Python Web Scraping

tutorialspoint

SIMPLY EASY LEARNING

www.tutorialspoint.com

https://www.facebook.com/tutorialspointindia

https://twitter.com/tutorialspoint

About the Tutorial

Web scraping, also called web data mining or web harvesting, is the process of constructing an agent which can extract, parse, download and organize useful information from the web automatically.

This tutorial will teach you various concepts of web scraping and makes you comfortable with scraping various types of websites and their data.

Audience

This tutorial will be useful for graduates, post graduates, and research students who either have an interest in this subject or have this subject as a part of their curriculum. The tutorial suits the learning needs of both a beginner or an advanced learner.

Prerequisites

The reader must have basic knowledge about HTML, CSS, and Java Script. He/she should also be aware about basic terminologies used in Web Technology along with Python programming concepts. If you do not have knowledge on these concepts, we suggest you to go through tutorials on these concepts first.

Copyright & Disclaimer

Table of Contents

1. Python Web Scraping – Introduction

Web scraping is an automatic process of extracting information from web. This chapter will give you an in-depth idea of web scraping, its comparison with web crawling, and why you should opt for web scraping. You will also learn about the components and working of a web scraper.

What is Web Scraping?

The dictionary meaning of word 'Scrapping' implies getting something from the web. Here two questions arise: What we can get from the web and How to get that.

The answer to the first question is '**data**'. Data is indispensable for any programmer and the basic requirement of every programming project is the large amount of useful data.

The answer to the second question is a bit tricky, because there are lots of ways to get data. In general, we may get data from a database or data file and other sources. But what if we need large amount of data that is available online? One way to get such kind of data is to manually search (clicking away in a web browser) and save (copy-pasting into a spreadsheet or file) the required data. This method is quite tedious and time consuming. Another way to get such data is using **web scraping**.

Web scraping, also called **web data mining** or **web harvesting**, is the process of constructing an agent which can extract, parse, download and organize useful information from the web automatically. In other words, we can say that instead of manually saving the data from websites, the web scraping software will automatically load and extract data from multiple websites as per our requirement.

Origin of Web Scraping

The origin of web scraping is screen scrapping, which was used to integrate non-web based applications or native windows applications. Originally screen scraping was used prior to the wide use of World Wide Web (WWW), but it could not scale up WWW expanded. This made it necessary to automate the approach of screen scraping and the technique called **'Web Scraping'** came into existence.

Web Crawling v/s Web Scraping

The terms Web Crawling and Scraping are often used interchangeably as the basic concept of them is to extract data. However, they are different from each other. We can understand the basic difference from their definitions.

Web crawling is basically used to index the information on the page using bots aka crawlers. It is also called **indexing**. On the hand, web scraping is an automated way of extracting the information using bots aka scrapers. It is also called **data extraction**.

To understand the difference between these two terms, let us look into the comparison table given hereunder:

Web Crawling	Web Scraping
Refers to downloading and storing the contents of a large number of websites.	Refers to extracting individual data elements from the website by using a site-specific structure.
Mostly done on large scale.	Can be implemented at any scale.
Yields generic information.	Yields specific information.
Used by major search engines like Google, Bing, Yahoo. **Googlebot** is an example of a web crawler.	The information extracted using web scraping can be used to replicate in some other website or can be used to perform data analysis. For example the data elements can be names, address, price etc.

Uses of Web Scraping

The uses and reasons for using web scraping are as endless as the uses of the World Wide Web. Web scrapers can do anything like ordering online food, scanning online shopping website for you and buying ticket of a match the moment they are available etc. just like a human can do. Some of the important uses of web scraping are discussed here:

- **E-commerce Websites:** Web scrapers can collect the data specially related to the price of a specific product from various e-commerce websites for their comparison.

- **Content Aggregators:** Web scraping is used widely by content aggregators like news aggregators and job aggregators for providing updated data to their users.

- **Marketing and Sales Campaigns:** Web scrapers can be used to get the data like emails, phone number etc. for sales and marketing campaigns.

- **Search Engine Optimization (SEO):** Web scraping is widely used by SEO tools like SEMRush, Majestic etc. to tell business how they rank for search keywords that matter to them.

- **Data for Machine Learning Projects:** Retrieval of data for machine learning projects depends upon web scraping.

