# **Scraping website using Scrapy**

**Scrapy:**

Scrapy is an application framework for crawling web sites and extracting structured data which can be used for a wide-range of useful applications, like data mining, information processing or historical archival.

**Install:**

Use the command: *pip install scrapy*

*(If you are using Ubuntu then don’t use the python-scrapy package provided, they are tipycally too old and slow to catch up with the latest version)*

**Some packages needed:**

Scrapy is written in pure Python and depends on a few key Python packages (among others):

**Lxml, parsel, w3lib, twisted, cryptography and pyOpenSSL**

More information on how to install on different system/details about needed packages, please follow the link:

[**https://doc.scrapy.org/en/1.3/**](https://doc.scrapy.org/en/1.3/)

**Basic Concepts:**

**Spiders**

Spiders are classes which define how a certain site (or a group of sites) will be scraped, including how to perform the crawl (i.e. follow links) and how to extract structured data from their pages (i.e. scraping items). In other words, Spiders are the place where you define the custom behaviour for crawling and parsing pages for a particular site (or, in some cases, a group of sites).

**Selectors**

When you’re scraping web pages, the most common task you need to perform is to extract data from the HTML source. Scrapy comes with its own mechanism for extracting data. They’re called selectors because they “select” certain parts of the HTML document specified either by XPath or CSS expressions.

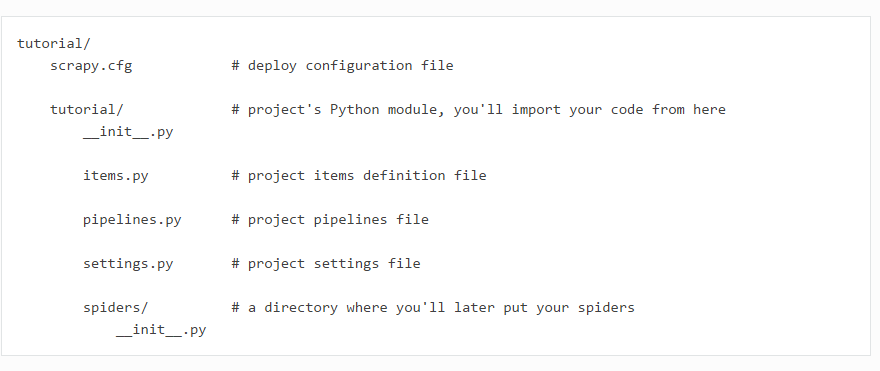
**Items**

To define common output data format Scrapy provides the Item class. **[Item](https://doc.scrapy.org/en/1.3/topics/items.html" \l "scrapy.item.Item)** objects are simple containers used to collect the scraped data. They provide a [dictionary-like](https://docs.python.org/2/library/stdtypes.html" \l "dict) API with a convenient syntax for declaring their available fields.

**Scrapy shell**

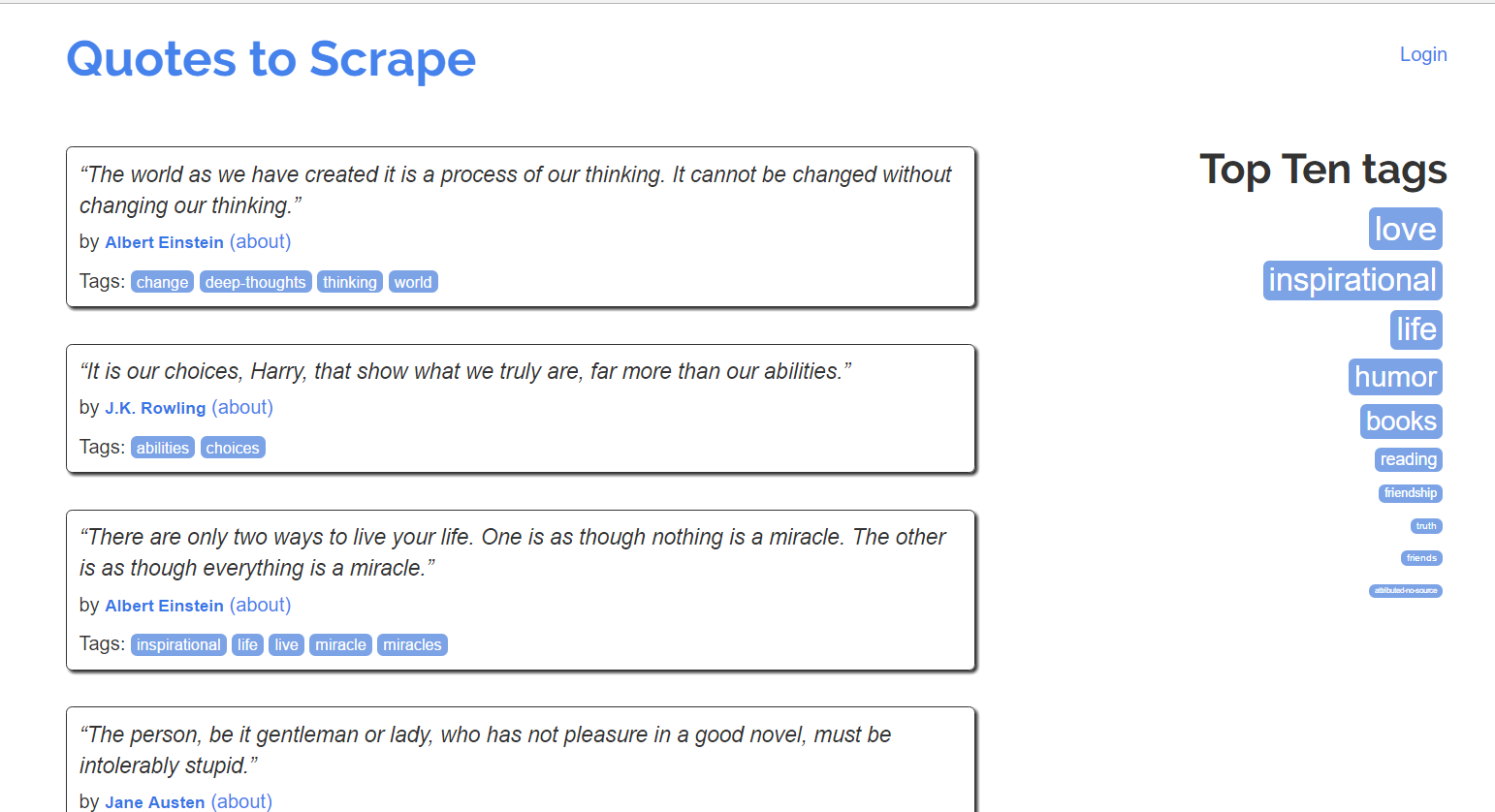
The Scrapy shell is an interactive shell where you can try and debug your scraping code very quickly, without having to run the spider. It’s meant to be used for testing data extraction code, but you can actually use it for testing any kind of code as it is also a regular Python shell.

