE).extract()

raw\_minimum\_order = product.xpath(XPATH\_PRODUCT\_MIN\_ORDER).extract()

raw\_seller\_years = product.xpath(XPATH\_SELLER\_YEARS).extract()

raw\_seller\_name = product.xpath(XPATH\_SELLER\_NAME).extract()

raw\_seller\_response\_rate = product.xpath(XPATH\_SELLER\_RESPONSE\_RATE).extract()

raw\_transaction\_level = product.xpath(XPATH\_TRANSACTION\_LEVEL).extract()

raw\_product\_link = product.xpath(XPATH\_PRODUCT\_LINK).extract()

*#getting the fraction part*

raw\_transaction\_level\_fraction = product.xpath(XPATH\_TRANSACTION\_LEVEL\_FRACTION)

*# cleaning the data*

product\_name = ''.join(raw\_product\_name).strip() if raw\_product\_name else None

product\_price = ''.join(raw\_product\_price).strip() if raw\_product\_price else None

minimum\_order = ''.join(raw\_minimum\_order).strip() if raw\_minimum\_order else None

seller\_years\_on\_alibaba = ''.join(raw\_seller\_years).strip() if raw\_seller\_years else None

seller\_name = ''.join(raw\_seller\_name).strip() if raw\_seller\_name else None

seller\_response\_rate = ''.join(raw\_seller\_response\_rate).strip() if raw\_seller\_response\_rate else None

*#getting actual transaction levels by adding the fraction part*

transaction\_level = len(raw\_transaction\_level)+.5 if raw\_transaction\_level\_fraction else len(raw\_transaction\_level)

product\_link = "https:"+raw\_product\_link[0] if raw\_product\_link else None

yield {

'product\_name':product\_name,

'product\_price':product\_price,

'minimum\_order':minimum\_order,

'seller\_years\_on\_alibaba':seller\_years\_on\_alibaba,

'seller\_name':seller\_name,

'seller\_response\_rate':seller\_response\_rate,

'transaction\_level':transaction\_level,

'product\_link':product\_link,

'search\_text':search\_keyword

}

**Reading search keywords from a file**

Let’s modify the spider to read keywords from a file from a folder called /resources in the project directory and get the products for all those keyword inputs. Lets create the folder and create a CSV file inside it called keywords.csv. Here is how the file looks like if we need to search seperately for headphones and then earplugs.

keyword

headphones

earplugs

Lets use Python’s standard CSV module to read the keywords file.

def parse(self, response):

      """Function to read keywords from keywords file"""

      keywords = csv.DictReader(open(os.path.join(os.path.dirname(\_\_file\_\_),"../resources/keywords.csv")))

      for keyword in keywords:

          search\_text = keyword["keyword"]

          url = "https://www.alibaba.com/trade/search?fsb=y&IndexArea=product\_en&CatId=&SearchText={0}&viewtype=G".format(search\_text)

          yield scrapy.Request(url, callback=self.parse\_listing, meta={"search\_text":search\_text})

**The Full Scrapy Spider Code**

Please see the full code here – <https://github.com/scrapehero/alibaba-scraper>

The spider alibaba\_crawler would look like <https://github.com/scrapehero/alibaba-scraper/blob/master/scrapy_alibaba/spiders/alibaba_crawler.py>

Let’s try to run our scraper using

scrapy crawl alibaba\_crawler

Your out would look similar to this

DEBUG: Forbidden by robots.txt: <GET <https://www.alibaba.com/trade/search?fsb=y&IndexArea=product_en&CatId=&SearchText=headphones&viewtype=G&page=1>>

This is because the Alibaba site has disallowed to crawl all URLs with pattern “/trade”. You can verify this by visiting the robots.txt page which is located at [https://www.alibaba.com/robots.tx](https://www.alibaba.com/robots.txt)t

All spiders created using Scrapy 1.1+ already respect robots.txt by default. You can disable this by setting the variable **ROBOTSTXT\_OBEY = False**.  Now scrapy knows there is no need to check the robots.txt file. It will start crawling the URLs specified the start\_urls list.

**Export Product Data into JSON or CSV using Scrapy**

Scrapy provides in-built CSV and JSON formats.

scrapy crawl <spidername> -o output\_filename.csv -t csv

scrapy crawl <spidername> -o output\_filename.json -t json

To store the output as a CSV file:

scrapy crawl alibaba\_crawler -o alibaba.csv -t csv

For a JSON file:

scrapy crawl alibaba\_crawler -o alibaba.csv -t json

This will create an output file that will be in the same folder as the script.

Here is some sample data extracted from Alibaba.com as CSV

**Known Limitations**

This code should be capable of scraping the details of most product listing pages of Alibaba as long as the structure remains same or similar. In case you see errors related to LXML while scraping, it could be because:

* Anti-Scraping measures of Alibaba.com might have flagged the crawler as a Bot.
* The structure of the website might have changed, making all the selectors we have invalid