



ESCUELA DE  
INGENIERÍA EN CIENCIAS Y SISTEMAS  
FACULTAD DE INGENIERÍA  
UNIVERSIDAD DE SAN CARLOS DE GUATEMALA

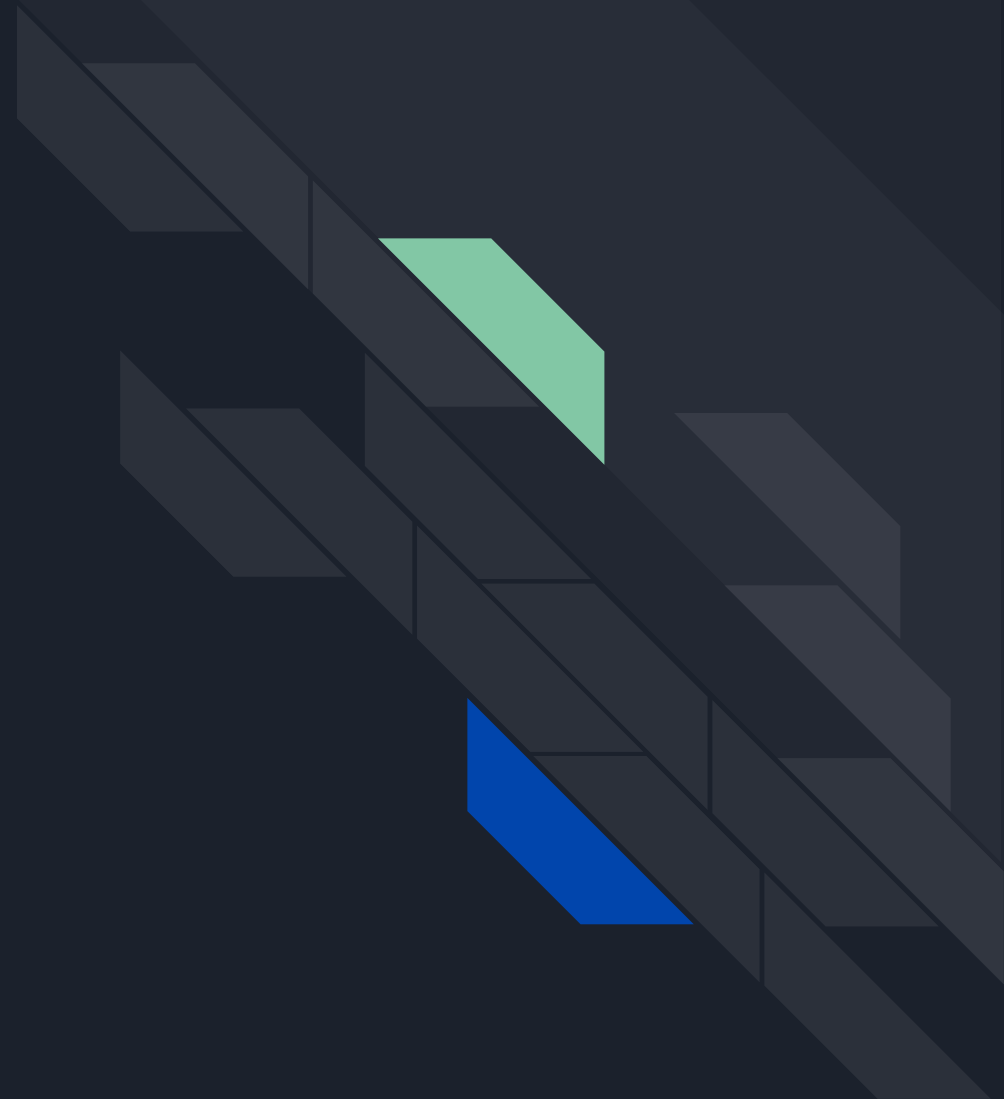


<b>Día, Fecha:</b>	Jueves, 08/08/2024
<b>Hora de inicio:</b>	10:40 - 12:20

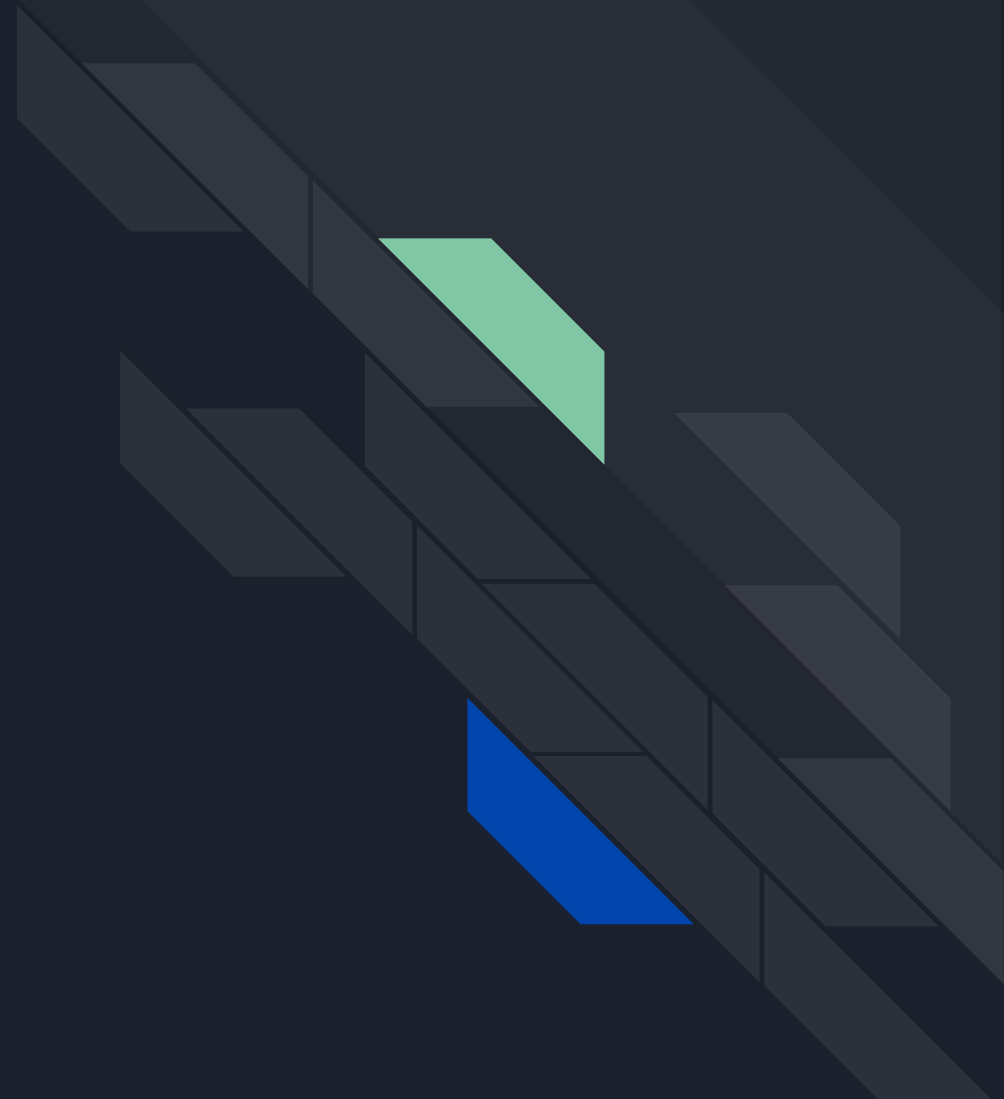
# Introducción a la Programación y Computación 2 [P]

Denilson Florentín de León Aguilar

# Práctica 1



# Proyecto 1



# UNIDAD 3

Procesamiento de datos XML



# Qué es DOM?

DOM o Document Object Model es un lenguaje para acceder y modificar documentos XML.

Con el Modelo de Objetos del Documento cualquier desarrollador puede construir documentos, navegar por su estructura, y añadir, modificar o eliminar elementos y contenido.

El mejor ejemplo del modelo DOM es HTML.



