熱血講義

프리렉의 열혈강의 시리즈

# Python
# 파이썬

# 열혈강의 Python 파이썬

❖제 21-2 강

주제 : XML 처리하기(DOM)

이강성( gslee@mail.kw.ac.kr )

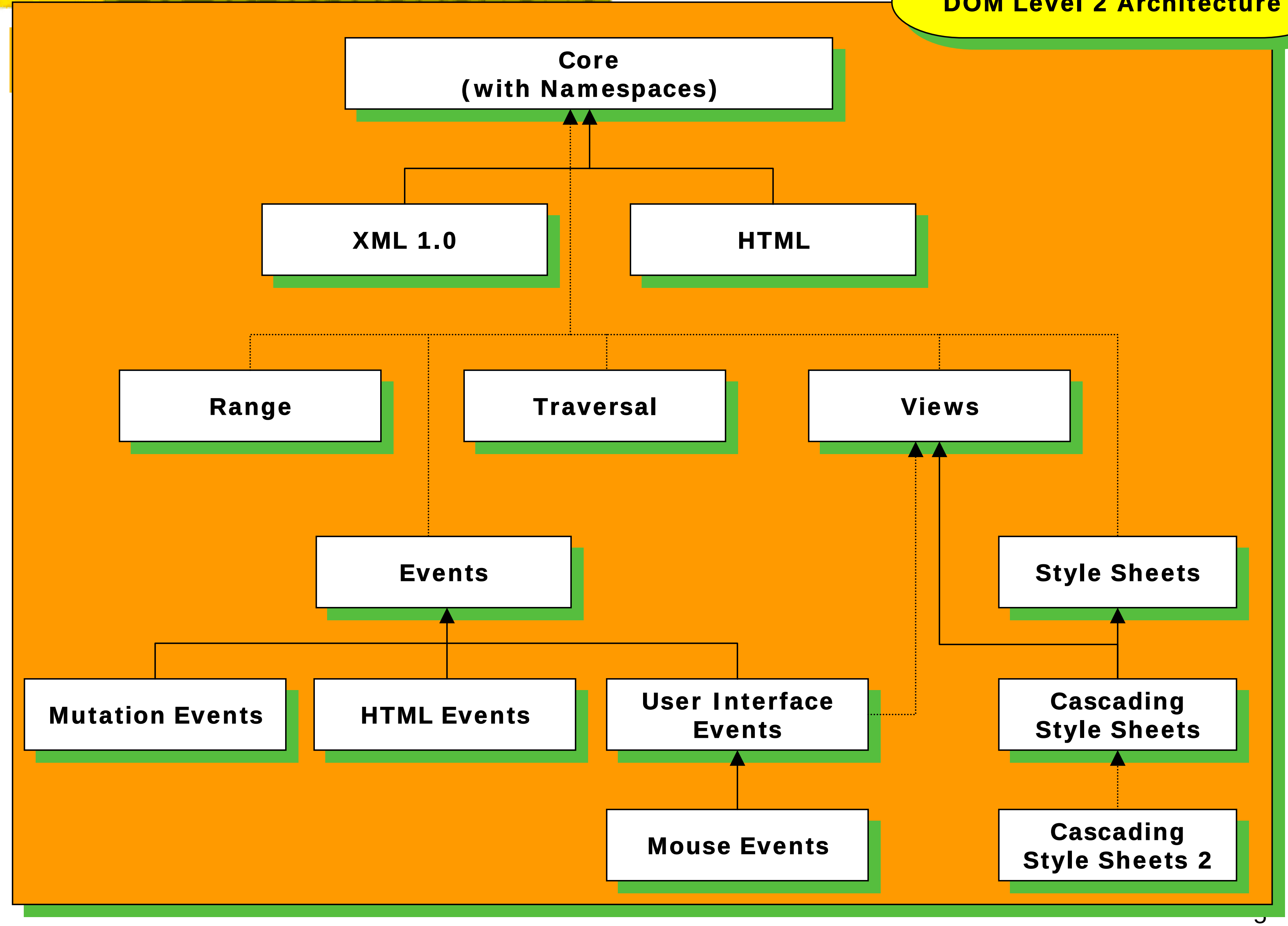
## ❖강의 목차

1. ..
2. SAX
3. DOM
4. Traversal
5. XPath
6. XSLT

## DOM

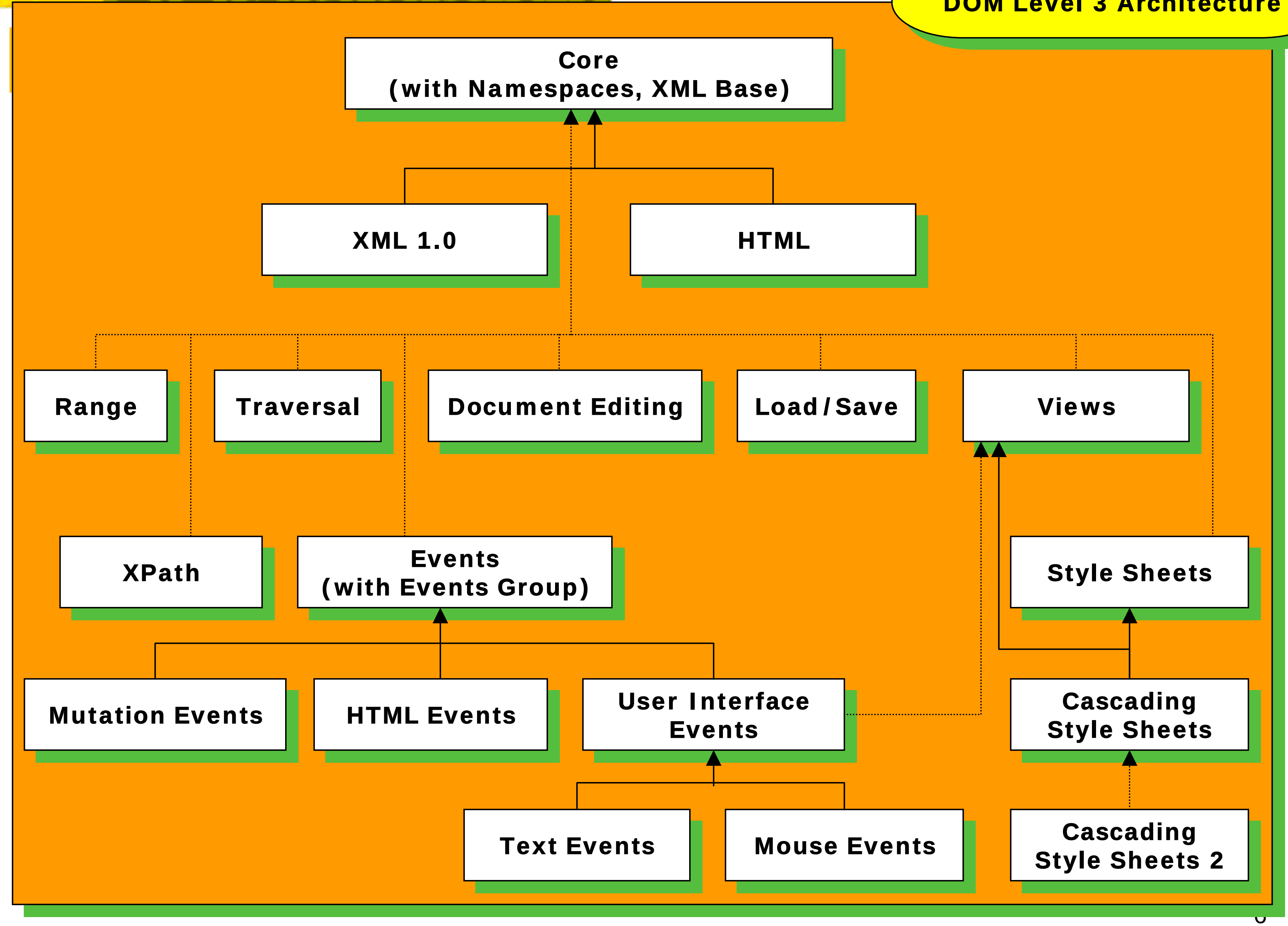
- DOM(Document Object Model)
  - W3C에서 만드는 플랫폼과 언어에 독립적인 API
  - 객체지향적 API
  - 다양한 binding 존재(Python, Java, C++, VB..)
  - 레벨1 → 레벨 2 → 레벨 3(Draft)

## DOM Level 2 Architecture



→ Inherits ⇢ Depends on

## DOM Level 3 Architecture



$\longrightarrow$ Inherits     $\dashrightarrow$ Depends on

# 파이썬 DOM 모듈들

- 표준 라이브러리
  - minidom
  - pulldom
- PyXML
  - http://pyxml.sourceforge.net/
  - 4DOM, javadom (for Jython)
- 4Suite
  - cDomlette

# Load and Save

- DOM Core 레벨 2까지는 Load and Save 기능 지원하지 않으므로 binding 마다 입출력 방법이 다르다.
- minidom의 예

```
# dom01.py
from xml.dom.minidom import *

dom1 = parse('sample01.xml')  # 파일 이름으로 DOM 트리 생성

f = open('sample01.xml')
dom2 = parse(f)      # 파일 객체로 DOM 트리 생성

dom3 = parseString('<value><int>1</int></value>')
```

# Load and Save

- 4DOM인 경우 Load

```
# dom03.py
from xml.dom.ext.reader.Sax2 import *

f = open('sample01.xml')
reader = Reader()  # Reader 객체를 생성한다.
dom1 = reader.fromStream(f)
dom2 = reader.fromUri('sample01.xml')
dom3 = reader.fromString('<value><int>1</int></value>')
```

## Load and Save

- 4DOM 인 경우 Save

```
# dom04.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()  # Reader 객체를 생성한다.
dom = reader.fromUri('sample01.xml')  # URI를 지정한다.
PrettyPrint(dom)        # dom 을 화면으로 출력한다.
```

```
f = open('test.xml', 'w')
PrettyPrint(dom, f)     # dom 을 파일 객체로 출력한다.
f.close()
```

