2. Web data collection basics

- DOM based parsing
- HTML, XML, CSS

Sogang University Department of Business Administration
Professor Myung Suk Kim

Web basic terms

◆ Client and server

A client / server represents a role relationship between two computer programs.
A client is a program that requests a service from another program, and a server is a program that responds to the request. In the case of the Internet, for example, a web browser is a client program that requests the transmission of a web page or file to a web server located somewhere on the internet.

◆ Web browser

Software for using Internet World Wide Web services. It supports HTTP, the protocol of the World Wide Web, and can open and display HTML documents. Originally, the term browser refers to user software used to search for information, but in most cases, a browser means a web browser to use the Internet World Wide Web. (Example : Google Chrome)

◆ URL (Universal Resource Locator)

Addresses of data (files) uploaded on the Internet . The address contains 3 elements.
The first is the protocol type used to access files such as HTTP for web pages and FTP for sites.
The second is the domain name or IP address of the server where the file is located.
The last element is the pathname of the file indicating the location of the file.
For example, for 'URL http://www.britannica.co.kr/world', the browser uses the HTTP protocol , moves to the web server 'www.britannica.co.kr' , and accesses the file name 'world'. indicates that.

Web analytics basic terms

◆ Parsing

The process of decomposing a sentence into its syntactic elements.
Parsing a string and splitting it into a family of elements that can be handled more easily the syntactic structure of a string.
A program that can determine is called a parser. Every compiler or interpreter includes parser.

◆ Scraping

Extracting the necessary information from each page . Programs that do scraping are called scrapers.
Ex) Extracting stock data from Naver Finance

◆ Crawler

A program that automatically repeats information collection
Example) A program that has the function of automatically taking pages of several items in turn, rather than extracting only one individual item.

❖ web page check whether to crawl check

Ex) twitter.com/robots.txt

DOM- based web data collection

- DOM (Documents Object Model) Parsing

  By using the web browsers, programs can retrieve the dynamic content generated by client-side scripts.

  It is also possible to parse web pages (HTML, XML,…) into a D O M tree, based on which programs can retrieve parts of these pages.

  ➡ web top various Resources for optionally collection will do number has exist

- Open API (Application Programming Interface)

  Facebook, Twitter, LinkedIn , etc. provides public and/ or private APIs which can be called using standard code f or retrieving the data in the prescribed format.

  ➡ beforehand decided data only to collect number has exist

Web document types

- HTML (Hyper Text Markup Language):

Data exchange between a client program (with Internet Explorer) and a server program on the web network program. A markup- based (expressed in tags) language

- XML

XML has a structure similar to HTML, but HTML uses a fixed tag name, while XML creates nodes arbitrarily and uses them as tags

- RSS

It is made based on XML and can be parsed simpler than HTML.
the website provides RSS , the webpage Links to RSS included.
Ex) Meteorological

- CSS

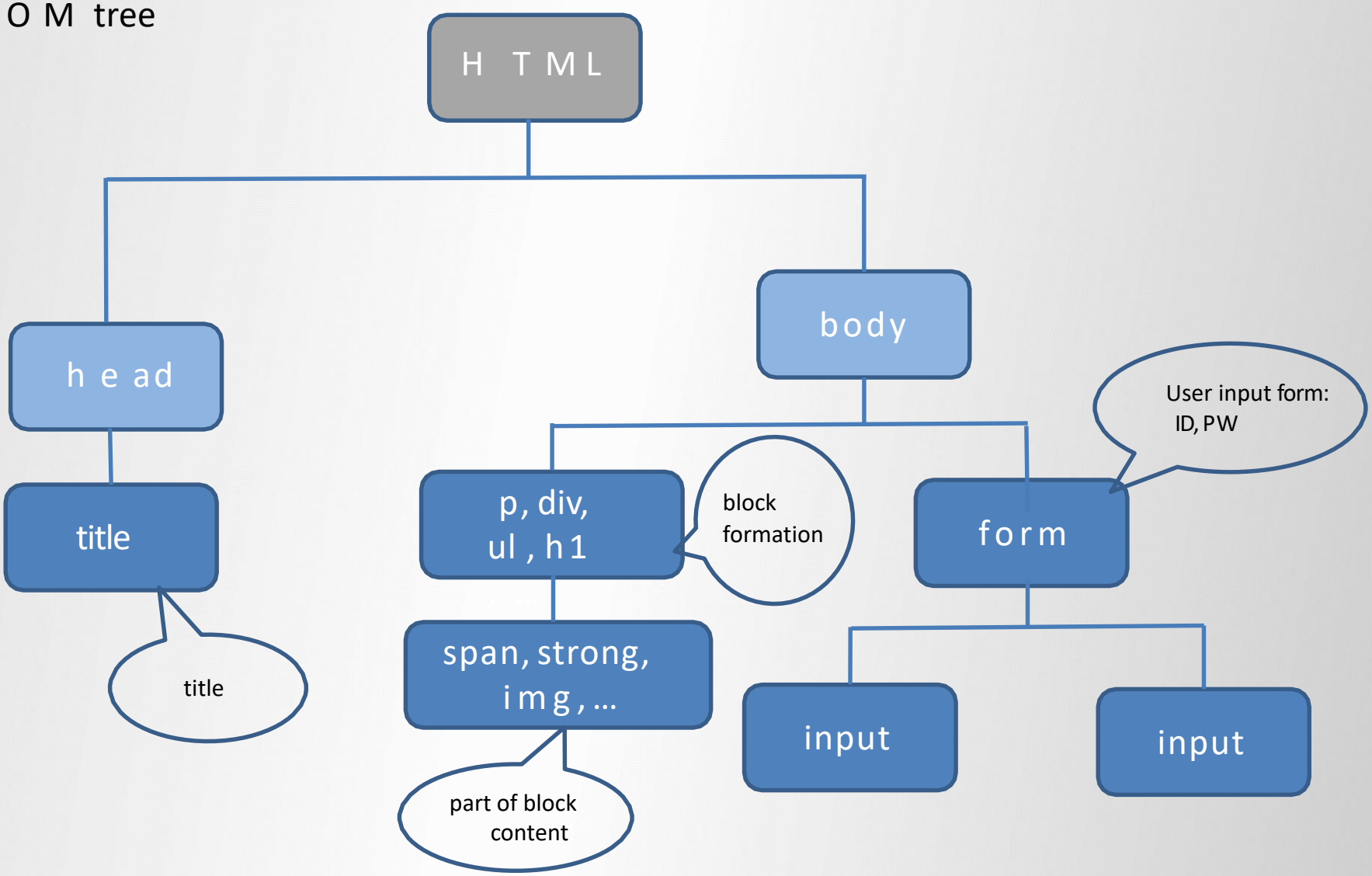
HTML _ document style by definition HTML the elements on screen output shape decision
CSS separates the content and style of a web page, making it easy to apply and change styles.
In an external CSS file, and changes to this file change the style of all web pages to which the style is applied.