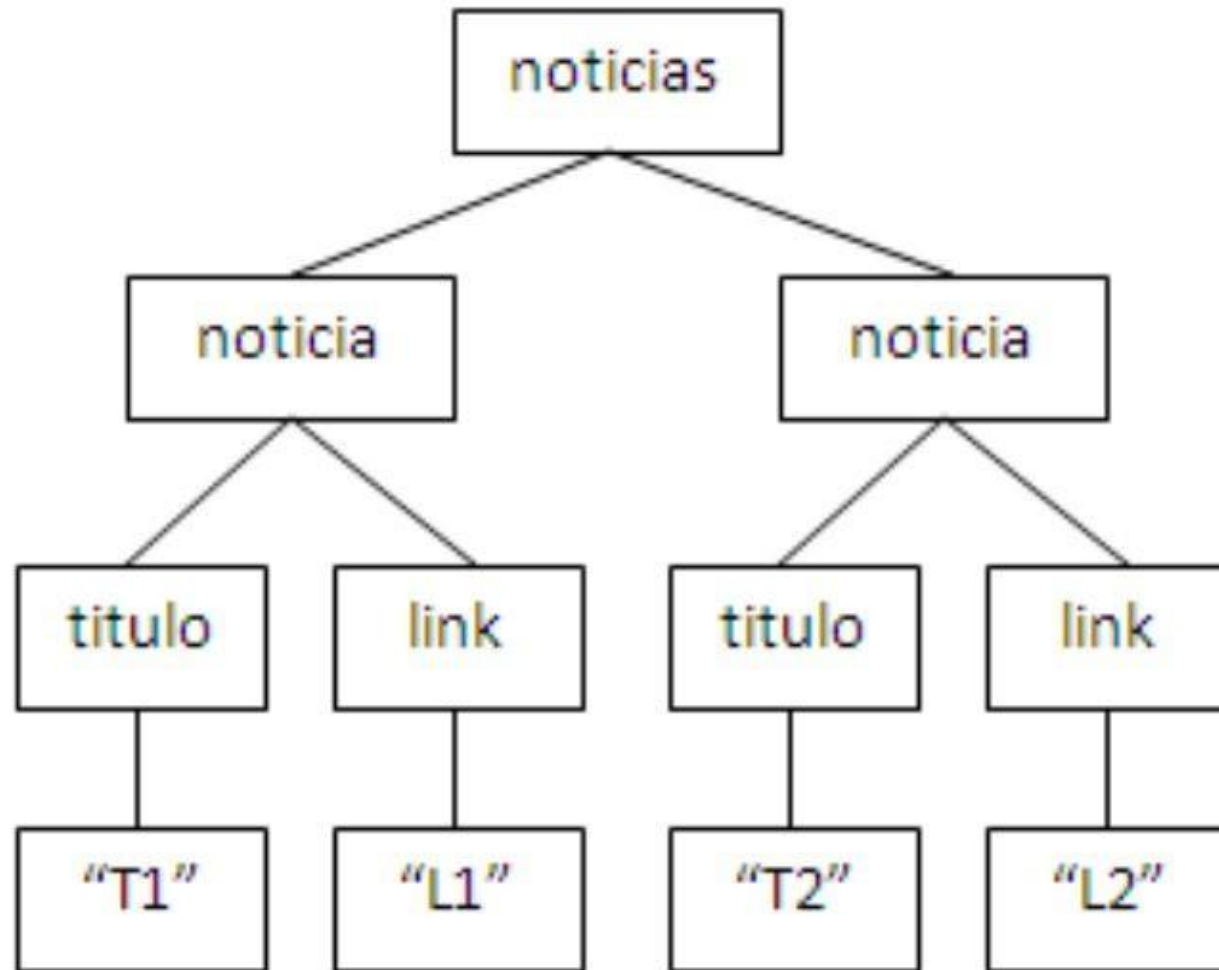
# Ejemplo del modelo DOM en archivo XML

```
1  <noticias>
2      <noticia>
3          <titulo>T1</titulo>
4          <link>L1</link>
5      </noticia>
6      <noticia>
7          <titulo>T2</titulo>
8          <link>L2</link>
9      </noticia>
10 </noticias>
```

# ¿Qué es DOM?

- En el DOM, los documentos tienen una estructura lógica que es muy parecida a un árbol sin embargo, no especifica que debe ser implementado como un árbol obligatoriamente.
- El DOM es un modelo lógico que puede implementarse de cualquier manera que sea conveniente.
- Una propiedad importante de los modelos de estructura del DOM es algo llamado “isomorfismo estructural”, el cual significa que si dos implementaciones cualesquiera del Modelo de Objetos del Documento se usan para crear una representación del mismo documento, ambas crearán el mismo modelo de estructura, con exactamente los mismos objetos y relaciones.