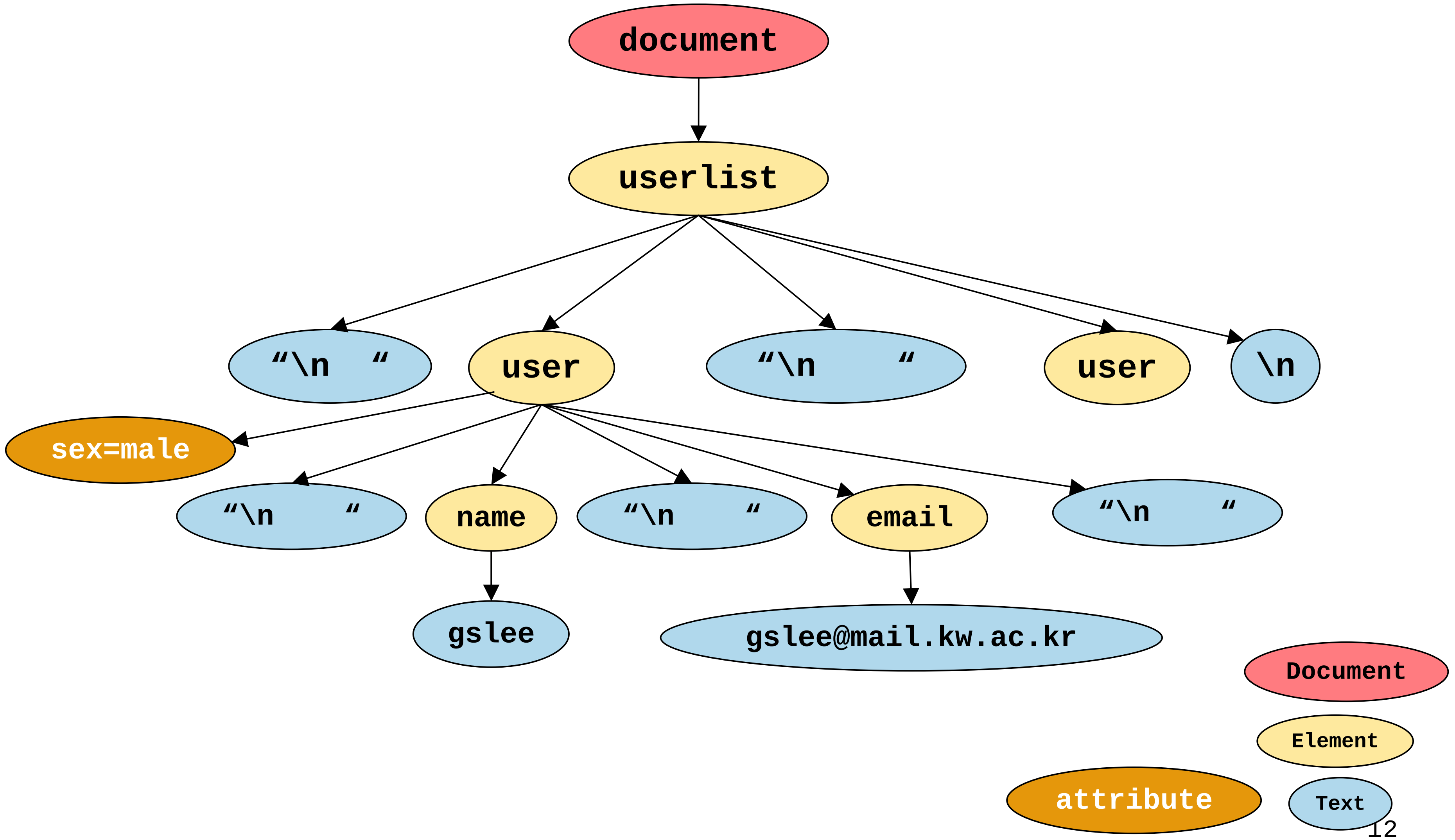
```
from StringIO import StringIO

strio = StringIO()   # 문자열 객체를 만든다.
PrettyPrint(dom, strio) # 문자열 객체로 출력된다
xmlstr = strio.getvalue() # 저장된 문자열을 얻어낸다.
```

## DOM 트리 구조

- sample02.xml

```
<?xml version="1.0" ?>
<userlist>
    <user sex="male">
        <name>gslee</name>
        <email>gslee@mail.kw.ac.kr</email>
    </user>
    <user sex="female">
        <name>spam</name>
        <email>spam@mail.kw.ac.kr</email>
    </user>
</userlist>
```

## DOM 트리 구조


document
userlist
“\n “
user
“\n “
user
\n
sex=male
“\n “
name
“\n “
email
“\n “
gslee
gslee@mail.kw.ac.kr
Document
Element
attribute
Text

