PYTHON XML 이해하기

Moon Yong Joon

1. XML 주요 요소
2. XML 모듈
3. LXML 모듈

1.XML 주요 요소

Moon Yong Joon

xml 주요 모듈 이해

xml 주요 모듈

두 모듈은 대부분 인터페이스가 유사하면 두가지를

xml.etree.ElementTree

lxml.etree

xml document 이해

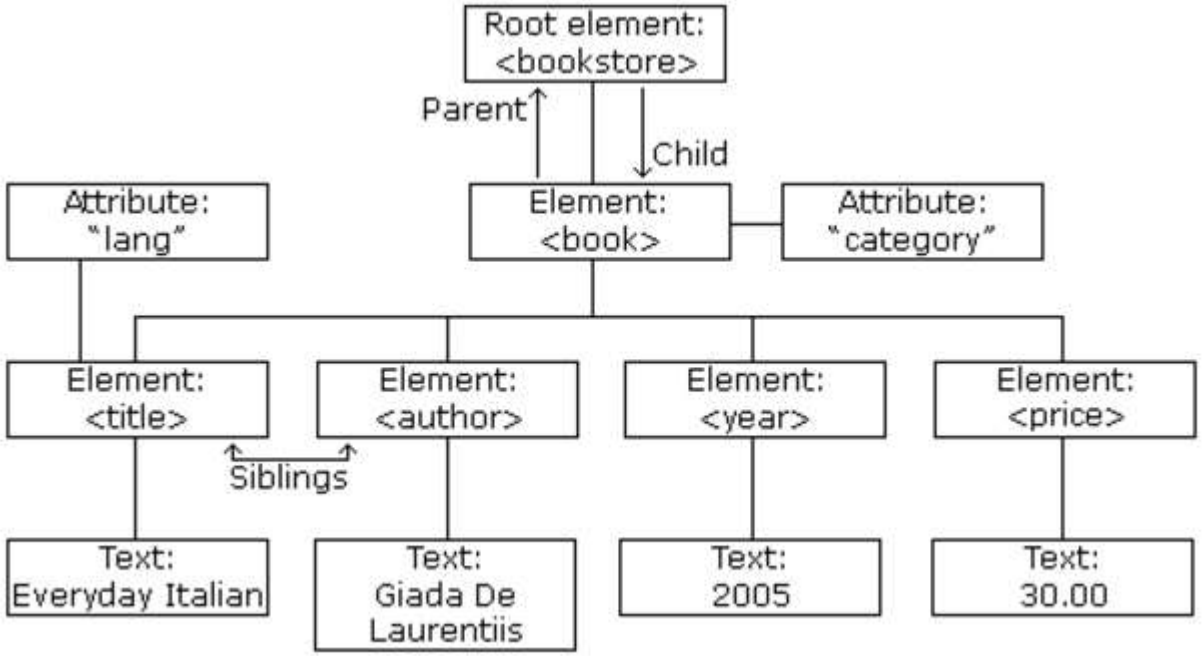
xml 주요 class

ElementTree는 전체 XML 문서를 트리로 나타내고 Element는 이 트리에서 단일 노드를 나타냄. 전체 문서와의 상호 작용 (파일 읽기 및 쓰기)은 일반적으로 ElementTree 수준에서 수행되고, 단일 XML 요소 및 해당 하위 요소와의 상호 작용은 요소 수준에서 수행

Element	단순하지만 유연한 컨테이너 객체로 단순화 된 XML 정보 세트와 같은 계층 적 데이터 구조를 메모리에 저장하도록 설계
ElementTree	XML 파일을 Element 객체들의 트리로 로드하고 다시 저장하기 위한 기능를 추가

xml tree : ElementTree

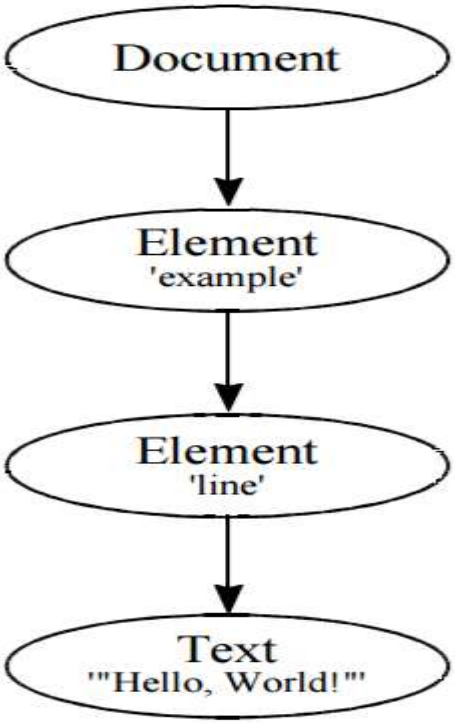
XML 문서는 요소 트리로 구성하며, XML 트리는 루트 요소에서 시작하여 루트 요소에서 하위 요소로 분기, 모든 요소는 하위 요소 (하위 요소)를 가짐



Document 구성

xml 문서를 Element로 구성해서 처리

```
<example>
<line>&quo;Hello,
   world!&quo;</line>
</example>
```



ElementTree type

xml.etree.ElementTree, lxml.etree 모듈에 ElementTree 타입의 공통된 tree 처리 기능

find, findall, findtext, getroot, getiterator, iter, iterfind: tree 내부 검색

parse : Parshing 처리

write : 파일 처리

Element type

계층적 데이터 구조를 메모리에 저장하도록 설계된 유연한 컨테이너 객체

tag : 이 요소가 나타내는 데이터의 종류 (요소 유형, 즉)를 나타내는 문자열

attrib : 파이썬 사전에 저장된 다수의 속성.

text : 내용을 담을 텍스트 문자열 및 후행 텍스트를 보관할 문자열

child element : 파이썬 시퀀스에 저장된 다수의 자식 요소들

Element vs attribute

속성에는 여러 값을 포함 하지 않고, 트리 구조를 포함하지 않으며, 쉽게 확장 할 수 없으므로 속성 생성이 특별히 주의해야 함

Element로 지정

```
<person>
  <gender>female</gender>
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

속성으로 지정

```
<person gender="female">
  <firstname>Anna</firstname>
  <lastname>Smith</lastname>
</person>
```

xml syntax

The XML Prolog

xml은 문서에 prolog를 가짐

```
<?xml version="1.0" encoding="UTF-8"?>
<note>
  <to>Tove</to>
  <from>Jani</from>
  <heading>Reminder</heading>
  <body>Don't forget me this weekend!</body>
</note>
```

반드시 root element 가짐

xml은 반드시 root element를 하나 가져야 함

```
<root>
  <child>
    <subchild>.....</subchild>
  </child>
</root>
```

Must Have a Closing Tag

xml은 반드시 element에 closing tag를 가져야 함

```
<p>This is a paragraph.</p>
<br />
```

Must be Properly Nested

xml은 반드시 element에 closing tag가 한쌍으로 구성되어 내포되어야 함

```
<b><i>This text is bold and italic</b></i>
```

Element 순서가 맞아야 함

```
<b><i>This text is bold and italic</i></b>
```

Entity References

xml 내부에 text 작성 특별한 문자를 사용하도록 제공

&lt;	<	less than
&gt;	>	greater than
&amp;	&	ampersand
&apos;	'	apostrophe
&quot;	"	quotation mark

Comments in XML

xml 내부 주석은 <!-- 시작하고 -->로 종료해야 함

