

# **INTRODUCCIÓN XML**

**ADOLFO SANZ DE DIEGO**

**NOVIEMBRE 2016**

# **1 ACERCA DE**

## 1.1 AUTOR

- Adolfo Sanz De Diego
  - Blog: [asanzdiego.blogspot.com.es](http://asanzdiego.blogspot.com.es)
  - Correo: [asanzdiego@gmail.com](mailto:asanzdiego@gmail.com)
  - GitHub: [github.com/asanzdiego](https://github.com/asanzdiego)
  - Twitter: [twitter.com/asanzdiego](https://twitter.com/asanzdiego)
  - LinkedIn: [in/asanzdiego](https://in/asanzdiego)
  - SlideShare: [slideshare.net/asanzdiego](https://slideshare.net/asanzdiego)

## 1.2 LICENCIA

- Copyright:
  - Antonio Sarasa Cabezuelo  
[<antoniosarasa@campusciff.net>](mailto:antoniosarasa@campusciff.net)

## 1.3 FUENTE

- Las slides y sus fuentes las podéis encontrar en:
  - <https://github.com/asanzdiego/curso-intro-linux-web-sql-2016>

# **2 INTRODUCCIÓN A XML**

## 2.1 ¿QUÉ ES?

- XML (Extensible Markup Language) es un metalenguaje que permite definir lenguajes de marcado.
- Los lenguajes de marcado permiten describir la estructura de los contenidos de un documento.

## 2.2 ETIQUETAS

- Un lenguaje de marcado está formado por un conjunto de etiquetas que se encierran entre **corchetes angulares**, <>, y se usan en pares.
- Cada par de etiquetas delimita el comienzo y el final de una porción de documento a la que se refiere la etiqueta. Por ejemplo:

```
<asignatura>Bases de datos</asignatura>
```

## 2.3 EJEMPLO XML

```
1 ①<banco>
2 ②<cuenta>
3   <numero_cuenta> C-101</numero_cuenta>
4   <nombre_sucursal> Centro</nombre_sucursal>
5   <saldo>500</saldo>
6   </cuenta>
7 ③<cuenta>
11 </cuenta>
12 ④<cuenta>
16 </cuenta>
17 ⑤<cliente>
18   <nombre_cliente> González</nombre_cliente>
19   <calle_cliente> Arenal</calle_cliente>
20   <ciudad_cliente> La Granja</ciudad_cliente>
21 </cliente>
22 ⑥<cliente>
26 </cliente>
27 ⑦<impositor>
28   <numero_cuenta> C-101</numero_cuenta>
29   <nombre_cliente>González</nombre_cliente>
30   </impositor>
31 ⑧<impositor>
34   </impositor>
35 ⑨<impositor>
38   </impositor>
39 </banco>
```

Ejemplo XML

## 2.4 VENTAJAS

- Permite que la información esté autodocumentada.
- Formato no rígido pues dispone de la capacidad de reconocer e ignorar nuevas etiquetas.
- Las etiquetas pueden aparecer varias veces facilitando la representación de atributos multivaluados.
- Permite el anidamiento de etiquetas.

# **3 ESTRUCTURA BÁSICA**

## 3.1 PROLOGO

- Consta de dos declaraciones:
  - La declaración XML que indica la versión de XML utilizada y el tipo de codificación de caracteres.
  - La declaración de tipo de documento que asocia el documento a una DTD o XSD respecto a la cual el documento es conforme.

## 3.2 ELEMENTOS

- Es un par de etiquetas de comienzo y final coincidentes que delimita una porción de información.

```
<título>introducción</título>
```

## 3.3 ELEMENTOS VACÍOS

- Existen elementos vacíos que no contienen contenido.

```
<Nombre etiqueta/>  
<Nombre etiqueta></Nombre etiqueta>
```

## 3.4 ELEMENTOS ANIDADOS

- Los elementos se pueden anidar:
  - Un texto aparece en el contexto de un elemento si aparece entre la etiqueta de inicio y final de dicho elemento.
  - Las etiquetas se anidan correctamente si toda etiqueta de inicio tiene un única etiqueta de finalización coincidente que está en el contexto del mismo elemento padre.
- Un elemento puede aparecer varias veces en un documento XML.