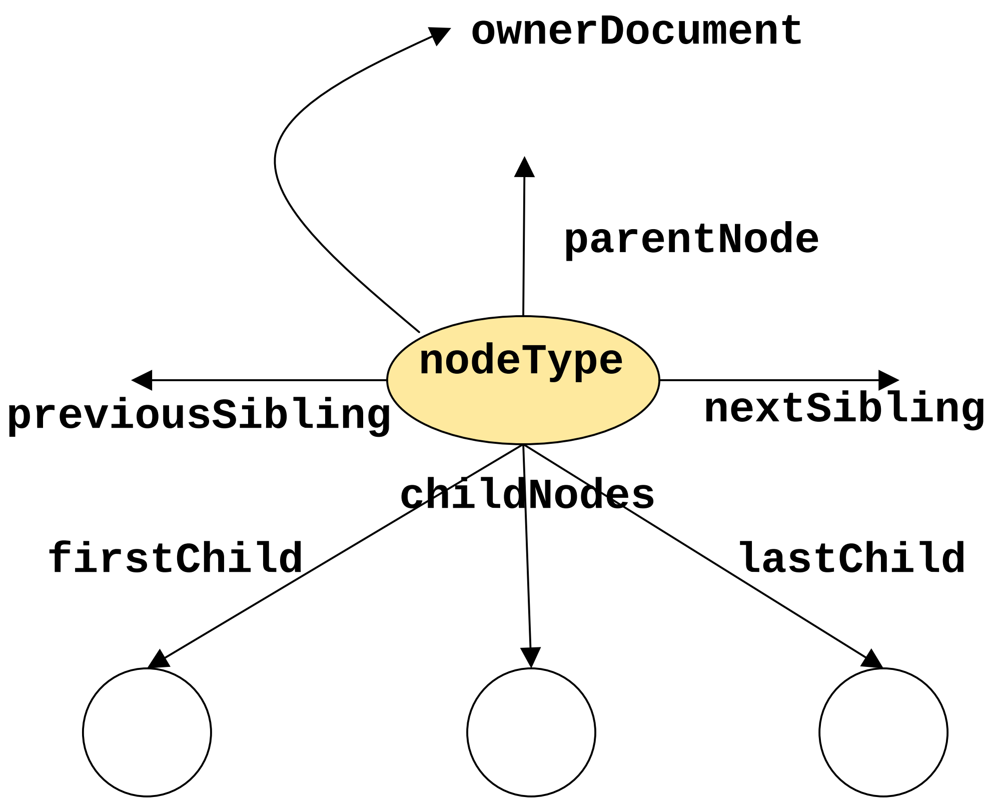
## 노드 탐색

- 노드 타입 확인하기
  - 노드.nodeType

```
ELEMENT_NODE                   = 1
ATTRIBUTE_NODE                 = 2
TEXT_NODE                      = 3
CDATA_SECTION_NODE             = 4
ENTITY_REFERENCE_NODE          = 5
ENTITY_NODE                    = 6
PROCESSING_INSTRUCTION_NODE    = 7
COMMENT_NODE                   = 8
DOCUMENT_NODE                  = 9
DOCUMENT_TYPE_NODE             = 10
DOCUMENT_FRAGMENT_NODE         = 11
NOTATION_NODE                  = 12
```

```
>>> dom.nodeType == dom.DOCUMENT_NODE
1
```

# 노드 이동 멤버 메쏘드


ownerDocument
parentNode
nodeType
previousSibling
nextSibling
childNodes
firstChild
lastChild

## 노드의 종류와 계층구조

```
Node ──┬── Document
       ├── DocumentFragment
       ├── DocumentType
       ├── EntityReference
       ├── Entity
       ├── Element
       ├── Attr
       ├── Notation
       ├── ProcessingInstruction
       └── CharacterData
                 ├── Comment
                 └── Text ──┐
                            └── CDATASection
```

# Node IDL

- IDL(interface definition language)

```
interface Node {
  // 노드 종류 상수
  const unsigned short      ELEMENT_NODE                 = 1;
  const unsigned short      ATTRIBUTE_NODE               = 2;
  const unsigned short      TEXT_NODE                    = 3;
  const unsigned short      CDATA_SECTION_NODE           = 4;
  const unsigned short      ENTITY_REFERENCE_NODE        = 5;
  const unsigned short      ENTITY_NODE                  = 6;
  const unsigned short      PROCESSING_INSTRUCTION_NODE  = 7;
  const unsigned short      COMMENT_NODE                 = 8;
  const unsigned short      DOCUMENT_NODE                = 9;
  const unsigned short      DOCUMENT_TYPE_NODE           = 10;
  const unsigned short      DOCUMENT_FRAGMENT_NODE       = 11;
  const unsigned short      NOTATION_NODE                = 12;
```