```
<!-- This is a -- comment -->
```

xml name 관리 기준

Element Naming Styles

XML Naming Styles

Style	Example	Description
Lower case	<firstname>	All letters lower case
Upper case	<FIRSTNAME>	All letters upper case
Underscore	<first_name>	Underscore separates words
Pascal case	<FirstName>	Uppercase first letter in each word
Camel case	<firstName>	Uppercase first letter in each word exce pt the first

Solving the Name Conflict

xml tag의 이름 충돌을 방지하기 위해 prefix로 tag를 식별하기 위해 xmlns 속성을 사용

➔ [xmlns:prefix="URI“]

```
<root>

<h:table xmlns:h="http://www.w3.org/TR/html4/">
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table xmlns:f="http://www.w3schools.com/furniture">
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

Root에 namespace 선언

Root element 내에 속성으로 하위 element 요소의 namespace를 선언해서 사용

```
<root
xmlns:h="http://www.w3.org/TR/html4/"
xmlns:f="http://www.w3schools.com/furniture">

<h:table>
  <h:tr>
    <h:td>Apples</h:td>
    <h:td>Bananas</h:td>
  </h:tr>
</h:table>

<f:table>
  <f:name>African Coffee Table</f:name>
  <f:width>80</f:width>
  <f:length>120</f:length>
</f:table>

</root>
```

xpath란

xpath

XPath는 XSLT 표준의 주요 요소, XPath는 XML 문서의 요소와 속성을 탐색하는 데 사용

- XPath는 XML 문서의 일부를 정의하는 구문
- XPath는 경로 표현식을 사용하여 XML 문서를 탐색
- XPath에는 표준 함수 라이브러리가 포함
- XPath는 XSLT 및 XQuery의 주요 요소
- XPath는 W3C 권장 사항

xpath notation 1

xpath notation

syntax	meaning
tag(node)	지정된 태그가있는 모든 자식 요소를 선택합니다. 예를 들어, "spam"은 "spam"이라는 이름의 모든 하위 요소를 선택하고 "spam / egg"는 "spam"이라는 이름의 모든 하위 요소에서 "egg"라는 이름의 모든 손자를 선택합니다. 범용 이름 ("{url} local")을 태그로 사용할 수 있습니다.
/	Root node로 부터 선택
//	현재 요소 아래의 모든 레벨에있는 모든 하위 요소를 선택합니다 (전체 하위 트리 검색). 예를 들어 ".//egg"는 전체 트리에서 모든 "egg"요소를 선택합니다.

xpath notation 2

xpath notation

syntax	meaning
.	현재 노드를 선택하십시오. 이것은 경로의 시작 부분에서 상대 경로임을 나타내기 위해 주로 유용합니다.
..	상위 요소를 선택합니다.
*	모든 하위 요소를 선택합니다. 예를 들어 "* / egg"는 "egg"라는 이름의 모든 손자를 선택합니다.
@	속성을 선택

xpath notation 3

xpath notation

syntax	meaning
[@attrib]	주어진 속성을 가진 모든 요소를 선택합니다. 예를 들어 ".//a[@href]"는 트리에서 "href"속성이있는 모든 "a"요소를 선택합니다.
[@attrib='value']	지정된 속성이 지정된 값을 가지는 모든 요소를 선택합니다. 예를 들어 ".//div[@class='sidebar ']"는 클래스의 "sidebar"가있는 트리의 모든 "div"요소를 선택합니다. 현재 릴리스에서는 값에 따옴표를 사용할 수 없습니다.
[tag]	tag라는 하위 요소가 있는 모든 요소를 선택합니다. 현재 버전에서는 태그 하나만 사용할 수 있습니다 (즉각적인 자식 만 지원됨).
[position]	(지정된 위치에 있는 모든 요소를 선택합니다. 위치는 정수 (1이 첫 번째 위치 임), 표현식 "last ()"(마지막 위치) 또는 last ()에 상대적인 위치 (예 : 두 번째 행의 "last () - 1") 일 수 있습니다. 마지막 위치). 이 술어에는 태그 이름이 있어야 합니다.

xpath 처리 예시

xpath 처리 예시

XPath Expression	Result
/bookstore/book[1]	bookstore 의 첫번째 자식 book 검색
/bookstore/book[last()]	bookstore 의 마지막 자식 book 검색
/bookstore/book[last()-1]	bookstore 의 마지막 -1 자식 book 검색
/bookstore/book[position()<3]	bookstore 의 첫번째와 두번째 자식 book 검색
//title[@lang]	title 중에 lang 속성 검색
//title[@lang='en']	title 중에 lang 속성='en' 검색
/bookstore/book[price>35.00]	bookstore 의 price 가격이 35.00보다 큰 자식 book 검색
/bookstore/book[price>35.00]/title	bookstore 의 price 가격이 35.00보다 큰 자식 book 내의 title tag검색

2.XML 모듈

Moon Yong Joon

XML 이해하기

Moon Yong Joon

xml 생성/로딩

xml 문서 : 파일 만들기

root는 하나이고 다양한 자식 node 들을 만듦

```
%%writefile doc1.xml
<?xml version="1.0"?>
<doc>
    <branch name="testing" hash="1cdf045c">
        text,source
    </branch>
    <branch name="release01" hash="f200013e">
        <sub-branch name="subrelease01">
            xml,sgml
        </sub-branch>
    </branch>
    <branch name="invalid">
    </branch>
</doc>
```

```
Writing doc1.xml
```

xml 문서 load

ElementTree는 하나의 api에 2개의 패키지를 제공하지만 동일한 결과를 처리

```
try:
    import xml.etree.cElementTree as ET
except ImportError:
    import xml.etree.ElementTree as ET
```

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000006787F28>
```

xml search

xml 문서 searching(읽기)

xml 문서를 ElementTree에 load한 후에 root를 읽고 child node searching해야 함

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)
root = tree.getroot()
print(root)
print(root.tag, root.attrib)
for child_of_root in root:
    print(child_of_root.tag, child_of_root.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000006751438>
<Element 'doc' at 0x0000000009921598>
doc {}
branch {'name': 'testing', 'hash': '1cdf045c'}
branch {'name': 'release01', 'hash': 'f200013e'}
branch {'name': 'invalid'}
```

searching : iter

searching한 결과가 depth-first iteration (DFS)로 처리

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)

for elem in tree.iter():
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x00000000067CCB38>
doc {}
branch {'name': 'testing', 'hash': '1cdf045c'}
branch {'name': 'release01', 'hash': 'f200013e'}
sub-branch {'name': 'subrelease01'}
branch {'name': 'invalid'}
```

Search & filter : iter(tag)

tag를 지정하고 searching한 결과가 depth-first iteration (DFS) 로 처리

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)

for elem in tree.iter(tag='branch'):
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x00000000067CC780>
branch {'name': 'testing', 'hash': '1cdf045c'}
branch {'name': 'release01', 'hash': 'f200013e'}
branch {'name': 'invalid'}
```

XML ELEMENTTREE CLASS 이해하기

Moon Yong Joon

xml 문서 parsing

xml 문서 만들기

xml 문서를 하나 만듦

```
%%writefile country_data.xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```
Writing country_data.xml
```

xml 문서 ElementTree parsing

xml.etree.ElementTree 내의 ElementTree class를 통해 parsing

```
import xml.etree.ElementTree as ET
tree = ET.ElementTree(file="country_data.xml")
root = tree.getroot()
print(type(root))
print(root.tag)
print(root.attrib)
for child in root:
    print(child.tag, child.attrib)
```

```
<class 'xml.etree.ElementTree.Element'>
data
{}
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

xml 문자열 parsing

문자열을 만들고 xml parsing

xml.etree.ElementTree 내의 fromstring 함수를 통해 parsing

```
import xml.etree.ElementTree as ET
country_data_as_string = '''
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
'''
```

처리 결과

xml.etree.ElementTree 내의 fromstring 함수를 통해 parsing

