

Universidad de San Carlos de Guatemala

Facultad de Ingeniería

Introducción a la Programación y Computación 2

Sección E

Catedrático: MsC Ing. Estuardo Zapeta

Tutor académico: Mónica Calderón

Actividad 2:

DOM XML y Utilización de Librería XPath para Python

Nombre: Gerson Sebastian Quintana Berganza

Carné: 201908686

Uso de DOM XML para Python

Para aprender a utilizar DOM XML para Python primero hay que aprender lo que es DOM.

¿Qué es DOM?

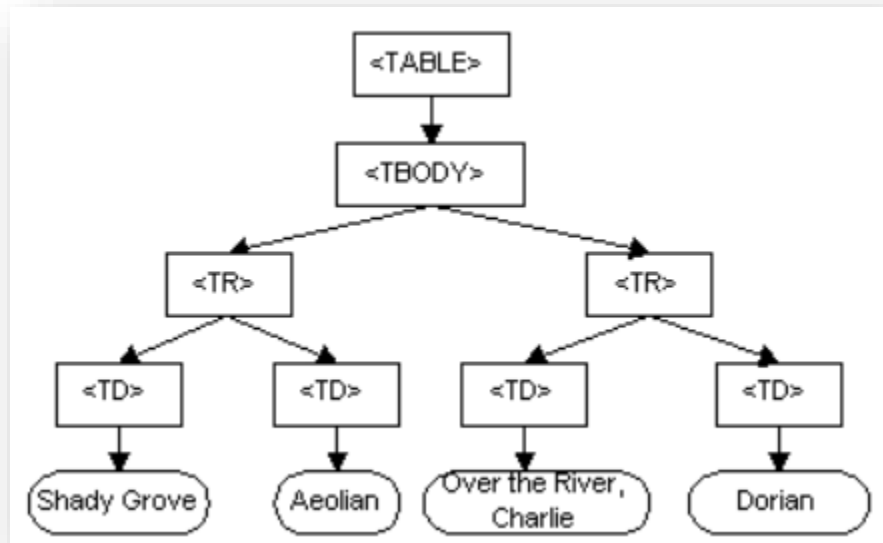
DOM (Modelo de Objetos del Documento, en español), es el que define la estructura lógica de los documentos y el modo en que se accede y manipula un documento. Con este los programadores pueden crear documentos, navegar en su estructura, añadir, crear o modificar elementos y su contenido. Se puede acceder a cualquier cosa que se encuentre en un documento HTML o XML, y se puede modificar, eliminar o añadir usando el Modelo de Objetos del Documento, salvo algunas excepciones.

El DOM proporciona una interfaz estándar de programación que puede utilizarse en diferentes entornos y aplicaciones. Una de las características más beneficiosas es que el DOM puede ser utilizado en cualquier lenguaje de programación.

El DOM, en general, tiene mucha similitud a la estructura del documento al que modeliza, por ejemplo:

```
<TABLE>
<TBODY>
<TR>
<TD>Shady Grove</TD>
<TD>Aeolian</TD>
</TR>
<TR>
<TD>Over the River, Charlie</TD>
<TD>Dorian</TD>
</TR>
</TBODY>
</TABLE>
```

La tabla anterior corresponde a un documento HTML. Y en la siguiente imagen se puede ver la similitud de la que se habló anteriormente:



En la imagen anterior se observa como el DOM modela el documento en una especie de árbol, esta representación se le llama “modelo de estructura”. Los nodos que se observan corresponden a objetos, los cuales pueden tener atributos y funciones.

¿Qué es XML?

XML (Extensible Markup Language) es un lenguaje de marcado que define un conjunto de reglas para la codificación de un documento, con el código escrito en este se pueden analizar datos o la lectura de datos creados en otras computadoras. Además, en XML se puede establecer diferentes elementos para establecer un formato y poder así, crear un lenguaje propio.

Algunas de las características de este lenguaje es lo fácil que es compartir datos, tanto en sistemas informáticos como en bases de datos que contengan información en un formato incompatible y, además, debido a que los datos se almacenan en un formato de texto simple permite compartirlos con otras aplicaciones. Esta es una de las características más populares de este lenguaje ya que anteriormente uno de los desafíos más difíciles para los desarrolladores era el intercambio de datos a través de internet lo que este lenguaje hace que esto sea menos complicado porque los datos pueden ser leídos por aplicaciones incompatibles.