# Node IDL

```
// 노드 자체 정보
 readonly attribute DOMString        nodeName;    // 노드의 이름
          attribute DOMString        nodeValue;   // 노드의 값
 readonly attribute unsigned short   nodeType;    // 노드의 유형

 // 노드 이동 메쏘드
 readonly attribute Node             parentNode;  // 부모 노드
 readonly attribute NodeList         childNodes;  // 자식 노드 리스트
 readonly attribute Node             firstChild;  // 첫 번째 자식 노드
 readonly attribute Node             lastChild;   // 마지막 자식 노드
 readonly attribute Node             previousSibling; // 다음 형제 노드
 readonly attribute Node             nextSibling;     // 이전 형제 노드
 readonly attribute NamedNodeMap     attributes;
 readonly attribute Document         ownerDocument;   // 최 상위 노드
```

# Node IDL

```
 // 노드 편집 메쏘드
  Node     insertBefore(in Node newChild, in Node refChild)  // 노드 삽입
  Node     replaceChild(in Node newChild, in Node oldChild)  // 노드 대치
  Node     removeChild(in Node oldChild)  // 노드 제거
  Node     appendChild(in Node newChild)  // 노드 추가
  boolean  hasChildNodes();               // 자식 노드 있는지 확인
  Node     cloneNode(in boolean deep);    // 노드 복제
  void     normalize();                   // 인접한 텍스트 노드들을 결합한다

  // 노드 질의 메쏘드
  boolean  isSupported(in DOMString feature, in DOMString version); //
지정한 기능이 지원되는지
  boolean  hasAttributes();               // 속성 값을 가지고 있는지

  // 이름 공간 관련 메쏘드
  readonly attribute DOMString       namespaceURI; // 이름 공간 URI
           attribute DOMString       prefix;       // 이름 공간 prefix
  readonly attribute DOMString       localName;
};
```

# Text 정보 추출

- 자식 노드의 텍스트 추출

```
>>> def getText(node):
        L = []
        for childNode in node.childNodes:
            if childNode.nodeType == childNode.TEXT_NODE:
                L.append(childNode.data)
        return ''.join(L)

>>> getText(dom.childNodes[1])
u' \n     \n     \n'
```

# Text 정보 추출

- 하위 노드들의 텍스트 추출

```
# dom07.py
from xml.dom.ext.reader.Sax2 import Reader

def getAllText(node):
    s = ''
    for node in node.childNodes:
        if node.nodeType == node.TEXT_NODE:
            s += node.nodeValue
        elif node.nodeType == node.ELEMENT_NODE:
            s += getAllText(node)
    return s

if __name__ == '__main__':
    reader = Reader()
    dom = reader.fromUri('sample02.xml')
    print getAllText(dom)
```

## 엘리먼트 노드 탐색하기

- getElementsByTagName(tagName)
  - tagName 을 갖는 엘리먼트 노드 리스트 리턴
- getElementsByTagNameNS(namespaceURI, localName)
  - 이름 공간의 ...

```
>>> dom.getElementsByTagName('user')   # user 엘리먼트 노드를 모두 얻는다.
<NodeList at 1925280: [<Element Node at 18c33a0: Name='user' with 1
attributes and 5 children>, ...
>>> dom.getElementsByTagNameNS(None, 'user')
<NodeList at 1928f40: [<Element Node at 18c33a0: Name='user' with 1
attributes and 5 children>, ...

>>> print [node.nodeName for node in doc.getElementsByTagNameNS('*', '*')]
[u'userlist', u'user', u'name', u'email', u'user', u'name', u'email']
```

## 속성 참조

- 노드에서 속성 얻어내는 메쏘드
  - **getAttribute(attrName)**
    - 정의된 속성이 없다면 공문자열 리턴
  - **getAttributeNS(NmSpace, attrName)**
- **attributes** 멤버는 사전 인터페이스의 **NameNodeMap** 객체를 리턴

## 속성 참조

```
>>> attr = elem.attributes
>>> attr[(None, 'sex')]    # 사전식 참조
<Attribute Node at 18c5a20: Name="sex", Value="male">
>>> for key, value in attr.items():
        print key, value


(None, u'sex') <Attribute Node at 18c5a20: Name="sex", Value="male">
>>> for key, value in attr.items():   # 속성 (이름공간, 이름)과 속성 노드 값
        print key, value.value


(None, u'sex') male
>>> for value in attr.values():       # 속성 이름과 속성 노드에 대한 루프
        print value.name, value.value


sex male
```

## DOM Core - 노드 생성 메쏘드

| 메쏘드 | 설명 |
|---|---|
| `createElement(tagname)` | 새로운 엘리먼트 노드를 만든다. |
| `createElementNS(namespaceURI, qualifiedName)` | createElement의 이름 공간 버전 |
| `createTextNode(data)` | 새로운 텍스트 노드를 만든다. |
| `createComment(data)` | 주석 노드를 만든다. |
| `createDocumentFragment()` | DocumentFragment 객체를 만든다. |
| `createCDATASection(data)` | CDATA 섹션을 만든다 |
| `createEntityReference(name)` | 엔티티 레퍼런스를 만든다. |
| `createProcessingInstruction(target, data)` | 주어진 이름과 데이터로 ProcssingInstruction 노드를 만든다 |