```
root = ET.fromstring(country_data_as_string)
print(root)
for node in root :
    print(node)
```

```
<Element 'data' at 0x00000000099625E8>
<Element 'country' at 0x00000000099EE4F8>
<Element 'country' at 0x00000000099F14A8>
<Element 'country' at 0x00000000099F1958>
```

XML ELEMENT CLASS 이해하기

Moon Yong Joon

Element Type

xml 문서 : Element

xml 문서의 모든 tag는 Element로 파싱됨

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
print(tree)
root = tree.getroot()
print(type(root))
print(root.tag)
print(root.attrib)
print(type(root.getchildren()[0]))
print(root.getchildren()[0].tag)
print(root.getchildren()[0].attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x000000000679D5F8>
<class 'xml.etree.ElementTree.Element'>
data
{}
<class 'xml.etree.ElementTree.Element'>
country
{'name': 'Liechtenstein'}
```

Element 조회

xml 문서 tag/attrib 단건 조회

data/country 태크에 대한 tag와 속성 조회

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
print(tree)
root = tree.getroot()
print(root.tag)
print(root.attrib)
print(root.getchildren()[0].tag)
print(root.getchildren()[0].attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x000000000679B710>
data
{}
country
{'name': 'Liechtenstein'}
```

xml 문서 tag/attrib 복수건 조회

data/country 태크에 대한 tag와 속성 조회

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
print(tree)
root = tree.getroot()

for child in root:
    print(child.tag, child.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x000000000679BE80>
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```

get 메소드로 attrib검색

root에서 get 메소드를 이용해서 속성을 검색

```
from xml.etree.ElementTree import parse
tree = parse("note.xml")
note = tree.getroot()

print(note.get("date"))
print(note.get("foo", "default"))
print(note.keys())
print(note.items())
```

```
20120104
default
['date']
[('date', '20120104')]
```

```
<?xml version="1.0"?>
- <note date="20120104">
    <to>Tove</to>
</note>
```

tag 내의 속성들 조회하기

root내의 자식 노드를 읽어 keys/items 메소드를 이용해서 속성들을 조회

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
root = tree.getroot()

con = tree.find("country")
print(con.keys())
print(con.items())
```

```
['name']
[('name', 'Liechtenstein')]
```

Element 생성

Element 생성

node들을 생성하고 note에 to 붙이기

```
from xml.etree.ElementTree import Element, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

# root에 자식node 붙이기
note.append(to)

#출력하기
dump(note)
```

```
<note><to>Tove</to></note>
```

SubElement 생성

node들을 생성하고 subelement로 note에 from 붙이기

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"

dump(note)
```

```
<note><to>Tove</to><from>Jani</from></note>
```

Element/SubElement 생성

element를 Element/SubElement로 생성해서 root에 붙이기

```
from xml.etree.ElementTree import Element, SubElement

root = Element("root")
root.append(Element("one"))
root.append(Element("two"))
root.append(Element("three"))

SubElement(root, "one1")
SubElement(root, "two1")
SubElement(root, "three1")

for node in root:
    print(node)
```

```
<Element 'one' at 0x000000000678EAE8>
<Element 'two' at 0x000000000678E688>
<Element 'three' at 0x000000000678E958>
<Element 'one1' at 0x000000000678E278>
<Element 'two1' at 0x000000000574C7C8>
<Element 'three1' at 0x00000000099D5278>
```

insert 메소드로 자식생성

node들을 생성하고 insert로 note에 dummy 붙이기

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"
dummy = Element("dummy")
note.insert(1, dummy)

dump(note)
```

```
<note><to>Tove</to><dummy /><from>Jani</from></note>
```

remove 메소드로 자식삭제

node들을 생성하고 insert로 note에 dummy 붙였다가 remove로 삭제

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"
dummy = Element("dummy")
note.insert(1, dummy)
dump(note)
note.remove(dummy)
dump(note)
```

```
<note><to>Tove</to><dummy /><from>Jani</from></note>
<note><to>Tove</to><from>Jani</from></note>
```

attribute 생성

indexing으로 속성 추가

node들을 생성하고 attrib 내에 date를 추가

```
from xml.etree.ElementTree import Element, SubElement, dump

note = Element("note")
to = Element("to")
to.text = "Tove"

note.append(to)
SubElement(note, "from").text = "Jani"
note.attrib["date"] = "20120104"

dump(note)
```

```
<note date="20120104"><to>Tove</to><from>Jani</from></note>
```

Element 생성시 속성 추가

node들을 생성시 속성을 초기값으로 넣어서 attrib 내에 date를 추가

```
from xml.etree.ElementTree import Element, SubElement, dump
note = Element("note", date="20120104")
to = Element("to")
to.text = "Tove"

note.append(to)
dump(note)
```

```
<note date="20120104"><to>Tove</to></note>
```

xml 구조 확인

dump로 xml 구조 확인

xml 문서가 만들어지면 dump 함수로 구조 확인

```
from xml.etree.ElementTree import Element, SubElement, dump
note = Element("note", date="20120104")
to = Element("to")
to.text = "Tove"

note.append(to)
dump(note)
```

```
<note date="20120104"><to>Tove</to></note>
```

tostring으로 xml 보기

xml로 완성된 것을 tostring 함수로 결과치 확인하기

```
from xml.etree.ElementTree import Element, SubElement, Comment, tostring

top = Element('top')

comment = Comment('Generated for PyMOTW')
top.append(comment)

child = SubElement(top, 'child')
child.text = 'This child contains text.'

child_with_tail = SubElement(top, 'child_with_tail')
child_with_tail.text = 'This child has regular text.'
child_with_tail.tail = 'And "tail" text.'

child_with_entity_ref = SubElement(top, 'child_with_entity_ref')
child_with_entity_ref.text = 'This & that'

print(tostring(top))
```

```
b'<top><!--Generated for PyMOTW--><child>This child contains text.</child><c
ith_tail>And "tail" text.<child_with_entity_ref>This &amp; that</child_with_
```

문자열을 xml 처리 후 문자열표시

문자열은 xml로 전환(XML, fromstring)하고 이를 다시 tostring 함수로 결과치 확인하기

```
from xml.etree.ElementTree import XML, fromstring, tostring
text = '''
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
</data>
'''
elem = XML(text)
print(elem)

elem = fromstring(text) # same as XML(text)

text = tostring(elem)
print(text)
```

```
<Element 'data' at 0x00000000006999F8>
b'<data>\n    <country name="Liechtenstein">\n        <rank>1</rank>
        <neighbor direction="E" name="Austria" />\n        <neighbor
>'
```

XML XMLPULLPARSER

Moon Yong Joon

XMLPullParser 이용

XMLPullParser 인스턴스를 만들고 feed로 데이터를 제공해서 read_events로 읽는다

```
s = '''<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
    </country>
</data>'''
```

```
parser = ET.XMLPullParser(['start', 'end'])
parser.feed(s)

print(parser.read_events())

for event, elem in parser.read_events():
    print(event)
    print(elem.tag, 'text=', elem.text)
```

XMLPullParser 실행 결과

XMLPullParser.read_events로 읽으면 getnerator 로 제공