Data for Research: Researchers can collect useful data for the purpose of their research work by saving their time by this automated process.

Components of a Web Scraper

A web scraper consists of the following components:

Web Crawler Module

A very necessary component of web scraper, web crawler module, is used to navigate the target website by making HTTP or HTTPS request to the URLs. The crawler downloads the unstructured data (HTML contents) and passes it to extractor, the next module.

Extractor

The extractor processes the fetched HTML content and extracts the data into semi-structured format. This is also called as a parser module and uses different parsing techniques like Regular expression, HTML Parsing, DOM parsing or Artificial Intelligence for its functioning.

Data Transformation and Cleaning Module

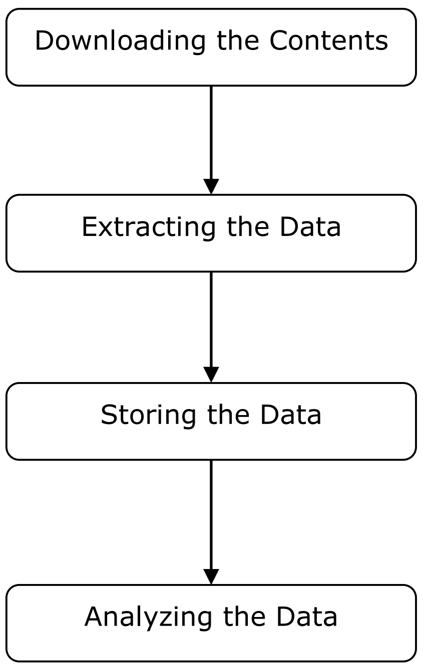
The data extracted above is not suitable for ready use. It must pass through some cleaning module so that we can use it. The methods like String manipulation or regular expression can be used for this purpose. Note that extraction and transformation can be performed in a single step also.

Storage Module

After extracting the data, we need to store it as per our requirement. The storage module will output the data in a standard format that can be stored in a database or JSON or CSV format.

Working of a Web Scraper

Web scraper may be defined as a software or script used to download the contents of multiple web pages and extracting data from it.



We can understand the working of a web scraper in simple steps as shown in the diagram given above.

Step 1: Downloading Contents from Web Pages

In this step, a web scraper will download the requested contents from multiple web pages.

Step 2: Extracting Data

The data on websites is HTML and mostly unstructured. Hence, in this step, web scraper will parse and extract structured data from the downloaded contents.

Step 3: Storing the Data

Here, a web scraper will store and save the extracted data in any of the format like CSV, JSON or in database.

Step 4: Analyzing the Data

After all these steps are successfully done, the web scraper will analyze the data thus obtained.

2. Python Web Scraping – Getting Started with Python

In the first chapter, we have learnt what web scraping is all about. In this chapter, let us see how to implement web scraping using Python.

Why Python for Web Scraping?

Python is a popular tool for implementing web scraping. Python programming language is also used for other useful projects related to cyber security, penetration testing as well as digital forensic applications. Using the base programming of Python, web scraping can be performed without using any other third party tool.

Python programming language is gaining huge popularity and the reasons that make Python a good fit for web scraping projects are as below:

Syntax Simplicity

Python has the simplest structure when compared to other programming languages. This feature of Python makes the testing easier and a developer can focus more on programming.

Inbuilt Modules

Another reason for using Python for web scraping is the inbuilt as well as external useful libraries it possesses. We can perform many implementations related to web scraping by using Python as the base for programming.

Open Source Programming Language

Python has huge support from the community because it is an open source programming language.

Wide range of Applications

Python can be used for various programming tasks ranging from small shell scripts to enterprise web applications.

Installation of Python

Python distribution is available for platforms like Windows, MAC and Unix/Linux. We need to download only the binary code applicable for our platform to install Python. But in case if the binary code for our platform is not available, we must have a C compiler so that source code can be compiled manually.

We can install Python on various platforms as follows:

Installing Python on Unix and Linux

You need to followings steps given below to install Python on Unix/Linux machines:

Step1: Go to the link https://www.python.org/downloads/.

Step2: Download the zipped source code available for Unix/Linux on above link.

Step3: Extract the files onto your computer.