## 3.5 EJEMPLO ANIDADO

```
1 □ <pedido_compra>
2   <identificador> P-101</identificador>
3 □ <comprador>
4   <nombre> Coyote Loco</nombre>
5   <dirección> Mesa Flat, Route 66, Arizona 12345, EEUU</dirección>
6   </comprador>
7 □ <proveedor>
8   <nombre> Proveedores Acme SA</nombre>
9   <dirección> 1, Broadway, Nueva York, NY, EEUU</dirección>
10  </proveedor>
11 □ <lista_elementos>
12 □ <elemento>
13   <identificador>EAl</identificador>
14   <descripción> Trineo propulsado por cohetes atómicos</descripción>
15   <cantidad>2</cantidad>
16   <precio>199.5</precio>
17 </elemento>
18 □ <elemento>
19   <identificador>PF2</identificador>
20   <descripción> Pegamento fuerte</descripción>
21   <cantidad>1</cantidad>
22   <precio>29.95</precio>
23 </elemento>
24 </lista_elementos>
25 <coste_total> 429.85</coste_total>
26 <forma_de_pago> Reembolso</forma_de_pago>
27 <forma_de_envio>Avión</forma_de_envio>
28 </pedido_compra>
```

Elementos anidados

## 3.6 ATRIBUTOS

- Las etiquetas de los elementos pueden incluir 1 o más **atributos que representan propiedades** de los elementos de la forma Nombre atributo="Valor atributo"

```
<cuenta tipo_cuenta="corriente">
```

- Los atributos pueden aparecer solamente una vez en una etiqueta dada.

## 3.7 MEZCLA

- El texto en un documento XML puede estar mezclado con los subelementos de otro elemento.

```
1 <cuenta>
2 Esta cuenta se usa muy rara vez, por no decir nunca
3 <numero_cuenta> C-102</numero_cuenta>
4 <nombre_sucursal> Navacerrada</nombre_sucursal>
5 <saldo>400</saldo>
6 </cuenta>
```

Mezcla texto con subelementos

## 3.8 RAÍZ

- Todo documento XML tiene un único elemento raíz que engloba al resto de elementos del documento.
- En el primer ejemplo el elemento era la raíz.

## 3.9 COMENTARIOS

- Es un texto que se escribe entre `<!-- y -->`
- La cadena `--` no puede aparecer dentro de un comentario.
- Los comentarios pueden aparecer en cualquier sitio salvo dentro de declaraciones, etiquetas y dentro de otros comentarios.

## 3.10 ESPACIO DE NOMBRES

- Es un mecanismo que permite especificar nombre únicos globalmente para que se usen como marcas de elementos en los documentos XML.
- Para ello se antepone a la etiqueta o atributo un identificador de recursos universal. En el ejemplo del banco podría ser <http://www.BancoPrincipal.com>
- Para abreviarlo se declaran abreviaturas del espacio de nombres mediante el atributo `xmlns`

## 3.11 EJEMPLOS ESPACIO DE NOMBRES

```
1 | <banco xmlns:BP="http://www.BancoPrincipal.com">
2 | ...
3 | <BP:sucursal>
4 |   <BP:nombre_sucursal> Centro </BP:nombre_sucursal>
5 |   <BP:ciudad_sucursal> Arganzuela</BP:ciudad_sucursal>
6 | </BP:sucursal>
7 | ...
8 | </banco>
```

Ejemplo espacio de nombres

## 3.12 VARIOS ESPACIOS DE NOMBRES

- Un documento puede tener más de un espacio de nombres declarado como parte del elemento raíz, de manera que se puede asociar **elementos diferentes con espacios de nombres distintos**.

## **3.13 ESPACIO DE NOMBRE PREDETERMINADO**

- Se puede definir un espacio de nombres predeterminado mediante el uso del atributo xmlns en el elemento raíz.
- Los elementos sin un prefijo de espacio de nombres explícito pertenecen entonces al espacio de nombres predeterminado.