- JSON

A data format designed for data storage and exchange, with roots in the JavaScript programming language.
Stored as key - value pairs

HTML (tag-based) hierarchical representation

D O M tree



HTML source code

- chrome/explorer on screen mouse right button by clicking < page sauce View> click

HTML source code

- HTML page source code

새 탭 | Web scraping - Wikipedia | view-source:https://en.wikipedia

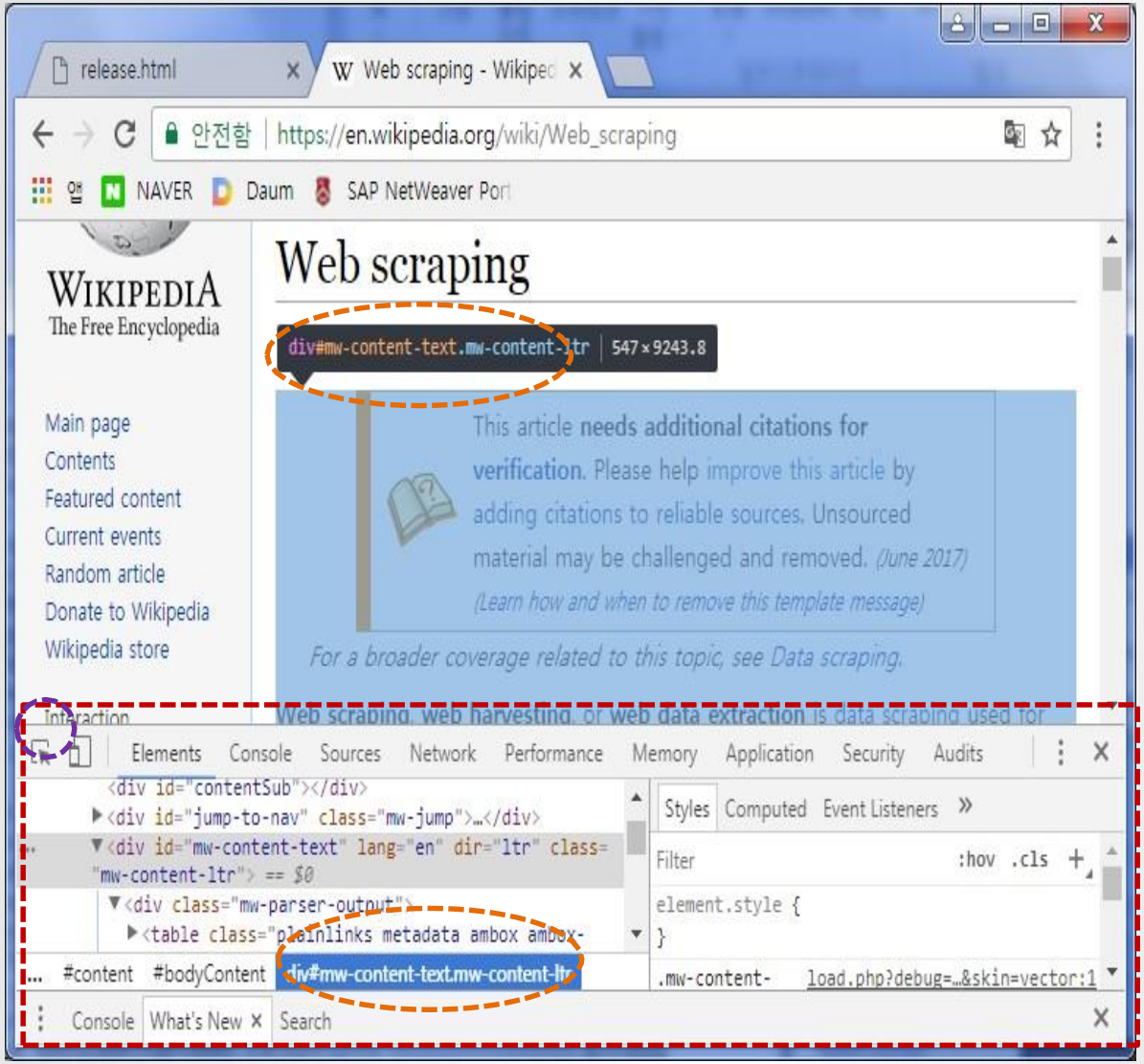
view-source:https://en.wikipedia.org/wiki/Web_scraping


```
<!DOCTYPE html>
<html class="client-nojs" lang="en" dir="ltr">
<head>
<meta charset="UTF-8"/>
<title>Web scraping - Wikipedia</title>
<script>document.documentElement.className="client-js";RLCONF={"wgBreakFrames":!1,"wgSeparatorTransformTable":
["",""],"wgDigitTransformTable":["",""],"wgDefaultDateFormat":"dmy","wgMonthNames":
["","January","February","March","April","May","June","July","August","September","October","November","December"],"wgMonthNamesShort":
["","Jan","Feb","Mar","Apr","May","Jun","Jul","Aug","Sep","Oct","Nov","Dec"],"wgRequestId":"XhTaUwpAAEQAADKPutsAAADR","wgCSPNonce":!1,"wgCan
onicalNamespace":"","wgCanonicalSpecialPageName":!1,"wgNamespaceNumber":0,"wgPageName":"Web_scraping","wgTitle":"Web
scraping","wgCurRevisionId":931071503,"wgRevisionId":931071503,"wgArticleId":2696619,"wgIsArticle":!0,"wgIsRedirect":!1,"wgAction":"view","w
gUserName":null,"wgUserGroups":["*"],"wgCategories":["CS1 Danish-language sources (da)","Articles needing additional references from June
2017","All articles needing additional references","Articles needing additional references from October 2018",
"Articles with limited geographic scope from October 2015","USA-centric","Web
scraping"],"wgPageContentLanguage":"en","wgPageContentModel":"wikitext","wgRelevantPageName":"Web_scraping","wgRelevantArticleId":2696619,"w
gIsProbablyEditable":!0,"wgRelevantPageIsProbablyEditable":!0,"wgRestrictionEdit":[],"wgRestrictionMove":
[],"wgMediaViewerOnClick":!0,"wgMediaViewerEnabledByDefault":!0,"wgPopupsReferencePreviews":!1,"wgPopupsConflictsWithNavPopupGadget":!1,"wgV
isualEditor":{"pageLanguageCode":"en","pageLanguageDir":"ltr","pageVariantFallbacks":"en"},"wgMFDisplayWikibaseDescriptions":
{"search":!0,"nearby":!0,"watchlist":!0,"tagline":!1},"wgWMESchemaEditAttemptStepOversample":!1,"wgULSCurrentAutonym":"English","wgNoticePro
ject":"wikipedia","wgWikibaseItemId":"Q665452","wgCentralAuthMobileDomain":!1,"wgEditSubmitButtonLabelPublish":!0}};RLSTATE=
{"ext.globalCssJs.user.styles":"ready","site.styles":"ready","noscript":"ready","user.styles":"ready",
"ext.globalCssJs.user":"ready","user":"ready","user.options":"ready","user.tokens":"loading","ext.cite.styles":"ready","mediawiki.legacy.sha
red":"ready","mediawiki.legacy.commonPrint":"ready","mediawiki.toc.styles":"ready","wikibase.client.init":"ready","ext.visualEditor.desktopA
rticleTarget.noscript":"ready","ext.uls.interlanguage":"ready","ext.wikimediaBadges":"ready","mediawiki.skinning.interface":"ready","skins.v
ector.styles":"ready"};RLPAGEMODULES=["ext.cite.ux-
enhancements","site","mediawiki.page.startup","mediawiki.page.ready","mediawiki.toc","mediawiki.searchSuggest","ext.gadget.ReferenceTooltips
","ext.gadget.watchlist-notice","ext.gadget.DRN-wizard","ext.gadget.charinsert","ext.gadget.refToolbar","ext.gadget.extra-toolbar-
buttons","ext.gadget.switcher","ext.centralauth.centralautologin","mmv.head","mmv.bootstrap.autostart","ext.popups","ext.visualEditor.deskto
pArticleTarget.init","ext.visualEditor.targetLoader","ext.eventLogging","ext.wikimediaEvents","ext.navigationTiming",
"ext.uls.compactlinks","ext.uls.interface","ext.cx.eventlogging.campaigns","ext.quicksurveys.init","ext.centralNotice.geoIP","ext.centralNot
ice.startUp","skins.vector.js"];</script>
<script>(RLQ=window.RLQ||[]).push(function(){mw.loader.implement("user.tokens@tffin",function($,jQuery,require,module)
```


