INTRODUCTION TO DATA SCIENCE

TA class V – BeautifulSoup

TA : Lee Chi-Hsuan

- Web Scraping vs Web Crawling
- HTML
- Regular Expression
- Requests
- BeautifulSoup

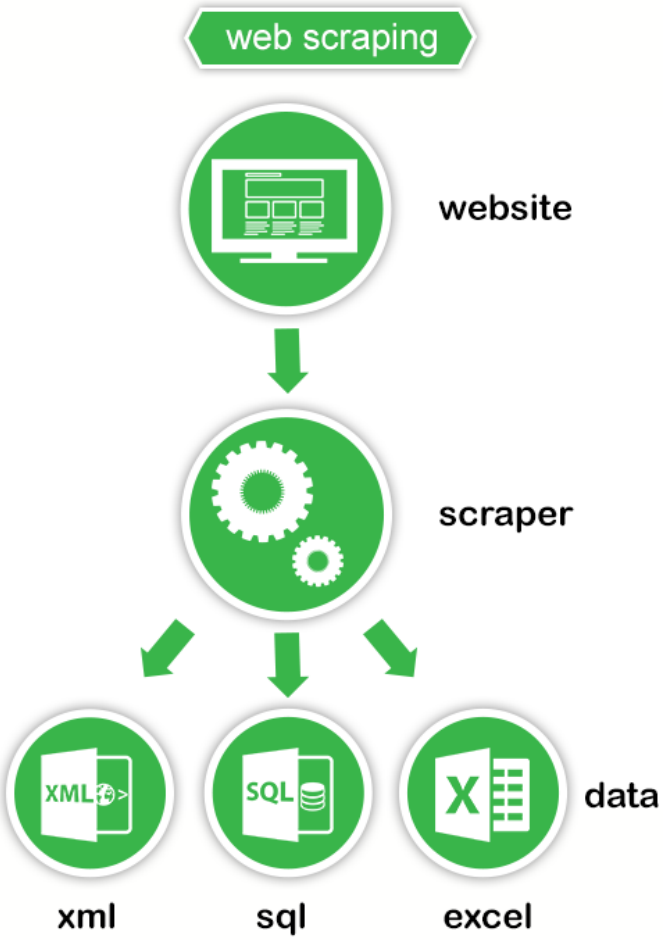
Web Scraping / Crawling

Web Scraping :

- Basically extracting data from websites in an automated manner
- Uses bots to scrape the information or content from websites

Web Crawling :

- An internet bot that systematically browses the World Wide Web, usually for the purpose of web indexing.
- It involves looking at a page in its entirety and indexing it, including its last letter and dot on the page, in the quest for information.