**Project contents**



**Example 1:**

**The first example is the sample code provided by the Scrapy offical document, it shows how to use Scrapy to extract data from a single website.(** [**http://quotes.toscrape.com/**](http://quotes.toscrape.com/) **)**



1. **Use command to create a project:**

*scrapy startproject tutorial*

1. **Create a spider inside the spider folder:**

**Code:**

**import** scrapy

**class** **QuotesSpider**(scrapy**.**Spider):

*//name identifies the spider, must be unique with a project, run spider with this name, not the documents name*

name **=** "quotes"

*//a list of urls you want to crawl from*

start\_urls **=** [

'http://quotes.toscrape.com/page/1/',

'http://quotes.toscrape.com/page/2/',

]

*//The****[parse()](https://doc.scrapy.org/en/1.3/topics/spiders.html" \l "scrapy.spiders.Spider.parse)****method usually parses the response, extracting the scraped data as dicts and also finding new URLs to follow and creating new requests (****[Request](https://doc.scrapy.org/en/1.3/topics/request-response.html" \l "scrapy.http.Request)****) from them.*

**def** **parse**(self, response):

**for** quote **in** response**.**css('div.quote'):

*//A spider typically generates many dictionaries containing the data extracted from the page, we used yield to do that*

**yield** {

'text': quote**.**css('span.text::text')**.**extract\_first(),

'author': quote**.**css('small.author::text')**.**extract\_first(),

'tags': quote**.**css('div.tags a.tag::text')**.**extract(),

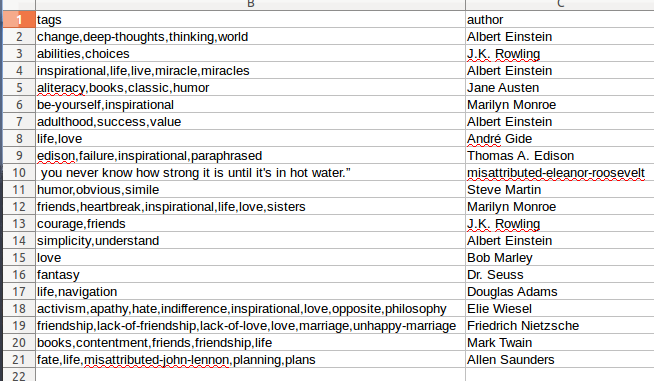
}

1. **Run the spider and store the scraped data:**

**Under the root folder(where the scrapy.cfg exists), type the command:**

*scrapy crawl quotes -o quotes.csv*

**Inside the same folder, we can find our result: quotes.csv**



**Example 2:**

**The second example is the sample code provided by the Data wrangling with Python book in Chapter 12 Spidering the web page 339.**

#### **XPath: a brief intro**

Besides [CSS](https://www.w3.org/TR/selectors), Scrapy selectors also support using [XPath](https://www.w3.org/TR/xpath) expressions.

XPath is a markup pattern selector combining the power of CSS selectors with the ability to traverse the DOM.

Right now we only need to understand the basic concepts about Xpath so we can understand what the code does, there will be a presentation about Xpath later. Also you can check the following link to see more information about Xpath in Scrapy.

[https://doc.scrapy.org/en/1.3/topics/selectors.html#topics-selectors](https://doc.scrapy.org/en/1.3/topics/selectors.html" \l "topics-selectors)

<http://zvon.org/comp/r/tut-XPath_1.html>

**Steps:**

1. **create start project:**

*scrapy startproject emopspider*

**2.Create a spider inside the spider folder:**

*Define the item.py*

*Define the emo\_spider.py*

1. **Run the spider and store the scraped data:**

*scrapy crawl emo -o quotes.csv*

*scrapy crawl quotes -o quotes.json*