오후 1:38
2020-01-10

HTML / CSS elements quest

- HTML / CSS Element Explorer (element inspector)



1) In the case of Chrome or Firefox, press [F12] in Developer Tools Heat

2) on the upper left of the developer tool and read a specific node of the selected

3) Right click and scroll into view to

HTML text

- HTML elements

division	tag and property
Heading	<h1>, <h2>,… ,<h6>: Largest heading, second largest heading, etc. letter size
Paragraph	<p>: Paragraph elements
Formatting	<b>: bold, : bold, <i>: italic, : italic, <del>: delete, <ins> insert, <mark>: highlight, <small>: small letter, <sub>: 아랫첨자, <sup>: 윗첨자
List	: Unordered bulleted list, : Ordered list, : Individual List item
Group	: Division or section (multiple other elements grouping), : Element middle part contents only separately grouping
table	<table>: row defined as <tr> , title defined as <th> , cell value as <td> Justice
Link	<a>: click and move linked text or image box.
Form	<form>: defines a form . Sending data to the server through a form relay, <input>: collect data from user , such as <select> <textarea> function <option>: <select> within user to choose number there is of items List Justice

HTML

- HTML representation (<tagname> content </tagname>)

Example) This document represents a typical HTML structure

- HTML Elements value

#The <book> element has three attributes: lang, format, and pages (examples 1 and 2 have the sa me result)

Example 1) <book lang =eng format =paperback pages = 128>

Example 2)
```
<book>
    <lang>eng</lang>
    <format>paperback</format>
    <pages>128</pages>
</book>
```

CSS

- **Cascading Styles (CSS) Sheet)**

  - HTML 's document style by definition HTML the elements on screen output shape decision
  - CSS makes it easy to apply and change styles by separating the content and style
  - Styles are usually stored in an external CSS file (css extension) , and the style can be changed by changing this file.

  Change the style of any web page you apply is possible

CSS

- With selectors and declaration blocks (property names and property values) composition

1. the selector It is used to find HTML elements, and the element name (HTML tag name), id , or class is used as a selector.