```
<generator object XMLPullParser.read_events at 0x000000000599AF10>
start
data text=

start
country text=

start
rank text= 1
end
rank text= 1
end
country text=

end
data text=
```

XML/XPATH SEARCHING

Moon Yong Joon

주요 메소드 특징

find 메소드 특징

find/findall/findtext 메소드 특징

find (pattern)는 주어진 패턴과 일치하는 첫 번째 하위 요소를 반환하고, 일치하는 요소가 없으면 None을 반환

findtext (pattern)은 주어진 패턴과 일치하는 첫 번째 하위 요소의 text 속성 값을 반환합니다. 일치하는 요소가 없으면이 메서드는 None 을 반환

findall (pattern)은 주어진 패턴과 일치하는 모든 서브 엘리먼트의리스트 (또는 또 다른 반복 가능한 객체)를 반환

get 메소드 특징

getiterator/getchildren 메소드 특징

getiterator (tag)는 서브 트리의 모든 레벨에서 주어진 태그를 가진 모든 서브 엘리먼트를 포함하는리스트 (또는 또 다른 반복 가능한 객체)를 리턴
요소는 문서 순서대로 반환 (즉, 트리를 XML 파일로 저장 한 경우 나타나는 순서와 동일한 순서로).

getiterator () (인수 없음)는 서브 트리에있는 모든 하위 요소의 목록 (또는 또 다른 반복 가능한 객체)을 반환

getchildren ()은 모든 직접 하위 요소의 목록 (또는 반복 가능한 다른 객체)을 반환합니다. 이 메소드는 더 이상 사용되지 않음
새로운 코드는 자식에 액세스하기 위해 인덱싱 또는 분할을 사용하거나 목록을 가져 오기 위해 목록 (elem)을 사용

Element indexing

indexing을 통한 Element 검색

root의 하위 tag를 [] 연산자를 통해 객체를 참조

```
import xml.etree.ElementTree as ET
tree = ET.parse('country_data.xml')
print(tree)
root = tree.getroot()
print(root)
print(root[0])
print(root[0][1])
print(root[0][1].text)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000006751B70>
<Element 'data' at 0x00000000099625E8>
<Element 'country' at 0x00000000099624F8>
<Element 'year' at 0x0000000009941D18>
2008
```

xml 문서 만들기

root는 하나이고 다양한 자식 node 들을 만듦

```
%%writefile doc1.xml
<?xml version="1.0"?>
<doc>
    <branch name="testing" hash="1cdf045c">
        text,source
    </branch>
    <branch name="release01" hash="f200013e">
        <sub-branch name="subrelease01">
            xml,sgml
        </sub-branch>
    </branch>
    <branch name="invalid">
    </branch>
</doc>
```

```
Writing doc1.xml
```

find 메소드 searching

find메소드를 통해 tag 접근

root의 하위 tag를 find/findall/findtext 메소드를 통해 객체를 참조

```
from xml.etree.ElementTree import parse
tree = parse("note.xml")
note = tree.getroot()
from_tag = note.find("to")
from_tags = note.findall("to")
from_text = note.findtext("to")

print(from_tag)
print(from_tags)
print(from_text)
```

```
<Element 'to' at 0x0000000009995F48>
[<Element 'to' at 0x0000000009995F48>, <Element 'to' at 0x00000000099C5BD8>]
Tove
```

find메소드 : xpath

xpath로 내부 위치 지정후 text를 조회

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)

print(tree.findtext('branch/sub-branch'))
```

```
<xml.etree.ElementTree.ElementTree object at 0x00000000067519B0>

            xml,sgml
```

find/findall메소드

xml을 읽고 users/user를 읽고 for문으로 처리

```
import xml.etree.ElementTree as ET

input = '''
<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
            </user>
        </users>
</stuff>'''

stuff = ET.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

get 메소드 searching

get 메소드를 통한 tag를 검색

root의 하위 tag 즉 자식을 getiterator, getchildren 메소드로 조회

```
from xml.etree.ElementTree import parse
tree = parse("note.xml")
note = tree.getroot()
childs1 = note.getiterator()
childs2 = note.getchildren()

print(childs1)
print(childs2)
```

```
<_elementtree._element_iterator object at 0x00000000099F2828>
[<Element 'to' at 0x000000000997E408>, <Element 'to' at 0x00000000099C5A48>]
```

XPATH SEARCHING

Moon Yong Joon

Iterfind searching

Xpath 사용하기 : tag

Xpath를 사용해서 searching

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)

for elem in tree.iterfind('branch/sub-branch'):
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000009920E80>
sub-branch {'name': 'subrelease01'}
```

Xpath 사용하기 : *

모든 하위 요소를 선택합니다. 예를 들어 "* / egg"는 "egg"라는 이름의 모든 손자를 선택합니다.

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)
for elem in tree.iterfind('*//sub-branch'):
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000009940668>
sub-branch {'name': 'subrelease01'}
```

Xpath 사용하기 : [@속성]

Xpath(branch내의 속성)를 사용해서 searching

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)
for elem in tree.iterfind('branch[@name]'):
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x0000000009920080>
branch {'name': 'testing', 'hash': '1cdf045c'}
branch {'name': 'release01', 'hash': 'f200013e'}
branch {'name': 'invalid'}
```

Xpath 사용하기 : [@속성=값]

Xpath(branch내의 속성)를 사용해서 searching

```
import xml.etree.cElementTree as ET
tree = ET.ElementTree(file='doc1.xml')
print(tree)
for elem in tree.iterfind('branch[@name="release01"]'):
    print(elem.tag, elem.attrib)
```

```
<xml.etree.ElementTree.ElementTree object at 0x000000000679B668>
branch {'name': 'release01', 'hash': 'f200013e'}
```

findall(xpath)

Xml file

xml 문서 생성

```
%%writefile lxml_test.xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```
Writing lxml_test.xml
```

xpath로 root 검색

findall 메소드로 xpath로 root 검색

```
import xml.etree.ElementTree as ET

tree = ET.parse('lxml_test.xml')
root = tree.getroot()
# Top-level elements
print(root.findall("."))
```

```
[<Element 'data' at 0x0000000005FFA9F8>]
```

xpath로 child검색

findall 메소드로 xpath로 countrytag검색

```
import xml.etree.ElementTree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()


# All country' children of the top-level
# elements
pp.pprint(root.findall("./country"))
```

```
[<Element 'country' at 0x0000000005FFA8B8>,
 <Element 'country' at 0x0000000005FF9138>,
 <Element 'country' at 0x0000000005FF9188>]
```

xpath로 grand-child 검색

findall 메소드로 xpath로 country내의 neighbor tag검색

```
import xml.etree.ElementTree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()


# All 'neighbor' grand-children of 'country' children of the top-level
# elements
pp.pprint(root.findall("./country/neighbor"))
```

```
[<Element 'neighbor' at 0x0000000005FF9638>,
 <Element 'neighbor' at 0x0000000005FF9778>,
 <Element 'neighbor' at 0x0000000005FF95E8>,
 <Element 'neighbor' at 0x0000000005FF9A48>,
 <Element 'neighbor' at 0x0000000005FF9A98>]
```

특정 tag 상위(속성) 검색

findall 메소드로 특정tag를 찾고 ..(상위) tag 중 name 속성에 singapore 값을 갖는 tag 검색

```
import xml.etree.ElementTree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//year"))
pp.pprint(root.findall(".//year/.."))
# Nodes with name='Singapore' that have a 'year' child
pp.pprint(root.findall(".//year/..[@name='Singapore']"))
```

```
[<Element 'year' at 0x0000000005FF9638>,
 <Element 'year' at 0x0000000005FF95E8>,
 <Element 'year' at 0x0000000005FF9408>]
[<Element 'country' at 0x0000000005FF9BD8>,
 <Element 'country' at 0x0000000005FF9688>,
 <Element 'country' at 0x0000000005FF94F8>]
[<Element 'country' at 0x0000000005FF9688>]
```