## 노드 추가, 삭제, 복사 메쏘드

| 메쏘드 | 설명 |
|---|---|
| **appendChild(newChild)** | 새로운 자식 노드를 자식 노드 리스트의 마지막에 추가한다. **newChild**를 리턴한다 |
| **insertBefore(newChild, refChild)** | 새 자식 노드를 지정된 노드(**refChild**) 앞에 삽입한다. **newChild**를 리턴한다 |
| **removeChild(oldChild)** | 자식 노드를 제거한다. **oldChild**는 현재 노드의 자식 노드이어야 한다. |
| **replaceChild(newChild, oldChild)** | 기존의 노드를 새 노드로 교환한다. **oldChild**는 현재 노드의 자식이어야 한다. |
| **cloneNode(deep)** | 현재의 노드를 복제한다. **deep**이 참이면 모든 자식 노드도 복제한다. 복제된 노드를 리턴한다. |
| **normalize()** | 인접한 텍스트 노드들을 하나로 결합한다. 트리를 단순화 하므로 텍스트 처리하는데 효율이 높아진다. |

## 문서 생성

- DOMImplementation 인터페이스를 이용하여 문서 객체 생성
  - createDocument(namespaceURI, qualifiedName, docType)

```
>>> from xml.dom import implementation
>>> root = implementation.createDocument(None, None, None)
>>> root
<XML Document at 17a4020>
```

## 노드 생성 및 추가

- createElement(tagName)
- createElementNS(namespaceURI, qualifiedName)

```
>>> rootElem = root.createElementNS(None, 'userlist')
>>> root.appendChild(rootElem)
```

## 하위 노드 더 추가하기

```
>>> user = root.createElementNS(None, 'user')
>>> name = root.createElementNS(None, 'name')
>>> email = root.createElementNS(None, 'email')
>>> user.appendChild(name)
<Element Node at 1873050: Name='name' with 0 attributes and 0
children>
>>> user.appendChild(email)
<Element Node at 187bb60: Name='email' with 0 attributes and
0 children>
>>> rootElem.appendChild(user)
<Element Node at 186c9c0: Name='user' with 0 attributes and 2
children>
>>> PrettyPrint(root)
<?xml version='1.0' encoding='UTF-8'?>
<userlist>
  <user>
    <name/>
    <email/>
  </user>
</userlist>
```

# 텍스트 노드 추가

```
>>> name.appendChild(root.createTextNode('gslee'))
<Text Node at 1899de0: 'gslee'>
>>>
email.appendChild(root.createTextNode('gslee@some.where'))
<Text Node at 18a6860: 'gslee@some.where'>
>>> PrettyPrint(root)
<?xml version='1.0' encoding='UTF-8'?>
<userlist>
  <user>
    <name>gslee</name>
    <email>gslee@some.where</email>
  </user>
</userlist>
```

# 텍스트 변경

- 텍스트 노드의 data 멤버

```
>>> email.firstChild.data = 'gslee@mail.gwu.ac.kr'
>>> PrettyPrint(root)
<?xml version='1.0' encoding='UTF-8'?>
<userlist>
  <user>
    <name>gslee</name>
    <email>gslee@mail.gwu.ac.kr</email>
  </user>
</userlist>
```

## 속성 추가 / 변경

- setAttributeNS

```
>>> user.setAttributeNS(None, 'sex', 'female')
>>> PrettyPrint(root)
<?xml version='1.0' encoding='UTF-8'?>
<userlist>
  <user sex='female'>
    <name>gslee</name>
    <email>gslee@mail.gwu.ac.kr</email>
  </user>
</userlist>
```

# 속성 삭제

- removeAttributeNS

```
>>> user.setAttributeNS(None, 'age', '40')  # 새로운 속성 값 설정
>>> user.removeAttributeNS(None, 'age')     # 속성 값 삭제
>>> PrettyPrint(root)
<?xml version='1.0' encoding='UTF-8'?>
<userlist>
  <user sex='male'>
    <name>gslee</name>
    <email>gslee@mail.gwu.ac.kr</email>
  </user>
</userlist>
```

## 노드 삭제

- removeChild(oldChild)

```
# dom08.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()
doc = reader.fromUri('sample02.xml')

def removeNode(name, node):
    for childNode in node.childNodes:
        if childNode.nodeName == name:
            node.removeChild(childNode)
        else:
            removeNode(name, childNode)

removeNode('email', doc)
PrettyPrint(doc)
```

## 노드 대치

- replaceChild(newChild, oldChild)

```
# dom09.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()
doc = reader.fromUri('sample02.xml')

email = doc.getElementsByTagName('email')[0]
oldtext = email.firstChild    # 텍스트 엘리먼트를 얻는다.

email.replaceChild(doc.createTextNode('gslee@pymail.net'),
oldtext)  # 대치한다.

PrettyPrint(doc)
```

## 노드 이동

- removeChild와 appendChild 혹은 insertBefore를 이용한다

```
# dom10.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()
doc = reader.fromUri('sample02.xml')

users = doc.getElementsByTagName('user')
parent = users[0].parentNode
parent.removeChild(users[0])
parent.appendChild(users[0])

PrettyPrint(doc)
```