Step4: Use the following commands to complete the installation:

```
run ./configure script
make
make install
```

You can find installed Python at the standard location **/usr/local/bin** and its libraries at **/usr/local/lib/pythonXX,** where XX is the version of Python.

Installing Python on Windows

You need to followings steps given below to install Python on Windows machines:

Step1: Go to the link https://www.python.org/downloads/.

Step2: Download the Windows installer **python-XYZ.msi** file, where XYZ is the version we need to install.

Step3: Now, save the installer file to your local machine and run the MSI file.

Step4: At last, run the downloaded file to bring up the Python install wizard.

Installing Python on Macintosh

We must use **Homebrew** for installing Python 3 on Mac OS X. Homebrew is easy to install and a great package installer.

Homebrew can also be installed by using the following command:

```
$ ruby -e "$(curl -fsSL
https://raw.githubusercontent.com/Homebrew/install/master/install)"
```

For updating the package manager, we can use the following command:

```
$ brew update
```

With the help of the following command, we can install Python3 on our MAC machine:

```
$ brew install python3
```

Setting Up the PATH

You can use the following instructions to set up the path on various environments:

Setting Up the Path on Unix/Linux

Use the following commands for setting up paths using various command shells:

For csh shell

```
setenv PATH "$PATH:/usr/local/bin/python".
```

For bash shell (Linux)

```
 ATH="$PATH:/usr/local/bin/python".
```

For sh or ksh shell

```
 PATH="$PATH:/usr/local/bin/python".
```

Setting Up the Path on Windows

For setting the path on Windows, we can use the path **%path%;C:\Python** at the command prompt and then press Enter.

Running Python

We can start Python using any of the following three ways:

Interactive Interpreter

An operating system such as UNIX and DOS that is providing a command-line interpreter or shell can be used for starting Python.

We can start coding in interactive interpreter as follows:

Step 1: Enter **python** at the command line.

Step 2: Then, we can start coding right away in the interactive interpreter.

```
$python # Unix/Linux

or

python% # Unix/Linux

or

C:> python # Windows/DOS
```

Script from the Command-line

We can execute a Python script at command line by invoking the interpreter. It can be understood as follows –

```
$python script.py # Unix/Linux

or

python% script.py # Unix/Linux

or

C: >python script.py # Windows/DOS
```

Integrated Development Environment

We can also run Python from GUI environment if the system is having GUI application that is supporting Python. Some IDEs that support Python on various platforms are given below:

IDE for UNIX: UNIX, for Python, has IDLE IDE.

IDE for Windows: Windows has PythonWin IDE which has GUI too.

IDE for Macintosh: Macintosh has IDLE IDE which is downloadable as either MacBinary or BinHex'd files from the main website.

3. Python Web Scraping – Python Modules for Web Scraping

In this chapter, let us learn various Python modules that we can use for web scraping.

Python Development Environments using virtualenv

Virtualenv is a tool to create isolated Python environments. With the help of virtualenv, we can create a folder that contains all necessary executables to use the packages that our Python project requires. It also allows us to add and modify Python modules without access to the global installation.

You can use the following command to install **virtualenv**:

```
(base) D:\ProgramData>pip install virtualenv

Collecting virtualenv

  Downloading
https://files.pythonhosted.org/packages/b6/30/96a02b2287098b23b875bc8c2f58071c3
5d2efe84f747b64d523721dc2b5/virtualenv-16.0.0-py2.py3-none-any.whl

(1.9MB)

    100% |████████████████████████████████| 1.9MB 86kB/s

Installing collected packages: virtualenv

Successfully installed virtualenv-16.0.0
```

Now, we need to create a directory which will represent the project with the help of following command:

```
(base) D:\ProgramData>mkdir webscrap
```

Now, enter into that directory with the help of this following command:

```
(base) D:\ProgramData>cd webscrap
```

Now, we need to initialize virtual environment folder of our choice as follows:

```
(base) D:\ProgramData\webscrap>virtualenv websc

Using base prefix 'd:\\programdata'

New python executable in D:\ProgramData\webscrap\websc\Scripts\python.exe

Installing setuptools, pip, wheel...done.
```

Now, activate the virtual environment with the command given below. Once successfully activated, you will see the name of it on the left hand side in brackets.

```
(base) D:\ProgramData\webscrap>websc\scripts\activate
```