Ya dentro del documento en sí, este se compone de un conjunto de etiquetas que proporcionan información acerca de la información que se quiere procesar. Dentro

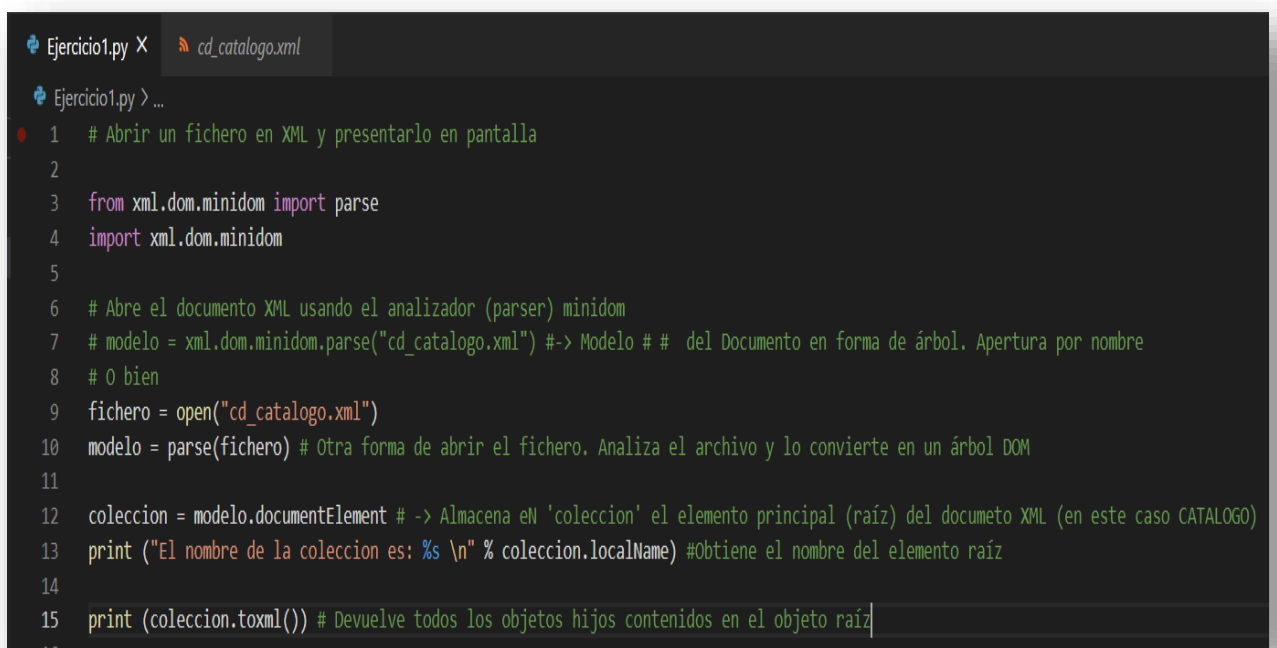
de estas etiquetas puede haber más etiquetas que puede estar anidadas o no. Estas etiquetas deben estar bien cerradas, nunca acortarse, cada una tiene su inicio y fin, y se eliminar alguna etiqueta podría generarse un error que cortaría el proceso de almacenamiento de información en cualquier tipo de lenguaje de programación que no se podría seguir procesando.

xml.dom para analizar documentos XML

Utilizar DOM para analizar un documento XML, permite leer todo el documento de una vez y almacenar todos los elementos que forman parte de ese documento se guardan en memoria en una estructura de árbol. DOM se utiliza para leer y modificar el documento, así como también la estructura de los elementos para escribir el contenido del archivo XML.

Ejemplos de análisis de documentos XML con DOC en Python

Ejemplo 1: Abrir un fichero en XML y presentarlo en pantalla



```
Ejercicio1.py X cd_catalogo.xml
Ejercicio1.py > ...
1 # Abrir un fichero en XML y presentarlo en pantalla
2
3 from xml.dom.minidom import parse
4 import xml.dom.minidom
5
6 # Abre el documento XML usando el analizador (parser) minidom
7 # modelo = xml.dom.minidom.parse("cd_catalogo.xml") #-> Modelo ## del Documento en forma de árbol. Apertura por nombre
8 # O bien
9 fichero = open("cd_catalogo.xml")
10 modelo = parse(fichero) # Otra forma de abrir el fichero. Analiza el archivo y lo convierte en un árbol DOM
11
12 coleccion = modelo.documentElement # -> Almacena en 'coleccion' el elemento principal (raíz) del documento XML (en este caso CATALOGO)
13 print ("El nombre de la coleccion es: %s \n" % coleccion.localName) #Obtiene el nombre del elemento raíz
14
15 print (coleccion.toxml()) # Devuelve todos los objetos hijos contenidos en el objeto raíz
```