# Representación gráfica del ejemplo anterior








# Orígen del modelo DOM

El DOM se originó como una especificación para permitir que los programas Java y los scripts de JavaScript fueran portables entre los navegadores web. La primera especificación del modelo surgió en 1997 llamada DOM Level 1 del cual se realizó una actualización en el año 2000.

Se han realizado nuevas especificaciones al modelo agregando más funcionalidades para el procesamiento de archivos con propiedades que necesitan de otras funciones.






# Orígen del modelo DOM

Los elementos que componen al archivo XML son tratados por el modelo DOM como objetos. Los atributos que contienen estos objetos son tratados por el modelo DOM como cadenas de caracteres o strings, por lo que tenemos que tomarlo en cuenta al realizar operaciones con los datos recién obtenidos del archivo de entrada.

Los atributos en el modelo DOM también pueden ser tratados como nodos en vez de strings aunque es bastante inusual hacerlo.



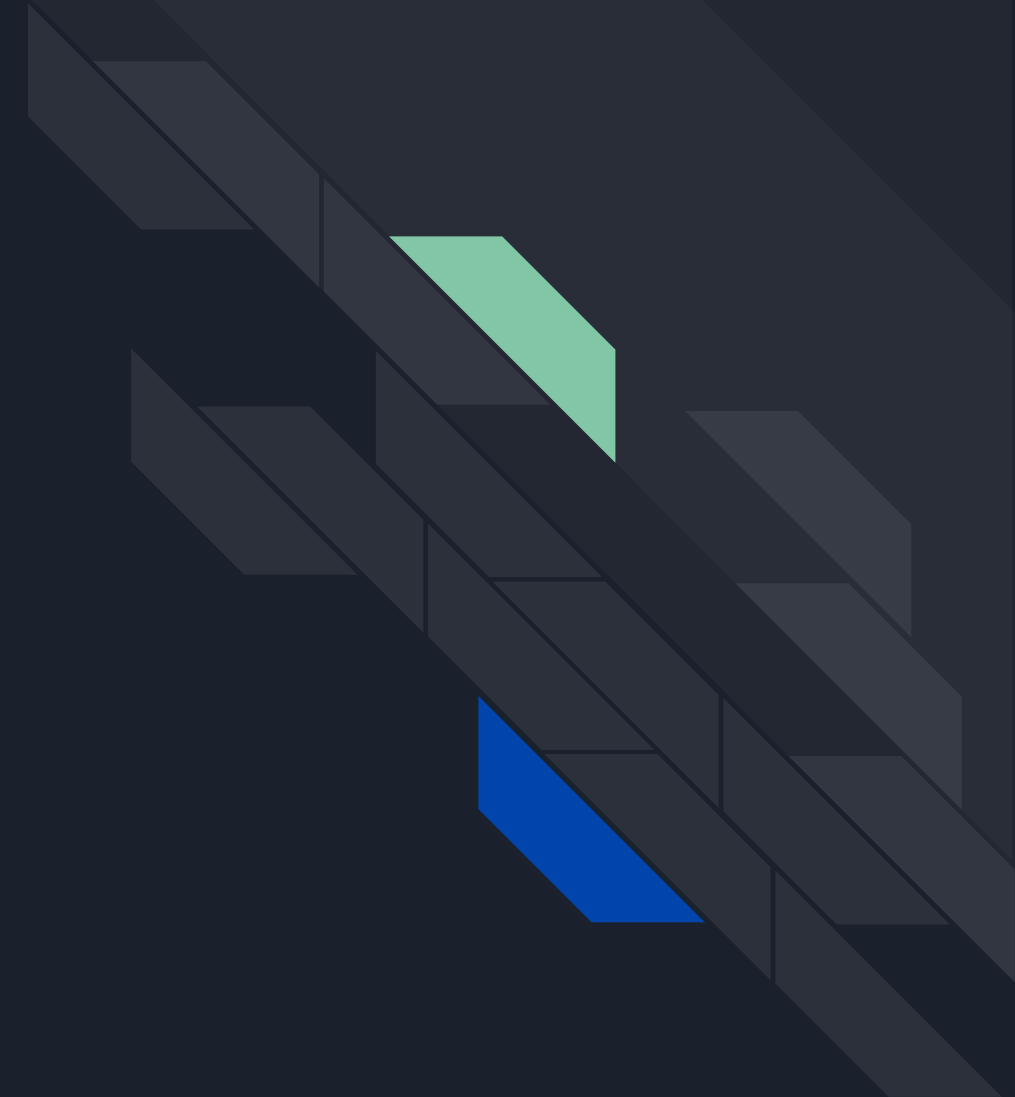
# Objetos del modelo DOM

Interfaz	Sección	Propósito
DOMImplementation	Objetos DOMImplementation	Interfaz para las implementaciones subyacentes.
Node	Objetos Nodo	Interfaz base para la mayoría de objetos en un documento.
NodeList	Objetos NodeList	Interfaz para una secuencia de nodos.
DocumentType	Objetos DocumentType	Información acerca de la declaraciones necesarias para procesar un documento.

# Objetos del modelo DOM

Document	Objetos Documento	Objeto que representa un documento entero.
Element	Objetos Elemento	Nodos elemento en la jerarquía del documento.
Attr	Objetos Atributo	Nodos de los valores de los atributos en los elementos nodo.
Comment	Objetos Comentario	Representación de los comentarios en el documento fuente.
Text	Objetos Texto y CDATASection	Nodos con contenido textual del documento.
ProcessingInstruction	Objetos ProcessingInstruction	Representación de instrucción del procesamiento.

# Recordatorio: Captura de pantalla



# MiniDom

MiniDOM o Minimal DOM Implementation es una simplificada del modelo DOM que puede utilizarse en Python.

Para utilizarlo es necesario importarlo desde el módulo xml.dom.

Este modelo usa la función “parse” para crear un objeto DOM directamente desde nuestro archivo XML. La función tiene la siguiente sintaxis:

```
xml.dom.minidom.parse(filename_or_file[, parser[, bufsize]])
```