## 노드 복제

- 어떤 노드의 부트리 전체를 복사
  - cloneNode(deep)

```
# dom11.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()
doc = reader.fromUri('sample02.xml')

firstUser = doc.getElementsByTagNameNS(None, 'user')[0]
clonedUser = firstUser.cloneNode(1)

doc.documentElement.appendChild(clonedUser)
PrettyPrint(doc)
```

# 노드 임포트 하기

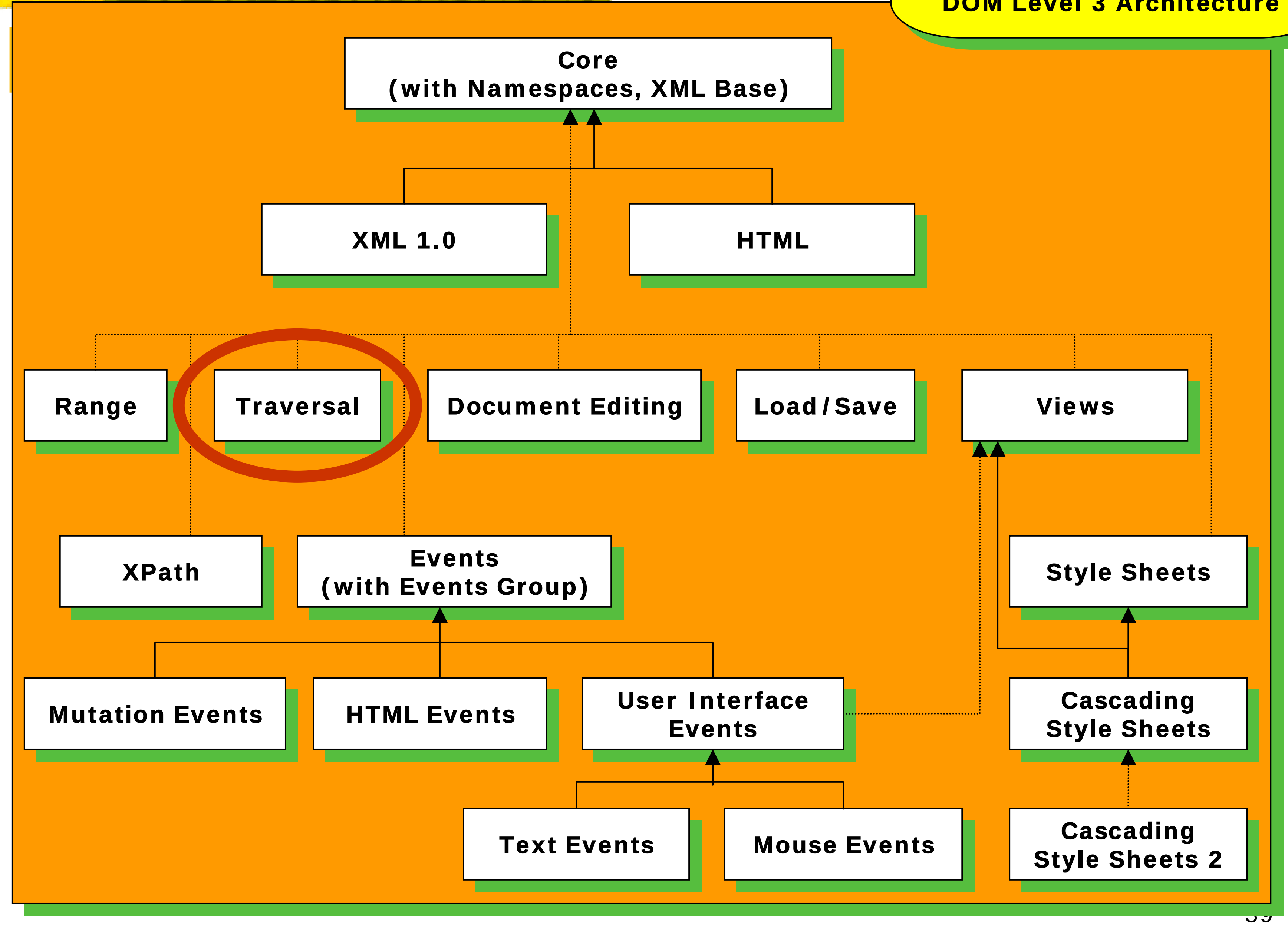
- 다른 Document에서 노드 임포트
  - importNode(importedNode, deep)

```
# dom12.py
from xml.dom.ext.reader.Sax2 import Reader
from xml.dom.ext import PrettyPrint

reader = Reader()
doc = reader.fromUri('sample02.xml')     # 문서 읽기
doc2 = reader.fromString('<userlist/>')  # 기본 문서 생성

node = doc2.importNode(doc.documentElement.childNodes[1], 1)
doc2.documentElement.appendChild(node)

PrettyPrint(doc2)
```