## 3.14 CDATA

- A veces es necesario almacenar valores que contienen etiquetas sin que se interpreten como etiquetas XML, es decir como texto normal. Para ello se usa la construcción:

```
<! [CDATA]<cuenta>...</cuenta> ]>
```

# **4 PROCESAMIENTO DE XML**

## 4.1 EJEMPLO

- Se va a considerar el siguiente documento XML de ejemplo para ilustrar las diferentes técnicas de procesamiento.

```
<catalogo>
  <Libro isbn="0-596-00128-2">
    <titulo>Python y XML</titulo>
    <fecha>Diciembre 2001</fecha>
    <autor>Pepito Perez</autor>
  </Libro>
  <Libro isbn="0-596-15810-6">
    <titulo>Programacion avanzada de XML</titulo>
    <fecha>Octubre 2010</fecha>
    <autor>Juan Garcia</autor>
  </Libro>
  <Libro isbn="0-596-15806-8">
    <titulo>Aprendiendo Java</titulo>
    <fecha>Septiembre 2009</fecha>
    <autor>Juan Garcia</autor>
  </Libro>
```

Ejemplo procesamiento

## 4.2 ELEMENTTREE

- ElementTree es una **librería estándar para procesar y crear documentos XML** que crea un árbol de objetos.
- El árbol generado esta formado por objetos "elemento" de tipo Element donde cada uno de ellos dispone de un conjunto de atributos: nombre, diccionario de atributos, valor textual y secuencia de elementos hijo.

## 4.3 ABRIR XML

- Para procesar un documento basta abrir el documento con el método `open()` como si se tratara de un fichero y usar el método `parse` de `ElementTree`.

```
from xml.etree import ElementTree  
  
f= open("Catalogo.xml", "rt")      <xml.etree.ElementTree.ElementTree object at 0x0219A070>  
arbol=ElementTree.parse(f)  
print arbol
```

Código abrir XML

## 4.4 ITERAR XML

- Si se quiere visitar todo el árbol se usa el método `iter()` que crea un generador que itera sobre todos los nodos del árbol.

```
catalogo {}
Libro {'isbn': '0-596-00128-2'}
titulo {}
fecha {}
autor {}

from xml.etree import ElementTree
Libro {'isbn': '0-596-15810-6'}
titulo {}
fecha {}
autor {}

f= open("Catalogo.xml", "rt")
arbol=ElementTree.parse(f)
Libro {'isbn': '0-596-15806-8'}
for nodo in arbol.iter():
    print nodo.tag, nodo.attrib
    titulo {}
    fecha {}
    autor {}

    Libro {'isbn': '0-596-15808-4'}
    titulo {}
    fecha {}
    autor {}
```

Código iterar XML

## 4.5 FILTRAR XML

- Puede que se esté interesado sólo en determinados elementos del árbol, y no en todos. Para ello **se pasa como parámetro del método iter()** el nombre del elemento de interés.

```
from xml.etree import ElementTree
f= open("Catalogo.xml", "rt")
arbol=ElementTree.parse(f)
i=1
for nodo in arbol.iter("Libro"):
    isbn=nodo.attrib.get("isbn")
    print nodo.tag, i, " con isbn:", isbn
    i=i+1
Libro 1 con isbn: 0-596-00128-2
Libro 2 con isbn: 0-596-15810-6
Libro 3 con isbn: 0-596-15806-8
Libro 4 con isbn: 0-596-15808-4
Libro 5 con isbn: 0-596-00797-3
Libro 6 con isbn: 0-596-10046-9
```