   1) element name selectors : specific HTML tag name

   2) Id Selector: Selects elements with a specific id attribute.

   Append # before the element's id attribute name

   1) Class Selector: Specific class properties having element selection.

   Attach . before the elements class property name

2. Element selector and class selector together using specific element only appointed possible

   Ex) < p calss =" center large">This is center and large.</p> # p and class: elements selector

3. Group using elements of the same style possible

CSS

selector { declaration block (property name : property value) }
selector { property1: value1 ; property2: value2; }

HTML tag name { attribute name 1: attribute value 1; attribute name 2: attribute value 2; }
p { color: blue ; text-align: center ; font-size: 10px ; } # element name with p _ case

#s1 { background-color: light blue ; font-style: italic ; } If you have #id="s1" attribute
.center { color: blue ; text-align: center ; } with #class="center" attribute case

center { color:blue ; text-align: center ; } Applies only p elements with #class="center" attribute
h1, h2, p { color: blue ; text-align: center ; } # h1 , h2, p grouping

Web scrapping : Xpath and CSS selectors

- Xpath and CSS Selectors

CSS selector	Xpath	detail
title	//title	All <title> nodes select
div i	//div/i	below the node <i>
p>i	//p/i	<i> immediately below the <p> node select
div[genre="drama"]	//div[@ genre="drama"]	g enre (attribute) value node with drama (value) select

- Nodes using CSS selectors select

selector	detail
*	all nodes select
name	Node with node name name select
.class1	Nodes with class="class1" select
#id1	Node with id="id1" select
[attr]	Select nodes with
[attr="value"]	A node whose attribute name is attr and whose attribute value is value1 select
:nth-child(n)	nth child node select

Web scrapping : CSS selectors

- Combining

combine selectors structure	
A B	Among the descendant nodes of the node matched with A, the node matched with B node
A>B	to A due to matched node's child node middle to b matched node
A to B	to A due to matched node's sibling node middle to b matched node
A+B	to A due to matched node's as soon as back sibling node to b matched node

CSS selector specification format description website
https://wickedmagic.tistory.com/563
https://junistory.blogspot.com/2017/08/css3.html

- Press F12 , select the corresponding part and right mouse Click Copy > Copy selector to check

Web Scrapping : Download

urllib.request Various in the library Using functions (planned using functions inside the reques module within the urllib package)
Data can be downloaded using HTTP or FTP , and includes a module that handles URLs

1. URL file (png etc) download and save
1) Method 1: urlretrieve () function
2) Method 2: urlopen () function :

2. IP address of your PC and client Output connection information
url = http://api.aoikujira.com/ip/ini
urlopen (url).read().decode("utf-8")
urllib.parse Using
Using a new url with parameters added to the URL Download

3.1. HTML scraping
(1) BeautifulSoup Various in the library Using function : No URL download function
Analyzing HTTP and XML (extracting the desired part)
(2) lxml Use

3.2. RSS scraping

Web Scrapping : Download + Analysis

4. Specify Parser : HTML case BeautifulSoup (html target , ' html.parser ')
Extract : html.body . tag (node) name
output : print (extract part.string)

5. Content extraction using
(1) Using the find() function Example) find(id="title") Specify only one id
(2) Using the find_all () function . In case of multiple contents
Example) find_all (" a ")[0][' href '] Linked url address pick up

Extract content after URL download : urlopen () + BeautifulSoup

Extract content using
BeautifulSoup.select_one (): Extract one element
Example) select_one (" div #meigen > h1") (Note) #id attribute value , .class attribute value
BeautifulSoup.select (): Extract multiple elements into a list with a CSS selector
Ex) Method 1: find(" ul ", {" class":"items "})
Method 2: select(" div #meigen > ul.items > li")

8. Download
https://en.wikisource.org/wiki/%EC%A0%80%EC%9E%90:%EC%9C%A4%EB%8F%99%EC%A3%BC
List of works : select("#mw-content-text > div > ul > li a") (reference) #id attribute value
.class attribute value

Web scrapping : using CSS + regular expressions

9. regular expression conjugation
Organize

```
li = soup.find_all(href= re.compile ( r"^ https://"))
for e in li: print(e.attrs['href'])
```

urljoin in urllib.parse specific using a function based on the principle url path conversion

Convert URL path (Convert relative path to absolute path)

```
base="http://example.com/html/a.html"
print( urljoin(base, "b.html") ) http://example.com/html/b.html
print( urljoin(base, " ../ index.html") ) http://example.com / index.html
print( urljoin (base, " http: //otherExample.com/wiki") ) http: //otherExample.com/wiki
print( urljoin (base, " // anotherExample.org/test") ) http: // anotherExample.org/test
```

11. Process HTML recursively (Download contents of relative path linked to base URL)
https://docs.python.org/3.5/library/

Download and Save URL File (Exercise)

Download and save

```
import urllib.request
from google.colab import drive
drive.mount ('/content/ gdrive ')

#Specify URL , file name and save path
url = "http://uta.pw/shodou/img/28/214.png"
savename = "/content/gdrive/My Drive/Colab Notebooks/Textmining/test.png"

urllib.request.urlretrieve(url, savename) # download and save

# other way
mem = urllib.request.urlopen(url).read()
with open(savename, mode=" wb ") as f:
    f. write (mem)
```

2. Accessing your PC 's IP verification API and outputting the result

```
import urllib.request
url = " http://api.aoikujira.com/ip/ini " _
res = urllib.request.urlopen(url)
data = res.read()

# convert binary to string
text = data.decode("utf-8")
print(text)

# other way
print( urllib.request.urlopen(url).read().decode("utf-8"))
```

HTML scrapping (Practice)