모든tag검색 후 특정 하위 검색

findall 메소드로 특정tag를 찾고 .//*(모든) tag 중 name 속성에 singapore 값을 갖는 tag 검색한 후 year 하위 tag 검색

```
import xml.etree.ElementTree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//*"))
pp.pprint(root.findall(".//*[@name='Singapore']"))
# 'year' nodes that are children of nodes with name='Singapore'
pp.pprint(root.findall(".//*[@name='Singapore']/year"))
```

```
[<Element 'country' at 0x0000000005FF9958>,
 <Element 'rank' at 0x0000000005FF9A98>,
 <Element 'year' at 0x0000000005FF9B88>,
 <Element 'gdppc' at 0x0000000005FF9AE8>,
 <Element 'neighbor' at 0x0000000005FF92C8>,
 <Element 'neighbor' at 0x0000000005FF9278>,
 <Element 'country' at 0x0000000005FF9228>,
 <Element 'rank' at 0x0000000005FF91D8>,
 <Element 'year' at 0x0000000005FF9188>,
 <Element 'gdppc' at 0x0000000005FF9098>,
 <Element 'neighbor' at 0x0000000005FF99A8>,
 <Element 'country' at 0x0000000005FF9818>,
 <Element 'rank' at 0x0000000005FF9048>,
 <Element 'year' at 0x0000000005FF9908>,
 <Element 'gdppc' at 0x0000000005FF9868>,
 <Element 'neighbor' at 0x0000000005FF9F48>,
 <Element 'neighbor' at 0x0000000005FF9F98>]
[<Element 'country' at 0x0000000005FF9228>]
[<Element 'year' at 0x0000000005FF9188>]
```

특정tag검색 후 위치 검색

findall 메소드로 특정tag를 찾고 .//태그명()으로 특정 tag를 검색하고 tag의 속한 위치 검색

```
import xml.etree.ElementTree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//neighbor"))
# All 'neighbor' nodes that are the second child of their parent
pp.pprint(root.findall(".//neighbor[2]"))
```

```
[<Element 'neighbor' at 0x000000000600C2C8>,
 <Element 'neighbor' at 0x000000000600C318>,
 <Element 'neighbor' at 0x000000000600C4A8>,
 <Element 'neighbor' at 0x000000000600C638>,
 <Element 'neighbor' at 0x000000000600C688>]
[<Element 'neighbor' at 0x000000000600C318>,
 <Element 'neighbor' at 0x000000000600C688>]
```

XML/HTML 파일 처리

Moon Yong Joon

파일 읽기

parse로 읽기 : file 처리

parse 함수를 통해 직접 접근하거나 파일을 읽고 전달 받아 처리

```
from xml.etree.ElementTree import parse

tree = parse("country_data.xml")
elem = tree.getroot()
print(elem)
```

```
<Element 'data' at 0x000000000997EF48>
```

```
from xml.etree.ElementTree import parse

file = open("country_data.xml", "r")
tree = parse(file)
elem = tree.getroot()
print(elem)
```

```
<Element 'data' at 0x00000000099F4098>
```

```
<?xml version="1.0"?>
- <data>
  + <country name="Liechtenstein">
  + <country name="Singapore">
  + <country name="Panama">
</data>
```

parse로 읽기 : file-like

parse 함수를 통해 file-like 즉 StringIO를 처리

```
import xml.etree.ElementTree as ET
from io import StringIO
M = StringIO('''
<foo>
   <bar>
      <type foobar="1"/>
      <type foobar="2"/>
   </bar>
</foo>''')

root = ET.parse(M).getroot() #element instance 생성

for i in root.findall('bar'):
    for j in i.findall('type'):
        print(j.get('foobar'))
```

```
1
2
```

ElementTree로 읽기 : file

ElementTree를 통해 직접 접근해서 파일을 읽기

```
from xml.etree.ElementTree import ElementTree

tree = ElementTree(file="country_data.xml")
elem = tree.getroot()
print(elem)
```

```
<Element 'data' at 0x00000000099EE2C8>
```

```
<?xml version="1.0"?>
- <data>
  + <country name="Liechtenstein">
  + <country name="Singapore">
  + <country name="Panama">
</data>
```

파일 생성

ElementTree로 xml파일 생성

ElementTree(root node).write(파일명)으로 새로운 파일 생성

```
from xml.etree.ElementTree import Element, SubElement, dump
note = Element("note", date="20120104")
to = Element("to")
to.text = "Tove"

note.append(to)
dump(note)

from xml.etree.ElementTree import ElementTree
ElementTree(note).write("note.xml")
```

```
<note date="20120104"><to>Tove</to></note>
```

```
<?xml version="1.0"?>
- <note date="20120104">
    <to>Tove</to>
</note>
```

ElementTree로 html파일 생성

ElementTree(root node).write(파일명)으로 새로운 파일 생성

```
import xml.etree.ElementTree as ET

# build a tree structure
root = ET.Element("html")

head = ET.SubElement(root, "head")

title = ET.SubElement(head, "title")
title.text = "Page Title"

body = ET.SubElement(root, "body")
body.set("bgcolor", "#ffffff")

body.text = "Hello, World!"

# wrap it in an ElementTree instance, and save as XML
tree = ET.ElementTree(root)
tree.write("page.xhtml")
```

```
▼<html>
  ▼<head>
     <title>Page Title</title>
   </head>
   <body bgcolor="#ffffff">Hello, World!</body>
 </html>
```

3.LXML 모듈

Moon Yong Joon

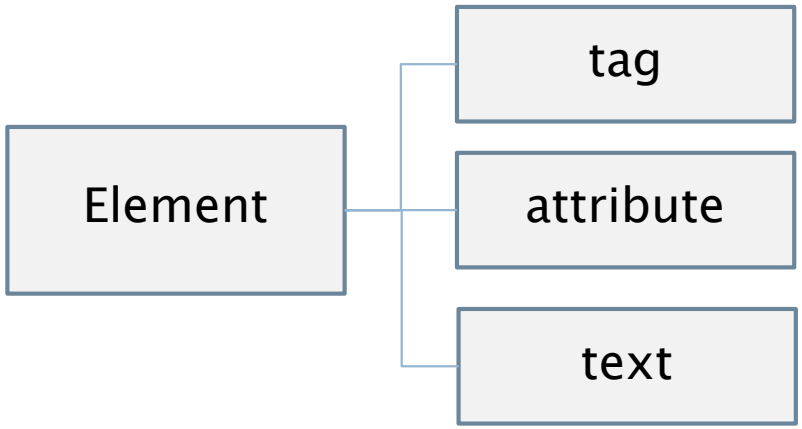
XML 문서 만들기

xml 기본 문서 만들기

Element/attribute 표현차이

element는 계층적 데이터 구조를 메모리에 저장하도록 설계된 유연한 컨테이너 객체이고,

attribute은 요소 내에 정보나 데이터를 표현하기 위한 방법으로 시작태그의 일부로 표현하며 속성명과 속성 값으로 이루어짐



element로 표현과 attribute으로 표현의 차이

- attribute으로 설계할 경우
  - 문서 크기를 줄일 수 있지만 속성 자체가 요소에 종속적이기 때문에 확장성이 떨어짐
- element로 설계할 경우
  - 가독성이 좋아 의미를 쉽게 파악할 수 있으며 향후 문서 내용을 추가하거나 확장할 때 용이

Xml 문서 만들기 흐름


```
from lxml import etree
#Root 생성
page = etree.Element('html')
#xml 문서 tree 처리
doc = etree.ElementTree(page)

#sub element 생성
headElt = etree.SubElement(page, 'head')
bodyElt = etree.SubElement(page, 'body')

#sub element의 sub element 생성
title = etree.SubElement(headElt, 'title')
#sub element의 sub element의 text 추가
title.text = 'Your page title here'

#sub element의 sub element의 속성 부여하고 생성
linkElt = etree.SubElement(headElt, 'link', rel='stylesheet',
    href='mystyle.css', type='text/css')

# tree를 이용해서 파일처리
outFile = open('homemade.xml', 'wb')
doc.write(outFile)

with open('homemade.xml', 'rb') as f :
    for i in f :
        print(i)
```

```
b'<html><head><title>Your page title here</title><link href="mystyle.css"
>'
```