web scraping
website
scraper
XML
SQL
X
data
xml
sql
excel

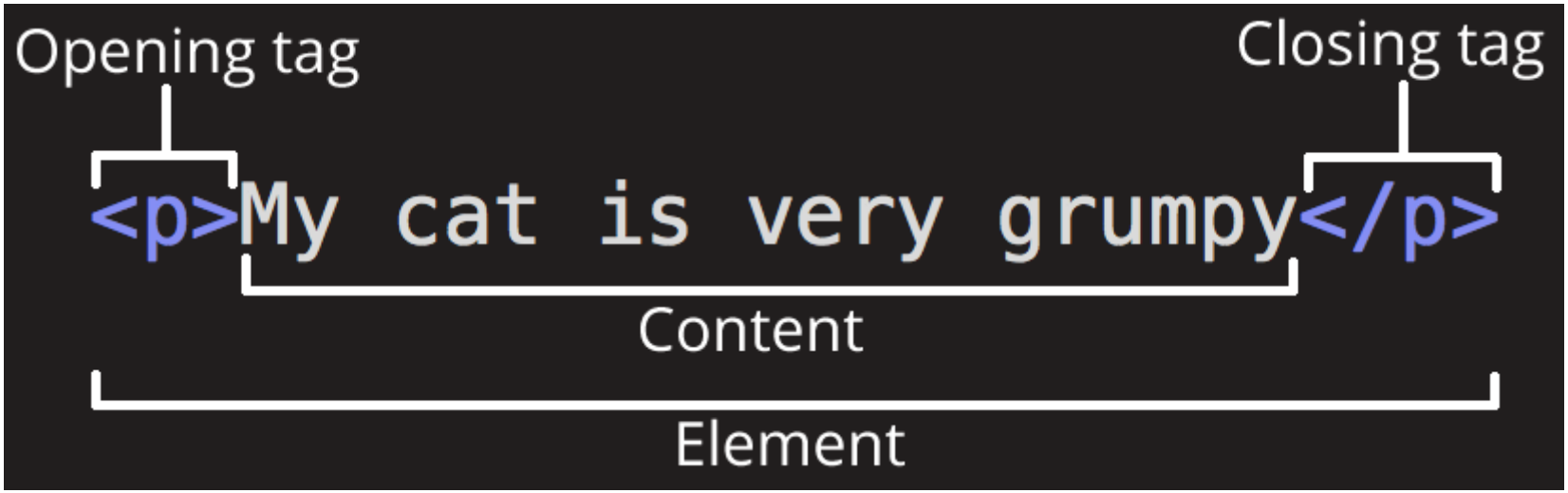

web crawler
crawler
visit all links
build list
indexing
store in database

Steps of Web Scraping :

1. Find the URL that you want to scrape
2. Inspecting the Page
3. Find the data you want to extract
4. Write the code
5. Run the code and extract the data
6. Store the data in the required format

HTML

- HyperText Markup Language
- The standard markup language for documents designed to be displayed in a web browser
- Can be assisted by CSS or JavaScript


Opening tag
Closing tag
My cat is very grumpy
Content
Element

Regular Expression

- A sequence of characters that specifies a search pattern.
- Such patterns are used by string-searching algorithms for "find" or "find and replace" operations on strings, or for input validation.

Metacharacter	Description
`^`	Matches the starting position within the string. In line-based tools, it matches the starting position of any line.
`.`	Matches any single character (many applications exclude newlines, and exactly which characters are considered newlines is flavor-, character-encoding-, and platform-specific, but it is safe to assume that the line feed character is included). Within POSIX bracket expressions, the dot character matches a literal dot. For example, `a.c` matches "abc", etc., but `[a.c]` matches only "a", ".", or "c".
`[ ]`	A bracket expression. Matches a single character that is contained within the brackets. For example, `[abc]` matches "a", "b", or "c". `[a-z]` specifies a range which matches any lowercase letter from "a" to "z". These forms can be mixed: `[abcx-z]` matches "a", "b", "c", "x", "y", or "z", as does `[a-cx-z]`. The `-` character is treated as a literal character if it is the last or the first (after the `^`, if present) character within the brackets: `[abc-]`, `[-abc]`. Note that backslash escapes are not allowed. The `]` character can be included in a bracket expression if it is the first (after the `^`) character: `[]abc]`.
`[^ ]`	Matches a single character that is not contained within the brackets. For example, `[^abc]` matches any character other than "a", "b", or "c". `[^a-z]` matches any single character that is not a lowercase letter from "a" to "z". Likewise, literal characters and ranges can be mixed.
`$`	Matches the ending position of the string or the position just before a string-ending newline. In line-based tools, it matches the ending position of any line.
`( )`	Defines a marked subexpression. The string matched within the parentheses can be recalled later (see the next entry, `\n`). A marked subexpression is also called a block or capturing group. **BRE mode requires** `\( \)`.
`\n`	Matches what the *n*th marked subexpression matched, where *n* is a digit from 1 to 9. This construct is vaguely defined in the POSIX.2 standard. Some tools allow referencing more than nine capturing groups. Also known as a backreference.
`*`	Matches the preceding element zero or more times. For example, `ab*c` matches "ac", "abc", "abbbc", etc. `[xyz]*` matches "", "x", "y", "z", "zx", "zyx", "xyzzy", and so on. `(ab)*` matches "", "ab", "abab", "ababab", and so on.
`{m,n}`	Matches the preceding element at least *m* and not more than *n* times. For example, `a{3,5}` matches only "aaa", "aaaa", and "aaaaa". This is not found in a few older instances of regexes. **BRE mode requires** `\{m,n\}`.