3. (1) HTML scrapping

```
### (1) BeautifulSoup Use : Reading library
From bs4 import BeautifulSoup

# Target HTML
html="""
<html><body>
<h1> What is scraping ?</h1>
<p> Analyzing web pages </p>
<p> Extracting the desired part </p>
</body></html>
"""

# HTML parsing
soup = BeautifulSoup(html, 'html.parser')

# Extract the desired part
h1 = soup.html.body.h1
p1 = soup.html.body.p
p2 = p1.next_sibling.next_sibling

# Print element's text
print("h1 = " + h1.string)
print("p = " + p1.string)
print("p = " + p2.string)
```

The difference between string and text : string outputs

HTML scrapping (Practice)

```
### (2) lxml Use : Reading library
!sudo apt-get install python3-lxml
!pip install lxml
!pip install cssselect
import lxml.html

tree = lxml.html.fromstring('''
<!DOCTYPE html>
<html>
<head>
    <title>lxml tutorials</title>
</head>
<div>
    <div class="cc cv"><i>Hello</i> World!!!</div>
</div>
</html>''')

# css selector
selectors = tree.cssselect('.cc')

print(len(selectors)) # 1 (iterators)
elements = selectors[0].cssselect('i')[0] # <i>Hello</i>
print(elements.text_content()) # Hello
print(selectors[0].text_content()) # Hello World!!!
print(selectors[0].attrib) # {'class': 'cc cv'}
print(selectors[0].get('class')) # 'cc cv'
```

HTML scrapping (Practice)

```
# xpath
from lxml import etree

root = etree.XML('''<root>
    <TEXT1 class="myitem">one</TEXT1>
    <TEXT2 class="myitem">two</TEXT2>
    <TEXT3 class="myitem">three</TEXT3>
    <v-TEXT4 class="v-list">four</v-TEXT4>
</root>''')

items = root.xpath("//*[re:test(local-name(), '^TEXT.*')]",  namespaces={'re': 'http://exslt.org/regular-expressions'})
for item in items:
    print(item.text)
    print(item.attrib)

items = root.xpath("//*[@class='myitem']")
for item in items:
    print(item.text)
    print(item.attrib)
```

RSS scrapping (Practice)

3. (2) RSS scrapping

```
!pip install feedparser
import feedparser
d = feedparser.parse ('http://www.aladin.co.kr/rss/special_new/351')

for entry in d.entries :
print('Title :', entry.title )
print('Category :', entry.category )
print('Link :', entry.link )
print()
```

HTML parsing (practice)

4. HTML parsing

```
from bs4 import BeautifulSoup

html="""
<html><body>
<h1 id="title"> What is scraping ?</h1>
<p id="body"> Analyzing web pages </p>
<p> Extracting the desired part </p>
</body></html>
"""

# HTML parsing
soup = BeautifulSoup (html, ' html.parser ')

# Extract the desired part with the find() method
title = soup.find(id="title")
body = soup.find(id="body")

# print the text part
print("#title=" + title.string )
print("#body=" + body.string )
```

The difference between string and text : string outputs

HTML parsing (practice)

```
from bs4 import BeautifulSoup
html="""
<html><body>
<ul> _ _
<li> <a href ="http://www.naver.com"> naver</a></li>
<li> <a href ="http://www.daum.net"> daum</a></li>
</ul> _ _
</body></html>
"""

# parse HTML
soup = BeautifulSoup (html, ' html.parser ')

# Extract with find_all () method
links = soup.find_all ("a")
links
linksA = soup.findAll ("a") # same as find_all
links[0].text
links[0].string
print(links[0]. text)
print(links[0].string)

# print the link list
for a in links:
    href = a.attrs['href']
    text = a.string
    print(text, ">", href)
```

HTML/XML (practice)

5. HTML/XML from webpage

```
from bs4 import BeautifulSoup
import urllib.request as req
url = "http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp"

# Get data with urlopen ()
res = req.urlopen(url)

# Analyze with BeautifulSoup
soup = BeautifulSoup (res, " html.parser ")

# Extract the desired data
title = soup.find ("title").string # node name title
wf = soup.find (" wf ").string # nodename wf
print(title)
print( wf )
```

CSS analysis (practice)

6. CSS selectors or Extract details using tag (id, class, etc.)

```
from bs4 import BeautifulSoup
# Analysis target HTML --- (※1)
# (One)
html="""
<html><body>
<div id=" meigen ">
<h1> Wikibooks books </h1>
< ul class="items">
<li> Introduction to Unity Game Effects </li>
<li> iPhone app development textbook starting with Swift </li>
<li> The classics of modern website design </li>
</ul> _ _
</div>
</body></html>
"""

# parse HTML
soup = BeautifulSoup (html, ' html.parser ')

# Extract the necessary parts with a CSS query
# Extract the title part
h1 = soup.select_one (" div#meigen > h1").string
print("h1 =", h1)
```

CSS analysis (practice)

```
# extract the list part
# Method 1: Go through
list1 = soup.find (" ul ", {" class":"items "}) # compare with li_list
list1.find("li").string
list2 = list1.find_all("li") # Since string cannot be used directly in list , use for statement
for one in list2:
print("li =", one. string )

# Method 2: Go to the target directly
li_list = soup.select (" div#meigen > ul.items > li")
li_list = soup.select (" ul.items > li") # Same result as above
for li in li_list :
print("li =", li. string )
```

CSS analysis (practice)

```
html="""
<html><body>
<ul class="list_nav type_fix">
<li class="nav_item">
<a href="https://mail.naver.com/" class="nav" data-clk="svc.mail"><i class="ico_mail"></i>메일</a>
</li>
<li class="nav_item"><a href="https://section.cafe.naver.com/" class="nav" data-clk="svc.cafe">카페</a></li>
<li class="nav_item"><a href="https://section.blog.naver.com/" class="nav" data-clk="svc.blog">블로그</a></li>
<li class="nav_item"><a href="https://kin.naver.com/" class="nav" data-clk="svc.kin">지식iN</a></li>
<li class="nav_item"><a href="https://shopping.naver.com/" class="nav shop" data-
clk="svc.shopping"><span class="blind">쇼핑</span></a></li>
<li class="nav_item"><a href="https://shoppinglive.naver.com/home" class="nav shoplive" data-
clk="svc.shoppinglive"><span class="blind">쇼핑LIVE</span></a></li>
<li class="nav_item"><a href="https://order.pay.naver.com/home" class="nav" data-clk="svc.pay">Pay</a></li>
<li class=" nav_item ">
<a href ="https://tv.naver.com/" class=" nav " data- clk =" svc.tvcast "> < i class=" ico_tv "></ i >TV</a>
</li>
</ul> _ _
</body></html>
"""
```