문서만들기 : Element

xml에 구성하는 원소를 Element로 생성함

```
: from lxml import etree
root = etree.Element("root")
print(root.tag)

child1 = etree.Element("child1")

root.append(child1)
print(root)
print(etree.tostring(root, pretty_print=True))
```

```
root
<Element root at 0x5f7b988>
b'<root>\n  <child1/>\n</root>\n'
```

문서만들기 : SubElement

html 문서를 만들고 각 Element 내의 text 추가

```
from lxml import etree
html = etree.Element("html")
body = etree.SubElement(html, "body")
body.text = "TEXT"


br = etree.SubElement(body, "br")
br.tail = "TAIL"
print(etree.tostring(html))
```

```
b'<html><body>TEXT<br/>TAIL</body></html>'
```

Element attribute 추가

xml 내의 요소에 대한 속성을 추가(set)하고 검색(get)하기

```
from lxml import etree
root = etree.Element("root", interesting="totally")
print(etree.tostring(root))

print(root.get("interesting"))

print(root.get("hello"))

root.set("hello", "Huhu")
print(root.get("hello"))
print(etree.tostring(root))
```

```
b'<root interesting="totally"/>'
totally
None
Huhu
b'<root interesting="totally" hello="Huhu"/>'
```

Element attribute 변경

xml 내의 요소에 대한 속성을 keys/items로 조회하고 속성에 대한 값 변경

```
from lxml import etree
root = etree.Element("root", interesting="totally")
root.set("hello", "Huhu")
print(sorted(root.keys()))

for name, value in sorted(root.items()):
    print('%s = %r' % (name, value))

attributes = root.attrib

print(attributes["interesting"])
print(attributes.get("no-such-attribute"))
attributes["hello"] = "Guten Tag"
print(attributes["hello"])
print(root.get("hello"))
```

```
['hello', 'interesting']
hello = 'Huhu'
interesting = 'totally'
totally
None
Guten Tag
Guten Tag
```

기타 XML 문서만들기

문서만들기 : XML

XML로 xml 문서 만들고 tostring함수에서 xml_declaration 세팅하면 헤더도 만들어 짐

```
from lxml import etree
root = etree.XML('<root><a><b/></a></root>')

print(etree.tostring(root))

print(etree.tostring(root, xml_declaration=True))
```

```
b'<root><a><b/></a></root>'
b"<?xml version='1.0' encoding='ASCII'?>\n<root><a><b/></a></root>"
```

문서만들기 : XMLParser

XMLparse 함수를 이용해서 파일을 xml 문서 만들기

```
from lxml import etree

parser = etree.XMLParser()

parser.feed("<roo")
parser.feed("t><")
parser.feed("a/")
parser.feed("><")
parser.feed("/root>")

root = parser.close()

print(etree.tostring(root))
```

```
b'<root><a/></root>'
```

ElementTree

XML 문서 Wrapping 하기

ElementTree는 주로 루트 노드가 있는 문서 래퍼이며, 직렬화 및 일반 문서 처리를 위한 두 가지 방법을 제공

```
from lxml import etree
root = etree.XML('''<?xml version="1.0"?>
<!DOCTYPE root SYSTEM "test" [ <!ENTITY tasty "parsnips"> ]>
<root>
<a>&tasty;</a>
</root>
''')
tree = etree.ElementTree(root)
print(etree.tostring(tree))
```

```
b'<!DOCTYPE root SYSTEM "test" [\n<!ENTITY tasty "parsnips">\n]>\n<root>\n<a>parsnips</a>\n</root>'
```

ElementTree : doctype 추가

xml 문서 doctype 속성 추가하기

```
from lxml import etree
root = etree.XML('''<?xml version="1.0"?>
<!DOCTYPE root SYSTEM "test" [ <!ENTITY tasty "parsnips"> ]>
<root>
<a>&tasty;</a>
</root>
''')

tree = etree.ElementTree(root)
print(tree.docinfo.xml_version)
print(tree.docinfo.doctype)

tree.docinfo.public_id = '-//W3C//DTD XHTML 1.0 Transitional//EN'
tree.docinfo.system_url = 'file://local.dtd'
print(tree.docinfo.doctype)
```

```
1.0
<!DOCTYPE root SYSTEM "test">
<!DOCTYPE root PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN" "file://local.dtd">
```

XML 문서 검색/갱신

Moon Yong Joon

Xml 문서 바꾸기

문서 검색 : index

xml 내의 요소를 indexing으로 검색 하기

```
from lxml import etree
root = etree.Element("root")
print(root)
print(root.tag)
root.append( etree.Element("child1") )
child2 = etree.SubElement(root, "child2")
child3 = etree.SubElement(root, "child3")

child = root[0]
print(child.tag)
print(len(root))

print(root[1].tag)
print(root.index(root[1]))
children = list(root)
print(children)
```

```
<Element root at 0x539f8c8>
root
child1
3
child2
1
[<Element child1 at 0x539e548>, <Element child2 at 0x539f9c8>, <Element child3 at 0x539f908>]
```

문서 삽입: insert

Element로 루트 노드가 있는 문서 만들기

```
from lxml import etree

note = etree.Element("note")
to = etree.Element("to")
to.text = "Tove"

note.append(to)
etree.SubElement(note, "from").text = "Jani"
dummy = etree.Element("dummy")
note.insert(1, dummy)

etree.dump(note)
```

```
<note>
  <to>Tove</to>
  <dummy/>
  <from>Jani</from>
</note>
```

문서 제거:remove

Element로 루트 노드가 있는 문서 만들기

```
from lxml import etree

note = etree.Element("note")
to =  etree.Element("to")
to.text = "Tove"

note.append(to)
etree.SubElement(note, "from").text = "Jani"
dummy =  etree.Element("dummy")
note.insert(1, dummy)
etree.dump(note)
note.remove(dummy)
etree.dump(note)
```

```
<note>
  <to>Tove</to>
  <dummy/>
  <from>Jani</from>
</note>
<note>
  <to>Tove</to>
  <from>Jani</from>
</note>
```

XML 문서 FILE 처리

Moon Yong Joon

파일 read

Xml file

xml 문서 생성

```
%%writefile lxml_test.xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```
Writing lxml_test.xml
```

Xml file 읽고 parsing

xml 문서를 읽고 ElementTree로 파싱하고 getroot()로 Element 처리한 후에 문자열로 출력

```
import lxml.etree as ET
tree = ET.parse('lxml_test.xml')
root = tree.getroot()
print(ET.tostring(root))
```

```
b'<data>\n    <country name="Liechtenstein">\n
        <neighbor name="Austria" direction="E"/>\n
ntry name="Singapore">\n        <rank>4</rank>\n
="Malaysia" direction="N"/>\n    </country>\n    <c
      <gdppc>13600</gdppc>\n        <neighbor name=
="E"/>\n    </country>\n</data>'
```

파일 write

ElementTree : file 생성

xml 문서를 만들고 ElementTree.write 메소드를 이용해서 파일 생성

```
from lxml import etree
note = etree.Element("note", date="20120104")
to = etree.Element("to")
to.text = "Tove"

note.append(to)

to1 = etree.Element("to")
to1.text = "Love"

note.append(to1)
etree.dump(note)

etree.ElementTree(note).write("note.xml")
```

```
<note date="20120104">
  <to>Tove</to>
  <to>Love</to>
</note>
```

```
<?xml version="1.0"?>
- <note date="20120104">
    <to>Tove</to>
    <to>Love</to>
  </note>
```