Salida:

```
PS C:\Users\gerso\Desktop\Lab_IPC2_TareaDos> & C:/Users/gerso/AppData,
El nombre de la coleccion es: CATALOGO

<CATALOGO>
  <CD>
    <TITULO>Empire Burlesque</TITULO>
    <ARTISTA>Bob Dylan</ARTISTA>
    <PAIS>USA</PAIS>
    <COMPANNIA>Columbia</COMPANNIA>
    <PRECIO>10.90</PRECIO>
    <ANNO>1985</ANNO>
  </CD>
  <CD>
    <TITULO>Hide your heart</TITULO>
    <ARTISTA>Bonnie Tylor</ARTISTA>
    <PAIS>UK</PAIS>
    <COMPANNIA>CBS Records</COMPANNIA>
    <PRECIO>9.90</PRECIO>
    <ANNO>1988</ANNO>
  </CD>
  <CD>
    <TITULO>Greatest Hits</TITULO>
    <ARTISTA>Dolly Parton</ARTISTA>
    <PAIS>USA</PAIS>
    <COMPANNIA>RCA</COMPANNIA>
    <PRECIO>9.90</PRECIO>
    <ANNO>1982</ANNO>
  </CD>
```

Ejercicio 2: Crear un fichero XML y guardarlo.

```
Ejercicio1.py • Ejercicio2.py X ordenadores.xml
Ejercicio2.py > ...
1  from xml.dom import minidom
2
3  Ordenador1 = ['Pentium M', '512MB']
4  Ordenador2 = ['Pentium Core 2', '1024MB']
5  Ordenador3 = ['Pentium Core Duo', '1024MB']
6  listaOrdenadores = [Ordenador1, Ordenador2, Ordenador3]
7
8  # Abro un modelo DOM en modo implementar
9  DOMimpl = minidom.getDOMImplementation()
10
11 # Crear el documento con la etiqueta principal estacionesTrabajo (etiqueta raiz)
12 # No se crear el documento como tal, sino que solo almacena en "xmldoc" la información que va a tener el documento
13 xmldoc = DOMimpl.createDocument(None, "estacionesTrabajo", None)
14 doc_root = xmldoc.documentElement # Almacena en 'doc_root' el nombre del elemento principal del documento
15
16 # Recorro la lista de ordenadores
17 for ordenador in listaOrdenadores:
18
19     #Crear Nodo...
20     nodo = xmldoc.createElement("Ordenador") # Crea una etiqueta 'Ordenador', la cual se pasa como parámetro
21
22     # Crear un subnodo, llamado procesador
23     elemento = xmldoc.createElement('Procesador')
```


Información almacenada en el fichero:

```
Ejercicio1.py • Ejercicio2.py ordenadores.xml X
ordenadores.xml
1 <?xml version="1.0" ?>
2 <estacionesTrabajo>
3   <Ordenador>
4     <Procesador>Pentium M</Procesador>
5     <Memoria>512MB</Memoria>
6   </Ordenador>
7   <Ordenador>
8     <Procesador>Pentium Core 2</Procesador>
9     <Memoria>1024MB</Memoria>
10  </Ordenador>
11  <Ordenador>
12    <Procesador>Pentium Core Duo</Procesador>
13    <Memoria>1024MB</Memoria>
14  </Ordenador>
15 </estacionesTrabajo>
```