CSS analysis (practice)

```
# parse HTML
soup = BeautifulSoup(html, 'html.parser')
ul = soup.find("ul", {"class":"list_nav type_fix"})
lis = ul.find_all("li")

dataf = []
for li in lis:
    tag = li.find("a")
    link = tag.attrs['href']
text = tag. text
print(text, link)
```

7. CSS from webpage

```
from bs4 import BeautifulSoup
import urllib.request as req
url = "https://en.wikisource.org/wiki/%EC%A0%80%EC%9E%90:%EC%9C%A4%EB%8F%99%EC%A3%BC"
res = req.urlopen ( url )
soup = BeautifulSoup (res, " html.parser ")
just below the #mw-content-text
#ul _ just below the tag
# under the li tag
# Select all a tags .
a_list = soup.select ("#mw-content-text > div > ul > li a")
for a in a_list :
name = a. string
print("-", name)
```

CSS analysis (practice)

8. CSS from HTML file

```
from bs4 import BeautifulSoup

# Pre-processing command for importing after uploading a file in colab ( path setting )
from google.colab import drive
drive.mount ('/content/ gdrive ')

Upload the file to the location below google drive (+New > File upload) and import
fp = open("/content/ gdrive /My Drive/ Colab Notebooks/ Textmining /books.html", encoding="utf-8")

soup = BeautifulSoup ( fp , " html.parser ")

# How to search with CSS selectors
sel = lambda q : print( soup. select_one (q). string)
sel ("#nu")
sel (" li#nu ")
sel (" ul > li#nu ")
sel ("#bible #nu")
sel ("#bible > #nu")
sel("ul#bible > li#nu")
sel("li[id='nu']")
sel("li:nth-of-type(4)")
# 그 밖의 방법
print(soup.select("li")[3].string)
print(soup.find_all("li")[3].string)
```

CSS analysis (practice)

```
from bs4 import BeautifulSoup
fp = open("fruits-vegetables.html", encoding="utf-8")
soup = BeautifulSoup(fp, "html.parser")

# CSS 선택자로 추출하기
print(soup.select_one("li:nth-of-type(6)").string)  # 작동 안됨
print(soup.select_one("#ve-list > li:nth-of-type(4)").string)
print(soup.select("#ve-list > li[data-lo='us']")[1].string)
print( soup.select ("# ve -list > li.black ")[1].string)

# Extract with find method
cond = {" data-lo":"us ", " class":"black "}
print( soup.find ("li", cond ).string)

# use the find method consecutively
print( soup.find (id=" ve -list").find("li", cond ).string)
```

Regular expression Application (practice)

9. Using regular expressions

```
from bs4 import BeautifulSoup
import re # when using regular expressions
html="""
<ul> _ _
<li> <a href ="hoge.html"> hoge </li>
<li> <a href ="https://example.com/ fuga "> fuga *</li>
<li> <a href ="https://example.com/foo"> foo*</li>
<li> <a href ="http://example.com/ aaa "> aaa </li>
</ul> _ _
"""
soup = BeautifulSoup (html, " html.parser ")

# Extract
li = soup.find_all ( href = re.compile ( r"^https ://"))
for e in li:
print( e.attrs [' href '])
```

URL specification method (practice)

10. URL designation method

```
from urllib.parse import urljoin
base="http://example.com/html/a.html"
print( urljoin (base, "b.html") )
print( urljoin(base, "sub/c.html") )
print( urljoin(base, "../index.html") )
print( urljoin(base, "../img/hoge.png") )
print( urljoin(base, "../css/hoge.css") )

print( urljoin(base, "/hoge.html") )
print( urljoin(base, "http://otherExample.com/wiki") )
print( urljoin(base, "//anotherExample.org/test") )
```

Loading and Saving XML Data (Hands-on)

11. XML analyze

```
from bs4 import BeautifulSoup
import urllib.request as req
import os.path
from google.colab import drive
drive.mount('/content/gdrive')

url = "http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp?stnId=108"
savename = "/content/gdrive/My Drive/Colab Notebooks/Textmining/forecast.xml"
if not os.path.exists(savename):
    req. urlretrieve ( url , savename ) # create XML data

# Analyze with BeautifulSoup
xml = open( savename , "r", encoding="utf-8").read()
soup = BeautifulSoup (xml, ' html.parser ')

info = {} # check each region
for location in soup.find_all ("location"):
name = location.find ('city').string
weather = location.find (' wf ').string
if not (weather in info):
info[weather] = []
info[weather].append(name)

# Output the weather for each region separately
for weather in info.keys ():
print("+", weather)
for name in info[weather]:
print("| - ", name)
```

region	number	region	number
Nationwide	108	Jeonbuk	146
Seoul / Gyeonggi	109	Jeonnam	156
Gangwon	105	Gyeongbuk	143
Chungbuk	131	Gyeongnam	159
Chungnam	133	Jeju	184

Loading and Saving XML Data (Hands-on)