Código filtrar XML

## 4.6 ITERAR DESDE RAÍZ

- Otra posibilidad de iterar sobre los elementos del árbol es acceder a la raíz del árbol y desde ella iterar sobre los hijos.

```
import xml.etree.ElementTree as ET      catalogo  {}
arbol=ET.parse("Catalogo.xml")          Libro   {'isbn': '0-596-00128-2'}
raiz=arbol.getroot()                   Libro   {'isbn': '0-596-15810-6'}
print raiz.tag, " ", raiz.attrib       Libro   {'isbn': '0-596-15806-8'}
for hijo in raiz:                     Libro   {'isbn': '0-596-15808-4'}
    print hijo.tag, " ", hijo.attrib    Libro   {'isbn': '0-596-00797-3'}
                                         Libro   {'isbn': '0-596-10046-9'}
```

Código iterar XML desde raíz

## 4.7 ACCESO INDEXADO

- También es posible acceder a los elementos de forma indexada.

```
import xml.etree.ElementTree as ET
arbol=ET.parse("Catalogo.xml")
raiz=arbol.getroot()           Titulo : Python y XML
print "Titulo :",raiz[0][0].text
|
```

Código acceso indexado

## 4.8 BUSCAR

- **find()**: recupera el primer subelemento del elemento actual encajando con la descripción dada
- **findall()**: recupera todos los subelementos del elemento actual encajando con la descripción dada
- **iterfind()**: recupera todos los elementos encajando con la descripción dada.
- **text**: accede al contenido textual de un elemento
- **get(atributo)**: accede al atributo dado del elemento.

## 4.9 EJEMPLO FINDALL()

- Se van a encontrar todos los títulos de los libros usando `findall()`.

```
from xml.etree import ElementTree
f= open("Catalogo.xml", "rt")
arbol=ElementTree.parse(f)
i=1
for nodo in arbol.findall("./Libro/titulo"):
    print "Titulo ",i," ",nodo.text
    i=i+1
```

Titulo 1 Python y XML
Titulo 2 Programacion avanzada de XML
Titulo 3 Aprendiendo Java
Titulo 4 Python para moviles
Titulo 5 R para estadistica
Titulo 6 Python en 100 paginas

Código ejemplo `findAll()`

## 4.10 USO DE EVENTOS

- Se puede realizar un procesamiento basado en eventos usando el método `iterparse()`:
  - Genera eventos "start" en las aperturas de elemento y eventos "end" en los cierres de elemento.
  - Además los datos pueden ser extraídos del documento durante la fase de parseo.

## 4.11 EJEMPLO USO EVENTOS

- Ejemplo de parseo dirigido por eventos:

```
from xml.etree.ElementTree import iterparse
for (event, element) in iterparse("Catalogo.xml",('start','end')):
    if event=="start":
        if element.tag=="Libro":
            print "****Libro****"
            print "isbn:",element.attrib["isbn"]
    if event=="end":
        if element.tag=="titulo":
            print "Titulo :",element.text
        if element.tag=="fecha":
            print "Fecha :",element.text
        if element.tag=="autor":
            print "Autor :",element.text
****Libro****
isbn: 0-596-00128-2
Titulo : Python y XML
Fecha : Diciembre 2001
Autor : Pepito Perez
****Libro****
isbn: 0-596-15810-6
Titulo : Programacion avanzada de XML
Fecha : Octubre 2010
Autor : Juan Garcia
****Libro****
isbn: 0-596-15806-8
Titulo : Aprendiendo Java
Fecha : Septiembre 2009
Autor : Juan Garcia
```

Código ejemplo interfase()

## 4.12 DESDE CADENA

- También es posible procesar cadenas que representan un documento XML usando el método `fromstring()` que toma como argumento la cadena que representa el documento XML.

```
import xml.etree.ElementTree as ET
cadena = '''
<catalogo>
    <Libro isbn="0-596-00128-2">
        <titulo>Python y XML</titulo>
        <fecha>Diciembre 2001</fecha>
        <autor>Pepito Perez</autor>
    </Libro>
</catalogo>
'''

doc=ET.fromstring(cadena)
lista=doc.findall("Libro")
for l in lista:
    print "***Libro***"
    print "isbn: ", l.get("isbn")
    print "Titulo :", l.find("titulo").text
    print "Fecha :", l.find("fecha").text
    print "Autor :", l.find("autor").text
```