We can install any module in this environment as follows:

```
(websc) (base) D:\ProgramData\webscrap>pip install requests

Collecting requests

  Downloading
https://files.pythonhosted.org/packages/65/47/7e02164a2a3db50ed6d8a6ab1d6d60b69
c4c3fdf57a284257925dfc12bda/requests-2.19.1-py2.py3-none-any.whl (9

1kB)

    100% |¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦| 92kB 148kB/s

Collecting chardet<3.1.0,>=3.0.2 (from requests)

  Downloading
https://files.pythonhosted.org/packages/bc/a9/01ffebfb562e4274b6487b4bb1ddec7ca
55ec7510b22e4c51f14098443b8/chardet-3.0.4-py2.py3-none-any.whl (133

kB)

    100% |¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦| 143kB 369kB/s

Collecting certifi>=2017.4.17 (from requests)

  Downloading
https://files.pythonhosted.org/packages/df/f7/04fee6ac349e915b82171f8e23cee6364
4d83663b34c539f7a09aed18f9e/certifi-2018.8.24-py2.py3-none-any.whl

(147kB)

    100% |¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦| 153kB 527kB/s

Collecting urllib3<1.24,>=1.21.1 (from requests)

  Downloading
https://files.pythonhosted.org/packages/bd/c9/6fdd990019071a4a32a5e7cb78a1d92c5
3851ef4f56f62a3486e6a7d8ffb/urllib3-1.23-py2.py3-none-any.whl (133k

B)

    100% |¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦| 143kB 517kB/s

Collecting idna<2.8,>=2.5 (from requests)

  Downloading
https://files.pythonhosted.org/packages/4b/2a/0276479a4b3caeb8a8c1af2f8e4355746
a97fab05a372e4a2c6a6b876165/idna-2.7-py2.py3-none-any.whl (58kB)

    100% |¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦¦| 61kB 339kB/s

Installing collected packages: chardet, certifi, urllib3, idna, requests

Successfully installed certifi-2018.8.24 chardet-3.0.4 idna-2.7 requests-2.19.1
urllib3-1.23
```

For deactivating the virtual environment, we can use the following command:

```
(websc) (base) D:\ProgramData\webscrap>deactivate

(base) D:\ProgramData\webscrap>
```

You can see that (websc) has been deactivated.

Python Modules for Web Scraping

Web scraping is the process of constructing an agent which can extract, parse, download and organize useful information from the web automatically. In other words, instead of manually saving the data from websites, the web scraping software will automatically load and extract data from multiple websites as per our requirement.

In this section, we are going to discuss about useful Python libraries for web scraping.

Requests

It is a simple python web scraping library. It is an efficient HTTP library used for accessing web pages. With the help of **Requests**, we can get the raw HTML of web pages which can then be parsed for retrieving the data. Before using **requests**, let us understand its installation.

Installing Requests

We can install it in either on our virtual environment or on the global installation. With the help of **pip** command, we can easily install it as follows:

```
(base) D:\ProgramData> pip install requests

Collecting requests

Using cached
https://files.pythonhosted.org/packages/65/47/7e02164a2a3db50ed6d8a6ab1d6d60b69
c4c3fdf57a284257925dfc12bda/requests-2.19.1-py2.py3-none-any.whl

Requirement already satisfied: idna<2.8,>=2.5 in d:\programdata\lib\site-
packages (from requests) (2.6)

Requirement already satisfied: urllib3<1.24,>=1.21.1 in
d:\programdata\lib\site-packages (from requests) (1.22)

Requirement already satisfied: certifi>=2017.4.17 in d:\programdata\lib\site-
packages (from requests) (2018.1.18)

Requirement already satisfied: chardet<3.1.0,>=3.0.2 in
d:\programdata\lib\site-packages (from requests) (3.0.4)

Installing collected packages: requests

Successfully installed requests-2.19.1
```

Example

In this example, we are making a GET HTTP request for a web page. For this we need to first import requests library as follows:

```
In [1]: import requests
```

In this following line of code, we use requests to make a GET HTTP requests for the url: https://authoraditiagarwal.com/ by making a GET request.

```
In [2]: r = requests.get('https://authoraditiagarwal.com/')
```

Now we can retrieve the content by using **.text** property as follows:

```
In [5]: r.text[:200]
```

Observe that in the following output, we got the first 200 characters.

```
Out[5]: '<!DOCTYPE html>\n<html lang="en-US"\n\titemscope
\n\titemtype="http://schema.org/WebSite" \n\tprefix="og: http://ogp.me/ns#"
>\n<head>\n\t<meta charset

="UTF-8" />\n\t<meta http-equiv="X-UA-Compatible" content="IE'
```

Urllib3

It is another Python library that can be used for retrieving data from URLs similar to the **requests** library. You can read more on this at its technical documentation at https://urllib3.readthedocs.io/en/latest/.