Textmini | data_typ | Success | preproc | 과제.ipy | www.km

주의 요함 | kma.go.kr/weather/forecast/mid-term-rss3.jsp?stnId=108

앱 SAINT NAVER Google deeplearning Crunchbase YouTube 읽기 목록


```
<tm>202111050600</tm>
▼<wf>
  <![CDATA[ ○ (강수) 8일(월)은 전국에 비가 오겠습니다. 9일(화)~10일(수)은 중부지방과 전라권에 비(강원영서 비 또는
  눈)가 오겠고, 10일(수)은 제주도에도 비가 오겠습니다. <br /> 11일(목)은 충남권과 전라권, 제주도에 비가 오겠습니다.
  <br />○ (기온) 8일(월) 아침 기온은 10~18도로 어제(4일, 아침 기온 4~12도)보다 높겠으나, 8일(월) 낮부터 기온이 큰 폭
  으로 떨어져<br /> 8일(월) 오후부터 15일(월)까지 아침 기온은 -1~9도, 낮 기온은 9~20도로 어제(4일, 낮 기온 16~22도)보
  다 크게 낮겠고, 바람도 강하게 불어 춥겠습니다.<br />○ (해상) 8일(월)~11일(목) 대부분 해상에서 물결이 1.0~5.0m로 높
  게 일겠습니다. ]]>
</wf>
</header>
▼<body>
▼<location wl_ver="3">
  <province>서울·인천·경기도</province>
  <city>서울</city>
  ▼<data>
    <mode>A02</mode>
    <tmEf>2021-11-08 00:00</tmEf>
    <wf>흐리고 비</wf>
    <tmn>11</tmn>
    <tmx>11</tmx>
    <reliability/>
    <rnSt>90</rnSt>
  </data>
  ▼<data>
    <mode>A02</mode>
    <tmEf>2021-11-08 12:00</tmEf>
    <wf>흐리고 비</wf>
    <tmn>11</tmn>
    <tmx>11</tmx>
    <reliability/>
    <rnSt>100</rnSt>
  </data>
```

web analytics application

1. How to read the URL

Using #requests

```
import requests
from bs4 import BeautifulSoup

url = "https://www.naver.com/"
req = requests.get ( url )
html = req. text
soup = BeautifulSoup (html, ' html.parser ')
print(soup)


# urllib.request Use
import urllib.request
from bs4 import BeautifulSoup

url = 'https://www.naver.com/'
req = urllib.request.Request ( url )
html = urllib.request.urlopen ( req )
soup = BeautifulSoup (html, ' html.parser ')
print(soup)
```

Web scrapping : download (URL+ parameters)

❖ URL Parameters : Insert parameters into the URL to allow

URL parameters are separated by an equal sign (=) and the first parameter is always followed by a question mark
(I'll try {URL parameter } by Googling)

❖ Example

Using Meteorological Agency RSS (XML data download)
The parameter (stnId) specifies

Korea Meteorological Administration RSS address : http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp

New address with parameters added :
url = http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp ?stnId=108

region	number	region	number
Nationwide	108	Jeonbuk	146
Seoul / Gyeonggi	109	Jeonnam	156
Gangwon	105	Gyeongbuk	143
Chungbuk	131	Gyeongnam	159
Chungnam	133	Jeju	184

Web analytics application

2. URL + parameters

```
import urllib.request
import urllib.parse
url0 = "http://www.kma.go.kr/weather/forecast/mid-term-rss3.jsp"

# URL encode
values = {
' stnId ': '109'
}
params = urllib.parse.urlencode (values)

# Create request-only URL
url = url0 + "?" + params
print(" url =", url )

# download
data = urllib.request.urlopen ( url ).read()
text = data.decode ("utf-8")
print(text)
```

Web analytics application

3. Search stocks on Google

```
############## Search stocks on Google ( eg : tesla stock) and extract the stock price ( two urls have the same result : use the shorter one )
#https://www.google.com/ search?q = tesla+stock&ei =Q9_eYILWCMS4mAX0_p3QBA&o...
#https://www.google.com/search?q=tesla+stock
import requests
from bs4 import BeautifulSoup

url = "https://www.google.com/search?q="
company = "tesla"
req = requests.get(url + company + "+stock")
html = req.text
soup = BeautifulSoup(html, "html.parser")
#tags = soup.select("span.lsqQVc.NprOob.XcVN5d.wT3VGc")  # class = news.tit
#print(price[0].text)

tags = soup.select("div.BNeawe.iBp4i.AP7Wnd")
print(tags)
for tag in tags:
print( tag. text )

tags[0]. text.split (' ')[0]
```