# MiniDom

La propiedad “file\_or\_filename” debe contener una ruta hacia el archivo XML o hacia un objeto de tipo archivo. Esta función retorna un documento el cual ya podremos procesar como un XML.

Para empezar a buscar elementos con el nombre específico de una tag utilizamos el siguiente comando: `getElementByTagName()`.

Debido a que cada nodo se trata como un objeto, podemos acceder a los valores de los tags usando las propiedades del objeto.

En el ejemplo siguiente podemos observar el procesamiento de un archivo XML.

# Ejemplo Minidom

```
<data>
  <items>
    <item name="item1">item1abc</item>
    <item name="item2">item2abc</item>
  </items>
</data>
```



# Ejemplo Minidom

Primero debemos importar la librería y mediante el comando “parse” creamos un objeto documento con la ruta del archivo XML.

```
from xml.dom import minidom
```

```
mydoc = minidom.parse('items.xml')
```

Para obtener los nodos con el tag “item” utilizamos el comando “getElementsByTagName”.

```
items = mydoc.getElementsByTagName('item')
```

# Ejemplo Minidom

Podemos obtener los valores de los atributos del tag de la siguiente manera:

```
print('Item #2 attribute:')  
print(items[1].attributes['name'].value)
```

Con lo cual tendremos de resultado:

```
Item #2 attribute:  
item2
```

Para imprimir todos los valores posibles de los atributos podemos usar instrucciones cíclicas.

```
print('\nAll attributes:')  
for elem in items:  
    print(elem.attributes['name'].value)
```

# Ejemplo Minidom

Lo cual nos dará el siguiente resultado:

```
All attributes:  
item1  
item2
```

Para obtener los valores dentro del tag tenemos dos opciones. Utilizar el comando “firstChild” o utilizar `childNodes[x]` donde `x` es el índice. Usualmente este índice será 0.

```
print('\nItem #2 data:')  
print(items[1].firstChild.data)  
print(items[1].childNodes[0].data)
```

De cualquier forma, el resultado será el mismo:

```
Item #2 data:  
item2abc  
item2abc
```

# Ejemplo Minidom

Ahora bien, si lo que desea es obtener todos los valores posibles dentro del tag especificado, también podemos usar instrucciones cíclicas de la siguiente manera:

```
print('\nAll item data:')  
for elem in items:  
    print(elem.firstChild.data)
```

El resultado será el siguiente:

```
All item data:  
item1abc  
item2abc
```




# Element Tree

Es otra alternativa simple para procesar archivos XML que se puede utilizar en Python. Igual que en MiniDOM, debemos primero importar la librería para utilizar el módulo.