Ejercicio 3: Analizar el contenido de un documento XML.

```
Ejercicio1.py Ejercicio3.py X cd_catalogo.xml Ejercicio2.py Release Notes: 1.53.0
Ejercicio3.py > ...
1 # Extraer un fichero y modificarlo
2
3 import xml.dom.minidom
4
5 # El método 'parse' lee todo el archivo, analiza la estructura contenida en XML
6 # y la reproduce en una representación accesible para Python, es decir,
7 # crear la estructura de árbol, al cual hay que referirse para extraer información
8 # contenida dentro del archivo XML.
9 arbolDom = xml.dom.minidom.parse("cd_catalogo.xml")
10
11 # Obtiene el elemento principal del documento XML, es decir, aquel
12 # que contiene a todos los demás elementos
13 objetoPrincipal = arbolDom.documentElement
14
15 # Imprime el nombre del objeto principal
16 print("El nombre del objeto raíz es ", objetoPrincipal.localName)
17
18 # Obtiene todos los elementos con el nombre de etiqueta "CD" que son hijos
19 # del objetoPrincipal
20 listaEtiquetas = objetoPrincipal.getElementsByTagName("CD")
21
22 for hijoCD in listaEtiquetas:
23     print("*****CD*****")
24     titulo = hijoCD.getElementsByTagName("TITULO")[0]
25     print("Título: ", titulo.childNodes[0].data) # En este caso es la posición 0, ya que obtiene todos los nodos hijos, incluyendo el nodo
26     # el nodo desde el que se invocó y obtiene su contenido
27
28     # Se coloca [0] ya que estamos accediendo a un elemento de una lista
29     artista = hijoCD.getElementsByTagName("ARTISTA")[0]
30     print("Artista: ", artista.childNodes[0].data)
31
32     pais = hijoCD.getElementsByTagName("PAIS")[0]
33     print("País: ", pais.childNodes[0].data)
```

```
Ejercicio1.py  Ejercicio3.py X  cd_catalogo.xml  Ejercicio2.py  Release Notes: 1.53.0
Ejercicio3.py > ...
17
18 # Obtiene todos los elementos con el nombre de etiqueta "CD" que son hijos
19 # del objetoPrincipal
20 listaEtiquetas = objetoPrincipal.getElementsByTagName("CD")
21
22 for hijoCD in listaEtiquetas:
23     print("*****CD*****")
24     titulo = hijoCD.getElementsByTagName("TITULO")[0]
25     print("Título: ", titulo.childNodes[0].data) # En este caso es la posición 0, ya que obtiene todos los nodos hijos, incluyendo el nodo
26     | | | | | | | | | | # el nodo desde el que se invocó y obtiene su contenido
27
28     # Se coloca [0] ya que estamos accediendo a un elemento de una lista
29     artista = hijoCD.getElementsByTagName("ARTISTA")[0]
30     print("Artista: ", artista.childNodes[0].data)
31
32     pais = hijoCD.getElementsByTagName("PAIS")[0]
33     print("País: ", pais.childNodes[0].data)
34
35     precio = hijoCD.getElementsByTagName("PRECIO")[0]
36     print("Precio: ", precio.childNodes[0].data)
37
38     anno = hijoCD.getElementsByTagName("ANNO")[0]
39     print("Año: ", anno.childNodes[0].data)
40
41     print()
```

Salida:

```
*****CD*****
Título: Picture book
Artista: Simply Red
País: EU
Precio: 7.20
Año: 1985

*****CD*****
Título: Red
Artista: The Communards
País: UK
Precio: 7.80
Año: 1987

*****CD*****
Título: Unchain my heart
Artista: Joe Cocker
País: USA
Precio: 8.20
Año: 1987
```


xPath

Es un módulo de la librería `xml.etree.ElementTree` que permite acceder a elementos de un árbol por medio de expresiones. Sirve para definir un subconjunto de nodos de XML e identificar partes específicas de un documento XML.

Utiliza un tipo de notación similar a las rutas de ficheros, pero haciendo referencia a los nodos de un documento XML.

Rutas

Sirven para localizar nodos dentro de una estructura jerárquica propia de XML. Cada expresión de ruta consta de varios pasos que se separan por la barra simple (/) o las barras dobles (//).

- La barra simple significa que el siguiente paso se encuentra adyacente a la jerarquía de nodos, es decir, es un hijo directo del nodo.
- La barra doble permite localizar cualquier nodo que sea descendiente sin importar el nivel.

Las rutas pueden ser:

- Absolutas: Empiezan con barra y hacen referencia al nodo raíz. Ejemplo: `/libro/titulo`. Para acceder a un elemento que se encuentre de la forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<libro>
|   <titulo>Computación</titulo>
</libro>
```

- Relativas: Hacen referencia al nodo en el que nos encontramos. Por ejemplo: `libro/titulo`. Para acceder a un elemento que se encuentre de la forma:

```
<?xml version="1.0" encoding="UTF-8"?>
<biblioteca>
  <libro>
    <titulo>Computación</titulo>
  </libro>
</biblioteca>
```

Formas de seleccionar un nodo

Sintaxis	Descripción
/	Selecciona desde el nodo raíz.
//	Selecciona nodos desde el nodo contextual (Sin importar donde se encuentren).
.	Selecciona el nodo actual. Esto es principalmente útil al comienzo de la ruta, para indicar que es una ruta relativa.
..	Selecciona el elemento padre.
//*	Selecciona todos los nodos del documento.
*	Selecciona todos los nodos elemento.
[@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 comilla
[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 ultima, o la posición relativa con respecto a la ultima posición last()-1.

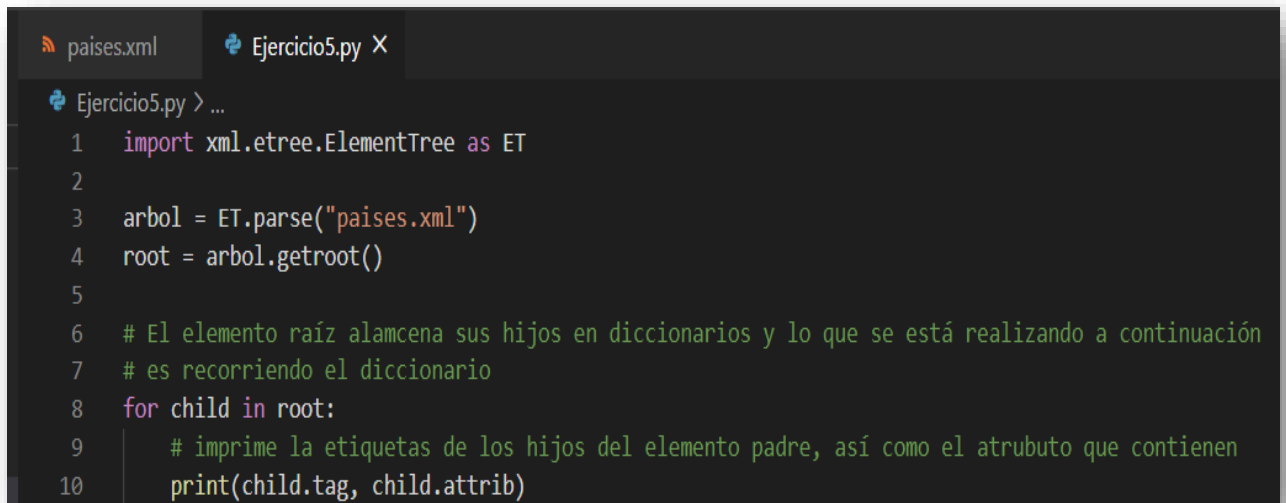
Ejemplos 4: Obtener el nombre del objeto raíz.

```
Ejemplo4.py X libros.xml
Ejemplo4.py > ...
1  # Obtener el nombre del objeto raíz
2
3  import xml.etree.ElementTree as ET
4
5  # El método 'parse' lee todo el archivo, analiza la estructura contenida en XML
6  # y la reproduce en una representación accesible para Python, es decir,
7  # crear la estructura de árbol, al cual hay que referirse para extraer información
8  # contenida dentro del archivo XML.
9  arbol = ET.parse("libros.xml")
10
11 #Obtiene el elemento raíz
12 root = arbol.getroot()
13
14 # Obtiene la etiqueta del elemento raíz
15 nombre = root.tag
16 print(nombre)
```

Salida:

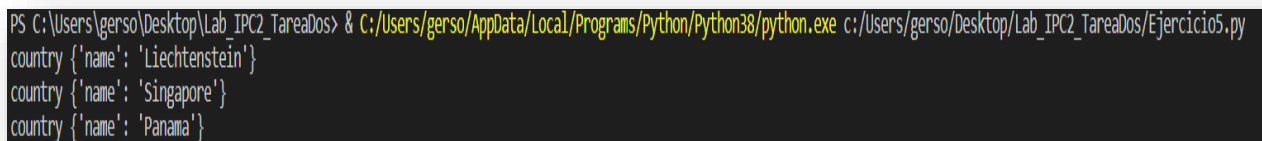
```
PS C:\Users\gerso\Desktop\Lab_IPC2_TareasDos> & C:/Users/gerso/AppData/Local/Programs/Python/Python38/python.exe c:/Users/gerso/Desktop/
Catalog
```

Ejemplo 5: Obtener los elementos hijos del elemento raíz.



```
países.xml  Ejercicio5.py X
Ejercicio5.py > ...
1  import xml.etree.ElementTree as ET
2
3  arbol = ET.parse("países.xml")
4  root = arbol.getroot()
5
6  # El elemento raíz almacena sus hijos en diccionarios y lo que se está realizando a continuación
7  # es recorriendo el diccionario
8  for child in root:
9      # imprime la etiquetas de los hijos del elemento padre, así como el atributo que contienen
10     print(child.tag, child.attrib)
```

Salida:



```
PS C:\Users\gerso\Desktop\Lab_IPC2_TareaDos> & C:/Users/gerso/AppData/Local/Programs/Python/Python38/python.exe c:/Users/gerso/Desktop/Lab_IPC2_TareaDos/Ejercicio5.py
country {'name': 'Liechtenstein'}
country {'name': 'Singapore'}
country {'name': 'Panama'}
```