Web analytics application

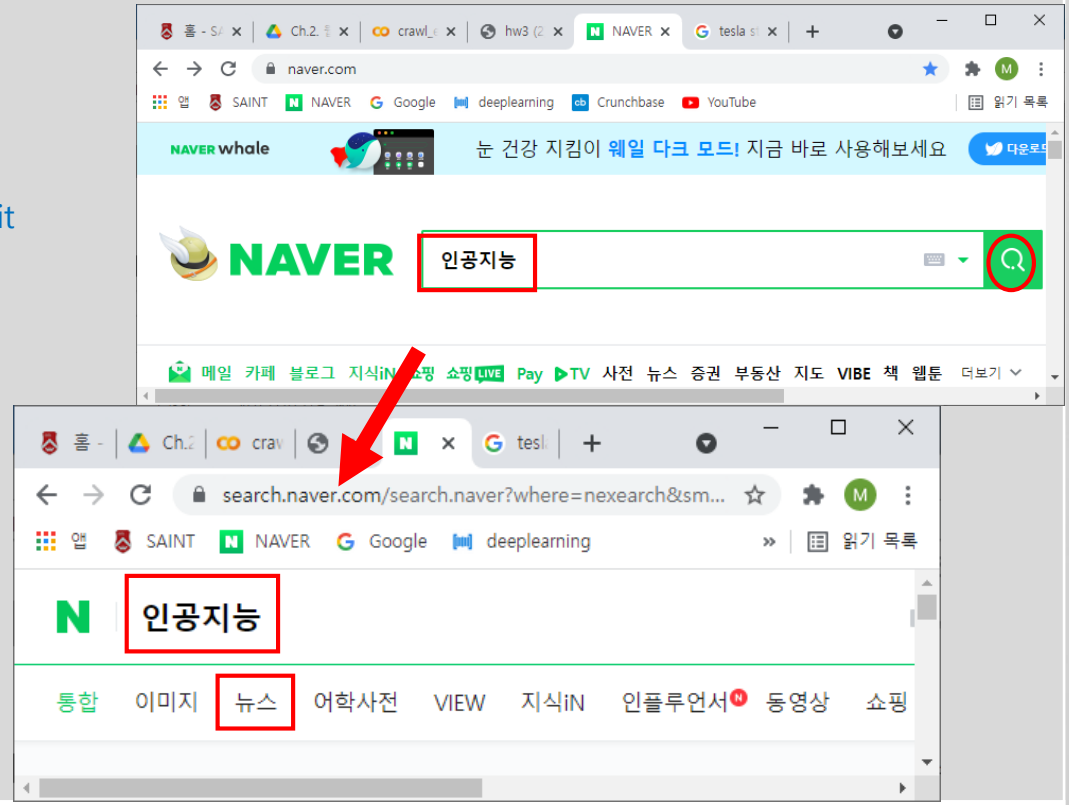
4. Naver keyword-based news search

```
############### Search artificial intelligence ( news ) on Naver
#https://search.naver.com/ search.naver?where = news&sm = tab_jum&query = artificial intelligence
import requests
from bs4 import BeautifulSoup

url = "https://search.naver.com/ search.naver?where = news&sm = tab_jum&query =?"
keyword = " artificial intelligence "
req = requests.get ( url + keyword)
html = req. text
soup = BeautifulSoup (html, " html.parser ")

############# news title and url extraction
titles = soup.select(".news_tit")  # class = news.tit
print(titles[0])
print(titles[0].text)
print(titles[0]["href"])
print(titles[0]["title"])
title_list = []
url_list = []
for title in titles:
    title_list.append(title.text)
    url_list.append(title["href"])

print(title_list)
len(title_list)
print(url_list)
len(url_list)
```



Web analytics application

```
#### Or
title_list = [title["title"] for title in titles]
print(title_list)
url_list = [title["href"] for title in titles]
print(url_list)

############## news content extraction
contents = soup.select(".api_txt_lines")  # class = api_txt_lines
print(contents[0])
print(contents[0].text)

content_list = []
for content in contents:
    content_list.append(content.text)
print(content_list)

#### Or
content_list = [content.text for content in contents]
print(content_list)
```

Web analytics application

```
############## news information building (combine title, url, content)
news_list = zip(title_list, url_list, content_list)
news_info_list = []
for news in news_list:
    news_dict = {
"title": news[0],
" url ": news[1],
"content": news[2]
}

    news_info_list.append ( news_dict )

print( news_info_list )
```

Web analytics application

5. Google play download app Review Scraper

```
##############
# https://strangefate.github.io/2022/01/06/AppReview-googleplayscraper/
# https://github.com/JoMingyu/google-play-scraper
!pip install google-play-scraper

# Google play pokemongo review
# https://play.google.com/store/apps/details?id=com.nianticlabs.pokemongo&hl=en&gl=US
# package name: com.nianticlabs.pokemongo/ Google search: Pokémon GO - Apps on Google Play

# Information about the app
from google_play_scraper import app

result = app(
    'com.nianticlabs.pokemongo',
    lang='en', # defaults to 'en'
    country='us' # defaults to 'us'
)

result
len(result)
type(result)
result['comments']
len(result['comments'])
```

Web analytics application

```
# https://play.google.com/store/apps/details?id=com.kakaobank.channel&hl=ko&gl=kr
# Google saerch: Kakao bank - Apps on Google Play

# Information about the reviews
from google_play_scraper import Sort, reviews

result, continuation_token = reviews(
    'com.kakaobank.channel',
    lang='ko', # defaults to 'en'
    country='kr', # defaults to 'us'
    sort=Sort.NEWEST, # defaults to Sort.NEWEST
    count=10, # defaults to 100
    filter_score_with=None # defaults to None(means all score)
)

result


# If you pass `continuation_token` as an argument to the reviews function at this point,
# it will crawl the items after the previous review items.
result, _ = reviews(
    'com.kakaobank.channel',
    continuation_token=continuation_token # defaults to None(load from the beginning)
)

result
```

Web analytics application

```
# Time based reviews
year = 2022
token = None
review_list = []
while year >= 2021:
  result, continuation_token = reviews(
      'com.kakaobank.channel',
      lang = 'ko', # defaults to 'en'
      country = 'kr', # defaults to 'us'
      sort = Sort.NEWEST, # defaults to Sort.NEWEST
      continuation_token = token,
      count = 10, # defaults to 100
      filter_score_with = None # defaults to None(means all score)
  )
  token = continuation_token
  year = result[-1]['at'].year
  for review in result:
    if review['at'].year >= 2019:
      temp_list = [review['score'], review['content'], review['at']]
      review_list.append(temp_list)

review_list
len(review_list)  # 5270
review_list[0]
```

Web analytics application

```
### Crawl all reviews
from google_play_scraper import Sort, reviews_all

result = reviews_all(
    'com.fantome.penguinisle',
    sleep_milliseconds=1000, # defaults to 0
    lang='en', # defaults to 'en'
    country='us', # defaults to 'us'
    sort=Sort.MOST_RELEVANT, # defaults to Sort.MOST_RELEVANT
    filter_score_with=5 # defaults to None(means all score)
)

result
```

Web analytics application

```
### Permission check
from google_play_scraper import permissions

result = permissions(
    'com.spotify.music',
    lang='en', # defaults to 'en'
    country='us', # defaults to 'us'
)

result



### App search
from google_play_scraper import search

result = search(
    "best Pikachu game",
    lang="en",  # defaults to 'en'
    country="us",  # defaults to 'us'
    n_hits=3  # defaults to 30 (= Google's maximum)
)

result
```