Código ejemplo `fromstring()`

## 4.13 MODIFICAR XML

- Se puede modificar un documento XML que ha sido leído:
  - A nivel de elemento se puede cambiar el contenido cambiando el **valor** de `Element.text`, añadir o modificar atributos con el **método** `Element.set()`, y añadir nuevos hijos con el **método** `Element.append()`.
  - A nivel de documento, se escribe el nuevo documento con el **método** `ElementTree.write()`

## **4.14 EXPLICAR EJEMPLO MODIFICAR XML**

- Se va a modificar el documento XML de ejemplo:
  - Se va añadir un nuevo atributo que indica el orden.
  - Se va añadir un nuevo elemento que indica la editorial.
  - Se va añadir un nuevo atributo que indica si hay ejemplares.

## 4.15 EJEMPLO MODIFICAR XML

```
<catalogo>
    <Libro ejemplares="si" isbn="0-596-00128-2" orden="1">
        <titulo>Python y XML</titulo>
        <fecha>Diciembre 2001</fecha>
        <autor>Pepito Perez</autor>
        <editorial>Anaya</editorial>
    </Libro>
    <Libro ejemplares="si" isbn="0-596-15810-6" orden="2">
        <titulo>Programacion avanzada de XML</titulo>
        <fecha>Octubre 2010</fecha>
        <autor>Juan Garcia</autor>
        <editorial>Anaya</editorial>
    </Libro>
    <Libro ejemplares="si" isbn="0-596-15806-8" orden="3">
        <titulo>Aprendiendo Java</titulo>
        <fecha>Septiembre 2009</fecha>
        <autor>Juan Garcia</autor>
        <editorial>Anaya</editorial>
    </Libro>

import xml.etree.ElementTree as ET
arbol=ET.parse("Catalogo.xml")
i=1
for libro in arbol.iter("Libro"):
    cadena=str(i)
    libro.set("orden",cadena)
    libro.set("ejemplares","si")
    editorial=ET.Element("editorial")
    editorial.text="Anaya"
    libro.append(editorial)
    i=i+1

arbol.write("Catalogo2.xml")
```

Código ejemplo modificar XML

## 4.16 ELIMINAR ELEMENTOS

- También es posible eliminar elementos con el método `Element.remove()`.
- Tomando como entrada la salida del ejemplo anterior se van a eliminar todos los elementos de tipo "Libro" que tengan un número de orden mayor que 3.

## 4.17 EJEMPLO ELIMINAR ELEMENTOS

```
<catalogo>
    <Libro ejemplares="si" isbn="0-596-00128-2" orden="1">
        <titulo>Python y XML</titulo>
        <fecha>Diciembre 2001</fecha>
        <autor>Pepito Perez</autor>
        <editorial>Anaya</editorial>
    </Libro>
    <Libro ejemplares="si" isbn="0-596-15810-6" orden="2">
        <titulo>Programacion avanzada de XML</titulo>
        <fecha>Octubre 2010</fecha>
        <autor>Juan Garcia</autor>
        <editorial>Anaya</editorial>
    </Libro>
    <Libro ejemplares="si" isbn="0-596-15806-8" orden="3">
        <titulo>Aprendiendo Java</titulo>
        <fecha>Septiembre 2009</fecha>
        <autor>Juan Garcia</autor>
        <editorial>Anaya</editorial>
    </Libro>

import xml.etree.ElementTree as ET
arbol=ET.parse("Catalogo2.xml")
raiz=arbol.getroot()
for libro in raiz.iter("Libro"):
    orden=int(libro.get("orden"))
    if orden== 3:
        raiz.remove(libro)
arbol.write("Catalogo3.xml")
```

Código ejemplo eliminar elementos

## 4.18 CREAR XML

- También es posible la creación de documentos XML desde cero. Para ello se disponen de los siguientes métodos en la clase Element:
  - **Element():** Crea un elemento nuevo.
  - **subElement():** Añade un nuevo elemento al padre.