Al utilizar Element Tree, se crea una estructura en forma de árbol utilizando el comando “parse” y se obtiene un elemento raíz. Una vez identificado este elemento, ya podremos recorrer el árbol debido a que todos los nodos del árbol están conectados.

Utilizaremos el mismo archivo de ejemplo que en MiniDOM para ejemplificar algunas funciones básicas de Element Tree.



# Element Tree

Primero debemos importar Element Tree, y mediante el comando “parse” creamos un objeto de tipo árbol. Con el comando “getroot()” obtenemos la raíz.

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

En este caso, con el Element Tree no utilizaremos cadenas para buscar los tags específicos, sino que utilizaremos una notación en la que especificamos índices de subárboles y posiciones dentro de sus nodos.

Utilizaremos el comando “root[X][Y].attribute”.

Donde X es el índice del nodo padre que buscamos y Y será el índice del subnodo.

# Element Tree

Para buscar un elemento en especial:

```
print('Item #2 attribute:')  
print(root[0][1].attrib)
```

Obtenemos el siguiente resultado:

```
Item #2 attribute:  
item2
```

Para obtener todos los valores de cierto atributo también utilizamos instrucciones cíclicas con la característica de que utilizamos los identificadores root para la raíz, element para los atributos en tags y subelement para los valores.



# Element Tree

Para buscar un elemento en especial:

```
print('Item #2 attribute:')  
print(root[0][1].attrib)
```

Obtenemos el siguiente resultado:

```
Item #2 attribute:  
item2
```

Para obtener todos los valores de cierto atributo también utilizamos instrucciones cíclicas con la característica de que utilizamos los identificadores root para la raíz, element para los atributos en tags y subelement para los valores.



# Element Tree

```
print('\nAll attributes:')  
for elem in root:  
    for subelem in elem:  
        print(subelem.attrib)
```

Obtenemos el siguiente resultado:

```
All attributes:  
item1  
item2
```

Para obtener un valor específico:

```
print('\nItem #2 data:')  
print(root[0][1].text)
```

# Element Tree

El resultado sería:

```
Item #2 data:  
item2abc
```

Para obtener todos los valores de las tags

```
print('\nAll item data:')  
for elem in root:  
    for subelem in elem:  
        print(subelem.text)
```

Y el resultado será:

```
All item data:  
item1abc  
item2abc
```

# Escribir Documentos XML (Element Tree)

ElementTree también es ideal para escribir datos en archivos XML. El siguiente código muestra cómo crear un archivo XML con la misma estructura que el archivo que usamos en los ejemplos anteriores.

1. Cree un elemento, que actuará como nuestro elemento raíz. En este caso se llamará "data".
2. Una vez que tenemos nuestro elemento raíz, podemos crear subelementos usando la SubElement función sintaxis:

```
import xml.etree.ElementTree as ET

# create the file structure
data = ET.Element('data')
```

```
items = ET.SubElement(data, 'items')
```

# Escribir Documentos XML (Element Tree)

Aquí parent está el nodo principal al que conectarse, attrib hay un diccionario que contiene los atributos del elemento y extra hay argumentos de palabras clave adicionales. Esta función nos devuelve un elemento, que puede usarse para adjuntar otros subelementos, como lo hacemos en las siguientes líneas al pasar elementos al SubElement constructor.

3. Aunque podemos agregar nuestros atributos con la SubElement función, también podemos usar la set() función, como hacemos en el siguiente código. El texto del elemento se crea con la propiedad text del Elemento objeto.

```
item1 = ET.SubElement(items, 'item')
item2 = ET.SubElement(items, 'item')
item1.set('name', 'item1')
item2.set('name', 'item2')
item1.text = 'item1abc'
item2.text = 'item2abc'
```

# Escribir Documentos XML (Element Tree)

4. En las últimas 3 líneas del código siguiente, creamos una cadena a partir del árbol XML y escribimos esos datos en un archivo que abrimos.

```
# create a new XML file with the results  
mydata = ET.tostring(data)  
myfile = open("items2.xml", "w")  
myfile.write(mydata)
```

La ejecución de este código dará como resultado un nuevo archivo, "items2.xml", que debería ser equivalente al archivo "items.xml" original, al menos en términos de la estructura de datos XML. Probablemente notará que la cadena resultante es solo una línea y no contiene sangría,



# ¿DUDAS?

Procesamiento de datos XML