Installing `Urllib3`

Using the **pip** command, we can install **urllib3** either in our virtual environment or in global installation.

```
(base) D:\ProgramData>pip install urllib3

Collecting urllib3

Using cached
https://files.pythonhosted.org/packages/bd/c9/6fdd990019071a4a32a5e7cb78a1d92c5
3851ef4f56f62a3486e6a7d8ffb/urllib3-1.23-py2.py3-none-any.whl

Installing collected packages: urllib3

Successfully installed urllib3-1.23
```

Example: Scraping using Urllib3 and BeautifulSoup

In the following example, we are scraping the web page by using **Urllib3** and **BeautifulSoup**. We are using **Urllib3** at the place of requests library for getting the raw data (HTML) from web page. Then we are using **BeautifulSoup** for parsing that HTML data.

```
import urllib3

from bs4 import BeautifulSoup

http = urllib3.PoolManager()

r = http.request('GET', 'https://authoraditiagarwal.com')

soup = BeautifulSoup(r.data, 'lxml')

print (soup.title)

print (soup.title.text)
```

This is the output you will observe when you run this code:

```
<title>Learn and Grow with Aditi Agarwal</title>

Learn and Grow with Aditi Agarwal
```

Selenium

It is an open source automated testing suite for web applications across different browsers and platforms. It is not a single tool but a suite of software. We have selenium bindings for Python, Java, C#, Ruby and JavaScript. Here we are going to perform web scraping by using selenium and its Python bindings. You can learn more about Selenium with Java on the link **https://www.tutorialspoint.com/selenium**.

Selenium Python bindings provide a convenient API to access Selenium WebDrivers like Firefox, IE, Chrome, Remote etc. The current supported Python versions are 2.7, 3.5 and above.

Installing Selenium

Using the **pip** command, we can install **urllib3** either in our virtual environment or in global installation.

```
pip install selenium
```

As selenium requires a driver to interface with the chosen browser, we need to download it. The following table shows different browsers and their links for downloading the same.

Chrome	https://sites.google.com/a/chromium.org/chromedriver/downloads
Edge	https://developer.microsoft.com/en-us/microsoft-edge/tools/webdriver/
Firefox	https://github.com/mozilla/geckodriver/releases
Safari	https://webkit.org/blog/6900/webdriver-support-in-safari-10/

Example

This example shows web scraping using selenium. It can also be used for testing which is called selenium testing.

After downloading the particular driver for the specified version of browser, we need to do programming in Python.

First, need to import **webdriver** from selenium as follows:

```
from selenium import webdriver
```

Now, provide the path of web driver which we have downloaded as per our requirement:

```
path = r'C:\\Users\\gaurav\\Desktop\\Chromedriver'

browser = webdriver.Chrome(executable_path = path)
```

Now, provide the url which we want to open in that web browser now controlled by our Python script.

```
browser.get('https://authoraditiagarwal.com/leadershipmanagement')
```

We can also scrape a particular element by providing the xpath as provided in lxml.

```
browser.find_element_by_xpath('/html/body').click()
```

You can check the browser, controlled by Python script, for output.

Scrapy

Scrapy is a fast, open-source web crawling framework written in Python, used to extract the data from the web page with the help of selectors based on XPath. Scrapy was first released on June 26, 2008 licensed under BSD, with a milestone 1.0 releasing in June 2015. It provides us all the tools we need to extract, process and structure the data from websites.

Installing Scrapy

Using the **pip** command, we can install **urllib3** either in our virtual environment or in global installation.

```
pip install scrapy
```

For more detail study of Scrapy you can go to the link https://www.tutorialspoint.com/scrapy/index.htm.