## 4.19 EJEMPLO CREAR XML

- En el siguiente ejemplo se va a crear un documento XML con información de un libro semejante a los ejemplos anteriores.

```
from xml.etree.ElementTree import Element,SubElement,Comment
from xml.etree import ElementTree
from xml.dom import minidom

def prettyify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')           <?xml version="1.0" ?>
    reparsed = minidom.parseString(rough_string)                 <Catalogo>
    return reparsed.toprettyxml(indent=" ")                      <Libro>
                                                       <titulo>Python y XML</titulo>
                                                       <fecha>Diciembre 2001</fecha>
                                                       <autor>Pepito Perez</autor>
                                                       </Libro>
</Catalogo>

raiz=Element("Catalogo")
Libro=SubElement(raiz, "Libro")
Titulo=SubElement(Libro, "titulo")
Titulo.text="Python y XML"
Fecha=SubElement(Libro, "fecha")
Fecha.text="Diciembre 2001"
Autor=SubElement(Libro, "autor")
Autor.text="Pepito Perez"
print prettyify(raiz)
```

Código ejemplo crear XML

## 4.20 AÑADIR ATRIBUTOS

- Para añadir atributos a un elemento que se está creando basta pasar como argumento del elemento o subelemento un diccionario con los atributos expresados en forma de parejas clave-valor.

# 4.21 EJEMPLO AÑADIR ATRIBUTOS

- Se va a modificar el código anterior para añadir atributos al elemento Libro. En concreto se va añadir el atributo isbn, orden y ejemplares.

```
from xml.etree.ElementTree import Element,SubElement,Comment
from xml.etree import ElementTree
from xml.dom import minidom

def prettyify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent="  ")

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro",{"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
Titulo=SubElement(Libro,"titulo")
Titulo.text="Python y XML"
Fecha=SubElement(Libro,"fecha")
Fecha.text="Diciembre 2001"
Autor=SubElement(Libro,"autor")
Autor.text="Pepito Perez"
print prettyify(raiz)

<?xml version="1.0" ?>
<Catalogo version="1.0">
  <Libro ejemplares="si" isbn="0-596-00128-2" orden="1">
    <titulo>Python y XML</titulo>
    <fecha>Diciembre 2001</fecha>
    <autor>Pepito Perez</autor>
  </Libro>
</Catalogo>
```

Código ejemplo añadir atributos

## 4.22 AÑADIR HIJOS

- Se pueden añadir múltiples hijos a un elemento mediante el método `extend()` que recibe como argumento algo que sea iterable tal como una lista o bien otra instancia de Element.
- En el caso de una instancia de Element, los hijos del elemento dado se añaden como hijos del nuevo padre. Sin embargo el padre actual no es añadido.

## 4.23 EJEMPLO AÑADIR HIJOS

- Se va a reconstruir el ejemplo anterior pero usando extend sobre una cadena dada.

```
from xml.etree.ElementTree import Element, SubElement, XML
from xml.etree import ElementTree
from xml.dom import minidom

def prettyify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent=" ")

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro", {"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
hijos=XML('''<hijos><titulo>Python y XML</titulo><fecha>Diciembre 2001</fecha><autor>Pepito Perez</autor></hijos>''')
Libro.extend(hijos)
print prettyify(raiz)
```

Código ejemplo añadir hijos

## 4.24 EJEMPLO AÑADIR HIJOS CON LISTA

- También se podría haber construido pasando una lista.

```
from xml.etree.ElementTree import Element, SubElement, XML
from xml.etree import ElementTree
from xml.dom import minidom

def prettyify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent=" ")

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro", {"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
titulo=Element("titulo")
titulo.text="Python y XML"
fecha=Element("fecha")
fecha.text="Diciembre 2001"
autor=Element("autor")
autor.text="Pepito Perez"
hijos=[titulo,fecha,autor]
Libro.extend(hijos)
print prettyify(raiz)
```

Código ejemplo añadir hijos con lista

## **4.25 GUARDAR XML**

- A veces interesa guardar un documento XML en un archivo. En estos casos se usará el método `write()` de `ElementTree`.

