DOM:

El modelo de objetos de documento, o "DOM" (Document Object Model), es una API entre idiomas del World Wide Web Consortium (W3C) para acceder y modificar documentos XML. Una implementación de DOM presenta un documento XML como una estructura de árbol o permite que el código de cliente compile una estructura de este tipo desde cero. A continuación, da acceso a la estructura a través de un conjunto de objetos que proporcionaba interfaces conocidas.

Para analizarlo en Python existen muchas librerías entre estas:

xml.dom.minidom: es una implementación mínima de la interfaz DOM con una API similar a la de otros lenguajes. Está destinada a ser más simple que una implementación completa del DOM y también significativamente más pequeña. las aplicaciones DOM suelen comenzar analizando algún XML en un DOM. Con xml.dom.minidom, esto se hace a través de las funciones de análisis sintáctico:

```
from xml.dom.minidom import parse, parseString

dom1 = parse('c:\\temp\\mydata.xml')  #Analiza un archivo xml  encontrándolo a través de una dirección y el nombre de un archivo.

datasource = open('c:\\temp\\mydata.xml')  #Obtiene la dirección del archivo
dom2 = parse(datasource)  #Analiza el archivo con la dirección anterior

dom3 = parseString('<myxml>Some data<empty/> some more data</myxml>') #Analiza un texto en formato xml.
```

La función parse() puede tomar un nombre de archivo o un objeto de archivo previamente abierto.

xml.dom.minidom.parse(*Nombrearchivo_o_direccion*, *analizador=None*, *bufsize=None*): Retorna un Document a partir de la entrada dada.. *parser*, si se proporciona, debe ser un objeto de un analizador sintáctico SAX2. Esta función intercambiará el controlador de documentos del analizador sintáctico y activará el soporte con el espacio de nombres.

Si tienes XML en una cadena de texto , puedes usar la función parseString() en su lugar:

xml.dom.minidom.parseString(*Cadena_texto*, *parser=None*):
Retorna un objeto Document que representa a *string*. Este método crea un objeto io.StringIO para la cadena de caracteres y lo pasa a parse().

Ambas funciones retornan un objeto Document que representa el contenido del documento.

Lo que hacen las funciones parse() y parseString() es conectar un analizador sintáctico de XML con un «constructor DOM» que puede aceptar eventos de análisis de cualquier analizador sintáctico SAX y convertirlos en un árbol DOM. El nombre de las funciones es quizás engañoso, pero es fácil de entender cuando se comprenden las interfaces. El análisis sintáctico del documento se completará antes de que retornen estas funciones, dichas funciones simplemente no proporcionan una implementación del analizador sintáctico por si mismas.

También puedes crear un objeto Document invocando a un método en un objeto de la «Implementación del DOM». Puedes obtener este objeto llamando a la función getDOMImplementation() del paquete xml.dom o del módulo xml.dom.minidom. Una vez que tengas un objeto Document, puedes agregarle nodos secundarios para llenar el DOM:

```
from xml.dom.minidom import getDOMImplementation
impl = getDOMImplementation()
newdoc = impl.createDocument(None, "some_tag", None)
top_element = newdoc.documentElement
text = newdoc.createTextNode('Some textual content.')
top_element.appendChild(text)
```

Para extraer valores de un xml ya analizado en python se utilizan los métodos clásicos de DOM:

METODOS:

- x.getElementsByTagName(*name*) - obtener todos los elementos con un nombre de etiqueta especificado (x es un objeto de nodo).
- x.appendChild(*node*) - inserte un nodo secundario en x.
- x.removeChild(*node*) - eliminar un nodo secundario de x.
- childNodes[0] - el primer elemento secundario del elemento <title> (el nodo de texto).
- nodeValue: Obtiene el valor del nodo.

PROPIEDADES:

- x.nodeName - el nombre de x (x es un objeto de nodo)
- x.nodeValue - el valor de x
- x.parentNode - el nodo primario de x
- x.childNodes - los nodos secundarios de x

- x.attributes - los nodos de atributos de x

EJEMPLOS:

1.

```
from xml.dom.minidom import paseString
xml="""\
<Libreria>
<Libro>
<titulo>Himnos Nacionales</titulo>
<Autor>Fernando Montes</Autor>
</Libro>
<Libro>
<titulo>Cantares de Hoy y Mañana</titulo>
<Autor>Jose Marroquin</Autor>
</Libro>
</Libreria >
"""
dom = parseString(xml)
name = dom.getElementsByTagName('titulo')[0].childNodes[0].nodeValue
print(name)
```

2.

```
from xml.dom.minidom import parseString
xml="""\
<Restaurante>
<Platillo>
<Nombre>Pollo a la Plancha</Nombre>
<Precio>30.00</Precio>
</Platillo>

<Platillo>
<Nombre>Carne Asada</Nombre>
<Precio>35.50</Precio>
</Platillo>
</Restaurante >
"""
dom = parseString(xml)
Nombre= dom.getElementsByTagName('Platillo')[1].firstChild.nodeValue

print(Nombre)
```