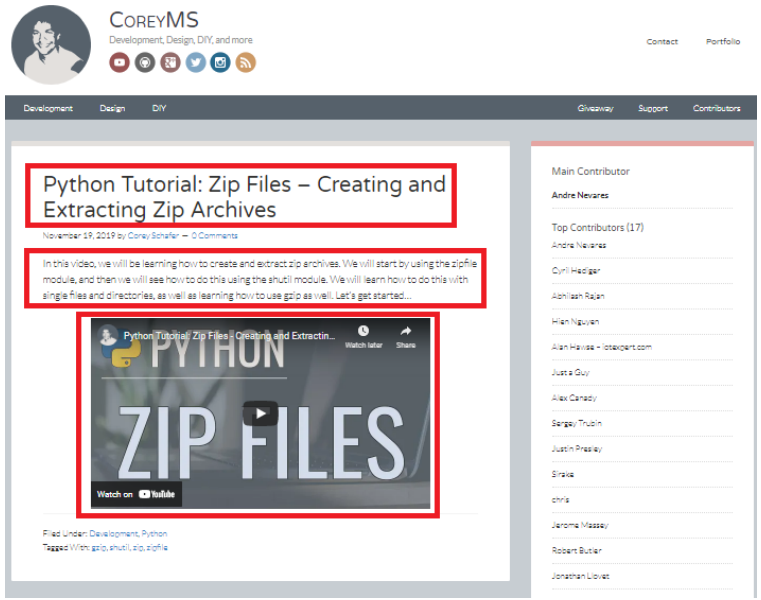
字元	描述
\	將下一個字元標記為一個特殊字元（File Format Escape，清單見本表）、或一個原義字元（Identity Escape，有^$()*+?.[\{|共計12個）、或一個向後參照（backreferences）、或一個八進位跳脫符。例如，「 n 」符合字元「 n 」。「 \n 」符合一個換行符。序列「 \\ 」符合「 \ 」而「 \(」則符合「 (」。
^	符合輸入字串的開始位置。如果設定了RegExp物件的Multiline屬性，^也符合「 \n 」或「 \r 」之後的位置。
$	符合輸入字串的結束位置。如果設定了RegExp物件的Multiline屬性，$也符合「 \n 」或「 \r 」之前的位置。
*	符合前面的子表達式零次或多次。例如，zo*能符合「 z 」、「 zo 」以及「 zoo 」。*等價於{0,}。
+	符合前面的子表達式一次或多次。例如，「 zo+ 」能符合「 zo 」以及「 zoo 」，但不能符合「 z 」。+等價於{1,}。
?	符合前面的子表達式零次或一次。例如，「 do(es)? 」可以符合「 does 」中的「 do 」和「 does 」。?等價於{0,1}。
{n}	n是一個非負整數。符合確定的n次。例如，「 o{2} 」不能符合「 Bob 」中的「 o 」，但是能符合「 food 」中的兩個o。
{n,}	n是一個非負整數。至少符合n次。例如，「 o{2,} 」不能符合「 Bob 」中的「 o 」，但能符合「 foooood 」中的所有o。「 o{1,} 」等價於「 o+ 」。「 o{0,} 」則等價於「 o* 」。
{n,m}	m和n均為非負整數，其中n<=m。最少符合n次且最多符合m次。例如，「 o{1,3} 」將符合「 fooooood 」中的前三個o。「 o{0,1} 」等價於「 o? 」。請注意在逗號和兩個數之間不能有空格。
?	非貪心量化（Non-greedy quantifiers）：當該字元緊跟在任何一個其他重複修飾詞（*,+,?，{n}，{n,}，{n,m}）後面時，符合模式是**非**貪婪的。非貪婪模式儘可能少的符合所搜尋的字串，而預設的貪婪模式則儘可能多的符合所搜尋的字串。例如，對於字串「 oooo 」，「 o+? 」將符合單個「 o 」，而「 o+ 」將符合所有「 o 」。
.	符合除「 \r 」「 \n 」之外的任何單個字元。要符合包括「 \r 」「 \n 」在內的任何字元，請使用像「 (.|\r|\n) 」的模式。
(pattern)	符合pattern並取得這一符合的子字串。該子字串用於向後參照。所取得的符合可以從產生的Matches集合得到，在VBScript中使用SubMatches集合，在JScript中則使用$0…$9屬性。要符合圓括號字元，請使用「 \(」或「 \) 」。可帶數量字尾。
(?:pattern)	符合pattern但不取得符合的子字串（shy groups），也就是說這是一個非取得符合，不儲存符合的子字串用於向後參照。這在使用或字元「 (|) 」來組合一個模式的各個部分是很有用。例如「 industr(?:y|ies) 」就是一個比「 industry|industries 」更簡略的表達式。
(?=pattern)	正向肯定預查（look ahead positive assert），在任何符合pattern的字串開始處符合尋找字串。這是一個非取得符合，也就是說，該符合不需要取得供以後使用。例如，「 Windows(?=95|98|NT|2000) 」能符合「 Windows2000 」中的「 Windows 」，但不能符合「 Windows3.1 」中的「 Windows 」。預查不消耗字元，也就是說，在一個符合發生後，在最後一次符合之後立即開始下一次符合的搜尋，而不是從包含預查的字元之後開始。
(?!pattern)	正向否定預查（negative assert），在任何不符合pattern的字串開始處符合尋找字串。這是一個非取得符合，也就是說，該符合不需要取得供以後使用。例如「 Windows(?!95|98|NT|2000) 」能符合「 Windows3.1 」中的「 Windows 」，但不能符合「 Windows2000 」中的「 Windows 」。預查不消耗字元，也就是說，在一個符合發生後，在最後一次符合之後立即開始下一次符合的搜尋，而不是從包含預查的字元之後開始