## 4.26 EJEMPLO GUARDAR XML

- Se va a realizar el mismo ejemplo de antes pero ahora el resultado se almacenará en un archivo.

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

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro",{"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
titulo=Element("titulo")
titulo.text="Python y XML"
fecha=Element("fecha")
fecha.text="Diciembre 2001"
autor=Element("autor")
autor.text="Pepito Perez"
hijos=[titulo,fecha,autor]
Libro.extend(hijos)
ElementTree(raiz).write("Ejemplo.xml")
```

<Catalogo version="1.0">  
 <Libro ejemplares="si" isbn="0-596-00128-2" orden="1">  
 <titulo>Python y XML</titulo>  
 <fecha>Diciembre 2001</fecha>  
 <autor>Pepito Perez</autor>  
 </Libro>  
</Catalogo>

Código ejemplo guardar XML

## 4.27 ELEMENTOS VACIOS

- El método write() de ElementTree tiene un segundo argumento que sirve para **controlar que se hace con elementos que son vacíos**. Existen tres posibilidades según el valor de dicho argumento:
  - **xml**: Genera un elemento vacío con una sola etiqueta
  - **html**: Genera un elemento vacío con dos etiquetas.
  - **text**: Imprime solo elementos con contenido, el resto se los salta.

## 4.28 EJEMPLO ELEMENTOS VACIOS (I)

- Siguiendo con el ejemplo anterior se va añadir un elemento vacío y se van a probar los tres argumentos.

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

def prettyify(elem):
    """Return a pretty-printed XML string for the Element.
    """
    rough_string = ElementTree.tostring(elem, 'utf-8')
    reparsed = minidom.parseString(rough_string)
    return reparsed.toprettyxml(indent=" ")

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro",{"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
titulo=Element("titulo")
titulo.text="Python y XML"
fecha=Element("fecha")
fecha.text="Diciembre 2001"
autor=Element("autor")
autor.text="Pepito Perez"
hijos=[titulo,fecha,autor]
Libro.extend(hijos)
ElemVacio=SubElement(Libro,"vacio")

print prettyify(raiz)

<?xml version="1.0" ?>
<Catalogo version="1.0">
  <Libro ejemplares="si" isbn="0-596-00128-2" orden="1">
    <titulo>Python y XML</titulo>
    <fecha>Diciembre 2001</fecha>
    <autor>Pepito Perez</autor>
    <vacio/>
  </Libro>
</Catalogo>
```

Código ejemplo elementos vacíos I

## 4.29 EJEMPLO ELEMENTOS VACIOS (II)

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

raiz=Element("Catalogo")
raiz.set("version","1.0")
Libro=SubElement(raiz,"Libro",{"orden":"1","ejemplares":"si","isbn":"0-596-00128-2"})
titulo=Element("titulo")
titulo.text="Python y XML"
fecha=Element("fecha")
fecha.text="Diciembre 2001"
autor=Element("autor")
autor.text="Pepito Perez"
hijos=[titulo,fecha,autor]
Libro.extend(hijos)
ElemVacio=SubElement(Libro,"vacio")

for metodo in ["xml","html","text"]:
    print metodo
    ElementTree(raiz).write(sys.stdout,method=metodo)
    print "\n"
```

Código ejemplo elementos vacíos II

## 4.30 EJEMPLO ELEMENTOS VACIOS (III)

```
xml
<Catalogo version="1.0"><Libro ejemplares="si" isbn="0-596-00128-2" orden="1"><titulo>Py
thon y XML</titulo><fecha>Diciembre 2001</fecha><autor>Pepito Perez</autor><vacio /></Li
bro></Catalogo>

html
<Catalogo version="1.0"><Libro ejemplares="si" isbn="0-596-00128-2" orden="1"><titulo>Py
thon y XML</titulo><fecha>Diciembre 2001</fecha><autor>Pepito Perez</autor><vacio></vaci
o></Libro></Catalogo>

text
Python y XMLDiciembre 2001Pepito Perez
```

Código ejemplo elementos vacíos III