## DOM Traversal 인터페이스

- DOM 레벨 2 인터페이스
- 지원 인터페이스
  - NodeIterator
    - 1차원 적으로 노드 접근
  - TreeWalker
    - 트리 방식으로 노드 접근
  - NodeFilter
    - 접근할 노드를 필터링

## DOM Level 3 Architecture



$\longrightarrow$ Inherits $\cdots\cdots\blacktriangleright$ Depends on

## Node Iterator

- 반복자 객체 생성
  - createNodeIterator(root, whatToShow, filter, entityReferenceExpansion)
- 반복자 메쏘드
  - nextNode()
    - 다음 노드로 진행
  - previousNode()
    - 이전 노드로 진행
  - detach()
    - 반복자 사용 종료

# Node Iterator

- 예제:

```
# dom13.py
from xml.dom.NodeFilter import NodeFilter
from xml.dom.ext.reader.Sax2 import Reader

reader = Reader()
doc = reader.fromUri('sample02.xml')

iterator = doc.createNodeIterator(doc, NodeFilter.SHOW_ELEMENT,
None, 0)
elem = iterator.nextNode()
while elem:
    print elem.nodeName
    elem = iterator.nextNode()
```

## Node Iterator

- 샘플 파일
  - sample02.xml

```
<?xml version="1.0" ?>
<userlist>
    <user sex="male">
        <name>gslee</name>
        <email>gslee@mail.gwu.ac.kr</email>
    </user>
    <user sex="female">
        <name>spam</name>
        <email>spam@mail.gwu.ac.kr</email>
    </user>
</userlist>
```

## Node Iterator

- 실행 결과

```
userlist
user
name
email
user
name
email
```

## Node filter

- 임포트
  - from xml.dom.NodeFilter import NodeFilter
- 필터 상수의 사용
  - iterator = doc.createNodeIterator(doc, NodeFilter.SHOW_ELEMENT, None, 0)

# Node filter

- 필터가 갖는 상수 값

```
SHOW_ALL
SHOW_ELEMENT
SHOW_ATTRIBUTE
SHOW_TEXT
SHOW_CDATA_SECTION
SHOW_ENTITY_REFERENCE
SHOW_ENTITY
SHOW_PROCESSING_INSTRUCTION
SHOW_COMMENT
SHOW_DOCUMENT
SHOW_DOCUMENT_TYPE
SHOW_DOCUMENT_FRAGMENT
SHOW_NOTATION
```

# Node filter

- 예제 : 텍스트 정보만 추출

```
# dom14.py
from xml.dom.NodeFilter import NodeFilter
from xml.dom.ext.reader.Sax2 import Reader

reader = Reader()
doc = reader.fromUri('sample02.xml')

iterator = doc.createNodeIterator(doc,
NodeFilter.SHOW_TEXT, None, 0)
elem = iterator.nextNode()
while elem:
    print elem.data,
    elem = iterator.nextNode()
```

## Node filter

- 필터의 설계
  - NodeFilter 인터페이스 상속
  - acceptNode 메쏘드 재정의
    - 전달되는 노드를 취하려면 FILTER_ACCEPT,
    - 버리려면 FILTER_REJECT 리턴

```
class emailFilter(NodeFilter):
    def acceptNode(self, node):
        if node.localName == 'email':
            return self.FILTER_ACCEPT
        else:
            return self.FILTER_REJECT
```

## Node filter

- 예제 : 이메일 주소 추출

```
# dom15.py
from xml.dom.NodeFilter import NodeFilter
from xml.dom.ext.reader.Sax2 import Reader

class emailFilter(NodeFilter):
    def acceptNode(self, node):
        if node.localName == 'email':
            return self.FILTER_ACCEPT
        else:
            return self.FILTER_REJECT

reader = Reader()
doc = reader.fromUri('sample02.xml')

iterator = doc.createNodeIterator(doc,
NodeFilter.SHOW_ELEMENT, emailFilter(), 0)
elem = iterator.nextNode()
while elem:
    print elem.firstChild.data
    elem = iterator.nextNode()
```

## Tree walker

- Node Iterator와 유사하나 트리 관점에서 이동가능
  - nextNode(), previousNode()
  - parentNode()
  - previousSibling(), nextSibling()
  - firstChild() lastChild()
- 예제 : Preorder 트리 탐색

## Tree walker

```
# dom17.py
from xml.dom.NodeFilter import NodeFilter
from xml.dom.ext.reader.Sax2 import Reader

def doSomething(node):
   if node.nodeName in ('email', 'name'):
       print node.nodeName, ':', node.firstChild.data
   else:
      print node.nodeName

def processMe(tw):
   n = tw.currentNode
   doSomething(n)
   child = tw.firstChild()
   while child:
       processMe(tw)
       child = tw.nextSibling()
   tw.currentNode = n

reader = Reader()
doc = reader.fromUri('sample02.xml')

tw = doc.createTreeWalker(doc, NodeFilter.SHOW_ELEMENT, None, 1)
processMe(tw)
```