3.

```
from xml.dom.minidom import parseString
xml="""\
<Hotel>
<Cuarto>
<No_Cuarto>354</ No_Cuarto >
<Precio>60.00</Precio>
</Cuarto>

<Cuarto>
<No_Cuarto>215</ No_Cuarto >
<Precio>75.00</Precio>
</Cuarto>
</Hotel >
"""
dom = parseString(xml)
NoCuarto = dom.getElementsByTagName('No_Cuarto')[1].lastChild.nodeValue
print(NoCuarto)
```

4.

```
from xml.dom.minidom import parse
dom = parse(“C:\\Notas.xml”)
nota = dom.getElementsByTagName('Alumno')[3].childNodes[2].nodeValue
print(nota)
```

5.

```
from xml.dom.minidom import parse
dom = parse(“C:\\Inscripciones.xml”)
Asignatura=
dom.getElementsByTagName(‘Asignatura')[3].childNodes[2].nodeValue
print(Asignatura)
```

XPATH

Xpath es un módulo que es parte de la librería xml.etree.ElementTree por lo general la misma se importa de la siguiente manera:

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

Xpath provee una serie de expresiones para localizar elementos en un árbol, su finalidad es proporcionar un conjunto de sintaxis, por lo que debido a su limitado alcance no se considera un motor en si mismo.

Para importar archivos de mejor forma se realiza lo siguiente:

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

o bien leyéndolo directamente como una cadena:

```
root = ET.fromstring(nombre_as_string)
```

fromstring() analiza XML de una cadena directamente en un Element, que es el elemento raíz del árbol analizado.

Xpath Sintaxis:

SINTAXIS	Descripción
tag	Selecciona todos los elementos hijos contenidos en la etiqueta "tag", Por ejemplo: spam, selecciona todos los elementos hijos de la etiqueta spam y así sucesivamente en un path de nodos spam/egg, /spam/egg/milk
*****	Selecciona todos los elementos hijos. Ejemplo: */egg, seleccionara todos los elementos nietos bajo la etiqueta egg
.	Selecciona el nodo actual, este es muy usado en el inicio del path, para indicar que es un path relativo.
//	Selecciona todos los sub elementos de todos los niveles bajo el nodo expresado. Por ejemplo: .//egg selecciona todos los elementos bajo egg a través de todo el arbol bajo la etiqueta
..	Selecciona el elemento padre
[@attrib]	Selecciona todos los elementos que contienen el atributo tras el "@"
[@attrib='value']	Seleccione todos los elementos para los cuales el atributo dado tenga un valor dado, el valor no puede contener comillas

[tag]	Selecciona todos los elementos que contienen una etiqueta hijo llamada tag. Solo los hijos inmediatos son admitidos
[tag='text']	Selecciona todos los elementos que tienen una etiqueta hijo llamada tag incluyendo descendientes que sean igual al texto dado
[position]	Selecciona todos los elementos que se encuentran en la posición dada. La posición puede contener un entero siendo 1 la primera posición, la expresión last( ) para la última, o la posición relativa con respecto a la ultima posición last( )-1

notas: las expresiones entre corchetes deben ser precedidas por un nombre de etiquta, un astertisco u otro comodín. Las referencias a position deben ser precedidas por una etiqueta xml válida.

EJEMPLOS:

XML Usado (País.xml):

```
<?xml version="1.0"?>
<data>
   <country name="Liechtenstein">
      <rank updated="yes">2</rank>
      <year>2008</year>
      <gdppc>141100</gdppc>
      <neighbor name="Austria" direction="E"/>
      <neighbor name="Switzerland" direction="W"/>
   </country>
   <country name="Singapore">
      <rank updated="yes">5</rank>
      <year>2011</year>
      <gdppc>59900</gdppc>
      <neighbor name="Malaysia" direction="N"/>
   </country>
   <country name="Panama">
      <rank updated="yes">69</rank>
      <year>2011</year>
      <gdppc>13600</gdppc>
      <neighbor name="Costa Rica" direction="W"/>
      <neighbor name="Colombia" direction="E"/>
   </country>
</data>
```

1.

```
import xml.etree.ElementTree as ET

root = ET.fromstring(País_as_string)

# Elementos de nivel superior
root.findall(".")
```

2.

```
import xml.etree.ElementTree as ET

root = ET.fromstring(País_as_string)

# todos los hijos de neighbor o nietos de country en el nivel superior
root.findall("./country/neighbor")
```

3.

```
import xml.etree.ElementTree as ET

root = ET.fromstring(País_as_string)

# Nodos xml con name='Singapore' que sean hijos de 'year'
raiz.findall(".//year/..[@name='Singapore']")
```

4.

```
import xml.etree.ElementTree as ET
root = ET.fromstring(País_as_string)
# nodos 'year' que son hijos de etiquetas xml donde name='Singapore'
root.findall(".//*[@name='Singapore']/year")
```

5.

```
import xml.etree.ElementTree as ET
root = ET.fromstring(País_as_string)
# todos los nodos 'neighbor'que son el segundo hijo de su padre
root.findall(".//neighbor[2]")
```