String 처리 하기

문서만들기 : fromstring

fromstring 함수를 이용해서 문자열을 xml 문서로 전환

```
some_xml_data = "<root>data</root>"

root = etree.fromstring(some_xml_data)
print(root.tag)
print(etree.tostring(root))
```

```
root
b'<root>data</root>'
```

tostring 처리

html 문서를 만들고 각 Element Tree를 전부 검색해서 출력

```
from lxml import etree
root = etree.XML('<root><a><b/></a></root>')

print(etree.tostring(root))

print(etree.tostring(root, xml_declaration=True))

print(etree.tostring(root, encoding='iso-8859-1'))
print(etree.tostring(root, pretty_print=True))
```

```
b'<root><a><b/></a></root>'
b"<?xml version='1.0' encoding='ASCII'?>\n<root><a><b/></a></root>"
b"<?xml version='1.0' encoding='iso-8859-1'?>\n<root><a><b/></a></root>"
b'<root>\n  <a>\n    <b/>\n  </a>\n</root>\n'
```

XML 문서 PARSING & SEARCHING

Moon Yong Joon

parsing

Parsing 하기 : parse

parse 함수를 이용해서 파일을 xml 문서로 전환

```
from io import BytesIO
from lxml import etree

file_like_object = BytesIO(b"<root>data</root>")
tree = etree.parse(file_like_object)

print(etree.tostring(tree))
```

```
b'<root>data</root>'
```

Parsing 하기 : XML

xml 문서를 만들때 parser 정보에 대한 config 세팅 변환

```
from lxml import etree

parser = etree.XMLParser(remove_blank_text=True)
root = etree.XML("<root>  <a/>   <b>  </b>     </root>", parser)

print(etree.tostring(root))
```

```
b'<root><a/><b>  </b></root>'
```

Parsing 하기 : iterparse

XMLparse 함수를 이용해서 파일을 xml 문서로 파싱(event 기반)

```
from lxml import etree
some_file_like = BytesIO(b"<root><a>data</a></root>")

for event, element in etree.iterparse(some_file_like,events=("start", "end")):
    print("%5s, %4s, %s" % (event, element.tag, element.text))
```

```
start, root, None
start,    a, data
  end,    a, data
  end, root, None
```

Parsing 하기 : 특정 tag 정리

XMLparse 함수를 이용해서 파일을 xml문서 정보 clear

```
from lxml import etree
from io import BytesIO
some_file_like = BytesIO(b"<root><a><b>data</b></a><a><b/></a></root>")

for event, element in etree.iterparse(some_file_like):
    if element.tag == 'b':
        print(element.text)
    elif element.tag == 'a':
        print("** cleaning up the subtree")
        element.clear()
```

```
data
** cleaning up the subtree
None
** cleaning up the subtree
```

Tree iteration

find 메소드

하위 요소들 중에 첫번째 검색 처리하기

```
from lxml import etree

input = '''
<stuff>
    <users>
        <user x="2">
            <id>001</id>
            <name>Chuck</name>
        </user>
        <user x="7">
            <id>009</id>
            <name>Brent</name>
            </user>
        </users>
</stuff>'''

stuff = etree.fromstring(input)
lst = stuff.findall('users/user')
print('User count:', len(lst))

for item in lst:
    print('Name', item.find('name').text)
    print('Id', item.find('id').text)
    print('Attribute', item.get("x"))
```

```
User count: 2
Name Chuck
Id 001
Attribute 2
Name Brent
Id 009
Attribute 7
```

findall 메소드

하위 요소들을 전부 검색하고 내부 속성을 처리하기

```
from lxml import etree

from io import StringIO
M = StringIO('''
<foo>
   <bar>
      <type foobar="1"/>
      <type foobar="2"/>
   </bar>
</foo>''')

root = etree.parse(M).getroot() #element instance 생성

for i in root.findall('bar'):
    for j in i.findall('type'):
        print(j.get('foobar'))
```

```
1
2
```

Elements tree : iter

html 문서를 만들고 각 Element Tree를 전부 검색해서 출력

```
from lxml import etree
root = etree.Element("root")
etree.SubElement(root, "child").text = "Child 1"
etree.SubElement(root, "child").text = "Child 2"
etree.SubElement(root, "another").text = "Child 3"

for element in root.iter():
    print("%s - %s" % (element.tag, element.text))
```

```
root - None
child - Child 1
child - Child 2
another - Child 3
```

Elements tree : iter(특정값)

html 문서를 만들고 각 Element Tree를 전부 검색하지만 주어진 값의 요소만 출력

```
from lxml import etree
root = etree.Element("root")
etree.SubElement(root, "child").text = "Child 1"
etree.SubElement(root, "child").text = "Child 2"
etree.SubElement(root, "another").text = "Child 3"

for element in root.iter("root"):
    print(" root only %s - %s" % (element.tag, element.text))

for element in root.iter("child"):
    print(" child only %s - %s" % (element.tag, element.text))

for element in root.iter("another", "child"):
    print(" another/child only  %s - %s" % (element.tag, element.text))
```

```
 root only root - None
 child only child - Child 1
 child only child - Child 2
 another/child only  child - Child 1
 another/child only  child - Child 2
 another/child only  another - Child 3
```

XML 문서 NAMESPACE

Moon Yong Joon

Namespace :QName

Qname 함수로 namespace가 들어간 tag 만들기

```
# localname, namespace, text
from lxml import etree

tag = etree.QName('http://www.w3.org/1999/xhtml', 'html')
print(tag.localname)
print(tag.namespace)
print(tag.text)

root = etree.Element('{http://www.w3.org/1999/xhtml}html')
tag = etree.QName(root)
print(tag.localname)
print(tag.namespace)
print(tag.text)
```

```
html
http://www.w3.org/1999/xhtml
{http://www.w3.org/1999/xhtml}html
html
http://www.w3.org/1999/xhtml
{http://www.w3.org/1999/xhtml}html
```

Namespace : Element 기초

Element 생성시 namespace를 직접 넣고 tag 만들기

```
from lxml import etree

XHTML_NAMESPACE = "http://www.w3.org/1999/xhtml"
XHTML = "{%s}" % XHTML_NAMESPACE

NSMAP = {None : XHTML_NAMESPACE} # the default namespace (no prefix)

xhtml = etree.Element(XHTML + "html", nsmap=NSMAP) # lxml only!
body = etree.SubElement(xhtml, XHTML + "body", nsmap=NSMAP)
body.text = "Hello World"

print(etree.tostring(xhtml, pretty_print=True))
print(xhtml.find('body'))
print(xhtml.find(XHTML + "body"))
```

```
b'<html xmlns="http://www.w3.org/1999/xhtml">\n  <body>Hello World</body>\n</html>\n'
None
<Element {http://www.w3.org/1999/xhtml}body at 0x5f7bb88>
```

Namespace : Element 활용

Element 생성시 namespace에 대한 map을 넣고 처리

```
from lxml import etree

root = etree.Element('root', nsmap={'a': 'http://a.b/c'})
child = etree.SubElement(root, 'child',
                         nsmap={'b': 'http://b.c/d'})
print(len(root.nsmap))
print(len(child.nsmap))
print(child.nsmap['a'])
print(child.nsmap['b'])


print(etree.tostring(root, pretty_print=True))
print(root.findall('child'))
for i in root.iter():
    print(i.tag)
```

```
1
2
http://a.b/c
http://b.c/d
b'<root xmlns:a="http://a.b/c">\n  <child xmlns:b="http://b.c/d"/>\n</root>\n'
[<Element child at 0x54a2308>]
root
child
```