Requests

- An elegant and simple HTTP library for Python, built for human beings.
- Get
- Post
- Header

BeautifulSoup

- A Python library for pulling data out of HTML and XML files.



	headline	summary	video_link
0	Python Tutorial: Zip Files – Creating and Extr...	In this video, we will be learning how to crea...	https://youtube.com/watch?v=z0gguhEmWiY
1	Python Data Science Tutorial: Analyzing the 20...	In this Python Programming video, we will be l...	https://youtube.com/watch?v=_P7X8tMplsw
2	Python Multiprocessing Tutorial: Run Code in P...	In this Python Programming video, we will be l...	https://youtube.com/watch?v=fKI2JW_qrso
3	Python Threading Tutorial: Run Code Concurrent...	In this Python Programming video, we will be l...	https://youtube.com/watch?v=IEEhzQoKtQU
4	Update (2019-09-03)	Hey everyone. I wanted to give you an update o...	NaN
5	Python Quick Tip: The Difference Between "==" ...	In this Python Programming Tutorial, we will b...	https://youtube.com/watch?v=mO_dS3rXDIs
6	Python Tutorial: Calling External Commands Usi...	In this Python Programming Tutorial, we will b...	https://youtube.com/watch?v=2Fp1N6dof0Y
7	Visual Studio Code (Windows) – Setting up a Py...	In this Python Programming Tutorial, we will b...	https://youtube.com/watch?v=-nh9rCzPJ20
8	Visual Studio Code (Mac) – Setting up a Python...	In this Python Programming Tutorial, we will b...	https://youtube.com/watch?v=06I63_p-2A4
9	Clarifying the Issues with Mutable Default Arg...	In this Python Programming Tutorial, we will b...	https://youtube.com/watch?v=_JGmemuINww

Let’s have a quick walkthrough

Imports

```
import pandas as pd
import numpy as np
import requests
import re
import csv
from bs4 import BeautifulSoup
```

Web Scraping vs Web Crawling

```
Web scraping :
The process of processing a web document and extracting information out of it.
You can do web scraping without doing web crawling.

Web crawling :
The process of iteratively finding and fetching web links starting from a list of seed URL's.
Strictly speaking, to do web crawling, you have to do some degree of web scraping (to extract the URL's)
```