Namespace : prefix 처리

namespace를 prefix를 주고 처리가 가능함

```
import io
fs = '''
<a:foo xmlns:a="http://codespeak.net/ns/test1"
        xmlns:b="http://codespeak.net/ns/test2">
<b:bar>Text</b:bar>
</a:foo> '''
f = io.StringIO(fs)
doc = etree.parse(f)
foo = doc.getroot()
print(etree.tostring(doc))
print(foo.nsmap)
print(foo.findall('b:bar', foo.nsmap))
```

```
b'<a:foo
xmlns:a="http://codespeak.net/ns/t
est1"
xmlns:b="http://codespeak.net/ns/t
est2">\n <b:bar>Text</b:bar>\n
</a:foo>'
{'a': 'http://codespeak.net/ns/test1',
'b': 'http://codespeak.net/ns/test2'}
[<Element
{http://codespeak.net/ns/test2}bar
at 0x54be6c8>]
```

XML 문서 XPATH

Moon Yong Joon

xpath 처리

xpath:

xpath를 이용해서 Element 검색

```
from lxml import etree
from io import StringIO
f = StringIO('<foo><bar></bar></foo>')
tree = etree.parse(f)

r = tree.xpath('/foo/bar')
print(len(r))
print(r[0].tag)

r = tree.xpath('bar')
print(r[0].tag)
```

```
1
bar
bar
```

xpath 로 css 처리하기

xpath로 CSS select를 하기

```
from lxml import etree
import io

fs = '''<div id="outer">
            <div id="inner" class="content body">
                    text
            </div>
        </div>'''

f = io.StringIO(fs)

doc = etree.parse(f)

sel = doc.xpath("//div[@class='content body']")
print(sel)
```

```
[<Element div at 0x5f1f7c8>]
```

Namespace : xpath함수

namespace를 prefix를 주고 처리가 가능함

```
import io
fs = '''
<a:foo xmlns:a="http://codespeak.net/ns/test1"
        xmlns:b="http://codespeak.net/ns/test2">
<b:bar>Text</b:bar>
</a:foo> '''

f = io.StringIO(fs)
doc = etree.parse(f)

r = doc.xpath('/x:foo/b:bar',
              namespaces={'x': 'http://codespeak.net/ns/test1',
                          'b': 'http://codespeak.net/ns/test2'})
print(len(r))
print(r[0].tag)
print(r[0].text)
```

```
1
{http://codespeak.net/ns/test2}bar
Text
```

Parsing 하기 :XPath

XPath 함수로 xpath를 선언하고 parsing xml 문서 정보를 넣고 조회해기

```
from lxml import etree
root = etree.XML("<root>world<a>TEXT</a><b>hello</b></root>")

find_text = etree.XPath("//text()")
print(type(find_text))
print(find_text(root))

text = find_text(root)[2]
print(text)

print("text tag",text.getparent())
print(text.getparent().text)

#string으로 처리하기
find_text = etree.XPath("//text()", smart_strings=False)
ext = find_text(root)[1]
print(ext)
print(hasattr(ext, 'getparent'))
```

```
<class 'lxml.etree.XPath'>
['world', 'TEXT', 'hello']
hello
text tag <Element b at 0x5f09ac8>
hello
TEXT
False
```

xml 문서 xpath 처리

Xml file

xml 문서 생성

```
%%writefile lxml_test.xml
<?xml version="1.0"?>
<data>
    <country name="Liechtenstein">
        <rank>1</rank>
        <year>2008</year>
        <gdppc>141100</gdppc>
        <neighbor name="Austria" direction="E"/>
        <neighbor name="Switzerland" direction="W"/>
    </country>
    <country name="Singapore">
        <rank>4</rank>
        <year>2011</year>
        <gdppc>59900</gdppc>
        <neighbor name="Malaysia" direction="N"/>
    </country>
    <country name="Panama">
        <rank>68</rank>
        <year>2011</year>
        <gdppc>13600</gdppc>
        <neighbor name="Costa Rica" direction="W"/>
        <neighbor name="Colombia" direction="E"/>
    </country>
</data>
```

```
Writing lxml_test.xml
```

Root 검색

root tag 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()
# Top-level elements
pp.pprint(root.findall("."))
```

```
[<Element data at 0x56e2588>]
```

자식 tag 검색

root tag의 자식 tag 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()


# All country' children of the top-level
# elements
pp.pprint(root.findall("./country"))
```

```
[<Element country at 0x56e20c8>,
 <Element country at 0x56e23c8>,
 <Element country at 0x56e2208>]
```

손자 tag 검색

root tag의 손자 tag 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()


# All 'neighbor' grand-children of 'country' children of the top-level
# elements
pp.pprint(root.findall("./country/neighbor"))
```

```
[<Element neighbor at 0x56e2048>,
 <Element neighbor at 0x56e2e08>,
 <Element neighbor at 0x56e2848>,
 <Element neighbor at 0x56e2388>,
 <Element neighbor at 0x56e20c8>]
```

특정 tag 선택 후 상위 tag 검색

특정 tag 선택 후 상위 tag 내의 속성이 name='Singpore'가 있는 tag 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//year"))
pp.pprint(root.findall(".//year/.."))
# Nodes with name='Singapore' that have a 'year' child
pp.pprint(root.findall(".//year/..[@name='Singapore']"))
```

```
[<Element year at 0x54c5488>,
 <Element year at 0x54c5448>,
 <Element year at 0x54c5148>]
[<Element country at 0x54c5108>,
 <Element country at 0x54c5488>,
 <Element country at 0x54c5148>]
[<Element country at 0x54c5148>]
```

모든tag 내의 특정속성후 하위 tag

모든tag 내의 특정속성후 하위 tag(year) 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//*"))
pp.pprint(root.findall(".//*[@name='Singapore']"))
# 'year' nodes that are children of nodes with name='Singapore'
pp.pprint(root.findall(".//*[@name='Singapore']/year"))
```

```
[<Element country at 0x56e2b88>,
 <Element rank at 0x56e2e88>,
 <Element year at 0x56e2cc8>,
 <Element gdppc at 0x56e2f88>,
 <Element neighbor at 0x56e2b08>,
 <Element neighbor at 0x56e2108>,
 <Element country at 0x56e2648>,
 <Element rank at 0x56e2488>,
 <Element year at 0x56e29c8>,
 <Element gdppc at 0x56e2408>,
 <Element neighbor at 0x56e2948>,
 <Element country at 0x56e2588>,
 <Element rank at 0x56e2788>,
 <Element year at 0x56e2988>,
 <Element gdppc at 0x56e2388>,
 <Element neighbor at 0x56e27c8>,
 <Element neighbor at 0x56e2448>]
[<Element country at 0x56e2cc8>]
[<Element year at 0x56e2108>]
```

특정 tag 의 포지션별 검색

특정 tag 의 포지션(2)별 검색

```
import lxml.etree as ET
import pprint as pp

tree = ET.parse('lxml_test.xml')
root = tree.getroot()

pp.pprint(root.findall(".//neighbor"))
# All 'neighbor' nodes that are the second child of their parent
pp.pprint(root.findall(".//neighbor[2]"))
```

```
[<Element neighbor at 0x56e29c8>,
 <Element neighbor at 0x56e2408>,
 <Element neighbor at 0x56e2948>,
 <Element neighbor at 0x56e2248>,
 <Element neighbor at 0x56e2588>]
[<Element neighbor at 0x56e2988>, <Element neighbor at 0x56e2408>]
```